Cryptanalysis of Symmetric Cryptographic Primitives

Aleksandar Kircanski

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy (Computer Science) at
Concordia University
Montréal, Québec, Canada

June 2013

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Aleksandar Kircanski

Entitled: Cryptanalysis of Symmetric Cryptographic Primitives

and submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY (Computer Science)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. W.F. Xie

_____ External Examiner
Dr. A. Miri

_____ External to Program.
Dr. A. Hamou-Lhadj

_____ Examiner
Dr. P. Grogono

_____ Examiner
Dr. D. Ford

_____ Thesis Supervisor
Dr. A. Youssef

Approved by _____
Chair of Department or Graduate Program Director
Dr. V. Haarslev , Graduate Program Director

September 9, 2013 _____
Dr. C. Trueman, Interim Dean
Faculty of Engineering & Computer Science

Abstract

Cryptanalysis of Symmetric Cryptographic Primitives

Aleksandar Kircanski, Ph.D.
Concordia University, 2013

Symmetric key cryptographic primitives are the essential building blocks in modern information security systems. The overall security of such systems is crucially dependent on these mathematical functions, which makes the analysis of symmetric key primitives a goal of critical importance. The security argument for the majority of such primitives in use is only a heuristic one and therefore their respective security evaluation continually remains an open question. In this thesis, we provide cryptanalytic results for several relevant cryptographic hash functions and stream ciphers.

First, we provide results concerning two hash functions: HAS-160 and SM3. In particular, we develop a new heuristic for finding compatible differential paths and apply it to the the Korean hash function standard HAS-160. Our heuristic leads to a practical second order collision attack over all of the HAS-160 function steps, which is the first practical-complexity distinguisher on this function. An example of a colliding quartet is provided. In case of SM3, which is a design that builds upon the SHA-2 hash and is published by the Chinese Commercial Cryptography Administration Office for the use in the electronic authentication service system, we study second order collision attacks over reduced-round versions and point out a structural slide-rotational property that exists in the function.

Next, we examine the security of the following three stream ciphers: Loiss, SNOW 3G and SNOW 2.0. Loiss stream cipher is designed by Dengguo Feng *et al.* aiming to be implemented in byte-oriented processors. By exploiting some differential properties of a particular component utilized in the cipher, we provide an attack of a practical complexity on Loiss in the related-key model. As confirmed by our experimental results, our attack recovers 92 bits of the 128-bit key in less than one hour on a PC with 3

GHz Intel Pentium 4 processor. SNOW 3G stream cipher is used in 3rd Generation Partnership Project (3GPP) and the SNOW 2.0 cipher is an ISO/IEC standard (IS 18033-4). For both of these two ciphers, we show that the initialization procedure admits a sliding property, resulting in several sets of related-key pairs. In addition to allowing related-key key recovery attacks against SNOW 2.0 with 256-bit keys, the presented properties reveal non-random behavior of the primitives, yield related-key distinguishers for the two ciphers and question the validity of the security proofs of protocols based on the assumption that these ciphers behave like perfect random functions of the key-IV.

Finally, we provide differential fault analysis attacks against two stream ciphers, namely, HC-128 and Rabbit. In this type of attacks, the attacker is assumed to have physical influence over the device that performs the encryption and is able to introduce random faults into the computational process. In case of HC-128, the fault model in which we analyze the cipher is the one in which the attacker is able to fault a random word of the inner state of the cipher but cannot control its exact location nor its new faulted value. Our attack requires about 7968 faults and recovers the complete internal state of HC-128 by solving a set of 32 systems of linear equations over Z_2 in 1024 variables. In case of Rabbit stream cipher, the fault model in which the cipher is analyzed is the one in which a random bit of the internal state of the cipher is faulted, however, without control over the location of the injected fault. Our attack requires around $128 - 256$ faults, precomputed table of size $2^{41.6}$ bytes and recovers the complete internal state of Rabbit in about 2^{38} steps.

Acknowledgments

I would like to express gratitude to my supervisor Dr. Amr Youssef for guiding me throughout the years and unselfishly sharing his knowledge, enthusiasm and ideas with me. After each single meeting with my supervisor I felt re-energized to attack cryptographic problems. The amount of hours we spent discussing my research topic made this work meaningful and possible.

My gratitude goes to my mother, father, sister and my wife during the past years for providing me with the everlasting love and understanding. Without their constant support, this work would not have been possible.

Next, I would like to thank Dr. Alex Biryukov who hosted me during an exciting three month research visit in 2012 at the Laboratory of Algorithmics, Cryptology and Security (LACS), at University of Luxembourg where we managed to produce a cryptanalytic result in only a short period of time.

I am indebted to my lab colleagues at the CIISE Crypto Lab for making my Ph.D. journey a personal and warm experience. Next, I thank my friend Jean Grondin for discussions on the graph theory problems related to my research in the cafes of Côtes-des-Neiges, Montréal. Finally, my gratitude goes to Benoit Coulombe for all the time spent in Dieu du Ciel.

ALEKSANDAR KIRCANSKI, 2013

He had been struggling with those 'empties' forever, and the way I see it, without any benefit to humanity or himself. In his shoes, I would have said leave it long ago and gone to work on something else for the same money. Of course, on the other hand, if you think about it, an 'empty' really is something mysterious and maybe even incomprehensible. I've handled quite a few of them, but I'm still surprised every time I see one.

– Arkady and Boris Strugatsky, *Roadside Picnic*, 1971

Table of Contents

List of Figures

List of Tables

1

Introduction

1.1 Background and motivation

Historically, the goal of cryptography was to allow two parties to communicate over an insecure channel with an adversary not able to understand what is being said. Nowdays, the goal of cryptography, understood in a broader sense, is to provide means to enforce diverse security goals such as integrity, authenticity and non-repudiation [104].

Modern information security systems widely implement cryptographic functions to enforce such security goals and the very basic constructions that are employed for this purpose are called cryptographic primitives. In the past twenty five years, a variety of efficient cryptographic primitives has been developed. The cryptographic primitives that use the same cryptographic key for both encryption of plaintext and decryption of ciphertext are called symmetric-key primitives. Examples of such primitives are block ciphers and stream ciphers. Although hash functions do not fall into the category of symmetric key primitives in the strict sense since they are keyless, many currently used hash functions are constructed based on block ciphers and in this sense are naturally related to the area of symmetric key cryptography.

Many of the symmetric-based cryptosystems designed in the last fifteen years owe their existance to public competitions for cryptographic primitives. In such a competition, a standardization body issues a submission request for the cryptographic primitive to be developed and coordinates the process of analysis/standardization of the primitive. So far, NIST has organized two such competitions, one for block cipher standard AES (1997-2000) and one for the cryptographic hashing standard SHA-3 (2007-2012). Another such competition called eStream (2004-2007) was organized for new stream cipher designs by ECRYPT,

the Network of Excellence within the European Information Societies Technology. These competitions sparked a great deal of cryptanalytic and design efforts and also had a strong impact on communities outside cryptography.

The security of a particular symmetric cryptographic primitive relies on the fact that experienced cryptanalysts have not been able to breach the security claims that come with the primitive in question. In other words, in most of the cases, there exists no proof that the cryptosystem is secure and the security claim is only a heuristic one. Devising cryptanalytic attacks can thus be seen as providing better lower bounds on the attack complexities for the primitive in question.

In the last 15 years, the cryptographic community witnessed several interesting cryptanalytic attacks. As for cryptographic hash functions, attacks against MD5 and SHA-1 hash functions [130] by Wang based on differential cryptanalysis have shown that MD5 and SHA-1 are not collision resistant. Given these cryptanalytic results as well as the dramatic attacks that built upon collision attacks (e.g., the construction of two X.509 certificates containing identical signatures and differing only in the public keys [128]), the National Institute of Standards and Technology (NIST) started a public hash function competition SHA-3 that resulted in choosing Keccak as the new hash function standard. In the area of stream ciphers, Fluhrer, Mantin and Shamir described an attack against RC4 used in the WEP mode, which allowed breaking the WEP key in real time. Finally, as for block ciphers, AES block cipher that has been believed to be secure for around 10 years, is now shown to be susceptible to boomerang related-key analysis [20] (Biryukov, Khovratovich and Nikolić) and also biclique attacks [29] (Bogdanov, Khovratovich and Rechberger) in the single-key model. It should be noted that, while the MD5/SHA-1 and the WEP attack are practically feasible, the AES analysis did not provide attacks that threaten security in practice so far.

Motivation: The cryptanalytic work done in this thesis is motivated by the fact that the security of symmetric cryptographic primitives continually remains an open question. This is especially true when diverse new cryptographic primitives are publicly proposed by different groups and there exists no assurance in the security of these primitives. *Our goal is to achieve better understanding of the real security of these primitives and to provide our independent findings to the public.*

1.2 Thesis contributions

The contributions of this thesis are as follows:

- We propose a heuristic algorithm [71] for searching for compatible differential paths and apply it to the Korean hash function standard HAS-160. Our heuristic leads to a practical second order collision attack over all of the HAS-160 function steps, which is the first practical-complexity distinguisher on this function. An example of a colliding quartet is provided. In the context of second order collision attacks, constructing compatible differentials paths plays a central role. Previously, searching for compatible differentials was done in an ad-hoc manner and in case of HAS-160 (and SHA-2), the known compatible paths for HAS-160 spanned over a suboptimal number of steps. Our proposed heuristic aims to provide a systematic and efficient way to search for compatible paths over large number of steps, which extends the overall number of attacked steps.

- SM3 is a hash function designed by Xiaoyun Wang *et al.*, and published by the Chinese Commercial Cryptography Administration Office for the use of electronic authentication service system. The design of SM3 builds upon the design of the SHA-2 hash function, but introduces additional strengthening features. In Chapter 4, using a higher order differential cryptanalysis approach, we present a practical 4-sum distinguisher against the compression function of SM3 reduced to 32 rounds [72] In addition, we point out a slide-rotational property of SM3-XOR, which exists due to the fact that constants used in the rounds are not independent.

- Loiss is a byte-oriented stream cipher designed by Dengguo Feng *et al*. Its design builds upon the design of the SNOW family of ciphers. The algorithm consists of a linear feedback shift register (LFSR) and a non-linear finite state machine (FSM). Loiss utilizes a new structure called Byte-Oriented Mixer with Memory (BOMM) in its filter generator, reminiscent of the RC4 S-box and aiming to improve resistance against algebraic attacks, linear distinguishing attacks and fast correlation attacks. In Chapter 5, by exploiting some differential properties of the BOMM structure during the cipher initialization phase, we provide an attack of a practical complexity on Loiss in the related-key model [21]. As confirmed by our experimental results, our attack recovers 92 bits of the 128-bit key in less than one hour on a PC with 3 GHz Intel Pentium 4 processor. The possibility of extending the attack to a resynchronization attack in a single-key model is discussed.

- SNOW 3G is a stream cipher chosen by the 3rd Generation Partnership Project (3GPP) as a cryptoprimitive to substitute KASUMI in case its security is compromised. SNOW 2.0 is one of the stream

ciphers chosen for the ISO/IEC standard IS 18033-4. In Chapter 6, we show that the initialization procedure of the two ciphers admits a sliding property, resulting in several sets of related-key pairs [77]. In case of SNOW 3G, a set of 2^{32} related key pairs is presented, whereas in case of SNOW 2.0, several such sets are found, out of which the largest are of size 2^{64} and 2^{192} for the 128-bit and 256-bit variant of the cipher, respectively. In addition to allowing related-key key recovery attacks against SNOW 2.0 with 256-bit keys, the presented properties reveal non-random behavior which yields related-key distinguishers and also questions the validity of the security proofs of protocols that are based on the assumption that SNOW 3G and SNOW 2.0 behave like perfect random functions of the key-IV.

- HC-128 is a high speed stream cipher with a 128-bit secret key and a 128-bit initialization vector. It has passed all the three stages of the ECRYPT stream cipher project and is a member of the eSTREAM software portfolio. In Chapter 7, we present a differential fault analysis attack on HC-128 [75]. The fault model in which we analyze the cipher is the one in which the attacker is able to fault a random word of the inner state of the cipher but cannot control its exact location nor its new faulted value. To perform the attack, we exploit the fact that some of the inner state words in HC-128 may be utilized several times without being updated. Our attack requires about 7968 faults and recovers the complete internal state of HC-128 by solving a set of 32 systems of linear equations over Z_2 in 1024 variables.

- Rabbit is a high speed scalable stream cipher with 128-bit key and a 64-bit initialization vector. It has passed all three stages of the ECRYPT stream cipher project and is a member of eSTREAM software portfolio. In Chapter 8, we present a practical fault analysis attack on Rabbit [74]. The fault model in which we analyze the cipher is the one in which the attacker is assumed to be able to fault a random bit of the internal state of the cipher but cannot control the exact location of injected faults. Our attack requires around $128 - 256$ faults, precomputed table of size $2^{41.6}$ bytes and recovers the complete internal state of Rabbit in about 2^{38} steps.

The work in this thesis was published in [21, 71, 72, 74, 75, 77]. In addition, the work which was executed during this Ph.D. but not included in the thesis appeared in [76, 78, 120].

2

Background

2.1 Basic symmetric key primitive design approaches

In this chapter, we provide a short overview of the basic symmetric key cryptography constructions and their respective cryptanalysis methods. We start by providing common design methods for block ciphers, stream ciphers and hash functions.

2.1.1 Block Ciphers

A block cipher is a function that maps n-bit plaintext into n-bit ciphertext, parameterized by a k-bit secret key. More formally [104], a block cipher is a mapping

$$E_K : \{0,1\}^n \times \{0,1\}^k \mapsto \{0,1\}^n$$

such that for each key $K \in \{0,1\}^k$, $E(P,K)$ is an invertible mapping (the *encryption function* for K) from $\{0,1\}^n \mapsto \{0,1\}^n$ written $E_K(P)$. The inverse mapping is the *decryption function*, denoted $D_K(C)$. Here, n is called the *blocksize* and k is the *keylength*. It is generally assumed that the key is chosen at random.

Modern block ciphers are built by cascading simple operations which are individually insufficient to provide the required properties, but, when combined, achieve a high degree of mixing of the plaintext and secret key bits. The schematic view of iterative block cipher design is shown in Fig. 2.1. Before use, the secret key is expanded according to the *key expansion*, which aims to mix the secret key bits into the different rounds of the block cipher. For instance, in the case of the DES block cipher [109], the key expansion is fully

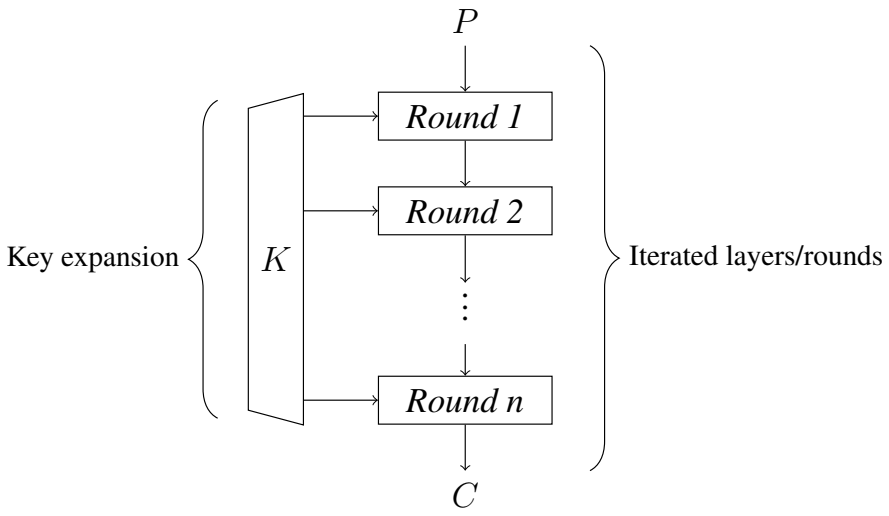

Figure 2.1: Block cipher: key expansion and layers of simple round functions

linear, but in the case of AES block cipher, the key expansion includes non-linear mixing operations. One iteration layer is called a *round* and typically consists of linear and non-linear sub-layer. The linear layer is specified using permutations, XOR, finite field multiplications by constants and similar linear operations. The non-linear layer is often implemented using S-boxes which implement operations that are non-linear with respect to the ones used in the linear layer.

In Fig. 2.2, two common approaches to construct a round function are provided. The Feistel Network round function, similar to the one used in the design of DES [109] block cipher and the Substitution Permutation Network, similar to the one used in AES [110] are presented. In the basic Feistel network construction, the input to the round function is divided in two words L_i, R_i and processed as follows:

$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \oplus F(R_i, K_i)$$

where K_i is the i-th round subkey and F is the mixing function. The advantage of the Feistel network construction lies in the fact that decryption function can be implemented by reusing the implementation of the encryption function where only the subkey expansion is different. In the Substitution Permutation Network construction depicted in Fig. 2.2, all of the input is uniformly mixed by applying a non-linear S-box layer followed by a linear layer, such as permutation.

Cascading a sufficient number of rounds in a block cipher may achieve the required mixing of plain-

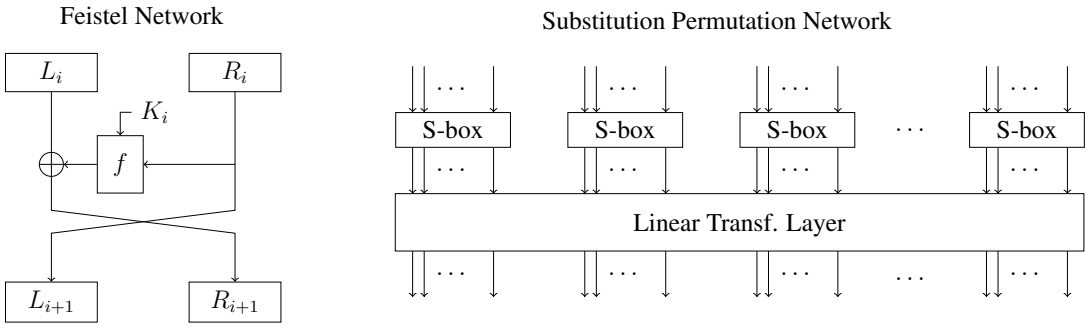

Figure 2.2: Approaches for constructing the round function in a cryptographic primitive

text and key bits and render the algorithm resistant to key or plaintext recovery algorithms. Classical security requirements for block ciphers include resistance to known-plaintext attacks, chosen-plaintext attacks and chosen-ciphertext attacks [104]. A more recent security requirement includes resistance to related-key attacks, in which the attacker is assumed to have access to the ciphertext typically for chosen plaintext encryption under multiple unknown keys that follow some pre-specified relation.

It is important to note that there exist even stronger security requirements, such as resistance to known-key attacks and chosen-key attacks. These scenarios are relevant to the analysis of the block ciphers in the hash function modes. As will be seen in subsection 2.1.3, when a block cipher is turned into a compression function according to the Davies-Meyer construction, the input message that is to be hashed is passed to the block cipher through the secret key mechanism and thus the attacker is in control of the key input.

2.1.2 Stream Ciphers

Instead of processing the plaintext block by iterating layers of transformations as done in block ciphers, stream ciphers generate pseudo-random sequences of data that is used to mask the plaintext. In that sense, stream ciphers can be seen pseudo-random number generators that depend on secret keys. Apart from the secret key, modern stream ciphers make use of another parameter called the initial vector (IV), which allows reusing the same secret key for different plaintexts. Unlike block ciphers, stream ciphers have memory, i.e., keep an inner state over different generated outputs.

Before the encryption starts, the secret inner state is initialized depending on the supplied secret key and the initial vector. Here, it is essential to achieve a high degree of mixing of these two variables to avoid

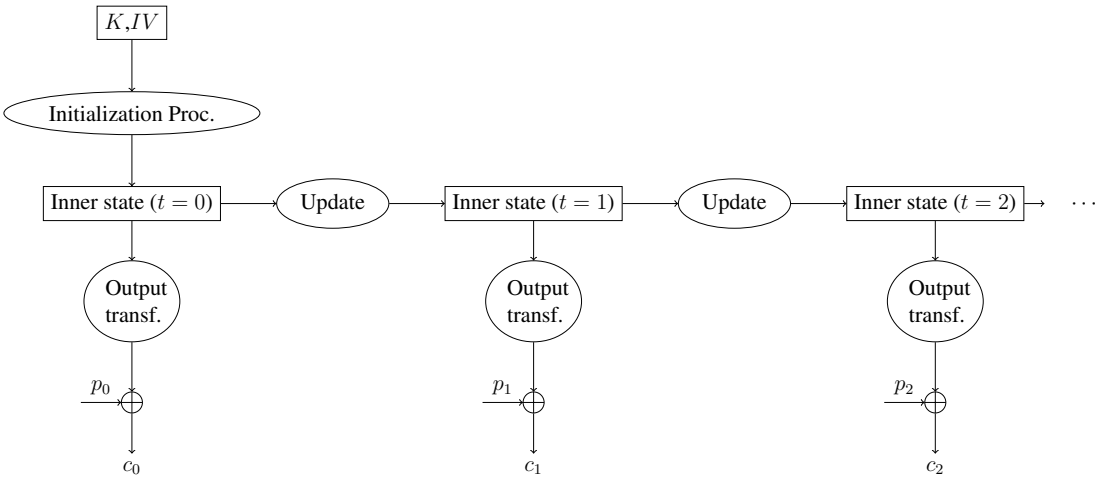

Figure 2.3: One possible stream cipher work flow

resynchronization attacks on the initialization phase of the cipher. It is important to point out the similarity between the stream cipher initialization phase and a hash function. Namely, the initialization can be seen as hashing of the key and the IV into the starting inner state. The schematic view of a stream cipher is provided on Fig. 2.3.

The design of some stream ciphers is reminiscent of the block cipher design explained in 2.1.1. For example, this is the case in the LFSR-based software-oriented SNOW family of stream ciphers [49,51]. In a block cipher, the plaintext is processed by iterating a comparatively large number of similar transformations (i.e., rounds) on the plaintext. Contrarily, the idea of a stream cipher is to release a small portion of the inner state after *each* stream cipher inner state update step. The state update step in the SNOW family of ciphers consists of a linear and a non-linear part. The linear part corresponds to an LFSR over GF(2^{32}) and the non-linear part to an FSM consisting of a transformation similar to one S-box based block cipher round. Apart from the fact that the FSM can be seen as one block cipher round, the analogy can be extended and the LFSR can be seen as the key schedule in a block cipher. The work flow of such a design approach is provided in Fig. 2.4, where NL and L denote non-linear and linear transformations, respectively. The length and the choice of the released keystream at each step have to be chosen carefully to eliminate the chances of inner state recovery algorithms while maximizing the performance of the cipher.

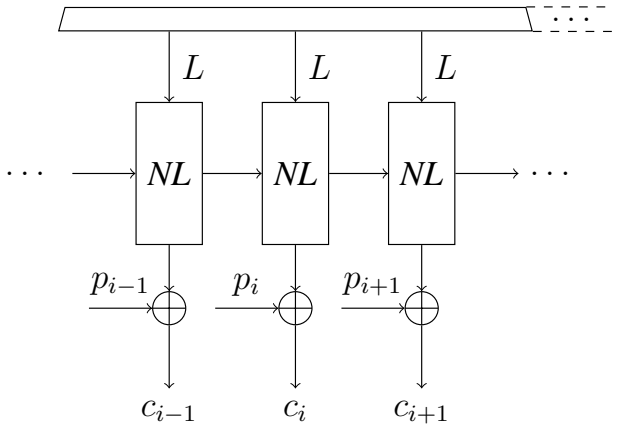

Figure 2.4: One possible approach to designing a stream cipher

2.1.3 Hash Functions

In this subsection, common ways to construct a hash function out of a block cipher are reviewed. The widely used hash functions such as SHA-2 [112], SHA-1 [111] and MD5 [119] are all built based on an underlying block cipher.

It is assumed that the input message to be hashed is divided into blocks of some fixed size. In the case the length of the message is not a multiple of the message block length, padding is applied. Now, a block cipher is used as a building block for a hash function as follows. First, a block cipher $E : \{0,1\}^n \times \{0,1\}^k \mapsto \{0,1\}^n$ is turned into a compression function, which compresses a single message block into a digest. Three common ways to achieve this are provided on Fig. 2.5. In the Davies-Meyer mode [104], which is utilized in SHA-1, SHA-2 and MD5, the compression function is specified by computing $H_i = E_{m_i}(H_{i-1}, K) \oplus H_{i-1}$. In other words, the input message is passed as a key to the block cipher and the output of the cipher is XOR-ed to its input. In the Matyas-Meyer-Oseas (MMO) mode [104], the function is specified by $H_i = E_{g(H_{i-1})}(m_i) \oplus m_i$. Finally, in the Miyaguchi-Preneel (MP) mode [104], we have $H_i = E_{g(H_{i-1})}(m_i) \oplus m_i \oplus H_{i-1}$. The g function used in the two latter modes accomodates the fact that the block cipher may have different block and key sizes and therefore the key length is adjusted by applying the function g to overcome this problem.

The compression function is then plugged in a mode that allows processing of an arbitrary number of message blocks. One way to achieve this functionality is the well-known Merkle-Damgård construction, shown in Fig. 2.6. Although limitations of this method have been exposed [64, 66], it is one of the most commonly used modes and it is applied in SHA-1, SHA-2, MD5 and SM3.

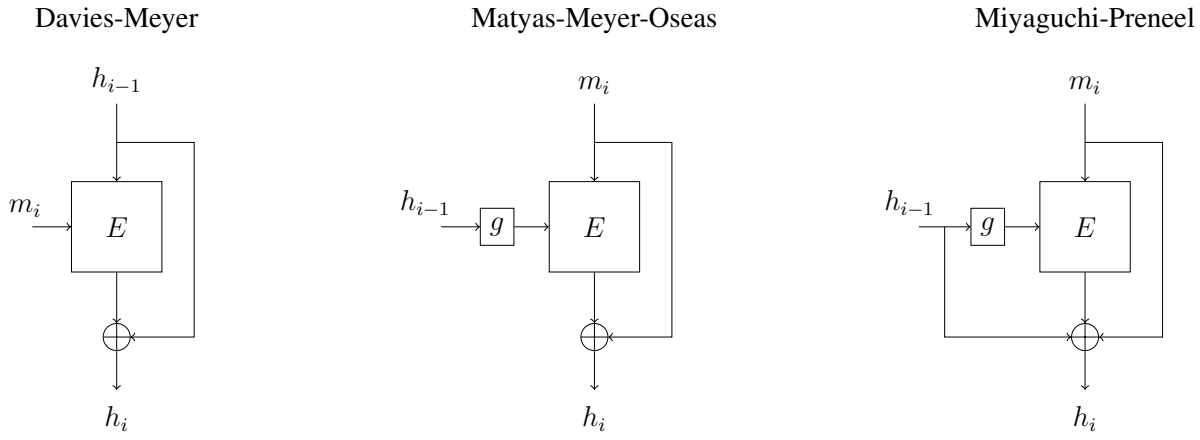

Figure 2.5: Turning a block cipher into a compression function

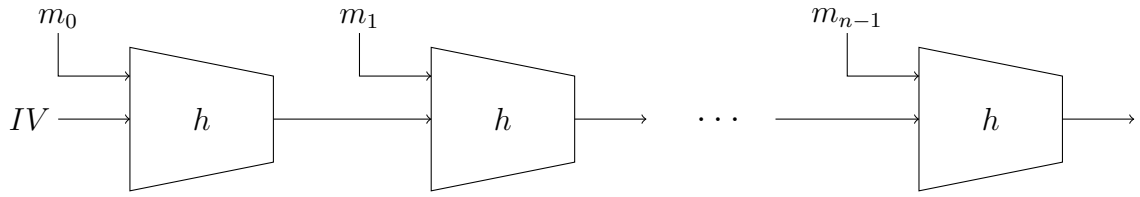

Figure 2.6: Merkle-Damgård mode

The main security goals of cryptographic hash functions are:

- *Preimage resistance:* Given a hash h, it is difficult to find any message m such that $h = hash(m)$.
- *Second Preimage resistance:* Given input m_1, it is difficult to find another m_2 s.t. $h(m_1) = h(m_2)$.
- *Collision resistance:* It is difficult to find any two messages m_1 and m_2 such that $h(m_1) = h(m_2)$.

The collision resistance property is the generic attack complexity of $2^{n/2}$, due to the birthday paradox [104], where n is the digest size. The second preimage and the preimage attacks have a higher generic complexity of 2^{n-1} operations.

2.2 Background on cryptanalysis of cryptographic primitives

In an effort to render cryptographic primitives more secure, researchers in the cryptanalytic community discovered many powerful attacks against symmetric key systems. Each proposed cryptanalytic attack is characterized by the following parameters:

- *Amount of required input data*: The number of input/output data required to successfully execute the attack.

- *Number of necessary operations*: The amount of necessary computations required to execute the attack, often measured in terms of the number of executions of the cryptographic primitive.

- *Storage complexity*: The amount of memory required to perform the cryptanalytic task.

- *Number of necessary physical actions on the encrypting device*: This can include the number of necessary measurements in the case of side channel analysis (such as power analysis attacks [84] and timing attacks [83]) or the number of induced faults in the memory of the cipher, in the case of fault analysis.

As stated above, apart from having available some amount of the input/output data related to the cryptographic primitive, sometimes the attack model includes physical access to the device. Thus, another classification of cryptanalytic attacks is with respect to whether or not the attacker has some sort of physical access to the encrypting device. In more detail, the classification is as follows:

- *Pure cryptanalytic attacks*: If there is no information that leaks from the physical implementation of the primitive, the attacker attempts to breach the security goals of the cryptographic primitive given some input/output data. In the case of stream and block ciphers, this can include an attempt to recover the key given plaintext/ciphertext material, whereas in the case of hash functions this may include attempting to find a preimage for a given hash value, or to construct a collision for the primitive, solely based on the specification of the primitive.

- *Physical access dependent attacks*: This attack model assumes that the attacker has some physical access to the device executing the cryptographic primitive in question. It should be noted that side channel attacks are applied against primitives that utilize a secret parameter, such as stream ciphers, block ciphers or hash functions in MAC modes [104]. The model includes side channel analysis where the attacker measures certain leaking parameters, e.g., the power consumption of the cryptographic device or the time used to perform the encryption operation. A careful analysis of this side channel information may enable the attacker to find some information about the inner workings of the encrypting process under the given secret parameter, which leads to information recovery. Another cryptanalytic model that falls into this category is differential fault analysis of ciphers in which the attacker induces faults (errors) by applying physical influence such as ionizing radiation to the de-

vice during the encryption. Similarly to side-channel analysis, a careful inspection of the results of encryption in a faulty environment may help the cryptanalyst to pin down the secret parameter.

In the sequel, some of the most important cryptanalytic techniques applied against symmetric key based primitives are reviewed, after which we provide a short introduction to side-channel attacks.

2.2.1 Pure cryptanalytic attacks

In the literature, cryptanalytic attacks are defined conservatively: a break of a cryptosystem is achieved if the effort required by the attacker is less than the effort required by the generic attacks (such as exhaustive key search or collision attack based on the birthday paradox). At the same time, in many practical scenarios, the attackers may have access to the cryptographic device. For such scenarios, side-channel attacks are expected to recover the secret information in practical complexities.

Differential cryptanalysis: Instead of following the input/output *values* through layers of mixing, in differential cryptanalysis, one follows the propagation of *differences* throughout the primitive. Differential cryptanalysis is one of the most often used tools in cryptanalysis of symmetric key primitives. It was introduced in [18] in the context of the block cipher DES, where efficient attacks on reduced-round variants of the cipher have been proposed by noting biases in the corresponding plaintext-ciphertext differences. Apart from block ciphers, differential cryptanalysis can be naturally applied to all other symmetric-key-based cryptographic primitives.

The first stage in the differential analysis of a primitive is to study differential properties of particular components used in the function. As for linear parts, such as bit permutations or key addition, the resulting differences are deterministically computed by the differences in the input and therefore their behavior is completely deterministic. In the case of non-linear components, such as s-boxes, knowledge of the input difference does not guarantee knowledge of the output difference. In this case, the *differentials*

$$\Delta X \xrightarrow{p} \Delta Y$$

are traced probabilistically. Here, p denotes the probability that the input difference ΔX will cause the output difference ΔY when passed through the considered non-linear function. To obtain a differential for $R-1$ rounds of the cipher, differentials are combined along a *differential path* and the final differential

probability is given by

$$\prod_{i=1}^{n} p_i$$

where n is the number of s-boxes used in the differential path and p_i the probability of the difference propagation within the i-th s-box.

In the case of block ciphers, portions of the last round subkey are guessed and if the distribution of differences for the current key guess does not correspond to the expected one, the key candidate is discarded. Since the said portion of the last round subkey is usually smaller than the full key, significant reduction of the key space is achieved. In the case of stream ciphers, differential attacks are typically applied on the initialization procedure of the cipher which depends on the secret key. Finally, in the case of collision attacks on hash functions, the attacker attempts to control the propagation of differences in order to find a colliding input pair.

Linear Cryptanalysis: This type of analysis uses probabilistic linear relations between the input and output of the non-linear components of the cryptographic primitive. It has been successfully applied to block ciphers [96], providing the first attack on the full DES block cipher. Also, it turns out to be effective against stream ciphers [32]. Recently, it has been applied to construct distinguishers in the context of hash functions [7].

When applied to block ciphers, linear cryptanalysis is a known plaintext attack that relies on linear relations between plaintext and ciphertext bits. The relation that is exploited needs to hold with biased probability, i.e., with probability different from $\frac{1}{2}$. To construct such a relation, the first stage is to find linear approximations of the non-linear building blocks of the cipher. In other words, the relation among input and output bits of a given non-linear function is established

$$X_{i_1} \oplus X_{i_2} \oplus \ldots \oplus X_{i_m} \oplus \ldots \oplus Y_{i_1} \oplus Y_{i_2} \oplus \ldots \oplus Y_{i_n}$$

where X and Y denote the input and output of the component. If the particular component is an s-box with n input bits and m output bits, there exist $(2^n - 1) \times (2^m - 1)$ possibilities and the cryptanalyst needs to investigate which ones hold with high bias, defined as $\epsilon = p - \frac{1}{2}$. Next, these linear approximations are combined using the Piling-Up Lemma [96] and a biased linear approximation among the key bits, the plaintext and ciphertext for the cipher reduced to $R - 1$ rounds is obtained. In particular, the Piling-Up

Lemma provides how the bias behaves when several biased random variables are XOR-ed together.

Lemma 1 *Let X_i be independent random variables of which the values are 0 with the probability of p_i and 1 with probability $1 - p_i$. Then, the probability that $X_1 \oplus \ldots \oplus X_n = 0$ is*

$$\frac{1}{2} + 2^{n-1} \prod_{i=1}^{n} (p_i - \frac{1}{2})$$

Once the distinguisher for $R - 1$ round of a block cipher is obtained using the linear relation, the key-recovery stage can proceed as follows. The R-th round subkey bits are then guessed and determined by the guesses that provide most of the plaintext-ciphertext pairs satisfying the linear relation. When applied to a stream cipher, similar to the way linear relations are constructed in block ciphers, one follows the evolution of the inner state bits that participate in keystream generation at different cipher clocks and combines the linear approximations of non-linear components. This allows construction of linear relations between bits at different clocks, which allows the keystream produced by the stream cipher to be distinguished from random and also in some cases secret inner state recovery [32]. Finally, linear relations have been used to construct distinguishers for hash functions [7]. An important difference is that, since there is no key involved, one does not aim to recover the key but rather attempts to construct a distinguisher for the function.

Higher-order differential cryptanalysis: While ordinary differential cryptanalysis utilizes differences of the form

$$\Delta_a f(x) = f(x \oplus a) \oplus f(x)$$

higher-order differential cryptanalysis attempts to generalize these first order difference to the i-th derivatives. It has been shown [80] that it is possible to construct a cipher that is unbreakable by means of classical differential cryptanalysis and at the same time weak with respect to higher order differential cryptanalysis, making the attack relevant. For instance, in [80] it was shown that for the function $f(x, k) = (x+k)^2 \bmod p$ with input/output size of $2log_2p$, where p is prime, every non-trivial one round differential has probability of $\frac{1}{p}$ and the second order derivative is a constant. The problem with high-order differentials is to combine them to more than two rounds, as is possible with first order differentials.

More precisely, the notion of higher order differentials as introduced by Lai in [85] is as follows.

Definition 1 *Let $(S, +)$ and $(T, +)$ be Abelian groups. For a function $f : S \to T$, the derivative of f at*

$a \in S$ *is defined as*

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

The i*-th derivative of* f *at* $(a_1, \ldots, a_i)$ *is then defined by*

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

In the case of the function $f : F_2^m \to F_2^n$, we have (Proposition 3, [85]):

Lemma 2 *Let* $L[a_1, \ldots, a_i]$ *be the list of all* 2^i *possible subsets of* $a_1, a_2, \ldots, a_i$*. Then,*

$$\Delta^{(i)}_{a_1,\ldots,a_i} f(x) = \bigoplus_{c \in L[a_1,\ldots,a_i]} f(x \oplus c)$$

Below, we review the higher-order analysis of hash functions. In particular, the two equivalent notions of second order collisions and zero-sums are defined. As defined in [22], an i-th order differential collision for f is an i-tuple $(a_1, \ldots, a_i)$, together with a value x such that

$$\Delta_{(a_1,\ldots,a_i)} f(x) = 0$$

As argued in [22], since the $i+1$ input parameters $a_1, \ldots, a_i$ and x can be chosen freely, the query complexity of finding an i-th order collision is $2^{n/(i+1)}$, where n denotes the bit-size of the output of the function f. Here, the query complexity denotes the number of queries made to the f function oracle. Thus, the query complexity of finding a second order collision for the function f, i.e., values x, a_1 and a_2, such that

$$f(x \oplus a_1 \oplus a_2) \oplus f(x \oplus a_1) \oplus f(x \oplus a_2) \oplus f(x) = 0 \tag{2.1}$$

is $2^{n/3}$. As for the computational complexity, which would include evaluating f around $2^{n/3}$ times and finding, among the outputs, a quartet that sums to 0, no algorithm with complexity better than $2^{n/2}$ is known. It should be noted that in [121], the boomerang technique is used to find a zero-sum quartet of inputs x_0, x_1,

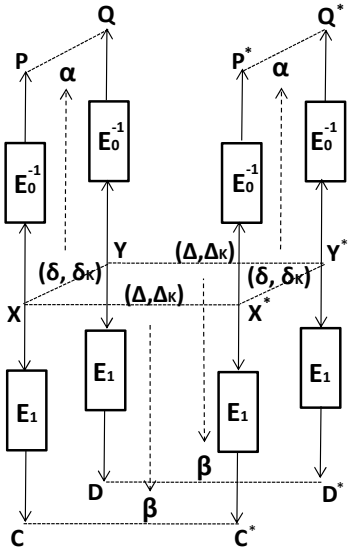

Figure 2.7: Boomerang attack against a compression function

x_2, x_3, introduced in [8], such that

$$\begin{aligned} x_0 \oplus x_1 \oplus x_2 \oplus x_3 &= 0 \\ f(x_0) \oplus f(x_1) \oplus f(x_2) \oplus f(x_3) &= 0 \end{aligned} \tag{2.2}$$

It is easy to verify that the notions zero-sum quartet and second order collision notions are equivalent. For example, given a zero-sum quartet, it suffices to put $x = x_0$, $a_1 = x_0 \oplus x_1$, $a_2 = x_0 \oplus x_2$ to have (2.1) satisfied. An efficient technique to construct second order collision utilizes the boomerang attack, as explained below.

Boomerang Attacks: This is a combined attack in which two different differential paths are combined in order to pass through a larger number of rounds. The attack was devised in 1999 by Wagner [129] and in its basic version, the attack required chosen-ciphertext queries, but in later versions [65] this requirement was removed.

Boomerang attacks were applied to construct second order collisions for hash functions in 2011 independently by Biryukov *et al.* [23] and Lamberger *et al.* [86]. The general idea is to construct a quartet that forms a boomerang structure [129] for a block cipher in the Davis-Meyer mode. The differentials used in the boomerang are related key differentials, where the secret key of the block cipher corresponds to the message block in the case of a compression function. The encryption function is divided into two parts, $E_1 \circ E_0$. As shown in Fig. 2.7, for the bottom part of the boomerang, a related-key differential $(\Delta, \Delta_K) \rightarrow \beta$ for E_1 with probability q is constructed. Similarly, another related-key differential $(\delta, \delta_K) \rightarrow \alpha$ with probability p is used for E_0^{-1}. Then, an attempt to randomly satisfy the differentials in the boomerang

structure with probability p^2q^2 would proceed as follows:

- Randomly choose X, the inner state in the middle of the hash function execution, representing the input to E_1 (and the output of E_0). Let $X^* = X \oplus \Delta$, $Y = X \oplus \delta$ and $Y^* = X \oplus \Delta \oplus \delta$.
- Compute backward from X, X^*, Y, Y^* using E_0^{-1} to obtain P, P^*, Q, Q^*, using keys K, $K \oplus \Delta_K$, $K \oplus \delta_K$, $K \oplus \delta_K \oplus \Delta_K$, respectively.
- Compute forward from X, X^*, Y, Y^* using E_1 to obtain C, C^*, D, D^* using keys K, $K \oplus \Delta_K$, $K \oplus \delta_K$, $K \oplus \delta_K \oplus \Delta_K$, respectively.
- Verify whether $C \oplus C^* = D \oplus D^*$ and $P \oplus Q = P^* \oplus Q^*$.

If the last condition is satisfied, a zero-sum quartet is found for the encryption function in Davis-Meyer mode, since $P \oplus Q \oplus P^* \oplus Q^* = 0$ and also $(C \oplus P) \oplus (C^* \oplus P^*) \oplus (D \oplus Q) \oplus (D^* \oplus Q^*) = 0$.

To improve the efficiency of the process above, instead of trying to satisfy the boomerang randomly, message modification can be used for some of the differential paths in the boomerang. For example, in [22], message modification is applied to satisfy the middle part of the boomerang, i.e., to satisfy the two differentials paths of the function E_1. The other paths in the boomerang are satisfied randomly.

Truncated differential cryptanalysis: Truncated differential cryptanalysis was introduced in 1994 by Knudsen [80]. The idea of this technique is to view the differences in an abstracted way, instead of fully specifying them. In particular, in a conventional differential attack, each bit of the input and output difference is specified. In truncated differential cryptanalysis, the differences are not fully specified. For instance, instead of specifying each bit in a 32-bit difference, one can only take into account whether each of the four bytes in a difference are active and write, say, 1001, if only the least and most significant bytes are active. Here, the particular 8-bit value of the difference is abstracted away from the picture.

Truncated differentials are a versatile tool in cryptanalysis. One of the first applications of truncated differentials was provided in [80], where truncated differentials are used to attack a 6-round DES with only 46 chosen plaintext-ciphertext pairs. Another interesting early truncated differential is the one used to attack the reduced round Skipjack block cipher [82] (an algorithm developed by the US National Security Agency), where it was shown that there exists a truncated differential that spans over 24 steps with probability 1. Over the last 20 years, truncated differentials have been extensively used. A recent truncated differential attack includes an attack against the recently proposed block cipher WIDEA [91], which is a successor of

the widely used IDEA block cipher. In the case of hash functions, truncated differentials are naturally used against block cipher based hash functions that rely on S-boxes, such as ECHO and Grøstl [117]. An example of an application of truncated differentials on a stream cipher is cryptanalysis of round-reduced versions of the Salsa stream cipher [115].

Differential-linear cryptanalysis: This type of attack was introduced in 1994 by Langford and Hellman [87]. It is a combined attack in the sense that it combines two different probabilistic patterns (a differential pattern and a linear pattern). Ideally, combining the two patterns outperforms a pure differential or linear attack over the same number of rounds. This is the case when this attack is applied against 8-round DES, as shown in [87]. In particular, a chosen-plaintext attack against 8-round DES is obtained, in which differential cryptanalysis is applied to the first three rounds and linear cryptanalysis is applied on the remaining five rounds. The main observation on which the attack relies is that inverting certain bits in the input of the first round leaves certain third round bits unchanged, implying that the XOR sum of these third round bits is also left unchanged. From rounds four to seven, a linear approximation involving exactly these unchanged four-round input bits and certain 7-th round bits is then used. For each 8-th round subkey portion, it is verified whether for each plaintext, the XOR sum of the bits in question changes or not. Using differential-linear cryptanalysis, the number of necessary chosen plaintext-ciphertext pairs was reduced to 512 to recover 10 bits of the key, for 8-round DES. In this case, the differential-linear combination yielded an attack that requires fewer plaintext-cipher text pairs, when compared to the the classical Biham-Shamir differential attack, which requires over 5000 chosen pairs. The differential-linear attack has mostly been applied to block ciphers. One recent application of this technique is a 12-round cryptanalysis of the Serpent block cipher [46], which was a finalist in the AES block cipher competition.

Impossible differential cryptanalysis: Instead of using biased differentials, differentials with probability 0 can also be used to mount key recovery attacks on cryptographic primitives [14]. Let the input difference δ_0 never propagate to the output difference δ_1. If this property holds for rounds 1 to $R-1$, then the final round subkey can be attacked as follows. We encrypt many plaintext pairs with difference δ_0 and guess the corresponding portion of the round R subkey. If the guess is correct, the difference δ_1 will *not* occur. If, however, such a difference occurs, we can prune the key space based on the current key guess. One notable early application of this attack was on the Skipjack block cipher [15], where 31 out of 32 rounds where shown to be susceptible to this technique.

Multiset attacks, Integral attacks: The first application of multiset attacks was against the predecessor of AES, a block cipher called Square [43]. The main probabilistic pattern utilized in such attacks is the preservance of certain properties of sets of values. For example, consider a set of inputs such that each two inputs differ in exactly one byte i_0 and this byte takes all possible values, i.e., a set of 2^8 inputs are considered. Namely, if a byte-wise bijective transformation is applied, the output set will also satisfy the said property, i.e., the corresponding output byte will take all possible values. Useful sets of another type are sets where each value occurs exactly k times. Multiset attacks trace this kind of properties through as many rounds as possible. Integral cryptanalysis refers to the attack when the used set property is that $\sum_{x \in G} x = 0$, where G is the group that the elements belong to, i.e., the set is *balanced*. The property is conserved after the addition of two sets and this is what is important for pushing the property through several rounds in the cipher. One relatively recent application of this technique is against AES [81], where it is shown that 7-round AES does not behave as a random oracle.

Slide attacks: A particularity of this type of attack is that it is independent of the number of rounds employed in the primitive. The idea is quite simple: one attempts to instantiate the inner state at round $i + 1$ with values exactly equal to values in state i of another instance of the primitive. If the same round functions are applied throughout the primitive, the property is preserved with probability 1 until the end of the execution of the primitive. Slide attacks have been introduced in the context of block ciphers by Biryukov and Wagner [26] and have been applied to stream ciphers [34] and hash function as well [56].

When applied to block ciphers, the attack relies on the key schedule weakness by which parts of the key are reused subsequently in different rounds. For simplicity, assume that the same key is used in each round transformation; let property of the cipher be called *self-similarity*. The first stage of the key-recovery attack on such a block cipher is to find *slid pairs*, defined as follows. Let F be the round function of an iterated block cipher. If a pair of known plaintexts $(P, C), (P', C')$ satisfies $F(P) = P'$, then due to the self-similarity of both the rounds and the key schedule, the corresponding ciphertexts also satisfy $F(C) = C'$. Such a pair is called a *slid pair*.

By finding a slid pair, which is possible using $O(2^{\frac{n}{2}})$ plaintexts due to the birthday paradox [104], the attacker obtains a plaintext P and its one-round ciphertext, P'. The key idea here is that the attacker is able to verify that the corresponding ciphertexts (C, C') are on a one-round difference. Now, if from the pair (P, P') it is possible to deduce the information about the key K, the secret key is compromised. A common

way to defend against slide attacks is to use different constants in each round of the transformation.

2.2.2 Side channel attacks

Side channel attacks usually exploit the information leaked by the physical characteristics of the cryptographic modules during the execution of the algorithm. This extra information can be extracted from timing, power consumption or electromagnetic radiation features. Other forms of side-channel information can be a result of hardware or software failures, changes in frequency or temperature and computational errors. In the following subsections, we briefly review some of the side channel cryptanalytic models.

Fault Analysis: In fault analysis, the cryptanalyst applies some kind of physical influence on the internal state of the cryptosystem, such as ionizing radiation which flips random bits in the memory of the device. By careful study of the results of computations under such faults, an attacker may be able to retrieve information about the secret key. Smartcards are especially susceptible to this kind of attack. Fault attacks were first introduced by Boneh *et al.* [30] in 1996 where they described attacks that targeted the RSA public key cryptosystem by exploiting a faulty Chinese Remainder Theorem computation to factor the modulus n. Subsequently, fault analysis attacks were extended to symmetric systems such as DES [19] and later to AES [48] and other primitives. Fault analysis attacks became a more serious threat after cheap and low-tech methods of applying faults were presented (e.g., [6, 126]).

Hoch and Shamir [58] addressed the problem of fault analysis of stream ciphers in 2004. Ciphers based on LSFRs, LILI-128, SOBER-t32 and also RC4 were analyzed and it has been shown that none of these constructions are secure in the *random-location* fault model, i.e., in the case where the attacker can not choose the exact location of induced faults. As for RC4, the key recovery attack required 2^{16} faults and 2^{26} keystream words.

In [17], Biham *et al.* assessed the RC4 stream cipher in the *chosen-location* model, where an attacker chooses a location at which a fault is induced. An interesting idea to push the cipher into a specific state called a Finney state, by means of inducing faults, is used to find the secret internal state of RC4. A Finney state is a state in which RC4 in normal mode of operation, i.e., without faults, can not enter. However, once the internal state is artificially pushed into one of the Finney states, it can not go out anymore, the length of a cycle becomes very small and what is more, the secret S table can be read solely by looking at the keystream output. The attack required 2^{16} *chosen-location* faults. Another, more advanced fault analysis

attack on RC4 which requires 2^{10} faults was also introduced in the same paper.

Timing Analysis: The majority of optimized implementations of cryptographic algorithms execute the computation in a non-constant time. If these operations involve secret parameters, these timing variations can leak some information that can provide enough critical knowledge to recover secret information. Timing attacks were first introduced in 1996 by Kocher [83] who demonstrated the power of these attacks against the RSA cryptosystem. Subsequently, Schindler [4] presented timing attacks on the implementation of RSA exponentiation that employs the Chinese Remainder Theorem. Other uses of timing attacks can be found in [45, 57].

A particular type of timing analysis, called cache-timing analysis was proposed in 2005 by Osvik *et al.* [113]. A simple cache-timing attack in a scenario where the attacker and the legitimate user share the same CPU, named *prime-then-probe*, is as follows. First, the attacker fills the cache with data and then stops using the CPU. Then, the legitimate user performs the encryption on the CPU. Finally, the attacker measures loading times and finds which of his data has been removed from the cache. It should be noted that the attacker does not learn the content of the cache registers, but only positions that have been used by the legitimate user. From such information, a cipher's internal values leak and can lead to the recovery of the secret key. An important example of ciphers particularly vulnerable to cache-timing analysis is the Advanced Encryption Standard (AES) [31, 113]. Bertoni *et al.* [13] describes how cache misses can be used for cryptanalysis.

As for the stream ciphers, a cache-timing model was applied to analyze the HC-256 stream cipher by Zenner [138], where, given 6148 precise cache measurements and computational, effort equivalent to around 2^{55} key tries in the brute force setting, the secret internal state of the stream cipher can be recovered. In [88], Leander *et al.* have shown how to apply cache-timing attacks against Linear Feedback Shift Register based stream ciphers. In particular, it was shown how to recover the secret key for SOSEMANUK [12], another software oriented eStream finalist, given the precise cache measurements in 40 and 60 clocks of the cipher, respectively.

Power Analysis Attacks: Useful information about the operations being executed in cryptographic hardware can leak through power consumption information. Power analysis has been shown to be effective against smart cards and embedded devices. In general, power analysis attacks [84] can be either simple power analysis (SPA) or differential power analysis (DPA). In SPA attacks, using the measured power

traces, the attacker guesses what instruction is being carried out at a specific time as well as the input and the output values of the instruction. Such analysis requires the attacker to know the exact structure of the implementation. In contrast, DPA attacks do not require detailed knowledge of the implementation and utilize statistical methods in the process. Experimental results of power analysis against smartcards have been reported in [5, 114].

3

A heuristic for finding compatible differential paths with application to HAS-160

In this chapter, a heuristic that searches for compatible differential paths is proposed. The application of the proposed heuristic in the case of HAS-160 yields a practical second order collision over all of the hash function steps, which is the first practical result that covers the full HAS-160 compression function.

In general, whenever two probabilistic patterns are combined for the purpose of passing through a maximal number of rounds of a cryptographic primitive, a natural question that arises is the question of compatibility of the two patterns. Particularly, the question of compatibility of differential paths in the context of boomerang attacks was tackled in 2011, by Murphy [107], who showed that care should be exercised when estimating the boomerang attack success probability, since there may exist dependency between the two combined differentials. The extreme case is the total impossibility of combining the two paths, where the corresponding probability is equal to 0.

In the context of constructing second order collisions for compression functions using the start-from-the-middle technique, due to availability of message modification in the steps where the primitive follows the two paths, the above mentioned probability plays less of a role as long as it is strictly greater than 0. In that case, the two paths are said to be compatible. Several paths that were previously believed to be compatible have been shown to be incompatible in the previously described sense, e.g., by Leurent [89] and Sasaki [123] for the BLAKE and RIPEMD-160 hash functions, respectively.

The compatibility requirement in this context can be stated with more precision as follows. Let ϕ and ω be two differential paths over some number of steps of an iterative function $f = f_{j+n} \circ \ldots \circ f_j$. If there

exists a quartet of f inputs x_0, x_1, x_2 and x_3 such that computations (x_0, x_1) and (x_2, x_3) follow ϕ whereas (x_0, x_2) and (x_1, x_3) follow ω, we say that ϕ and ω are compatible. Usually the path ϕ is left unspecified over the last k steps (backward path) and ω is unspecified over the remaining steps (forward path). Such paths have also been previously called *independent* [23]. Another closely related notion is the concept of *non-interleaving* paths in the context of biclique attacks [67].

New Contributions. In this chapter, we present a heuristic that allows us to search for compatible differential paths. The heuristic builds on the previous de Cannière and Rechberger automatic differential path search method. Instead of working with pairs, our proposed heuristic operates on quartets of hash executions and includes cross-path propagations. We present detailed examples of particular propagations applied during the search. As an application of our proposed heuristic, a second order collision for the full HAS-160 compression function is found. The best previous practical distinguisher for this function covered steps 5 to 80 [124]. This is the first practical distinguisher for the full HAS-160. This particular hash function is relevant as it has been standardized by the Korean government (TTAS.KO-12.0011/R1) [1].

Related Work. The differential paths used in groundbreaking attacks on MD4, MD5 and SHA-1 [130, 131] were found manually. Subsequently, several techniques for automatic differential path search have been studied [35, 54, 125, 128]. The de Cannière and Rechberger heuristic [35] was subsequently applied to many MDx/SHA-x based hash functions, such as RIPEMD-128, HAS-160, SHA-2 and SM3 [99–102]. To keep track of the current information in the system, the heuristic relies on 1-bit constraints that express the relations between pairs of bits in the differential setting. This was generalized to multi-bit constraints by Leurent [89], where the finite state machine approach allowed uniform representation of different constraint types. Multi-bit constraints have been used in the context of differential path search in [90].

The boomerang attack [129], originally applied to block ciphers, has been adapted to the hash function setting independently by Biryukov *et al.* [23] and by Lamberger and Mendel [86]. In particular, in [23], a distinguisher for the 7-round BLAKE-32 was provided, whereas in [86] a distinguisher for the 46-step reduced SHA-2 compression function was provided. The latter SHA-2 result was extended to 47 steps [22]. Subsequently, boomerang distinguishers have been applied to many hash functions, such as HAVAL, RIPEMD-160, SIMD, HAS-160, SM3 and Skein [72, 92, 98, 121, 123, 124, 136]. Outside of the boomerang context, the zero-sum property as a distinguishing property was first used by Aumasson [8].

As for the previous HAS-160 analysis, in 2005, Yun *et al.* [137] found a practical collision for

the 45-step (out of 80) reduced hash function. Their attack was extended in 2006 to 53 steps by Cho *et al.* [36], however, with computational complexity of 2^{55} 53-step compression function computations. In 2007, Mendel and Rijmen [103] improved the latter attack complexity to 2^{35}, providing a practical two-block message collision for the 53-step compression function. Preimage attacks on 52-step HAS-160 with complexity of 2^{152} was provided in 2008 by Sasaki and Aoki [122]. Subsequently, in 2009, this result was extended by Hong *al.* to 68 steps [61] where the attack required a complexity of $2^{156.3}$. In 2011, Mendel *et al.* provided a practical semi-free-start collision for 65-step reduced compression function [99]. Finally, in 2012, Sasaki *et al.* [124] provided a theoretical boomerang distinguisher for the full HAS-160 compression function, requiring $2^{76.6}$ function computations. In the same work, a practical second order collision was given for steps 5 to 80 of the function.

In the next section, we provide a review of boomerang distingiushers and a recapitulation of the de Cannière and Rechberger search heuristic, along with the HAS-160 specification. In Section 3.2, the general form of our search heuristic is provided and its application to HAS-160 is discussed. The three propagation types used in the heuristic are explained in Section 3.3. Concluding remarks are in given Section 3.4.

3.1 Review of related work and the specification of HAS-160

In the following subsections, we provide a description of a commonly used strategy to construct second order collisions, an overview of the de Cannière and Rechberger path search heuristic and finally the specification of the HAS-160 hash function.

3.1.1 Review of boomerang distinguishers for hash functions

First, we provide a generic definition of the property used for distinguishing the compression function from a random function. Let h be a function with n-bit output. A *second order collision* for h is a set $\{x, \Delta, \nabla\}$ consisting of an input for h and two differences, such that

$$h(x + \Delta + \nabla) - h(x + \Delta) - h(x + \nabla) + h(x) = 0 \tag{3.1}$$

As explained in [22], the query complexity for finding a second order collision is $3 \cdot 2^{n/3}$ where n denotes the bit-size of the output of the function f. By the query complexity, the number of queries required to be made

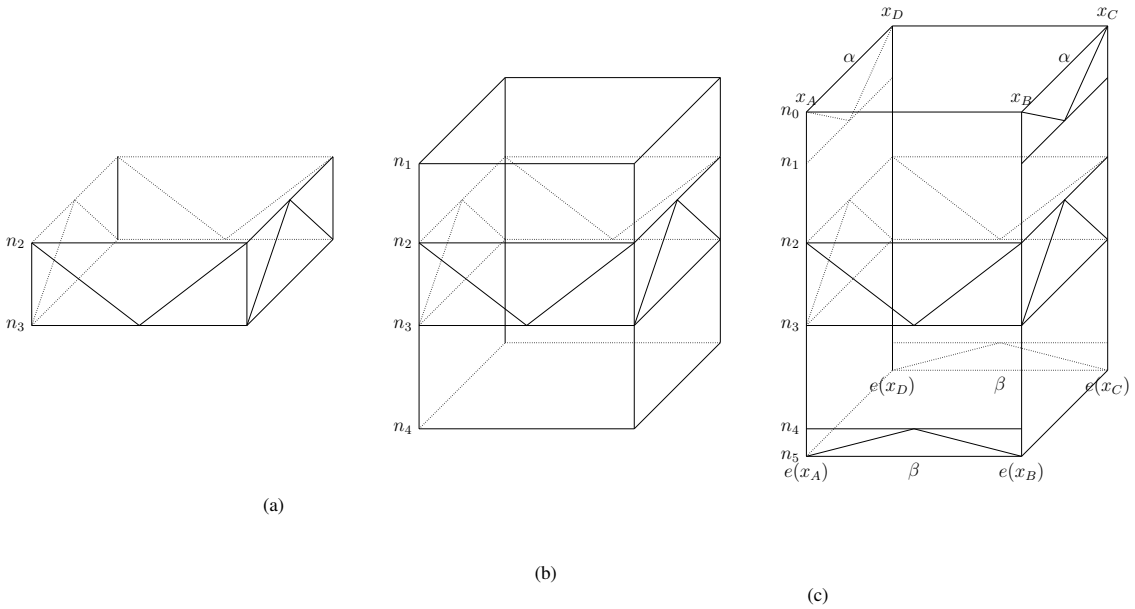

Figure 3.1: Start-from-the-middle approach for constructing second-order collisions

to the function h is considered. On the other hand, for the computational complexity, which would include evaluating h around $3 \cdot 2^{n/3}$ times and finding a quartet that sums to 0, the best currently known algorithm runs in complexity no better than $2^{n/2}$. If for a particular function a second order collision is obtained with a complexity lower than $2^{n/2}$, then this hash function deviates from the random function oracle.

Next, we explain the strategy to construct quartets satisfying (3.1) for Davies-Meyer based functions, as commonly applied in the previous literature. An overview of the strategy is provided in Fig. 3.1. We write $h(x) = e(x) + x$, where e is an iterative function consisting of n steps. The goal is to find four inputs x_A, x_B, x_C and x_D that constitute the inputs in (3.1) according to Fig. 3.1 (c). In particular, the goal is to have

$$\begin{aligned} x_A - x_D &= x_B - x_C \\ e(x_A) - e(x_B) &= e(x_D) - e(x_C) \end{aligned} \tag{3.2}$$

where the two values specified by (3.2) are denoted respectively by α and β in Fig. 3.1 (c). In this case, we have $h(x_A) - h(x_B) + h(x_C) - h(x_D) = e(x_A) + x_A - e(x_B) - x_B + e(x_C) + x_C - e(x_D) - x_D = 0$. Now, one can put $x_A = x$, $\Delta = x_D - x_A$ and $\nabla = x_B - x_A$ and (3.1) is satisfied.

A preliminary step is to decide on two paths, called the *forward* path and the *backward* path. As shown in Fig. 3.1, these paths are chosen so that for some $n_0 < n_1 < n_2 < n_3 < n_4 < n_5$, the forward path has no active bits between steps n_3 and n_4 and the backward path has no active bits between steps n_1 and

Compression function	n_0	n_1	n_2	n_3	n_4	n_5	Reference	Message block size
SHA-2	0	6	22	31	47	47	[22]	16×32
HAVAL	0	2	61	97	157	160	[121]	32×32
HAS-160	5	13	38	53	78	80	[124]	16×32

Table 3.1: Overview of some of the previously used boomerang paths

n_2. The forward path is enforced on faces (x_A, x_B) and (x_D, x_C) (front and back) whereas the backward differential is enforced on faces (x_A, x_D) and (x_B, x_C) (left and right). In the case of MDx-based designs, the particular n values depend mostly on the message schedule specification.

The procedure can be summarized as follows:

(a) The first step is to construct the middle part of the quartet structure, as shown in Fig. 3.1 (a). The forward and backward paths end at steps n_3 and n_2, respectively. In steps n_2 to n_3, the two paths need to be compatible for this stage to succeed.

(b) Following Fig. 3.1 (b), the paths are extended to steps n_1 backward and n_4 forward with probability 1, due to the absence of disturbances in the corresponding steps.

(c) Some of the middle-step words are randomized and the quartet is recomputed backward and forward, verifying if (3.2) is satisfied. If yes (see Fig. 3.1 (c)), return the quartet, otherwise, repeat this step.

This strategy, with variations, has been applied in several previous works, such as [22, 121, 123, 124]. In Table 3.1, we provide the forward/backward path parameters for the previous boomerang distinguishers on some of the MDx/SHA-x based compression functions following the single-pipe design strategy.

In [22,124], the number of steps in the middle was 9 and 16 steps, respectively. It can be observed that these numbers of middle steps are suboptimal, since simple message modification allows trivially satisfying 16 steps in the case of SHA-2 and HAS-160. Since the forward and backward paths are sparse towards steps n_3 and n_2, one can easily imagine satisfying more than 16 steps, while there remains enough freedom to randomize the inner state although some penalty in probability has to be paid. In the case of HAVAL [121], simple message modification allows passing through 32 steps and the middle part consists of as many as 36 steps. However, it should be noted that this is due to the particular property of HAVAL that allows narrow paths [70].

$\delta(x, x')$	meaning	(0,0)	(0,1)	(1,0)	(1,1)
?	anything	√	√	√	√
-	$x = x'$	√	-	-	√
x	$x \neq x'$	-	√	√	-
0	$x = x' = 0$	√	-	-	-
u	$(x, x') = (0, 1)$	-	√	-	-
n	$(x, x') = (1, 0)$	-	-	√	-
1	$x = x' = 1$	-	-	-	√
#		-	-	-	-

$\delta(x, x')$	meaning	(0,0)	(0,1)	(1,0)	(1,1)
3	$x = 0$	√	√	-	-
5	$x' = 0$	√	-	√	-
7		√	√	√	-
A	$x' = 1$	-	√	-	√
B		√	√	-	√
C	$x = 1$	-	-	√	√
D		√	-	√	√
E		-	√	√	√

Table 3.2: Symbols used to express 1-bit conditions [35]

3.1.2 Review of the de Cannière and Rechberger search heuristic

This search heuristic is used to find differential paths that describe pairs of compression function executions. The symbols used for expressing differential paths are provided in Table 3.2. For example, when we write `-x-u`, we mean a set of 4-bit pairs

$$\texttt{-x-u} = \{T, T' \in F_2^4 | T_3 = T'_3, T_2 \neq T'_2, T_1 = T'_1, T_0 = 0, T'_0 = 1\}$$

where T_i denotes i-th bit in word T.

Next, an example of *condition propagation* is provided. Suppose that a small differential path over one modular addition is given by

$$\texttt{----} + \texttt{---x} = \texttt{---x} \tag{3.3}$$

Here (3.3) describes a pair of additions: $x + y = z$ and $x' + y' = z'$, and from this "path" we have that $x = x'$ and also that y and y' are different only in the least significant bit (likewise for z and z'). However, this can happen only if $x_0 = x'_0 = 0$, i.e. if the lsb of x and x' is equal to 0. We thus *propagate a condition*

by replacing (3.3) with

$$---0 + ---x = ---x$$

The de Cannière and Rechberger heuristic [35] searches for differential paths over some number of compression function steps. It starts from a partially specified path which typically means that the path is fully specified at some steps (i.e., consisting of symbols $\{-, u, n\}$) and unspecified at other steps (i.e., symbol '?'). The heuristic attempts to complete the path, so that the final result is non-contradictory by proceeding as follows:

- *Guess*: select randomly a bit position containing '?' or 'x'. Substitute the symbol in the chosen bit position by '−' and $\{u, n\}$, respectively.

- *Propagate*: deduce new information introduced by the *Guess* step.

When a contradiction is detected, the search backtracks by jumping back to one of the guesses and attempts different choices.

3.1.3 HAS-160 specification

The HAS-160 hash function follows the MDx/SHA-x hash function design strategy. Its compression function can be seen as a block cipher in Davies-Meyer mode, mapping 160-bit chaining values and 512-bit messages into 160-bit digests. To process arbitrary-length messages, the compression function is plugged in the Merkle-Damgård mode.

Before hashing, the message is padded so that its length becomes multiple of 512 bits. Since here padding is not relevant, we refer the reader to [1] for further details. The underlying HAS-160 block cipher consists of two parts: message expansion and state update transformation.

Message expansion: The input to the compression function is a message $m = (m_0, \ldots m_{15})$ represented as 16 32-bit words. The output of the message expansion is a sequence of 32-bit words $W_0, \ldots W_{79}$. The expansion is specified in Table 3.3. For example, $W_{26} = m_{15}$.

State update: One compression function step is schematically described by Fig. 3.2 (a). The Boolean

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$m_8 \oplus m_9$ $\oplus m_{10} \oplus m_{11}$	m_0	m_1	m_2	m_3	$m_{12} \oplus m_{13}$ $\oplus m_{14} \oplus m_{15}$	m_4	m_5	m_6	m_7	$m_0 \oplus m_1$ $\oplus m_2 \oplus m_3$	m_8	m_9	m_{10}	m_{11}	$m_4 \oplus m_5$ $\oplus m_6 \oplus m_7$	m_{12}	m_{13}	m_{14}	m_{15}
$m_{11} \oplus m_{14}$ $\oplus m_1 \oplus m_4$	m_3	m_6	m_9	m_{12}	$m_7 \oplus m_{10}$ $\oplus m_{13} \oplus m_0$	m_{15}	m_2	m_5	m_8	$m_3 \oplus m_6$ $\oplus m_9 \oplus m_{12}$	m_{11}	m_{14}	m_1	m_4	$m_{15} \oplus m_2$ $\oplus m_5 \oplus m_8$	m_7	m_{10}	m_{13}	m_0
$m_4 \oplus m_{13}$ $\oplus m_6 \oplus m_{15}$	m_{12}	m_5	m_{14}	m_7	$m_8 \oplus m_1$ $\oplus m_{10} \oplus m_3$	m_0	m_9	m_2	m_{11}	$m_{12} \oplus m_5$ $\oplus m_{14} \oplus m_7$	m_4	m_{13}	m_6	m_{15}	$m_0 \oplus m_9$ $\oplus m_2 \oplus m_{11}$	m_8	m_1	m_{10}	m_3
$m_{15} \oplus m_{10}$ $\oplus m_5 \oplus m_0$	m_7	m_2	m_{13}	m_8	$m_{11} \oplus m_6$ $\oplus m_1 \oplus m_{12}$	m_3	m_{14}	m_9	m_4	$m_7 \oplus m_2$ $\oplus m_{13} \oplus m_8$	m_{15}	m_{10}	m_5	m_0	$m_3 \oplus m_{14}$ $\oplus m_9 \oplus m_4$	m_{11}	m_6	m_1	m_{12}

Table 3.3: Message expansion in HAS-160

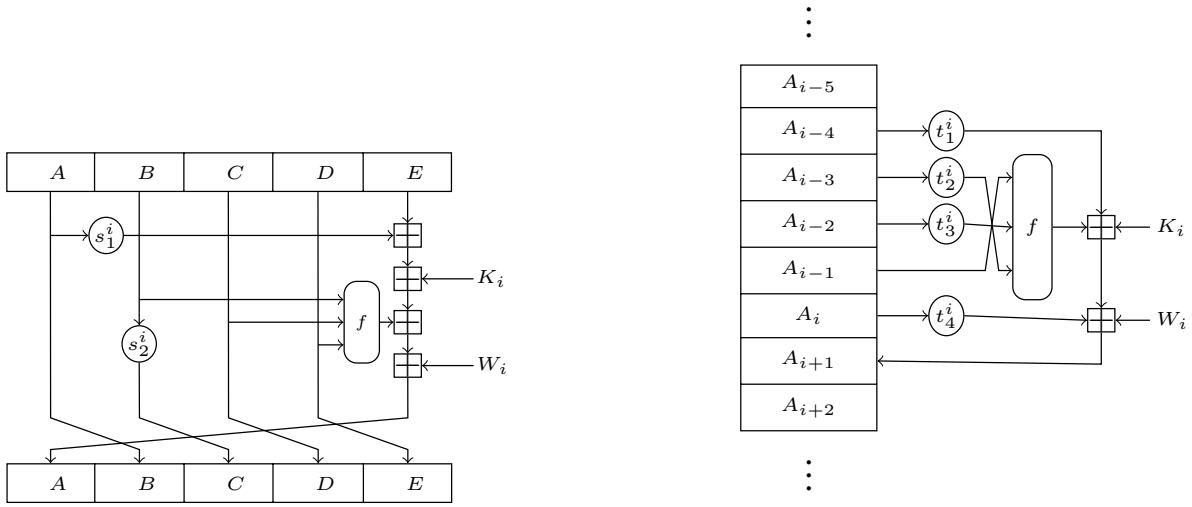

Figure 3.2: Two equivalent representations of the state update

functions f used in each step are given by

$$\begin{aligned} f_0(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\ f_1(x,y,z) &= x \oplus y \oplus z \\ f_2(x,y,z) &= (x \vee \neg z) \oplus y \end{aligned}$$

where f_0 is used in steps 0-19, f_1 is used in steps 20-39 and 60-79 and f_2 is used in steps 40-59. The constant K_i that is added in each step changes every 20 steps, taking the values 0, $5a827999$, $6ed9eba1$ and $8f1bbcdc$. The rotational constant s_1^i is specified by the following table

i mod 20	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
s_1^i	5	11	7	15	6	13	8	14	7	12	9	11	8	15	6	12	9	14	5	13

The other rotational constant s_2^i changes only each 20 steps and $s_2^i \in \{10, 17, 25, 30\}$. Following the Davies-

Message quartet								
M_A	F6513317	810F1084	FFB71009	78CC955E	C3C09F18	5379FC99	435586DA	9C9AD3B4
	00440C80	E174316A	006D1670	2B5CF68A	AB3DE600	02C9E9D3	5FE95AFF	E351DE04
M_B	F6513317	810F1084	FFB71009	78CC955E	C3C09f18	5379FC99	435786DA	9C9AD3B4
	00440C80	E174316A	006D1670	2B5CF68A	AB3FE600	02C9E9D3	5FE95AFF	E351DE04
M_C	76513317	010F1084	FFB71009	78CC955E	43C09F18	5379FC99	435786DA	1C9AD3B4
	00440C80	E174316A	006D1670	2B5CF68A	AB3FE600	02C9E9D3	5FE95AFF	E351DE04
M_D	76513317	010F1084	FFB71009	78CC955E	43C09f18	5379FC99	435586DA	1C9AD3B4
	00440C80	E174316A	006D1670	2B5CF68A	AB3DE600	02C9E9D3	5FE95AFF	E351DE04
Chaining values quartet								
IV_A	1143BE75	9A9CA381	85B3F526	DA6ABE66	70EBE920			
IV_B	3AF7BD99	D08E2E63	245C2AF0	C4456954	CAC046EA			
IV_C	3AF7B599	D08E2E63	B45C2AF0	C425694C	3BE146F2			
IV_D	1143B675	9A9CA381	15B3F526	DA4ABE5E	E20CE928			

Table 3.4: Second order collision for the full HAS-160 compression function

Meyer mode, feedforward is applied and the output of the compression is

$$(A_{80} + A_0, B_{80} + B_0, C_{80} + C_0, D_{80} + D_0, E_{80} + E_0)$$

Alternative description of HAS-160: In Fig. 3.2 (b), the compression function is shown as a recurrence relation, where A_{i+1} plays the role of A in the usual step representation. Namely, A can be considered as the only new computed word, since the rotation that is applied to B can be compensated by properly adjusting the rotation constants in the recurrence relation specification. One starts from A_{-4}, A_{-3}, A_{-2}, A_{-1} and A_0, putting these values to the previous chaining value (or the IV for the first message block) and computes the recurrence until A_{80} according to

$$A_{i+1} = A_{i-4} \lll t_1^i + K_i + f_i(A_{i-1}, A_{i-2} \lll t_3^i, A_{i-3} \lll t_2^i) + W_i + A_i \lll t_4^i \tag{3.4}$$

The rotational values t_j^i, $1 \leq j \leq 4$ are derived from s_1^i and s_2^i, where the constants related to the rotation of B in the usual representation change around the steps $20 \times k$, $k = 0, 1, 2, 3$. For instance, to compute A_{42}, we have $t_1^{41} = 17$, $t_2^{41} = 17$, $t_3^{41} = 25$ and $t_4^{41} = 11$.

3.2 Compatible paths search heuristic and application to HAS-160

In this section, we provide a new search heuristic that can be used to find compatible paths in the boomerang setting. The particular colliding quartet found by applying the heuristic on HAS-160 is provided in Table 3.4.

The heuristic uses quartets of 1-bit conditions from Table 3.2 to keep track of the bit differences in each of the four compression function executions. Apart from the single-path propagations proposed in [35],

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$m_8 \oplus m_9$ $\oplus m_{10} \oplus m_{11}$	$\textcircled{m_0}$	$\textcircled{m_1}$	m_2	m_3	$m_{12} \oplus m_{13}$ $\oplus m_{14} \oplus m_{15}$	$\textcircled{m_4}$	m_5	m_6	$\textcircled{m_7}$	$\textcircled{m_0} \oplus \textcircled{m_1}$ $\oplus m_2 \oplus m_3$	m_8	m_9	m_{10}	m_{11}	$\textcircled{m_4} \oplus m_5$ $\oplus m_6 \oplus \textcircled{m_7}$	m_{12}	m_{13}	m_{14}	m_{15}
$m_{11} \oplus m_{14}$ $\oplus \textcircled{m_1} \oplus \textcircled{m_4}$	m_3	m_6	m_9	m_{12}	$\textcircled{m_7} \oplus m_{10}$ $\oplus m_{13} \oplus \textcircled{m_0}$	m_{15}	m_2	m_5	m_8	$m_3 \oplus m_6$ $\oplus m_9 \oplus m_{12}$	m_{11}	m_{14}	$\textcircled{m_1}$	$\textcircled{m_4}$	$m_{15} \oplus m_2$ $\oplus m_5 \oplus m_8$	$\textcircled{m_7}$	m_{10}	m_{13}	$\textcircled{m_0}$
$m_4 \oplus m_{13}$ $\oplus \boxed{m_6} \oplus m_{15}$	$\boxed{m_{12}}$	m_5	m_{14}	m_7	$m_8 \oplus m_1$ $\oplus m_{10} \oplus m_3$	m_0	m_9	m_2	m_{11}	$\boxed{m_{12}} \oplus m_5$ $\oplus m_{14} \oplus m_7$	m_4	m_{13}	$\boxed{m_6}$	m_{15}	$m_0 \oplus m_9$ $\oplus m_2 \oplus m_{11}$	m_8	m_1	m_{10}	m_3
$m_{15} \oplus m_{10}$ $\oplus m_5 \oplus m_0$	m_7	m_2	m_{13}	m_8	$m_{11} \oplus \boxed{m_6}$ $\oplus m_1 \oplus \boxed{m_{12}}$	m_3	m_{14}	m_9	m_4	$m_7 \oplus m_2$ $\oplus m_{13} \oplus m_8$	m_{15}	m_{10}	m_5	m_0	$m_3 \oplus m_{14}$ $\oplus m_9 \oplus m_4$	m_{11}	$\boxed{m_6}$	m_1	$\boxed{m_{12}}$

Table 3.5: Message differentials. Backward: steps 0-39, forward: steps 40-79

step	$\Delta[A,B]$	$\Delta[D,C]$	$\Delta[B,C]$	$\Delta[A,D]$	step
9	??????????????????????????????????	??????????????????????????????????	----------	----------	⑨
10	??????????????????????????????????	??????????????????????????????????	----------	----------	10
11	??????????????????????????????????	??????????????????????????????????	----------	----------	11
12	??????????????????????????????????	??????????????????????????????????	----------	----------	12
13	??????????????????????????????????	??????????????????????????????????	----------	----------	13
⋮	⋮	⋮	⋮ [NO DIFFERENCE] ⋮		⋮
29	??????????????????????????????????	??????????????????????????????????	----------	----------	29
30	??????????????????????????????????	??????????????????????????????????	----------	----------	30
31	??????????????????????????????????	??????????????????????????????????	----------	----------	31
32	??????????????????????????????????	??????????????????????????????????	----------	----------	32
33	??????????????????????????????????	??????????????????????????????????	----------	----------	㉝
34	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	㉞
35	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	35
36	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	㊱
37	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	37
38	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	38
39	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	㊴
40	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	㊵
41	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	41
42	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	42
43	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	43
44	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	㊹
45	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	㊺
46	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	㊻
47	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	47
48	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	48
49	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????	49
50	----------	----------	??????????????????????????????????	??????????????????????????????????	㊿
51	----------	----------	??????????????????????????????????	??????????????????????????????????	(51)
52	----------	----------	??????????????????????????????????	??????????????????????????????????	52
53	----------	----------	??????????????????????????????????	??????????????????????????????????	53
54	----------	----------	??????????????????????????????????	??????????????????????????????????	54
⋮	⋮ [NO DIFFERENCE] ⋮		⋮	⋮	⋮
76	----------	----------	??????????????????????????????????	??????????????????????????????????	76
77	----------	----------	??????????????????????????????????	??????????????????????????????????	77

Table 3.6: Input for the search heuristic

two additional types of boomerang (cross-path) propagations are added. These boomerang propagations were previously listed in [89].

To specify the problem on which the heuristic is applied in the context of HAS-160, the forward and backward message differentials are provided next. Let the forward message differential consist of a one-bit difference in messages m_6 and m_{12} and the backward differential of a one-bit difference in m_0, m_1, m_4 and m_7, as shown in Table 3.5. The particular bit-position of differences is left unspecified. The choice of these difference positions is justified by the following start/end points of the expanded message differences, expressed in terms of the notation used in Fig. 3.1: $(n_0, n_1, n_2, n_3, n_4, n_5) = (0, 8, 34, 53, 78, 80)$. It can be observed that the middle part consists of 20 steps.

Now, the particular problem schematically described by Fig. 3.1 (a) is represented more specifically by Table 3.6, where the backward and forward message differentials are indicated in the first and the last

step	$\Delta[A,B]$	$\Delta[D,C]$	$\Delta[B,C]$	$\Delta[A,D]$	step
29	????????????????????????????????	????????????????????????????????	----------------------------------	----------------------------------	29
30	????????????????????????????????	????????????????????????????????	----------------------------------	----------------------------------	30
31	????????????????????????????????	????????????????????????????????	----------------------------------	----------------------------------	31
32	????????????????????????????????	????????????????????????????????	----------------------------------	----------------------------------	32
33	????????????????????????????????	????????????????????????????????	----------------------------------	----------------------------------	(33)
34	0???????????????????????????????	1???????????????????????????????	u---------------------------------	u---------------------------------	(34)
35	0??????????u???????x0???x-0?????	1??????????u???????x0???x-1?????	u----------1--------0-----u------	u----------0--------0-----u------	35
36	1x????????xu?-01B?--0Bx--u0D????	0x????????xu?-11B?--1Bx--u0D????	n----------1--u1----u----10------	n----------0--u1----u----00------	(36)
37	11-0D0B??0n0?101-x-10-01u01C???x	11-0D1B??0n1?100-x-10-00u10C???x	11-0-u---00u-10n---10-0n1un------	11-0-u---01u-10n---10-0n0un1----	37
38	00u0nn-1n01uu000uu-011u00nnn-01-	01u0nn-1n01uu110uu-001u10nnn-11-	0u1000-100111uu011-0n11u0000-u1-	0u0011-1101100uu000-0n10u0111-u1-	38
39	n101-1000100-0-0000-1---100-010n	n110-0010101-0-1001-1---001-100n	01un-n0u010u-0-u00u-1---n0u-un00	11un-n0u010u-0-u00u-1---n0u-un01	(39)
[40]	1-100010001-01--0n1-u-0-00--11-1	1-010011101-00--1n1-u-0-10--11-1	1-nu001uu01-0n--u01-1-0-u0--11-1	1-nu001uu01-0n--u11-0-0-u0--11-1	(40)
[41]	u--1--00--0-01--0--0u--001-0---1	u--0--00--0-11--1--1u--001-0---1	1--n--00--0-u1--u--u1--001-0---1	0--n--00--0-u1--u--u0--001-0---1	41
42	u---1-01001-110--n01011--n10---1	u---0-11110-011--n00000--n00---0	1---n-u1uun-n1u--00n0nn--0n0---n	0---n-u1uun-n1u--10n0nn--1n0---n	42
43	n------01----0------u------00-un	n------00----0------u------01-un	0????--0nD???0x?????1x??x--0u-10	1????--0nD???0x?????0x??x--0u-01	43
44	0-----10----------1u------	0------0----------1u------	0?????C0??????????????11?????x	0?????C0??????????????10?????x	(44)
45	------00----------u------1----	------00----------u------1----	??????00???????????????1??????1????	??????00???????????????0??????1????	(45)
46	u----------------------------	u----------------------------	1???????????????????????????????	0???????????????????????????????	(46)
47	----------------------------	----------------------------	???????????????????????????????	???????????????????????????????	47
48	-------u--------------------	-------u--------------------	???????1???????????????????????	???????0???????????????????????	48
49	-------n--------------------	-------n--------------------	???????0???????????????????????	???????1???????????????????????	49
[50]	----------------------------	----------------------------	???????????????????????????????	???????????????????????????????	(50)
51	----------------------------	----------------------------	???????????????????????????????	???????????????????????????????	(51)
52	----------------------------	----------------------------	???????????????????????????????	???????????????????????????????	52
[53]	----------------------------	----------------------------	???????????????????????????????	???????????????????????????????	53
54	----------------------------	----------------------------	???????????????????????????????	???????????????????????????????	54

Table 3.7: Output of the heuristic: compatible paths for HAS-160

Step	Conditions
33	$A_{33,14} \neq A_{32,14}$
34	$A_{34,20} = A_{33,20}$
35	$A_{35,0} \neq A_{34,0}, A_{35,16} \neq A_{33,31}, A_{35,26} \neq A_{34,26}$
36	$A_{36,3} = A_{35,3}, A_{36,9} \neq A_{35,9}, A_{36,21} = A_{35,21}, A_{36,22} = A_{34,5}, A_{36,23} = A_{35,23}$
37	$A_{37,0} = A_{36,0}, A_{37,1} = A_{36,1}, A_{37,2} \neq A_{35,17}, A_{37,13} = A_{36,13}, A_{37,23} \neq A_{36,23}$
38	$A_{38,25} = A_{36,8}$
39	$A_{39,19} \vee \overline{A}_{37,2} = 1$
40	$A_{40,17} \vee \overline{A}_{38,0} = 1, A_{40,30} \vee \overline{A}_{38,13} = 1$
41	$A_{41,16} \vee \overline{A}_{39,23} = 1$

Table 3.8: Backward differential conditions not shown in Table 3.7

column, respectively. At this point, the only information that is present in the system is that the two paths end at the corresponding steps $n_2 = 34$ and $n_3 = 53$. The output of the heuristic in the case of HAS-160 is given in Table 3.7. The full specifications of the two paths intersect at 5 steps, which is the number of inner state registers in HAS-160. Provided that the paths are compatible, one can now start from step 42 and apply the usual message modification technique to satisfy both paths, which resolves the middle of the boomerang as shown in Fig. 3.1 (a).

3.2.1 Search strategy

The approach consists of varying the position of the message difference bit, gradually extending the two paths, propagating the conditions in the quartet and backtracking in the case of a contradiction. In more detail, the heuristic proceeds as follows:

(1) Randomize the positions of active bits in the active message words.

Step	Conditions
37	$A_{37,2} = A_{36,2}, A_{37,3} \neq A_{36,3}, A_{37,10} \neq A_{36,10}, A_{37,13} = A_{36,28}, A_{37,15} = 0, A_{37,25} = A_{36,8}, A_{37,29} = A_{36,12}$
38	$A_{38,0} = 1$
39	$A_{39,4} = 1, A_{39,8} = 0, A_{39,9} = 1, A_{39,12} = 0, A_{39,17} = 0, A_{39,19} = 1$
40	$A_{40,4} = 0, A_{40,5} = 0, A_{40,8} = 0, A_{40,12} = 1$
41	$A_{41,13} = 0, A_{41,14} = 0$
42	$A_{42,7} = 0,$
43	$A_{43,6} = 0, A_{43,7} \vee \overline{A}_{41,14} = 1$
44	$A_{44,0} = 0, A_{44,1} = 0, A_{44,4} \vee \overline{A}_{42,11} = 1, A_{44,26} \vee \overline{A}_{42,1} = 1$
45	$A_{45,26} = 0$
46	$A_{46,4} \vee \overline{A}_{44,11} = 1$
47	$A_{47,4} = 1, A_{47,24} \vee \overline{A}_{45,31} = 1, A_{47,31} = 1$
48	$A_{48,31} = 0$
49	$A_{49,17} = 0$
50	$A_{50,17} = 0, A_{50,24} = 1$
51	$A_{51,17} = 0$

Table 3.9: Forward differential conditions not shown in Table 3.7

step	$\Delta[W_A, W_B]$	$\Delta[W_D, W_C]$	$\Delta[W_B, W_C]$	$\Delta[W_A, W_D]$
33	1----------	0----------	n----------	n----------
34	1----------	0----------	n----------	n----------
35	----------	----------	----------	----------
36	1----------	0----------	n----------	n----------
37	----------	----------	----------	----------
38	----------	----------	----------	----------
39	1----------	0----------	n----------	n----------
40	0----------u----------	1----------u----------	u----------1----------	u----------0----------
41	--------0-----u----------	--------0-----u----------	--------0-----1----------	--------0-----0----------
42	----------1	----------1	----------1	----------1
43	----------	----------	----------	----------
44	1----------	0----------	n----------	n----------
45	1----------	0----------	n----------	n----------
46	1----------	0----------	n----------	n----------
47	----------	----------	----------	----------
48	----------	----------	----------	----------

Table 3.10: Message differences after propagation

(2) Extend the specification of the forward/backward path backward/forward, respectively. Ensure that paths are randomized over different step invocations.

(3) Propagate all new conditions. In the case of contradiction, backtrack

(4) If the paths are fully specified on a sufficient number of steps, return the two paths

In step (1), the message disturbance position in the two differentials is randomized to achieve variation in the paths. Alternatively, one position can be fixed to bit 31 and the other position randomized at each step invocation. As for step (2), at the point where the probability of contradiction between the two paths is negligible, one can extend paths simply by randomly sampling them in the required steps and discarding non-narrow ones. Once the probability of contradiction becomes significant, the substitute/backtrack strategy according to the Table 3.11 is applied to the remaining steps. In step (3), apart from propagations

1.	???? $\mapsto$ --??
2.	??-- $\mapsto$ ----
3.	??xx $\mapsto$ --xx
4.	xx?? $\mapsto$ {uu10,nn01}
5.	xx-- $\mapsto$ {uu10,nn01}
6.	xxxx $\mapsto$ {unnu,nuun}

1.	???? $\mapsto$??--
2.	--?? $\mapsto$ ----
3.	xx?? $\mapsto$ xx--
4.	??xx $\mapsto$ {01uu,10nn}
5.	--xx $\mapsto$ {01uu, 10nn}
6.	xxxx $\mapsto$ {unnu,nuun}

Table 3.11: Substitution rules: adding information to the forward path (left) and backward path (right)

on a single path [35], quartet and quartet addition propagations (explained in Section 3.3) are applied. The heuristic ends when the full specification of two paths (containing only {-,u,n}) intersects on the number of words equal to the number of registers in the compression function inner state, as is the case in Table 3.7.

When new constraint information is to be added at a particular bit position, one can either add information to the forward path or to the backward path. Here, a clarification is necessary regarding the fact that in Table 3.7, four paths are shown, whereas the heuristic searches for a pair of paths (forward and backward). This is due to the fact that the paths on the opposite faces of the boomerang are equal (up to 0 and 1 symbols) and thus one can consider a pair of paths. Nonetheless, the inner state of the search algorithm keeps all the four paths explicitly.

The substitutions provided in Table 3.11 represent generalizations of the substitutions used in [35]. The choice whether the information will be added to the forward or the backward path is made randomly each time. The left-hand and the right-hand tables correspond to adding constraints to the forward and the backward path, respectively. Consider for example rule xx-- $\mapsto$ {uu10,nn01}. In this notation, the symbols xx-- describe a bit position for which $\delta[A_i^j, B_i^j] =$ x, $\delta[D_i^j, C_i^j] =$ x, $\delta[B_i^j, C_i^j] =$ -, $\delta[A_i^j, D_i^j] =$ -. The rule simply replaces the 'x' symbol in the forward path by 'u' or 'n', while at the same time applying the immediate propagation of the '-' symbols to '0' and '1', respectively. This rule represents a generalization of the x $\mapsto$ {u,n} rule used in [35]. Other rules can be explained in a similar manner.

One possible variation of the general heuristic above is as follows. Once the two paths are sufficiently specified so that the contradictions are likely to occur, instead of adding new constraints randomly, XYZXYZ[graduality] one can choose a parameter k and extend both paths by only k steps. If the heuristic succeeds in extending the paths by k steps, reporting that there is no contradiction in the system, more steps can be attempted. If in the intermediate steps of the search, the path was in fact contradictory and this was

not reported by 1-bit conditions, further attempts to extend or find the messages satisfying the paths will fail.

3.2.2 Application to HAS-160

In this section, we describe how the above heuristic can be applied in the case of HAS-160. First, we fix the position of the active bit in the backward differential to $b_1 = 31$. The following sequence of steps randomizes steps in the light-gray area in Table 3.6:

- Randomize the position of the forward message difference active bit b_2.
- With the message difference fully specified by b_1, b_2, sample narrow paths in the inner state words in steps denoted by light-gray in Table 3.6.
- Propagate conditions with respect to the three propagation types explained in Section 3.3. This step is applied repeatedly until none of the three propagation types can be applied at any of the bit positions.

Here, the path sampling is performed simply by initializing randomly the two instances of the path at the given step, calculating the recurrence over the required number of steps and extracting the path. If the Hamming weight of the path is greater than some pre-specified threshold, it is discarded and a new path is sampled. Using the sampling above of partial solution to the paths, the following procedure aims to find the full solution:

(1) Randomize steps in the light-gray area according to the procedure above (steps 43-49 and 34-37 in the forward and backward paths, respectively).

(2) Randomly choose (i, j), $0 \leq i \leq 31$, $38 \leq j \leq 42$, a position within the steps denoted by dark-grey in Table 3.6. If applicable, apply the substitution specified by Table 3.11. If not, choose another position. In the case there is none, return the state.

(3) Propagate conditions and backtrack in the case of contradiction. After a contradiction is reached a sufficient number of times, go to step (1).

After reducing the number of steps in which the two differentials meet from 5 to 3 (i.e., putting $k = 4$, where it should be noted that after the propagation the number of unconstrained bits will be relatively

small), we receive several paths reported as non-contradictory. At that point, there are two possible routes to verify the actual correctness of the intermediate result. One is to switch from 1-bit conditions to multi-bit conditions (such as 1.5-bit or 2.5-bit conditions [89]) that capture more information. ARXtools [89] can readily be used for this purpose. Each 2.5-bit verification using ARXtools for checking the compatibility of two paths took around 3-5 minutes. Another option is to continue with the search heuristic towards extending the specification of the paths to more steps, restarting always from the saved intermediate path state. As the knowledge in the system grows, the propagations turn a high proportion of bits into 0 and 1, which diminishes the possibility of contradiction. If the solution cannot be found after some time threshold t, the path can be abandoned. We experimented with both options above and concluded that both approaches are successful.

3.2.3 Full complexity of finding the HAS-160 second order collision

Our implementation of the heuristic found a correct pair of compatible paths in less than 5 days of execution on an 8-core Intel i7 CPU running at 2.67GHz. In more detail, as explained in Section 3.2.2, we ran the heuristic to search for paths that meet at 3 instead at 5 steps. It should be noted that due to many propagations, after the search stops, the resulting paths in fact have a small number of remaining unspecified bits in steps 38-42 (less than 32). The heuristic yielded around 8 solutions per day and among 40 returned path pairs, one turned out to be compatible and was successfully extended by one step more, as shown in Table 3.7.

The conditions for the two paths that are not explicitly given as `u,n,0,1` bits in Table 3.7 are provided in Tables 3.8 and 3.9. To find the quartet of message words and inner states that follow the two differentials in steps 34 to 49, inner state registers in step 42 are chosen to follow the conditions specified by Tables 3.8,3.9 and Table 3.7 and then the usual message modification procedure is applied backward and forward.

Once the middle steps of the quartet structure $n_2 = 34$ to $n_3 = 53$ are satisfied, the second order collision property extends to steps $n_1 = 8$ to $n_4 = 78$ with probability 1 (see Fig. 3.1 (b)). To cover all of the compression function steps, the middle steps are kept constant and the remaining ones are randomized until the second order collision property is satisfied. In particular, if m_6 and m_{15} are randomized while $m_6 \oplus m_{15}$ is kept constant, according to the message expansion specification, the inner state will be randomized for

$54 \leq i \leq 80$ and $0 \leq i \leq 35$. Similarly, if m_6 and m_4 are randomized where $m_6 \oplus m_4$ is kept constant, the randomization will happen for $52 \leq i \leq 79$ and $0 \leq i \leq 34$. Here, a small penalty in probability is paid due to the fact that the paths may be corrupted towards the start/end points. The two mentioned randomizations provide around 64 bits of freedom.

The probability that one randomization explained above yields a second-order collision can be bounded from below by p^2q^2, where p and q are the probabilities of two selected sparse differentials in steps $0 \leq i \leq n_1$ and $n_4 \leq i < 80$, respectively. By counting the number of conditions in sparse paths that happened in the quartet in Table 3.4, we obtain $p = 2^{-22}$ and $q = 2^{-3}$ and the probability lower bound $p^2q^2 = 2^{-50}$. The actual time of execution on the above mentioned PC was less than two days, due to the additional differential paths which contribute to the exact probability of achieving the second order collision property (previously named *amplified probability* [22, 89]).

3.3 Details on condition propagation

The heuristic keeps track of the current state of the system by keeping the following information in memory:

- Four differential path tables keeping the current state of bit-conditions

- $4 \times r$ carry graphs [106] (one carry graph for each of four paths consisting of r steps)

In our implementation, we used $r = 16$, keeping the information about steps 33-48. The carry graphs model the carry transitions allowed by the knowledge present in the system. Below, the three types of knowledge propagation are described. The propagations are applied as long as the system is not fully propagated with respect to all three types below.

3.3.1 Single-path propagations

An explicit example of a single-path propagation [35] (see also [106, 116]) is provided below. The constraints and the corresponding carry graphs for a particular bit position are all explicitly shown. The new propagated constraints as well as the omitted carry graph edges are indicated.

Throughout the compression function execution specified by (3.4), for any $1 \leq i \leq 80$ and $0 \leq j \leq 31$, bit A_i^j is computed based on the 5 input bits in A_{i-j}, $1 \leq j \leq 5$, the message word bit as well

δK	0110111011011001111010111010000**0**1
$\delta[W_{B,41}, W_{C,41}]$	--------0-----0---------------**-**-
$\delta[B_{37}, C_{37}]$	11-0-u---00u-10**n**---10-0n1un-----
$\delta[B_{38}, C_{38}]$	0u1000-100111uu**0**11-0n11u0000-u1-
$\delta[B_{39}, C_{39}]$	01un-n0u010u-0-u00u-1--**-**n0u-un00
$\delta[B_{40}, C_{40}]$	1-nu001uu01-0n--u01-1-0-u0--11-**1**
$\delta[B_{41}, C_{41}]$	1--n--00-**-**0-u1--u--u1--001-0---1
$\delta[B_{42}, C_{42}]$	1---n-u1uun-n1u--00n0nn--0n0--**-**n

δK	0110111011011001111010111010000**1**
$\delta[W_{B,41}, W_{C,41}]$	--------0-----0---------------**-**
$\delta[B_{37}, C_{37}]$	11-0-u---00u-10n**-**--10-0n1un-----
$\delta[B_{38}, C_{38}]$	0u1000-100111uu0**1**1-0n11u0000-u1-
$\delta[B_{39}, C_{39}]$	01un-n0u010u-0-u00u-1---**n**0u-un00
$\delta[B_{40}, C_{40}]$	1-nu001uu01-0n--u01-1-0-u0--11-**1**
$\delta[B_{41}, C_{41}]$	1--n--00--**0**-u1--u--u1--001-0---1
$\delta[B_{42}, C_{42}]$	1---n-u1uun-n1u--00n0nn--0n0---**n**

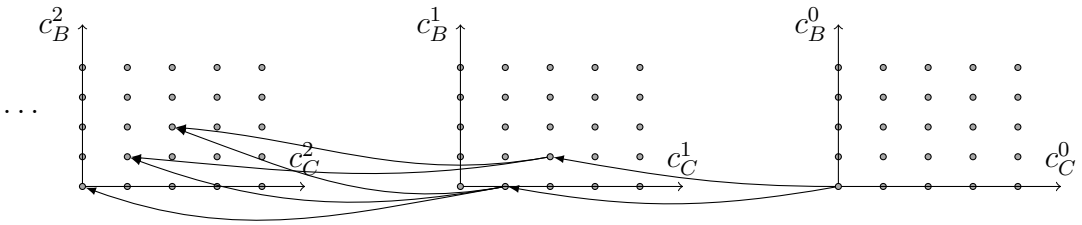

Figure 3.3: Extract of single-path path constraints

as a particular constant bit. Moreover, bit A_i^j depends on the carries coming from the computations at bit positions $j < k \leq 0$.

In Fig. 3.3, an extract of the path is provided, borrowed from the $\Delta[B, C]$ path in Table 3.7. The bit positions treated in this case are $\delta[B_{42}^1, C_{42}^1]$ (left) and $\delta[B_{42}^0, C_{42}^0]$ (right). The shaded bits are the bit positions participating in the computation of the two bits. As for the carry graph, it consists of 32 subgraphs, each comprising of 5×5 nodes. In Fig. 3.3, only the subgraphs corresponding to bit positions 1 (left) and 0 (right) are shown. Each subgraph node represents a particular carry configuration at the particular bit position. Due to the fact that there are 5 summands in (3.4), the carry value is limited to $\{0, \ldots 4\}$ and thus each subgraph contains 5×5 nodes. The edges in the graphs represent possible carry configuration transitions from bit position i to $i + 1$.

Next, the edges connecting subgraphs for bit positions $i = 0$ to $i = 1$ in Fig. 3.3 are explained. The edges shown and the corresponding bit-conditions are aligned in the sense that there are no possible propagations at the particular positions, neither from the bit-conditions to graphs nor vice-versa. According to the bit-conditions on position 0, we have

$$c_B^1 | B_{42}^0 = c_B^1 | 1 = 1 + W_{B,41}^0 + B_{37}^{15} + f_2(1, 1, 1) + 0 = 1 + W_{B,41}^0 + B_{37}^{15}$$

$$c_C^1 | C_{42}^0 = c_C^1 | 0 = 1 + W_{C,41}^0 + C_{37}^{15} + f_2(1, 0, 1) + 0 = 1 + W_{C,41}^0 + C_{37}^{15} + 1$$

From the above two equalities, it follows that $W^0_{B,41} = B^{15}_{37}$ and $W^0_{C,41} = C^{15}_{37}$. Since $\delta[W^0_{B,41}, W^0_{C,41}]$ and $\delta[B^{15}_{37}, C^{15}_{37}]$ are set to $-$, the possible carry configurations are $(c^1_B, c^1_C) \in \{(0,1),(1,2)\}$, which correspond to the two edges between the two subgraphs.

Whenever it is possible to deduce new information from what is already present in the system, propagations need to be carried out until no new information can be derived. Continuing with the setting in Fig. 3.3, assume that during the heuristic, the symbol $-$ at position $\delta[W^0_{B,41}, W^0_{C,41}]$ is replaced by `0`. Then, the propagation at this bit consists of replacing $-$ at position $\delta[B^{15}_{37}, C^{15}_{37}]$ by `0` and deleting the $(0,0) \mapsto (1,2)$ graph edge. The edge deletion continues to the left and to the right. in the case of Fig. 3.3, this amounts to deleting the edges coming out of node $(1,2)$ and continuing in the same manner throughout the rest of the subgraphs. Next, all of the influenced bit positions, either through carry graphs or through bit-conditions, need to be repropagated in a manner similar to the process described above.

3.3.2 Quartet propagations

This type of propagation is the simplest of all three types presented in this section, since it does not involve carry graphs. An example of this type of propagation is as follows. Let (i,j) denote a specific bit position in the range of the steps being considered. Let the bit-conditions $\delta[A^j_i, B^j_i]$, $\delta[D^j_i, C^j_i]$, $\delta[B^j_i, C^j_i]$, $\delta[A^j_i, D^j_i]$ in the four paths be equal to `u`, `x`, `-`, and `?`, respectively. It follows that $A^j_i = 0$, $B^j_i = 1$, $C^j_i = 1$ and $D^j_i = 0$ and thus the quartet can be readily replaced by a new one

`(ux-?)` $\mapsto$ `(uu10)`

Given a quartet of conditions, the substitution quartet is found by going through all the bit value quartets that satisfy the given condition quartet. The new quartet consists of the symbols from Table 3.2 that represent minimal sets containing the valid bit value pairs.

3.3.3 Quartet addition propagations

In this subsection, the following terminology is adopted: carry subgraphs as shown in Fig. 3.3 are called *2-graphs*. Nodes with at least one input/output edge in the 2-graphs are called *active* nodes. During the execution of the heuristic, each active 2-graph node corresponds to a possible carry configuration that

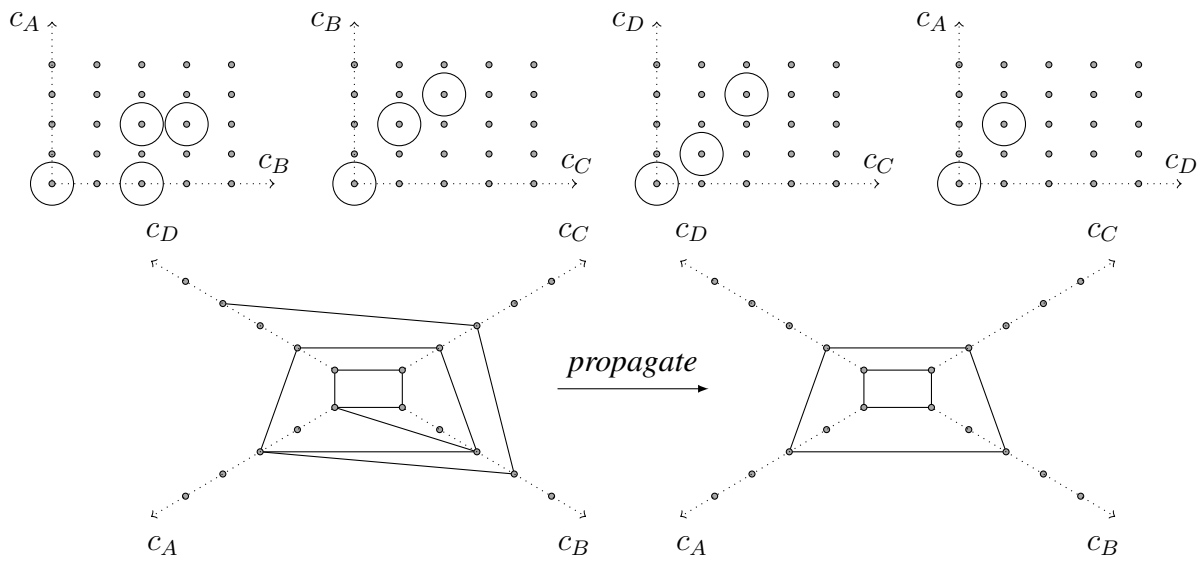

Figure 3.4: Example: 2-carry graphs and the corresponding 4-carry graph before and after propagation

has not yet been ruled out by the heuristic.

Quartet addition propagation is illustrated in Fig. 3.4. The four graphs in the top part represent a particular case of the 2-graphs that correspond to a single bit position (i, j) on paths $[A, B]$, $[B, C]$, $[D, C]$, $[A, D]$, respectively from left to right. The active nodes are circled and the information about the number of input/output edges is abstracted from the picture. The quartet addition propagation is based on the fact that the four different 2-graphs may impose incompatible constraints on the carry configurations at the considered bit position. For instance, according to the 2-graph corresponding to the path $[D, C]$ (third graph from the left in Fig. 3.4), since node $(c_D, c_C) = (3, 2)$ is active, it follows that having a carry equal to 3 at this bit position in the branch D is not ruled out. However, since there are no active nodes in the third column of the (c_A, c_D) graph, the node $(c_D, c_C) = (3, 2)$ should be deactivated.

For the purpose of deciding which 2-carry graph nodes should be deactivated, it is convenient to introduce another type of carry graph that will be called a *4-carry* graph. For each bit-position covered by the heuristic, the four 2-carry graphs are represented as one 4-carry graph, as shown in the bottom part of Fig. 3.4. The 4-carry graphs abstract the information about active nodes in the 2-carry graphs.

As shown in Fig. 3.4, the 4-carry graph has four groups of nodes that simply represent the carry values c_A, c_B, c_C and c_D, respectively. The edges in the 4-carry graph are constructed simply by mapping the active nodes in the corresponding 2-carry graphs to the edges between the corresponding node groups. This mapping is specified by an example as follows. The active nodes in the (c_A, c_D) 2-carry graph are $(0, 0)$

and $(2,1)$. This is translated to the edges $(0,0)$ and $(2,1)$ between the c_A and c_D branches in the 4-carry graph. The other three 2-carry graph active nodes are mapped to the edges analogously.

The 4-carry graph representation allows the quartet addition propagation rules to be expressed in a natural way. For that purpose, let a *cycle* denote a closed path connecting four nodes, where no two nodes are members of the same node group in the 4-graph. The propagation rules are then as follows:

(R1) Remove all "dead-end" edges, i.e., the ones with an end node of degree 1

(R2) Remove all edges that do not participate in any cycle

In the case of the propagation given in Fig. 3.4, the quartet addition propagation consisted of three applications of (R1) and one application of (R2). Since each 4-graph edge corresponds to a node in the corresponding 2-graph, the edge removal according to rules (R1) and (R2) amounts to deactivating the corresponding nodes in the 2-graph. The node deactivation is done by deleting all input and output edges for the corresponding 2-graph node. In the case of our HAS-160 search, implementing only rule (R1) turned out to be sufficient.

3.4 Conclusion

We have proposed a heuristic for searching for compatible differential paths and have applied it to HAS-160. Instead of working with 0/1 bit values, we used the reasoning on sets of bits described by 1-bit constraints. The three types of propagation used during the search (single-path propagation, quartet propagation and quartet addition propagation) are explained through particular examples. Using the 1-bit constraints along with these propagations yielded an acceptable rate of false positives and the second order collision was found. One possible future research direction is to evaluate the performance of the proposed heuristic in the case of SHA-2 with a goal of improving the attack [22] and to assess the impact of a high rate of contradictory paths reported in [100] in this context.

4

Boomerang and slide-rotational analysis of the SM3 hash function

In December of 2007, the Chinese National Cryptographic Administration Bureau released the specification of a Trusted Cryptography Module, detailing a cryptoprocessor to be used within the Trusted Computing framework in China. The module specifies a set of cryptographic algorithms that include the SMS4 block cipher, the SM2 asymmetric algorithm and SM3, a new cryptographic hash function designed by Xiaoyun Wang *et al.* [62]. The design of SM3 resembles the design of SHA-2 but includes additional fortifying features such as feeding two message-derived words into each round, as opposed to only one in the case of SHA-2.

In this chapter, we present a practical 4-sum distinguisher against the compression function of SM3 reduced to 32 rounds. In addition, we point out a slide-rotational property of SM3-XOR, which exists due to the fact that the constants used in the rounds are not independent. As explained in the previous chapter, the main idea of the second order differential cryptanalysis of hash functions [22, 23] is to use the boomerang technique [129], previously used for block ciphers, whereby the additional freedom to choose the key is exploited by using message modification techniques. Unlike in the context of first order analysis, where message modification is applied on a pair of messages, in second order analysis, this technique is applied to a quartet of values. Generally, the aim here is to find zero-sum quartets, i.e., quartets of input-output function values, for which the four inputs as well as the four outputs sum to zero. In case of a compression function that follows Davies-Meyer mode, a zero-sum can be seen as a second order collision for the compression function [22]. Finally, the zero-sum condition can be considered as an *evasive* property [55]. Such a property

is impossible to achieve with a non-negligible probability using oracle accesses to an ideal primitive. Thus, if it can be shown that the property can be satisfied for a particular construction, then it can be used for disproving its indifferentiability claims [37]. Another example of evasive properties in the context of hash functions are rotational properties [67]. Two words are said to be rotational if they are equal up to bit-wise rotation by some number of positions. If among the outputs for some carefully chosen inputs, the rotational relations hold with probability higher than the corresponding one for ideal function, then a distinguisher can be mounted [69].

In the first part of this chapter, we present a practical algorithm to find a second order collision for 32 rounds reduced version of the SM3 compression function. An interesting feature of our approach is that the two differential paths that are used for the bottom and the top part of the boomerang are not independent, as was required in [22]. This results in seemingly conflicting bit conditions [107] in the early rounds of the bottom part of the boomerang. However, as will be shown by the analysis in this chapter, the bit conflict that occurs is recoverable and it can be bypassed by using a long carry propagation on the left and the right face of the boomerang. The long carry propagation that is required to happen in order to resolve the conflicting condition is a relatively low-probability event. However, in our approach, it is ensured by message modification and does not affect the overall probability of the second-order collision search.

In the second part of the chapter, we note a slide-rotational property of SM3 and, we analyze the SM3-XOR compression function, which is the SM3 compression function with the addition mod 2^{32} replaced by XOR. In particular, we show that, for SM3-XOR, one can easily construct input-output pairs satisfying a simple rotational property. Such a property exists due to the fact that, unlike in SHA-2, the constants in rounds i, $i+1$, for $i = 0, \ldots, 63$, $i \neq 15$ are computed by bitwise rotation starting from two predefined independent values. Previously, SHA2-XOR was analyzed in [135].

As for previous work related to the work in this chapter, in [22, 23], Biryukov *et al.* presented second order analysis of SHA-2 and BLAKE. In particular, for the SHA-2 hash function, a second order collision for its compression function reduced to 46 rounds was computed [22]. The BLAKE hash function reduced to 8 rounds was shown to be suspectable to a second order attack which requires around 2^{242} compression function calls [23]. Sasaki [121] provided a second order collision for the compression function of the 5-pass HAVAL. A distinguisher for 32-round Skein-256 [92] requiring 2^{114} compression function calls was presented by Leurent *et al.* Rotational cryptanalysis was introduced by Khovratovich *et al* [67]. The SHA2-

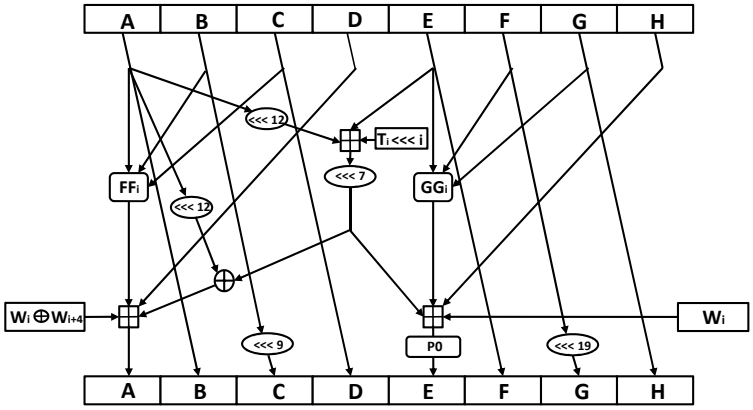

Figure 4.1: One round of the SM3 hash function

XOR compression function was analyzed by Yoshida *et al.* [135], where it was shown that an iterative differential can be used to detect non-randomness for up to 31 rounds of SHA2-XOR. The probability of the 31-round iterative differential for SHA2-XOR is 2^{-246} whereas for a random function the corresponding probability should be 2^{-256}. This allowed an attack against 32-round SHACAL-2-XOR and also a pseudo-collision attack for SHA2-XOR reduced to 34 rounds.

The rest of the chapter is organized as follows. The relevant specifications of the SM3 hash function are briefly reviewed in the next section. In Section 4.2, relevant background on higher order analysis of hash functions and the notation used throughout the chapter are given. Our second order attack against a reduced-round SM3 compression function is described in Section 4.3. The slide-rotational property of SM3 is discussed in Section 4.4. Finally, our conclusion is given in Section 4.5.

4.1 Specifications of the SM3 hash function

SM3 is a Merkle-Damgård construction that processes 512-bit input message blocks and returns a 256-bit hash value. Before hashing, the message of length l is padded by a bit set to 1, followed by k bits set to 0, where k is the smallest integer such that $l + 1 + k = 448 \bmod 512$. Finally, the remaining 64 bits are set to the value of l in the binary form. SM3 consists of two parts: the message expansion and the state update function (see Fig. 4.1). Below, we describe the two parts. The auxiliary functions, P_0 and P_1, both

operating on 32-bit words are used in the specifications and are defined by:

$$\begin{aligned} P_0(X) &= X \oplus (X \lll 9) \oplus (X \lll 17) \\ P_1(X) &= X \oplus (X \lll 15) \oplus (X \lll 23). \end{aligned}$$

Message expansion: The input here is the 512 message block represented as 16 32-bit words, $M_0, \ldots, M_{15}$. It is expanded to 68 32-bit words by letting $W_i = M_i$ for $0 \leq i < 16$ and

$$W_i = P_1(W_{j-16} \oplus W_{j-9} \oplus (W_{j-3} \lll 15)) \oplus (W_{j-13} \lll 7) \oplus W_{j-6} \tag{4.1}$$

for $16 \leq i < 68$. Another expanded message array used in SM3 is W'_i, $0 \leq i < 64$, defined by

$$W'_i = W_i \oplus W_{i+4}$$

State update transformation: In SM3, the state update starts from the fixed initial value of 8 32-bit words [62] and updates them in 64 rounds. Let A, B, C, D, E, F, G and H denote the inner state registers. As shown in Fig. 4.1, the j-th round transformation is given by

$$\begin{aligned} SS1 &= ((A \lll 12) + E + (T_j \lll j)) \lll 7, \\ SS2 &= SS1 \oplus (A \lll 12) \\ TT1 &= FF_j(A, B, C) + D + SS2 + W'_j \\ TT2 &= GG_j(E, F, G) + H + SS1 + W_j \\ D = C,\ \ C &= B \lll 9,\ \ B = A,\ \ A = TT1 \\ H = G,\ \ G &= F \lll 19,\ \ F = E,\ \ E = P_0(TT2) \end{aligned} \tag{4.2}$$

where the functions FF_j and GG_j are defined by

$$FF_j(X,Y,Z) = \begin{cases} X \oplus Y \oplus Z, & 0 \leq j \leq 15 \\ (X \wedge Y) \vee (Y \wedge Z) \vee (X \wedge Z) & 16 \leq j < 64 \end{cases}$$

$$GG_j(X,Y,Z) = \begin{cases} X \oplus Y \oplus Z, & 0 \leq j \leq 15 \\ (X \wedge Y) \vee (\neg X \wedge Z) & 16 \leq j < 64 \end{cases}$$

The round constants are $T_j = 0x79cc4519$ for $j \in \{0, \ldots, 15\}$ and $T_j = 0x7a879d8a$, for $j \in \{16, \ldots, 63\}$.

Comparison with SHA-2: The major difference between SHA-2 and SM3 is that in each round of SM3, two expanded message words are fed to the inner state, as opposed to just one in SHA-2. Also, the maximal distance between taps in the message expansion mechanism in SM3 is 4, whereas in SHA-2, it is 8. Another difference is that while addition modulo 2^{32} is used in the message expansion and the feedforward mechanisms in case of SHA-2, only XOR is used in SM3. Finally, one round of the SM3 hash function contains 8 mod 2^{32} additions, as opposed to 7 such additions in the case of SHA-2.

4.2 Background and notation

The following notation is used throughout the chapter:

- $x^{(b)}$: the b^{th} bit of an n-bit word x
- $x^{(c \cdots b)}$: the word $x^{(c)}x^{(c-1)} \cdots x^{(b)}$
- e_i: an n-bit unit vector with 1 in the i^{th} bit position
- $\overline{x}$: the bit-wise complemented word (or bit) corresponding to x
- W_j^i, $1 \leq i \leq 4$, $0 \leq j \leq 63$: expanded message words, where i denotes the boomerang branch. More precisely, $i = 1, 2$ signify the left and right branches on the front face and $i = 3, 4$ signify the left and right branches on the back face of the boomerang

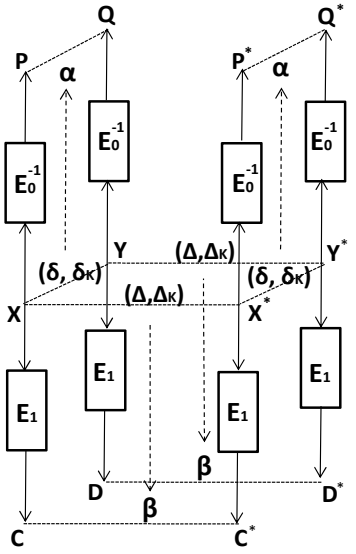

Figure 4.2: Boomerang attack against a compression function

- $h_j, 0 \leq j \leq 63$: the 256-bit compression function inner state after j rounds (e.g., h_0 is the state before any round has been executed)

- i-th round: the transformation that maps h_i into h_{i+1}

- $h_j^i, 1 \leq i \leq 4, 0 \leq j \leq 63$: the 256-bit inner state after j rounds at the i-th boomerang branch, where h_i^1 and h_i^2 correspond to the front face branches and h_i^3 and h_i^4 correspond to the back face branches.

4.2.1 Higher-order analysis of hash functions

In this section, the main idea of how to use the boomerang attack in the context of compression functions is provided. The goal of this attack on the function f is to find a quartet (x_0, x_1, x_2, x_3) such that

$$\begin{aligned} x_0 \oplus x_1 \oplus x_2 \oplus x_3 &= 0 \\ f(x_0) \oplus f(x_1) \oplus f(x_2) \oplus f(x_3) &= 0 \end{aligned} \tag{4.3}$$

which is called a *zero-sum* or equivalently, a *second-order collision*. For a more detailed exposition of these notions, please refer to the higher order cryptanalysis part of the Section 2.2. The strategies to construct a second order collision previously applied to SHA-2 and BLAKE [22, 23] varied to some degree. Here, we review the approach used in [22]. The general idea is to construct a quartet that forms boomerang structure [129] for a block cipher in the Davis-Meyer mode. The differentials used in the boomerang are related key differentials, where the secret key of the block cipher corresponds to the message block in the

case of a compression function. The encryption function is divided into two parts, $E_1 \circ E_0$. As shown in Fig. 4.2, for the bottom part of the boomerang, a related-key differential $(\Delta, \Delta_K) \rightarrow \beta$ for E_1 with probability q is constructed. Similarly, another related-key differential $(\delta, \delta_K) \rightarrow \alpha$ with probability p is used for E_0^{-1}. Then, an attempt to randomly satisfy the differentials in the boomerang structure, with probability p^2q^2 would proceed as follows:

- Randomly choose X, the inner state in the middle of the hash function execution, representing the input to E_1 (and the output of E_0). Let $X^* = X \oplus \Delta$, $Y = X \oplus \delta$ and $Y^* = X \oplus \Delta \oplus \delta$.

- Compute backward from X, X^*, Y, Y^* using E_0^{-1} to obtain P, P^*, Q, Q^*, using keys K, $K \oplus \Delta_K$, $K \oplus \delta_K$, $K \oplus \delta_K \oplus \Delta_K$, respectively.

- Compute forward from X, X^*, Y, Y^* using E_1 to obtain C, C^*, D, D^* using keys K, $K \oplus \Delta_K$, $K \oplus \delta_K$, $K \oplus \delta_K \oplus \Delta_K$, respectively.

- Verify whether $C \oplus C^* = D \oplus D^*$ and $P \oplus Q = P^* \oplus Q^*$.

If the last condition is satisfied, a zero-sum quartet is found for the encryption function in Davis-Meyer mode, since $P \oplus Q \oplus P^* \oplus Q^* = 0$ and also $(C \oplus P) \oplus (C^* \oplus P^*) \oplus (D \oplus Q) \oplus (D^* \oplus Q^*) = 0$.

To improve the efficiency of the above process, instead of trying to satisfy the boomerang randomly, message modification can be used for some of the differential paths in the boomerang. For example, in [22], message modification is applied to satisfy the middle part of the boomerang, i.e., to satisfy the two differentials paths of the function E_1. The other paths in the boomerang are satisfied randomly.

4.3 Zero-sum for reduced-round SM3

Here, a method for finding a zero-sum for the 32-round SM3 compression function is detailed. An example for the found zero-sum for the 32-round reduced SM3 compression function is given in Table 4.3.

4.3.1 Choosing the differential paths

In what follows, the backward and the forward differential paths used in the boomerang are provided and we explain why the two chosen paths are favorable. The 32-round block cipher used in the Davis-Meyer mode in the SM3 compression function is decomposed into $E_1 \circ E_0$. The function E_0 consists

of rounds $r = 0, \ldots, 14$ and the function E_1 consists of rounds $15, \ldots, 31$. The forward and backward differential paths for E_1 and E_0 used in the attack are given at the end of the Appendix (Tables 4.1 and 4.2, respectively). For example, the last row in Table 4.1 denotes that there is no active bits between the two inner states representing the output of round 14.

The paths have been found by linearizing the compression function and then applying CodingTool [108], a tool for effective search for low Hamming weight codewords of a given linear code. The linearization amounted to replacing addition mod 2^{32} by XOR and the functions FF_i and GG_i, $16 \leq i < 64$ by zero functions. Functions FF_i and GG_i, $0 \leq i < 16$, have been left unchanged, since they are already linear. The input to CodingTool is a generating matrix of a linear code and the output is a low Hamming-weight codeword. Here, the linear code in question is the linear mapping from the message to the concatenated bit-vectors representing consecutive inner states of the compression function. The matrix describing this mapping, i.e., the generating matrix of the linear code, is obtained by applying the linearized compression function to the unit vectors. Then, the matrix is fed to the low Hamming weight codeword search algorithm. The search is done for both functions E_0^{-1} and E_1.

The probabilities for the provided differentials are obtained by multiplying the round probabilities provided in Tables 4.1 and 4.2. As shown in the tables, the overall probabilities for E_0^{-1} and E_1 paths are 2^{-25} and 2^{-57}, respectively. Therefore, assuming that the events of satisfying the two differential paths are independent, by using a naive search, the probability of finding a quartet that yields a zero-sum would be $2^{-2\times(25+57)} = 2^{-164}$. Instead of applying a naive search, message modification allows a major improvement to this complexity. The differential path for E_1 in rounds $15 - 19$ is satisfied by using message modification and the rest, that is, rounds $0 - 4$ and also round 31, is satisfied by a random search. Then, the search complexity drops to $2^{2\times(25+2)} = 2^{54}$.

To clarify the advantage of the paths given in Tables 4.1 and 4.2, first we note that the Hamming weight of the two paths does not change if the paths are rotated by some fixed number of bit-positions. Therefore, the rotation amount is a free parameter that can be chosen to maximize the probability of success. As for the backward path shown in Table 4.1, the choice of the rotation amount is simply due to the fact that the number of active most significant bits is maximized, which improves the differential probability, given that the active most significant bits do not affect such probability.

As for the rotational amount used for the forward differential, the number of most significant bits in

rounds $15 - 19$ is less important, because in these rounds, the path is satisfied by message modification and there is no need to minimize the path probability by forcing bits to be on the most significant bit position. What matters for the forward differential is that one of the active bits in round 31 would correspond to the most significant bit since, as explained above, this condition is satisfied by a random search, as is the case with the particular paths in Table 4.2. This reduces the exhaustive search by more than a factor of 2 and when the alternative paths for the path in round 31 are taken into account, the reduction is by a factor of around 3. This reduction is relatively significant, since our goal is to find a zero-sum efficiently with a practical complexity.

In addition to the path given in Table 4.2, it can be easily verified that there exist two more paths obtained by rotating the one in Table 4.2 such that, in round 31, one of the active bits is the most significant bit. These two paths are obtained using rotation to the left by 17 and 9 positions. However, each of the three paths have conflicting bit conditions in rounds $15 - 16$ with respect to the backward differential, including the path given in Table 4.2. In the next subsection, we show that the conflict between the backward path and the forward path given in Table 4.2 is recoverable, i.e., it can be bypassed by message modification, which is the reason why we focus on this path.

4.3.2 Message modification and the conflicting bits

In this section, we provide some details on the message modification technique in the context of the boomerang, i.e., where the message modification is performed on a quartet of values, instead of on a pair of values. The focus is put on resolving the particular conflict between the bit-conditions on the two faces of boomerang that occurs in our SM3 analysis. A simple general tool for bypassing such conflicts, in the case when this is possible, is provided.

The message modification procedure that satisfies rounds $15 - 19$ of the forward differential on the front and the back face of the boomerang proceeds as follows. Here, by message modification, we also assume the modification of the middle inner state registers. Following the notation specified in Section 4.2, let h_{15}^1 and h_{15}^2 denote the inner states satisfying the difference specified by the first row of Table 4.2. In the back face of the boomerang, the corresponding inner states are $h_{15}^3 = h_{15}^1$ and $h_{15}^4 = h_{15}^2$. The goal is to have both the front face and back face differences propagate according to Table 4.2.

The modification procedure can start from h_{15}^1. The message words W_{15}^1, W_{15}^2 and the inner states

h_{15}^1, h_{15}^2 are modified so that the difference on the front face between h_{15}^1 and h_{15}^2 propagates according to Table 4.2. Now, if the bit-conditions for controlling the propagation between h_{15}^1 and h_{15}^3 do not conflict with the bit-conditions for h_{15}^1 and h_{15}^2, the message modification procedure can be applied again, but this time on h_{15}^1 and h_{15}^3. Then, clearly, the difference between h_{15}^3 and h_{15}^4 propagates in the exact same way as the difference between h_{15}^1 and h_{15}^2 and the goal is fulfilled. Also, due to the boomerang property, the difference between h_{15}^2 and h_{15}^4 propagates in the same way as h_{15}^1 and h_{15}^3.

However, the conflicting bit condition occurs due to the backward and forward paths in Tables 4.1 and 4.2. The conflicting condition and how it is resolved is explained below and also visualized in Fig. 4.3, where rounds 15 and 16 are shown. As depicted in the figure, the front-side and back-side differences (due to the forward path) are denoted by Δ and the left-side and right-side differences (due to the backward path) by δ. The conflict arises when one attempts to force the bit 22 difference coming from D_{15} not propagate to more than one bit in both front-side and back-side of A_{16}. The problem is that bit 22 is also active on the left-side and the front-side, as shown in Fig. 4.3 beside the $W_{15} \oplus W_{19}$ word. In particular, due to the message difference specified by the backward differential, we have

$$(W_{15}^1 \oplus W_{19}^1) \oplus (W_{15}^3 \oplus W_{19}^3) = e_{14} \oplus e_{22} \oplus e_{31} \tag{4.4}$$

At the same time, due to (4.2), for the active bit 22 of D_{15} to propagate only to a 1-bit difference in A_{16} in both the front and the back face of the boomerang, bits 22 of both

$$\alpha^1 = FF_{15}(A_{15}^1, B_{15}^1, C_{15}^1) + SS2_{15}^1 + (W_{15}^1 \oplus W_{19}^1)$$
$$\alpha^3 = FF_{15}(A_{15}^3, B_{15}^3, C_{15}^3) + SS2_{15}^3 + (W_{15}^3 \oplus W_{19}^3)$$

have to be fixed to the bit value $b = 0$ if there is no carry generated at bit position 21 in neither of the two additions

$$A_{16}^1 = \alpha^1 + D_{15}^1 \tag{4.5}$$
$$A_{16}^3 = \alpha^3 + D_{15}^3 \tag{4.6}$$

If, however, there is a carry generated at position 21 in both (4.5) and (4.6), then the above bit b needs to

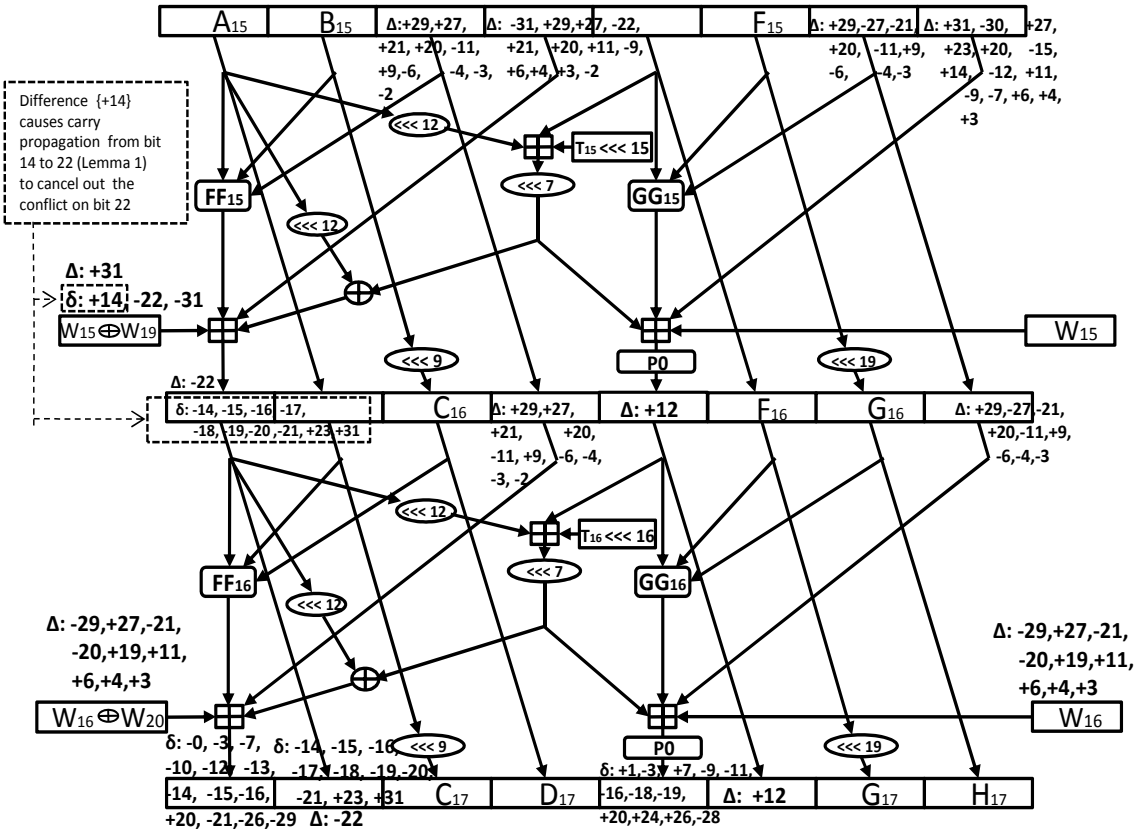

Figure 4.3: Resolving the conflicting bit condition in round 16

be fixed to 1. Thus, under the assumption that the carry is either generated at position 21 in both (4.5) and (4.6), or not generated in neither of these two additions, both the front and the back face of the boomerang cannot be satisfied, since bit 22 is active in (4.4).

Thus, to bypass the conflicting bit, the fact that, in (4.4), bit 14 is also active should be utilized. Then, by using an 8-bit long carry propagation from bit 14 to bit 22 on the left and the right face of the boomerang, the existence of carry at bit 21 can be ensured in exactly one of the additions (4.5) and (4.6), which cancels out the activation of bit 22 on the left and the right face of the boomerang.

Next, a simple lemma that ensures long carry propagation for the purpose of deactivating a particular bit is provided. The lemma can be used during the message modification process, i.e., whenever a deactivation of a bit by a carry chain is needed. Consider sums of n-bit words $X + S$ and $X' + S$ and suppose that bits k and l, where $k < l$ are active in X. The lemma specifies how to perform message modification on X so that the bit l in $X + S$ and $X' + S$ remains inactive.

Lemma 3 *Let X, X', and S be n-bit words. Also, let $0 \leq k < l \leq n$ and $X \oplus X' = e_l \oplus e_k$. If*

$$2^k \leq X^{(k-1..0)} + S^{(k..0)} < 2^{k+1} \tag{4.7}$$

$$X^{(k)} = \overline{X^{(l)}} \tag{4.8}$$

$$X^{(l-1..k+1)} = \overline{S^{(l-1..k+1)}} \tag{4.9}$$

then

$$(X + S) \oplus (X' + S) = e_{l-1} \oplus \ldots \oplus e_{k+1} \oplus e_k. \tag{4.10}$$

Proof: Due to (4.7) and the fact that $X^{(k)} \neq X'^{(k)}$, exactly one of the values in $\{X^{(k..0)} + S^{(k..0)}, X'^{(k..0)} + S^{(k..0)}\}$ will have a carry propagation from bit position k to $k+1$. Therefore, using (4.9), it is clear that $(X + S)^{(l-1..k)} \oplus (X' + S)^{(l-1..k)} = e_{l-1} \oplus \ldots \oplus e_{k+1} \oplus e_k$. Since $X^{(l)} \neq X'^{(l)}$, $X + S$ and $X' + S$ will be equal on bit l. Finally, from (4.8), $(X + S)^{(m)} = (X' + S)^{(m)}$ for $m > l$, $m < n$, which completes the proof. □

To resolve the round 15 bit conflict explained above, the Lemma can be applied by letting $X = (W^1_{15} \oplus W^1_{19})$, $X' = (W^3_{15} \oplus W^3_{19})$, $S = (FF_{15}(A^1_{15}, B^1_{15}, C^1_{15}) + SS2^1_{15} + D^1_{15})$, $k = 14$, $l = 22$, $n = 32$, where the active bit 31 in $(W^1_{15} \oplus W^1_{19})$ can be ignored since it is on the most significant position. Then, $X = (W^1_{15} \oplus W^1_{19})$ is modified to satisfy requirements (4.7)-(4.9). This message modification is done by only modifying a subset of bits $\{21, 20 \ldots, 0\}$ of W^1_{19}. Now, on the front face of the boomerang, A^1_{16} and A^2_{16} will have a signed difference of -22, while, at the back face of the boomerang, A^3_{16} and A^4_{16} will have the same signed difference, as specified by the forward differential, i.e., the conflicting bit condition has been bypassed.

As can be verified, this is the only conflicting condition in rounds 15 and 16. The application of Lemma 3 can also be seen as a way to increase the probability of satisfying the paths in the boomerang by 2^8, since the event of the carry propagation takes place with probability of around 2^{-8}. As for the conflicting conditions in rounds $17 - 19$, they are resolved by repeated applications of Lemma 3. The result of the message modification procedure described above is a quartet that satisfies the differences in rounds $15 - 19$. The quartet is used as input for the next stage of the zero-sum search procedure.

i	Inner state	$W_i \oplus W_{i+4}$	W_i	Prob
0	A: $+22$ B: -22 C: $-31, -22$ D: $-31, +29, +27,$ $+22, +21, +20,$ $+11, +9, -6,$ $-4, +3, -2$ E: $+12$ F: -12 G: $-31, +12$ H: $+31, -29, -27,$ $+21, +20, +12,$ $-11, -9, +6,$ $-4, +3$	$-29, +27,$ $+21, +20,$ $+19, -11,$ $-6, +4, -3$	$-29, +27,$ $+21, +20$ $-19, +11,$ $+6, -4, -3$	2^{-23}
1	B: $+22$ C: -31 D: $-31, -22$ F: $+12$ G: -31 H: $-31, +12$			2^{-2}
2	C: $+31$ D: -31 G: $+31$ H: -31			1
3	D: $+31$ H: $+31$	-31	-31	1
4				
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
15				

Table 4.1: Backward differential path with probability 2^{-25}

4.3.3 Searching for the zero-sum

After the differences in rounds $15 - 19$ have been satisfied, the remaining paths in the boomerang are satisfied randomly. The corresponding search procedure is facilitated due to the existence of many neutral bits with respect to the majority of already satisfied conditions. Namely, all the bits in words $W_8, \ldots, W_{14}$ are neutral with respect to the rounds $15 - 19$. The search proceeds by randomly satisfying the remaining conditions, i.e., the path given in Table 4.1 and also the last round of the path in Table 4.2. Although the nominal probability is as low as $2^{2\times(25+2)} = 2^{-54}$, this probability does not take into account the alternative differential paths that are similar to the ones specified above. Due to these additional paths, the actual probability is much larger, as was confirmed by executing the search procedure. Similar observation have been reported in for example in [23, 92].

The zero-sum for 32 rounds of the SM3 compression function, given in Table 4.3, was found after around 20 days of computation using 4 workstations, each with four 2.4 GHz Dual-Core AMD Opteron processors.

A natural way to extend the attack to more rounds would be to increase the number of rounds on

i	Inner state	$W_i \oplus W_{i+4}$	W_i	Prob
15	C: $+29, +27, +21,$ $+20, -11, +9,$ $-6, -4, -3, -2$ D: $-31, +29, +27,$ $-22, +21, +20,$ $+11, -9, +6,$ $+4, +3, -2$ G: $+29, -27, -21,$ $+20, -11, +9,$ $-6, -4, -3,$ H: $+31, -30, +27,$ $+23, +20, -15,$ $+14, -12, +11,$ $-9, -7, +6, +4, +3$	$+31$		2^{-26}
16	A: -22 D: $+29, +27, +21,$ $+20, -11, +9,$ $-6, -4, -3, -2$ E: $+12$ H: $+29, -27, -21,$ $+20, -11, +9,$ $-6, -4, -3$	$-29, +27,$ $-21, -20,$ $+19, +11,$ $+6, +4, +3$	$-29, +27,$ $+21, -20,$ $+19, +11,$ $+6, +4, +3$	2^{-25}
17	B: -22 F: $+12$			2^{-2}
18	C: -31 G: $+31$			2^{-2}
19	D: -31 H: $+31$	-31	-31	1
20				
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
31		$-31, -22, +14$		2^{-2}
32	A: $-31, -22, +14$			

Table 4.2: Forward differential path with probability 2^{-57}

which the message modification is performed. Namely, it should be noted that the consequence of the neutral bits described above is the fact that the complexities to satisfy the bottom and the top differential add up and do not multiply, similar as in [22]. Thus, one can imagine extending the middle rounds so that the message modification procedure terminates after a practical amount of computation. However, caution should be exercised when estimating the number of rounds that can be added since the number of conflicting bits grows quickly which consequently increases the number of necessary applications of Lemma 3. According to our experience, satisfying the conditions in the boomerang becomes increasingly difficult without an automated systematic procedure as the number of conflicting conditions grows. Thus, extending the number of rounds to be dealt with by an automated message modification procedure is the goal of our future research. Such a procedure would also be relevant for the SHA-2 boomerang analysis [22].

$A^1, B^1, \ldots, H^1$	$0x7a0d7b2f$ $0x776a25d5$ $0xcff768ac$ $0xd2eb20d5$ $0xd2c08d9b$ $0x744d3e5c$ $0xdf04e2ba$ $0x2cd3bb94$
$W_0^1, \ldots, W_{15}^1$	$0x31c2ba4f$ $0x336fa0d6$ $0x94a32431$ $0x9d3caeaa$ $0x814d29d5$ $0xc8ebf2e6$ $0x7a41c51f$ $0x3aa0bedd$ $0xaac4fb81$ $0xd584f8b$ $0x619690c2$ $0xfac9a4d1$ $0x2a28a333$ $0x175fb61c$ $0x6133d9ab$ $0x81e48a5e$
$A^2, B^2, \ldots, H^2$	$0x3c6a7d6d$ $0xbbdf98c0$ $0x5da6c569$ $0x89c62255$ $0xd75bec0e$ $0x6117de9c$ $0xb3f56bd3$ $0x3445d8b4$
$W_0^2, \ldots, W_{15}^2$	$0x2dc2be4f$ $0x33efa256$ $0xbc9b2c69$ $0x3d6cfeaa$ $0x3cea950$ $0x48ebf2e6$ $0x7241ad1f$ $0x32a0bedd$ $0xaac4fb81$ $0xbd083f9b$ $0x618698ca$ $0xfac9a4d1$ $0xaa28a333$ $0x1f5f9e1c$ $0x6133d9ab$ $0x81e48a5e$
$A^3, B^3, \ldots, H^3$	$0x7a4d7b2f$ $0x772a25d5$ $0x4fb768ac$ $0x6a7b1aa9$ $0xd2c09d9b$ $0x744d2e5c$ $0x5f04d2ba$ $0xc493b1cc$
$W_0^3, \ldots, W_{15}^3$	$0x19fab217$ $0x336fa0d6$ $0x94a32431$ $0x1d3caeaa$ $0x814d29d5$ $0xc8ebf2e6$ $0x7a41c51f$ $0x3aa0bedd$ $0xaac4fb81$ $0xd584f8b$ $0x619690c2$ $0xfac9a4d1$ $0x2a28a333$ $0x175fb61c$ $0x6133d9ab$ $0x81e48a5e$
$A^4, B^4, \ldots, H^4$	$0x3c2a7d6d$ $0xbb9f98c0$ $0xdde6c569$ $0x31561829$ $0xd75bfc0e$ $0x6117ce9c$ $0x33f55bd3$ $0xdc05d2ec$
$W_0^4, \ldots, W_{15}^4$	$0x5fab617$ $0x33efa256$ $0xbc9b2c69$ $0xbd6cfeaa$ $0x3cea950$ $0x48ebf2e6$ $0x7241ad1f$ $0x32a0bedd$ $0xaac4fb81$ $0xbd083f9b$ $0x618698ca$ $0xfac9a4d1$ $0xaa28a333$ $0x1f5f9e1c$ $0x6133d9ab$ $0x81e48a5e$

Table 4.3: An example for a zero-sum for 32 rounds of the SM3 compression function

$A^1, B^1, \ldots, H^1$	$0x565060b7$ $0x125d5655$ $0x285c7653$ $0xeaf5fe1e$ $0xda8bd7dd$ $0xb8bb1904$ $0x43bcaf18$ $0x7cf88895$
$W_0^1, \ldots, W_{15}^1$	$0x8f450bbd$ $0x4a0c9922$ $0x73dd44f8$ $0x9eceaaf8$ $0x33b13e20$ $0xb59d9c33$ $0x6b5a5f23$ $0xc0d2b468$ $0x7a9a1e16$ $0xaff62878$ $0x3fbb01f4$ $0x75278787$ $0xac0b849e$ $0x498f3045$ $0x62687c15$ $0xd3498eb$
$A^2, B^2, \ldots, H^2$	$0x24baacaa$ $0x53285c76$ $0xd5ebfc3d$ $0xdf1ee2a6$ $0x71763209$ $0x2bc610ef$ $0xf9f1112a$ $0xffeb86a4$
$W_0^2, \ldots, W_{15}^2$	$0x7efa7542$ $0x1e8a177b$ $0x94193244$ $0xe7ba89f0$ $0x3d9d55f1$ $0x67627c40$ $0x6b3b3867$ $0xd6b4be46$ $0x81a568d1$ $0xf5343c2c$ $0x5fec50f1$ $0x7f7603e8$ $0xea4f0f0e$ $0x5817093d$ $0x931e608a$ $0xc4d0f82a$

Table 4.4: An example for a slide-rotational pair for the SM3-XOR compression function

4.4 A slide-rotational property of SM3-XOR

As mentioned at the beginning of this chapter, the SHA2-XOR compression function was previously studied by Yoshida *et al.* [135]. In this section, we show that, in the case of the full SM3-XOR, pairs satisfying a certain rotational relation can be easily generated. An example of such a pair for the SM3-XOR is provided in Table 4.4. The possibility of practical generation of such evasive [55] SM3-XOR pairs demonstrates the existence of a non-trivial property which is not known to exist in SHA2-XOR.

The above mentioned property exists due to the fact that the constants over the 64 rounds of SM3 are related. According to the SM3 specification, in rounds $j \in \{0, \ldots, 15\}$, one constant rotated by j is utilized, whereas the other constant rotated by j is used in rounds $j \in \{16, \ldots, 63\}$. Since operations like XOR,

FF_i, GG_i, $0 \leq i < 64$, that are used in the SM3-XOR round function preserve the rotational property, it is natural to attempt a rotational attack, as provided below. We note that if instead of SM3-XOR, the original SM3 compression function is used, the addition mod 2^{32} transforms the attack into a probabilistic one, as outlined below. Due to the high number of additions per round, it appears difficult to exploit this rotational property directly and therefore the security of the SM3 compression function, at this stage of analysis, does not seem to be directly affected.

Two 32-bit words X, Y are said to be *rotational* if $X = Y \lll n$. Let messages W and W^* satisfy $W_1^* = W_0 \lll 1, W_2^* = W_1 \lll 1, \ldots, W_{16}^* = W_{15} \lll 1$. Below, a procedure for the instant generation of pairs v, v^* such that

$$\begin{aligned} v_1^* = v_0 \lll 1, v_2^* = v_1 \lll 8, v_3^* = v_2 \lll 1 \\ v_5^* = v_4 \lll 1, v_6^* = v_5 \lll 18, v_7^* = v_6 \lll 1 \\ V_1^* = V_0 \lll 1, V_2^* = V_1 \lll 8, V_3^* = V_2 \lll 1 \\ V_5^* = V_4 \lll 1, V_6^* = V_5 \lll 18, V_7^* = V_6 \lll 1 \end{aligned} \tag{4.11}$$

is provided, where $V = \text{SM3-XOR}(v, W)$, $V^* = \text{SM3-XOR}(v^*, W^*)$ and v_i, V_i for $0 \leq i \leq 7$ denote i-th 32-bit word in the v and V, respectively. For a random function, a random (v, W), (v^*, W^*) satisfying the above constraints will yield the corresponding V and V^* with probability $2^{-6\times 32} = 2^{-192}$, since (4.11) imposes 6 32-bit conditions on V, V^*.

4.4.1 Constructing a slide-rotational pair

We start by the following observations:

- The slide rotational messages expand to slide-rotational expanded messages with probability 1. In particular, fix $W_0, \ldots, W_{15}$ and let

$$W_1^* = W_0 \lll 1, W_2^* = W_1 \lll 1, \ldots, W_{16}^* = W_{15} \lll 1 \tag{4.12}$$

After expanding both W and W^*, we have $W_{i+1}^* = W_i \lll 1$, for $i = \{0, 1, \ldots, 62\}$ and also $W_{i+1}'^* = W_i' \lll 1$, for $i = \{0, 1, \ldots, 66\}$.

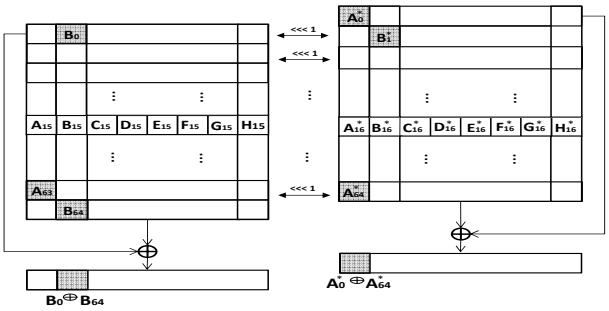

Figure 4.4: The slide-rotational attack against SM3-XOR

- We recall that T_i, $0 \leq i \leq 63$ are the round constants. If we have

$$W^*_{i+1} = W_i \lll 1, W'^*_{i+1} = W'_i \lll 1, T_{i+1} = T_i \lll 1 \tag{4.13}$$

$$A^*_{i+1} = A_i \lll 1, B^*_{i+1} = B_i \lll 1, \ldots, H^*_{i+1} = H_i \lll 1 \tag{4.14}$$

for $i = k$, then (4.14) will also hold for $i = k + 1$, where $k = 0, \ldots, 62$.

The observations above suggest that sliding can be introduced, as depicted in Fig. 4.4. Namely, consider randomly initializing W and letting W^* satisfy (4.12). Moreover, $A_0, B_0 \ldots, H_0$ is chosen randomly and the inner state registers after the first round in the second instance of the hash function are initialized according to (4.14). Then, until round 15, due to (4.13), the rotational property in the inner state registers will be preserved. Once the two instances reach rounds 15 and 16, respectively, a different round transformation is applied in the two instances and the rotational property may discontinue. This problem is bypassed by starting from the middle, i.e., by populating the inner states entering the critical rounds 15 and 16 (see Fig. 4.4).

Next, we explain how to bypass the critical rounds 15 and 16 which may discontinue the rotational property. That way, it is possible to have the slide property hold over all rounds of the two hash function instances. The idea is to start by populating the inner states entering the critical rounds 15 and 16 (see Fig. 4.4). In particular, a rotational pair $(A_{15}, \ldots, H_{15})$, $(A^*_{16}, \ldots, H^*_{16})$ is carefully chosen so that $(A_{16}, \ldots, H_{16})$ and $(A^*_{17}, \ldots, H^*_{17})$ satisfy relation (4.14). It should be noted that the rotational property may be destroyed only between A_{16} and A^*_{17} and between E_{16} and E^*_{17}, since the other registers go through identical rotational-preserving transformations in round 15 and round 16. As for A_{16} and A^*_{17}, for the purpose of tracking the possible rotational disturbance between the two registers, the equation to compute these two registers can

be rewritten as

$$A_{16} = FF_{15}(A_{15}, B_{15}, C_{15}) \oplus (T_{15} \lll 22) \oplus \alpha \tag{4.15}$$

$$A^*_{17} = FF_{16}(A^*_{16}, B^*_{16}, C^*_{16}) \oplus (T_{16} \lll 23) \oplus \alpha^* \tag{4.16}$$

where $\alpha = D_{15} \oplus W_{15} \oplus W_{19} \oplus (((A_{15} \lll 12) \oplus E_{15}) \lll 7) \oplus (A_{15} \lll 12)$ and $\alpha^* = D^*_{16} \oplus W^*_{16} \oplus W^*_{20} \oplus (((A^*_{16} \lll 12) \oplus E^*_{16}) \lll 7) \oplus (A^*_{16} \lll 12)$. Since (4.14) and (4.13) hold for $i = 15$, $\alpha^* = \alpha \lll 1$. Therefore, to have A_{16} and A^*_{17} be a rotational pair, it suffices to make $FF_{15}(A_{15}, B_{15}, C_{15}) \oplus (T_{15} \lll 22)$ and $FF_{16}(A^*_{16}, B^*_{16}, C^*_{16}) \oplus (T_{16} \lll 23)$ satisfy the rotational property. After expressing A^*_{16}, B^*_{16}, C^*_{16} in terms of A_{15}, B_{15}, C_{15} and using that FF_{15} and FF_{16} preserve the rotational property, the condition can be expressed in terms of A_{15}, B_{15}, C_{15} as follows:

$$FF_{15}(A_{15}, B_{15}, C_{15}) \oplus FF_{16}(A_{15}, B_{15}, C_{15}) = (T_{15} \oplus T_{16}) \lll 22 \tag{4.17}$$

When applied on 1-bit values X, Y and Z, the equation $FF_{15}(X, Y, Z) \oplus FF_{16}(X, Y, Z) = 0$ is satisfied for 2 out of 8 (X, Y, Z) values. Since the Hamming weight of the right-hand side of (4.17) is equal to 14, the number of solutions to the equation is $2^{18} \times 6^{14} = 2^{32} \times 3^{14}$. As for preserving the rotational property between E_{16} and E^*_{17}, developing the registers as in (4.15) and then forming the equation of the form (4.17) yields that the number of solutions E_{15}, F_{15} and G_{15} is $4^{32} = 2^{64}$. Therefore, the number of solutions for $(A_{15}, \ldots, H_{15})$ that pass the disturbance in rounds 15 and 16 is $2^{32} \times 3^{14} \times 2^{64} \times 2^{64} \approx 2^{182.19}$, since D_{15} and H_{15} are free variables. For such pairs, it follows that relations (4.11) are satisfied.

When instead of SM3-XOR, the SM3 compression function is considered, this property turns into a probabilistic one. Following [68], if $p_r = P[(x \lll r) + (y \lll r) = (x + y) \lll r]$ where x and y are 32-bit words, then $p_1 = 2^{-1.415}$. Since there exists 8 additions in one SM3 round, the probability that one round and its corresponding slided round will preserve the rotational property is given by $(p_1)^8 = 2^{-11.320}$ [68].

4.5 Conclusion

In this chapter, a second order collision for the SM3 compression function reduced to 32 rounds is presented. The top and the bottom differentials, used in the boomerang, impose seemingly conflicting

conditions. The novelty of our method is that these conditions are resolved during message modification by using long carry propagation on the left and right face of the boomerang. In the second part of the chapter, a slide-rotational property of SM3-XOR function is exposed and an example of a slide-rotational pair for SM3-XOR compression function is given.

5

Cryptanalysis of the Loiss stream cipher

Several word-oriented LFSR-based stream ciphers have been recently proposed and standardized. Examples include ZUC [50], proposed for use in the 4G mobile networks and also SNOW 3G [51], which is deployed in the 3GPP networks. The usual word-oriented LFSR-based design consists of a linear part, which produces sequences with good statistical properties and a finite state machine which provides non-linearity for the state transition function.

In 2011, the Loiss stream cipher [53] was proposed by a team from the State Key Laboratory of Information Security in China. The cipher follows the above mentioned design approach: it includes a byte-oriented LFSR and an FSM. The novelty in the design of Loiss is that its FSM includes a structure called a Byte Oriented-Mixer with Memory (BOMM) which is a 16 byte array adopted from the idea of the RC4 inner state. The BOMM structure is updated in a pseudorandom manner.

The Loiss key scheduling algorithm utilizes a usual approach to provide non-linearity over all the inner state bits. During the initialization phase, the FSM output is connected to the LFSR update function. This ensures that after the initialization process, the LFSR content depends non-linearly on the key and the IV. Such an approach has been previously used in several LFSR-based word-oriented constructions such as the SNOW family of ciphers [51]. In Loiss, however, the FSM contains the BOMM element which is updated slowly in a pseudo-random manner. The feedback to the LFSR, used in the initialization phase, passes through this BOMM which turns out to be exploitable in a differential-style attack since the BOMM does not properly diffuse differences.

In this chapter, we provide a related-key attack of a practical complexity against the Loiss stream cipher by exploiting this weakness in its key scheduling algorithm (see also [93] for a work that was done

independently of our results). The attack requires two related keys differing in one byte, a computational work of around 2^{26} Loiss initializations, $2^{25.8}$ chosen-IVs for both of the related keys, offline precomputation of around 2^{26} Loiss initializations and a storage space of 2^{32} words. This shows that the additional design complication, i.e., the addition of the BOMM mechanism, weakens the cipher instead of strengthening it. We also discuss the possibility of extending such a related-key attack into a resynchronization single-key attack. Finally, we show that Loiss does not properly resist to slide attacks.

The rest of the chapter is organized as follows. In section 5.1, we briefly review relevant specifications of the Loiss stream cipher. Our related-key attack is detailed in section 5.2 where we also discuss the possibility of extending the attack to the single-key scenario. In section 5.3, we show that Loiss is not resistant to slide attacks. Finally, our conclusion is given in section 5.4.

5.1 Specifications of Loiss

Figure 5.1 shows a schematic description of the Loiss stream cipher. In here, we briefly review

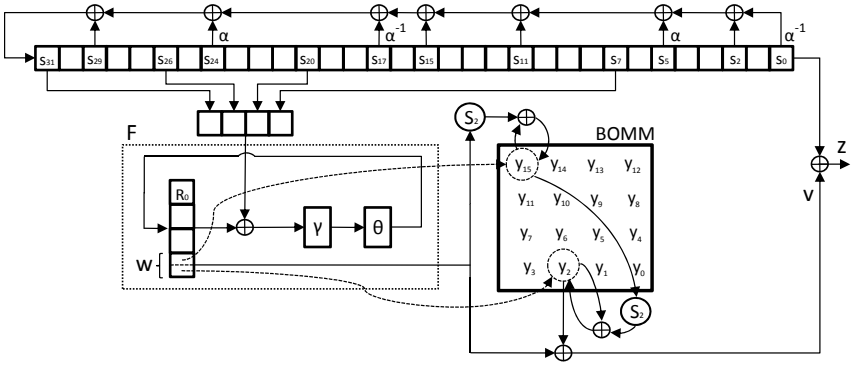

Figure 5.1: Loiss stream cipher

relevant components of the cipher. Let F_{2^8} denote the quotient field $F_2[x]/(\pi(x))$, where the corresponding primitive polynomial $\pi(x) = x^8 + x^7 + x^5 + x^3 + 1$. If α is a root of the polynomial $\pi(x)$ in F_{2^8}, then the LFSR of Loiss is defined over F_{2^8} using the characteristic polynomial

$$f(x) = x^{32} + x^{29} + \alpha x^{24} + \alpha^{-1}x^{17} + x^{15} + x^{11} + \alpha x^5 + x^2 + \alpha^{-1}.$$

The usual bijection between the elements of F_{2^8} and 8-bit binary values is used. The LFSR consists of 32 byte registers denoted by s_i, $0 \leq i \leq 31$. Restating the above equation, if $s_0^t, \ldots, s_{31}^t$ denote the LFSR

registers after t LFSR clocks, then the LFSR update function is defined by

$$s_{31}^{t+1} = s_{29}^t \oplus \alpha s_{24}^t \oplus \alpha^{-1} s_{17}^t \oplus s_{15}^t \oplus s_{11}^t \oplus \alpha s_5^t \oplus s_2^t \oplus \alpha^{-1} s_0^t \tag{5.1}$$

and $s_i^{t+1} = s_{i+1}^t$ for $0 \leq i \leq 30$.

The FSM consists of the function F and the BOMM. The function F compresses 32-bit words into 8-bit values. It utilizes a 32-bit memory unit R and takes LFSR registers s_{31}, s_{26}, s_{20} and s_7 as input. In particular, in each step, the output of F is taken to be the 8 leftmost bits of the register R, after which the R value is updated by

$$X = s_{31}^t | s_{26}^t | s_{20}^t | s_7^t$$
$$R^{t+1} = \theta(\gamma(X \oplus R^t))$$

where γ is the S-box layer which uses 8×8 S-box S_1 and is defined by

$$\gamma(x_1|x_2|x_3|x_4) = S_1(x_1)|S_1(x_2)|S_1(x_3)|S_1(x_4)$$

and θ is a linear transformation layer defined by

$$\theta(x) = x \oplus (x \lll 2) \oplus (x \lll 10) \oplus (x \lll 18) \oplus (x \lll 24)$$

Since the attack technique provided in this work does not depend on the particular choice of the used S-boxes, we refer the reader to [53] for the specifications of S_1 and S_2.

As for the BOMM structure, it utilizes 16 memory units, i.e., bytes $y_0, \ldots, y_{15}$. The BOMM function maps 8-bit values to 8-bit values. Let w and v denote the input and output of the BOMM function. Denote the nibbles of its input w as $h = w \gg 4$ and $l = w \bmod 16$. Then, the BOMM function returns $v = y_h^t \oplus w$, after which the update of its memory units takes place as follows:

$y_l^{t+1} = y_l^t \oplus S_2(w)$

If $h \neq l$, then

$y_h^{t+1} = y_h^t \oplus S_2(y_l^{t+1})$

else

$$y_h^{t+1} = y_l^{t+1} \oplus S_2(y_l^{t+1})$$

$$y_i^{t+1} = y_i^t, \text{ for } 0 \leq i \leq 15 \text{ and } i \notin \{h, l\}$$

where S_2 is an 8×8 S-box. In the FSM update step, the input to the BOMM function, i.e., the w value, is taken to be leftmost byte of the output of the F function.

The initialization procedure of Loiss proceeds as follows. The register R is set to zero, i.e., $R^0 = 0$. If the key K and the initialization vector IV are represented byte-wise as

$$\begin{aligned} K &= K_{15}|K_{14}|\cdots|K_0 \\ IV &= IV_{15}|IV_{14}|\cdots|IV_0, \end{aligned} \tag{5.2}$$

then the starting inner state $(s_{31}^0, \ldots, s_0^0, R^0, y_{15}^0, \ldots, y_0^0)$ is loaded with the K and IV as follows:

$$s_i^0 = K_i, \;\; s_{i+16}^0 = K_i \oplus IV_i, \;\; y_i^0 = IV_i \tag{5.3}$$

for $0 \leq i \leq 15$. Then, Loiss runs for 64 steps and the output of the BOMM takes part in the LFSR update step. In other words, instead of (5.1), the following LFSR update function is used:

$$s_{31}^{t+1} = s_{29}^t \oplus \alpha s_{24}^t \oplus \alpha^{-1} s_{17}^t \oplus s_{15}^t \oplus s_{11}^t \oplus \alpha s_5^t \oplus s_2^t \oplus \alpha^{-1} s_0^t \oplus \boldsymbol{v^t} \tag{5.4}$$

Then, the keystream generation stage starts. Loiss generator produces one byte of keystream per step:

$$z^t = s_0^t \oplus v^t.$$

In general, except for the new BOMM component, the whole Loiss design is very similar to the design of the SNOW 3G cipher. It is also interesting to note that the same θ linear layer has been used in the SMS4 block cipher [2] and also in ZUC [50].

5.2 Proposed Attack

In this section, a differential-style attack against the Loiss key scheduling algorithm is presented. The attack requires two related keys that differ in one byte. It also requires the ability to resynchronize the cipher under the two keys with chosen IV values.

The attack starts by having the pair of inner states right after the key loading step differ only in one LFSR byte and one BOMM byte. Then, the idea is to have the LFSR difference fully cancelled. We use the fact that the BOMM output participates in the LFSR update step during the initialization and the BOMM difference helps us to cancel out the LFSR difference through the feedback. Once the difference in the LFSR is fully cancelled, only the BOMM component is active and moreover, with a single byte difference. Then, since the BOMM does not have proper diffusion properties, the single-byte difference stays localized in the BOMM until the end of the initialization, which can be detected from the keystream.

The probability of the event that a given BOMM byte is not used during the initialization is $(\frac{15}{16})^{128} \approx 2^{-12}$, since a BOMM element is consulted 128 times during the 64 initialization steps. If the active byte has not been used until the end of the initialization, the two instances of the cipher generate several equal keystream bytes with high probability. Namely, the difference at the point where the keystream is to be produced will be of low-weight and localized in the BOMM. Therefore, spotting large number of zero bytes in the starting keystream byte difference indicates that the LFSR difference cancellations described above took place. These cancellations happen only when certain equations in the starting LFSR bytes are satisfied and consequently, since the starting LFSR bits are related to the key bits, information about the key bits leaks.

Let K and K' differ only in the byte K_3. The steps of the attack can be summarized as follows:

- Construct a list of (IV, IV') pairs for which the LFSR state difference cancellation happens. The cancellation event is described in section 5.2.1, the distinguisher used to detect this event is given in section 5.2.2 and a procedure for collecting the (IV, IV') pairs is provided in section 5.2.3.

- Use this collection of IVs as input to the filtering procedure to filter the wrong key candidates, as described in section 5.2.4.

The attack recovers 92 bits of the key and the remaining $128 - 92 = 36$ bits can be obtained by brute force. In another variant of the attack, 112 bits of the key are recovered and the rest are found by brute-force.

5.2.1 Cancelling the LFSR difference

In this section, a necessary and sufficient condition for the starting inner state difference to be fully cancelled in the LFSR after 4 steps is provided. The condition is specified in terms of the leftmost byte of the R register in the first 4 steps. Then, the conditions on the R register as provided by Observation 1 below leak information on the early LFSR bytes and thus about the secret key.

The key-loading mechanism (5.3) allows having a chosen difference only at bytes s_3 and y_3 at time $t = 0$. Namely, it suffices to have

$$K_3 \oplus K'_3 = IV_3 \oplus IV'_3 = \delta \tag{5.5}$$

and the rest of the K, K' and also IV, IV' bytes to have a zero-difference. Moreover, the key-loading mechanism trivially allows choosing the starting values of the y_3 register. This is done by choosing the IV_3 byte, since the IV is simply copied into the BOMM. This shows that the assumptions required by Observation 1 (i.e., the particular difference value $0x02$ in s_3 and y_3 and also the $y_3 = 0x9d$ constant) can be satisfied. Recall that w^t denotes the leftmost byte of the R register at time $t \geq 0$.

Observation 1 *Let a pair of Loiss inner states have only s_3 and y_3 bytes active, both with difference $0x02$. Also, let $y_3 = 0x9d$. Then, after 4 steps, the LFSR does not contain any active byte if and only if*

$$(w^0, w^1, w^2, w^3) = (0x00, 0x33, 0xK?, 0x3?) \tag{5.6}$$

where K is any hexadecimal digit different from $0x3$ and the symbol "?" denotes any hexadecimal digit.

Proof: From the cipher specification, $w^0 = 0x00$ is true regardless of the condition on the left-hand side. The two directions of the proof are provided as follows.

($\Leftarrow$): The change of the difference in the BOMM is described in Figure 5.2. In the first step, since $w^0 = 0x00$, both the value and the difference of y_3^0 remain unchanged and the LFSR difference is moved from s_3 to s_2. Since $w^1 = 0x33$ and both s_2 and y_3^1 are active with the same difference, they cancel out and the corresponding LFSR byte becomes inactive. As for the LFSR difference, it is just moved to s_1. Another effect of the second step is the change of the difference in y_3 byte from $0x02$ to $\alpha^{-1} \times 2$. Namely, expanding the difference in the y_3 byte and substituting the initial choice of $y_3^0 = 0x9d$ and also the choice of the

starting difference $\delta = 0x02$ gives

$$y_3^2 \oplus y_3'^2 = \delta \oplus S_2(y_3^0 \oplus S_2(0x33)) \oplus S_2(y_3^0 \oplus \delta \oplus S_2(0x33)) = \alpha^{-1} \times 0x02 \tag{5.7}$$

The third step moves the s_1 active byte to s_0, since $w^2 \gg 4 \neq 3$ and leaves the y_3 difference unchanged. Finally, since $w^3 \gg 4 = 0x3$, the difference in y_3 cancels out the difference in the LFSR update function (5.4) in the fourth step and this direction of the proof follows.

($\Rightarrow$): Clearly, $w^1 \gg 4 = 0x3$ since otherwise s_{31}^1 would be active and the LFSR after 4 steps would necessarily have at least one active byte. Moreover, $K = w^2 \gg 4 \neq 0x3$ holds since y_3^2 is necessarily active and otherwise there would be a difference introduced to the LFSR on byte s_{31}^2.

To show that $w^1 \bmod 4 = 0x3$, assume the contrary. In that case, the full LFSR cancellation in the fourth step cannot happen. Namely, in the second step, the difference in register y_3^1 remains unchanged, i.e., it remains equal to $0x02$. Therefore, during the third step, the existing one byte difference in the BOMM has to evolve to $\alpha^{-1} \times 2$ in order for the LFSR cancellation to happen in the fourth step. However, according to the S_2 specification, the input S_2 difference $0x02$ cannot be transformed to the output difference $\alpha^{-1} \times 2$ and thus $w^1 \bmod 4 = 0x3$.

Now, according to the ($\Leftarrow$) direction of the proof, (5.7) holds. To show that $w^3 \gg 4 = 0x3$, suppose the contrary. Since the LFSR byte s_0 is active at the fourth step (with the difference $0x2$), for this difference to be cancelled out, the BOMM output byte at step four has to be active with the same difference. Thus, the difference in y_3^2 which is equal to $\alpha^{-1} \times 0x02$ has to remain $\alpha^{-1} \times 0x02$ after passing through the S_2 S-box. This difference will necessarily be induced on some other BOMM byte since $K \neq 3$. However, such a possibility is ruled out by the S_2 specification: the S_2 S-box cannot map the input difference $\alpha^{-1} \times 2$ to $\alpha^{-1} \times 2$ output difference. It should be noted that this was possible in (5.7), since the same byte was updated twice in step 1. Therefore, $w^3 \gg 4 = 0x3$ has to hold. □

A descriptive overview of the cancellation specified by Observation 1 is as follows. In Figure 5.2, the BOMM and the LFSR bytes s_3, s_2, s_1, s_0 are shown during the first four steps. In the second and the fourth states in the figure, the cancellation of the LFSR difference by the feedback byte to the LFSR update is denoted. In the first step, the difference does not enter neither the LFSR update function nor the feedback value (since $w^0 = 0x00$). In the second step, it is required that $w^1 = 0x33$ for the difference to be cancelled and also to be updated to the next necessary BOMM difference value, $2\alpha^{-1}$. In the third step, the difference

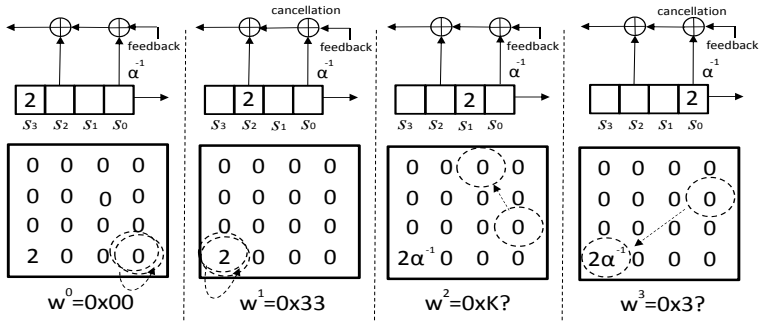

Figure 5.2: Illustration of the differences in the BOMM structure at times $t = 0, 1, 2, 3$

is neither passed to the LFSR nor changed in the BOMM. Finally, in the fourth step, the difference in the LFSR byte s_0 is cancelled and the LFSR becomes fully inactive.

It should be noted that Observation 1 holds for other difference values apart from $\delta = 0x02$. This set is give explicitely:

$$\Delta = \{2, 5, 7, 9, d, 10, 11, 13, 15, 16, 18, 19, 1a, 1c, 1d, 1f, 20, 21, 25, 27, 2a, 2b, 2c, 2e, 2f, 31,$$
$$32, 37, 38, 39, 3d, 3e, 45, 48, 4a, 4b, 4d, 4f, 50, 54, 56, 57, 5b, 5c, 5d, 60, 61, 63, 64, 65, 66, 69, 6a,$$
$$6b, 6c, 6f, 70, 72, 74, 75, 77, 79, 7a, 7b, 7d, 7f, 80, 81, 82, 87, 89, 8b, 8d, 8e, 92, 94, 96, 97, 98, 99,$$
$$9a, 9c, 9d, 9e, a0, a1, a9, aa, ac, ae, af, b0, b2, b5, b8, ba, bc, bd, bf, c0, c1, c3, c4, c5, c7, ca, cd,$$
$$d1, d2, d3, d4, d6, d7, d8, da, dc, de, df, e1, e2, e8, eb, ed, f0, f1, f2, f3, f4, f7, f9, fb, fc, ff\}$$

In particular, Observation 1 is true for any $\delta \in F_2^8$ such that the input differences $\alpha^{-1} \times \delta$ and δ cannot be mapped to the output differences $\alpha^{-1} \times \delta$ by the S_2 S-box (see the ($\Rightarrow$) part of the proof). For each difference from the set Δ, the initial constant for y_0^3 is calculated from (5.7).

The overall probability that there will be only one BOMM byte, y_3, active after all of the 64 steps of the key scheduling procedure is estimated next. For this event to happen, it suffices to have (5.6) satisfied in addition to ensuring that the y_3 difference does not propagate to other bytes during the initialization procedure. The event (5.6) happens with probability $p_w = 2^{-8} \times \frac{15}{16} \times 2^{-4} \approx 2^{-12.1}$. The event by which the y_3 difference does not propagate to any other byte is equivalent to the event of $w^t \bmod 16 \neq 0x3$ and $w^t \gg 4 \neq 0x3$ for $4 \leq t \leq 63$, and $w^2 \bmod 16 \neq 3$. The latter condition is included since Observation 1 does not rule out the possibility of the spreading of the y_3 difference to another byte during step 3. Thus, the probability that y_3 does not spread to any other byte is $p_s = (\frac{15}{16})^{2\times 60+1} \approx 2^{-11.3}$. Thus, a randomly chosen

key-IV pair satisfying (5.5) such that the assumptions of Observation 1 are satisfied produces a pair of inner states with only one active byte with probability

$$p = p_s \times p_w = 2^{-12.1} \times 2^{-11.3} = 2^{-23.4} \tag{5.8}$$

under the usual independence assumption.

5.2.2 Distinguishing Loiss pairs

In the previous subsection, we showed that it is possible to have a pair of Loiss inner states with only one active byte (located in the BOMM) after the initialization. Here, a distinguisher for the keystreams generated by a pair of such states is provided. The goal is to minimize the probability of false positives and false negatives.

Let the time at which the two instances of the cipher differ by only one BOMM byte be $t = 0$. Since at this time most of the words are inactive, it is natural to attempt distinguishing Loiss key stream pairs from random keystream pairs by simply counting the number of equal bytes in the two outputs. Such a distinguisher depends on parameters n and m, where n is the number of keystream generation steps that will be considered and m is the number of equal corresponding words in steps $0, \ldots, n-1$. The distinguisher can be formulated as:

- Count the number of indices $0 \leq i < n$ such that $z_i = z'_i$
- If this count is $\geq m$ return *Loiss keystreams*, otherwise return *Random*.

Good values for (n, m) can be chosen by consulting Table 5.1. In this table, the probability of false positives and false negatives for some representative (n, m) points has been tabulated. Details on how the values in the table have been calculated are provided below.

The false positive probability signifies the probability that in two random sequences of n bytes, more than m corresponding bytes will be equal. On the other hand, the false negative probability signifies the probability that two Loiss instances with only one active byte located in the BOMM, will produce strictly less than m equal bytes. For the purpose of the attack above, it is necessary to keep the probability of false positives low, since a false positive would lead to generating equations that have incorrect key values as solutions.

(n, m)	P[false positive] $\approx$	P[false negative] $\approx$
$(16, 6)$	$2^{-35.1}$	$2^{-22.41}$
$(16, 8)$	$2^{-50.4}$	$2^{-16.00}$
$(24, 8)$	$2^{-44.6}$	$2^{-24.01}$
$(24, 10)$	$2^{-59.2}$	$2^{-19.91}$
$(32, 10)$	$2^{-54.2}$	$2^{-27.6}$
$(32, 12)$	$2^{-68.3}$	$2^{-20.68}$

Table 5.1: Effectiveness of the distinguisher for different (n, m) parameters

As for the false positive probability, it has been calculated by using the formula describing the probability that in n randomly generated bytes, at least m of them are equal to zero. Namely, if l denotes the number of zeros in the sample, then $P[\text{false positive}] = P[l \geq m] = \sum_{l=m,\ldots,n} \binom{n}{l} \left(\frac{1}{256}\right)^l \left(\frac{255}{256}\right)^{n-l}$.

The false negative probability has been calculated experimentally by randomly generating a pair of equal Loiss inner states and then inducing a random difference at a random BOMM byte. After running the cipher for n steps, the number of equal bytes is counted. If such number is strictly smaller than m, a counter is incremented. After repeating the previous procedure for 2^{28} times and dividing the resulting counter by 2^{28}, an approximation of the probability of a false negatives is obtained.

For the purpose of the distinguisher used in the next subsection, taking $(n, m) = (32, 10)$ makes the probability of the attack failure marginally small, i.e., equal to around $2^{25.8} \times 2^{-54.2}$, since the distinguisher is applied for around $2^{25.8}$ times and a false positive answer would lead to wrong conclusions about the value of key bytes.

5.2.3 Finding the correct IVs

According to the cancellation probability (5.8), for around one in $2^{23.4}$ randomly chosen IVs, if the key-IV pair satisfies (5.5), the inner state right after the initialization will have only the y_3 BOMM byte active. Given the choices for the distinguisher given in Table 5.1, such event can be reliably detected. Hereafter, such IVs will be called *correct* IVs. In this section, it is shown that the correct IVs can be found with probability better than $2^{-23.4}$, which helps us reduce the final number of chosen-IVs required for the attack.

In particular, once one correct IV is obtained, more such correct IVs can be found with better prob-

ability. Namely, changing certain IV bytes in a correct IV does not influence all w_1, w_2 and w_3 bytes. For instance, perturbing byte IV_{11} in a correct IV does not change $w^1 = 0x33$ value and the the probability (5.8) that the new IV will also be a correct one increases by a factor of 2^8. More precisely, let T_1 denote a collection of IV bytes such that any change in bytes from T_1 leaves R^1 unchanged, but changes R^t, $t \geq 2$. It is easy to verify that $T_1 = \{IV_1, IV_5, IV_8, IV_{11}, IV_{13}\}$.

Thus, after finding one correct IV, varying only the bytes from T_1 can serve to find more correct IVs with better probability. Such a set of IVs would result in the IVs for which the R^1 word is constant. However, the attack step provided in subsection 5.2.4, which takes the correct IV set as its input, requires that the IVs produce about 5 different R^1 values. Similarly, there have to be around 360 different R^2 values. These two numbers of required different R^1 and R^2 values are necessary to minimize the number of key byte candidates that will be recovered, as will be explained in the next subsection. Therefore, the search procedure that produces the input to the procedure in the next subsection can proceed as follows:

- Let sets $L_0 = L_1 = L_2 = L_3 = L_4 = \emptyset$.
- Generate 5 correct IVs randomly and place them in sets L_i, $0 \leq i \leq 4$, respectively. In more detail, for each randomly generated IV, compute IV' according to (5.5) and apply the distinguisher from subsection 5.2.2. If the distinguisher returns a positive answer, a correct IV has been found.
- For $0 \leq i \leq 4$
    - Using the IV from each L_i, generate more corrects IVs such that the L_i sets contain 72 IVs each. In particular, the new correct IVs are generated by randomizing the starting IV bytes specified by T_1 and applying the distinguisher.

The output of the above procedure are sets L_i, $0 \leq i \leq 3$, each containing 72 IVs for which the R^1 is constant. This procedure takes around $5 \times 2^{23.4} + 5 \times 72 \times 2^{23.4-8} \approx 2^{26}$ chosen-IV queries on both Loiss instances. If instead of applying the previous procedure, all of the $5 \times 72 = 360$ correct IVs were generated randomly, the number of chosen IV queries would be $360 \times 2^{23.4} \approx 2^{31.9}$.

5.2.4 Filtering the key bytes

In each Loiss step, the function F updates the register R by a transformation similar to one round of a block cipher, where the R value plays the role of the plaintext and the four LFSR registers play the role

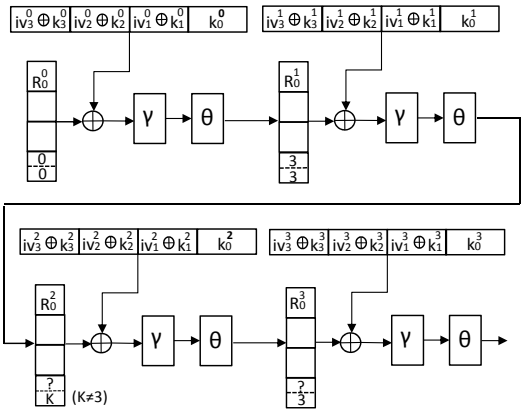

Figure 5.3: The R register in times $0 \leq t \leq 3$

of the round key. The goal hereafter is to recover the LFSR registers fed to F in the first three initialization steps, i.e., s_{7+i}, s_{20+i}, s_{26+i}, s_{31+i} for $0 \leq i \leq 2$. In particular, since the LFSR bytes in question can be represented as a sum of the key and the IV, the goal is to recover the key part in these bytes. First, the application of the F function in the first three steps is represented in the form of

$$R^{i+1} = F(R^i, k_3^i \oplus iv_3^i | k_2^i \oplus iv_2^i | k_1^i \oplus iv_1^i | k_0^i) \tag{5.9}$$

for $0 \leq i \leq 2$, where k_3^i, k_2^i, k_1^i and k_0^i depend only on the original key bytes and iv_3^i, iv_2^i and iv_1^i depend only on the IV bytes. More precisely, in the first step

$$\begin{aligned} k_3^0 = K_{15}, k_2^0 = K_{10}, k_1^0 = K_4, k_0^0 = K_7 \\ iv_3^0 = IV_{15}, iv_2^0 = IV_{10}, iv_1^0 = IV_4 \end{aligned} \tag{5.10}$$

In the second step, we have

$$\begin{aligned} k_3^1 &= K_{13} \oplus \alpha K_8 \oplus \alpha^{-1} K_1 \oplus K_{15} \oplus K_{11} \oplus \alpha K_5 \oplus K_2 \oplus \alpha^{-1} K_0 \\ k_2^1 &= K_{11}, k_1^1 = K_5, k_0^1 = K_8 \\ iv_3^1 &= IV_{13} \oplus \alpha IV_8 \oplus \alpha^{-1} IV_1 \oplus IV_{15} \oplus IV_{11} \oplus \alpha IV_5 \oplus \\ &\qquad IV_2 \oplus \alpha^{-1} IV_0 \oplus f^1 \\ iv_2^1 &= IV_{11}, iv_1^1 = IV_5 \end{aligned} \tag{5.11}$$

and in the third step

$$
\begin{aligned}
k_3^2 &= K_{14} \oplus \alpha K_9 \oplus \alpha^{-1} K_2 \oplus K_0 \oplus K_{12} \oplus \alpha K_6 \oplus K_3 \oplus \alpha^{-1} K_1 \\
k_2^2 &= K_{12}, k_1^2 = K_6, k_0^2 = K_9 \\
iv_3^2 &= IV_{14} \oplus \alpha IV_9 \oplus \alpha^{-1} IV_2 \oplus IV_0 \oplus IV_{12} \oplus \alpha IV_6 \oplus IV_3 \oplus \\
&\qquad \alpha^{-1} IV_1 \oplus f^2 \\
iv_2^2 &= IV_{12}, iv_1^2 = IV_6
\end{aligned}
\tag{5.12}
$$

where f^1, f^2 represent the feedback bytes. If the IV bytes in the right-hand side of (5.10), (5.11) and (5.12) are taken from a correct IV, then (5.6) will hold. In that case, also, the feedback bytes will be $f^1 = IV_0$ and $f^2 = IV_3 \oplus 0x33$. The first three steps of the F function when a correct IV is used are represented schematically in Figure 5.3.

Then, the filtering procedure for recovering k_j^i, $0 \leq i \leq 2$, $0 \leq j \leq 3$ amounts to substituting the F function key guesses into (5.9) along with the iv bytes derived from a correct IV and then verifying whether (5.6) holds. In particular, the filtering procedure is done round by round. As for the first F round, (5.6) amounts to $R^1 \gg 24 = 0x33$ and thus a candidate for $k^0 = k_3^0|k_2^0|k_1^0|k_0^0$ passes the criterion with probability 2^{-8}, which implies that 5 correct IVs are sufficient to uniquely determine k^0 with a good probability. We have verified experimentally that there is enough diffusion in one F-round to find the key uniquely with just 5 correct IVs.

As for the second step of the initialization phase, where (5.9) is executed for $i = 1$, first it should be noted that R^1 is known for each IV since $k_3^0|k_2^0|k_1^0|k_0^0$ is known. According to (5.6), the second F round criterion amounts to $R^2 \gg 28 \neq 3$. Thus, a guess for $k^1 = k_3^1|k_2^1|k_1^1|k_0^1$ passes the criterion with probability $\frac{15}{16}$. Assuming that all the wrong key bits can be eliminated, around 332 correct IV values will be required, since $2^{32} \times (\frac{15}{16})^{332} \approx 1$. In the previous section, 360 correct IVs has been generated, which ensures the unique recovery of k^1 with good probability. Throughout all our experiments, the number of candidates for k^1 that pass the test was consistently equal to 16. Without going into why 16 candidates always pass the test, it is noted that these candidates can be eliminated during the third F round filtering. The third F round criterion is $R^3 \gg 28 = 3$ and one can expect that the candidate for $k^2 = k_3^2|k_2^2|k_1^2|k_0^2$ passes with probability 2^{-4}, meaning that around 8 correct IV values will be required. The filtering is done for each of

the 16 candidates for k^1. Again, experimentally, it was found that 16 candidates for k^2 always pass the test and therefore there will be 16 candidates at the end of the filtering procedure.

It remains to state how the correct IVs are drawn from L_i, $0 \leq i \leq 4$ to derive the iv^i values specified by (5.10), (5.11) and (5.12). For the first F round filtering, the 5 IVs are chosen from L_0, L_1, L_2, L_3 and L_4, respectively, which ensures that different 5 iv^0 values will be derived and that the filtering procedure will properly work. The second and third round choice of the IVs is arbitrary.

Attack complexity: After the filtering procedure described above, there will remain 16 candidates for k^i_j, $0 \leq i \leq 2$, $0 \leq j \leq 3$ (96 bits). Each of the 16 candidates yields a linear system in the cipher key bytes determined by (5.10), (5.11) and (5.12). Since the linear equations in the system are independent, it follows that a $96 - 4 = 92$-bit constraint on the key K is specified. At this point, the attacker can either brute-force the remaining $128 - 92 = 36$ key bits or continue with the filtering process described above to deduce more key bits. In case of brute-forcing the 36 bits, the total complexity of the attack is dominated by around 2^{36} Loiss initialization procedures.

In the case where the filtering process is continued, the criterion $R^4 \gg 28 \neq 3$ can be used to filter out more key bits. Namely, expanding the corresponding iv^3 and k^3 values in a way analogous to (5.10)-(5.12), while taking into account the feedback byte in the LFSR update, reveals that altogether 20 more key bits can be recovered. In that case, the total complexity is dominated by the complexity of the above filtering procedures. The most expensive step is the filtering based on the second F round. We recall that in this filtering step, for each of the 360 correct IVs, each 32-bit key value is tested and eliminated if $R^2 \gg 28 \neq 3$ does not hold. Instead of applying the F function $2^{32} \times 360 \approx 2^{40.5}$ times, one can go through all key candidates for a particular IV, eliminate around $\frac{15}{16}$ of them and then, for the next IV, only go through the remaining candidates. In such a case, the number of applications of F is $\sum_{i=0}^{360}(\frac{15}{16})^i 2^{32} \approx 2^{36}$. To have further optimization, a table containing 2^{32} entries and representing F function can be prepared in advance. To measure the computational complexity of the attack in terms of Loiss initializations, a conservative estimate that one table lookup costs around 2^{-4} of a reasonable implementation of one Loiss initialization step could be accepted. Then, since there are $64 = 2^6$ steps in the initialization, the final complexity amounts to around 2^{26} Loiss initializations, $2^{25.8}$ chosen-IVs for both keys, storage space of 2^{32} 32-bit words and offline precomputation of 2^{32} applications of F, which is less than 2^{26} Loiss initializations, since each Loiss initialization includes 2^6 F computations.

Our attack was implemented and tested on a PC with 3 GHz Intel Pentium 4 processor with one core. Our implementation takes less than one hour to recover 92 bits of the key information and the attack procedure was successful on all the tested 32 randomly generated keys.

5.2.5 Towards a resynchronization attack

Here, some preliminary observations on the possibility of adapting the above attack to the single-key model are provided. In the single-key resynchronization attack, only the IV can have active bytes, which means that only the left-hand half of the LFSR, i.e., registers $s_{16}, \ldots, s_{31}$ as well as the BOMM will contain active bytes. As in the related-key attack above, the strategy is to have the difference cancelled out in the LFSR and localized only in the BOMM early during the initialization. One of the obstacles is that the R register will necessarily be activated when the difference reaches byte s_7, since the left-hand half of the LFSR contains active bytes. We note that this obstacle can be bypassed by cancelling the introduced R difference by having more than one LFSR byte active. Let LFSR bytes s_9, s_8 and s_7 be active with differences δ_2, δ_1, δ_0 at some time t during the initialization procedure. Also, assume that the word R and the BOMM bytes to be used in the next three steps are inactive. Below, we determine how many of the $(\delta_2, \delta_1, \delta_0)$ values can leave R inactive after 3 steps (after having passed through s_7) and also the probability of occurrence of such an event. For this purpose, note that the R cancellation event occurs if

$$\gamma(F(x^t) \oplus u^{t+1}) \oplus \gamma(F(x^t \oplus \delta_0) \oplus u^{t+1} \oplus \delta_1) = \theta^{-1}\delta_2 \tag{5.13}$$

where $x^t = R^t \oplus u^t$ and u^t denotes the 32-bit words fed to the F function from the LFSR in t-th step. By using a precomputed table for the S-box S_1 that, for each input and output difference, contains the information whether it is possible to achieve the input-output difference pair or not, we exhaustively checked for which values of $(\delta_2, \delta_1, \delta_0)$ equation (5.13) has solutions in x^t and u^{t+1}. The result of the finding is that only $2^{-12.861}$ of $(\delta_2, \delta_1, \delta_0)$ values cannot yield an R difference cancellation event. For the remaining $(\delta_2, \delta_1, \delta_0)$, for which (5.13) does have a solution, the probability of the R difference cancellation is $2^{-4} \times 2^{-28} = 2^{-32}$.

The analysis above indicates that attackers can choose almost any $(\delta_2, \delta_1, \delta_0)$ starting difference at three consecutive LFSR bytes and then bypass an R activation with a probability of 2^{-32}. A possible favor-

able position to introduce such $(\delta_2, \delta_1, \delta_0)$ difference can be in registers s_{18}, s_{17}, s_{16}, since the R register will only be activated through byte s_7. This can be done by activating IV_2, IV_1, IV_0 bytes. The 3-byte difference that arises in the BOMM then needs to be used for cancellations whenever some of the active LFSR bytes pass through the taps. Due to the relatively high number of cancellations that need to happen as the difference moves towards the right, we have not been able to bring the cancellation probability sufficiently high enough to have a practical attack. Controlling the difference propagation as done in [79] may be useful for that purpose. It is left for future research to verify whether a practical resynchronization single-key attack can be mounted against Loiss.

5.3 Sliding properties of Loiss

In [73], a slide attack on SNOW 3G and SNOW 2.0 was provided. This attack is a related-key attack and involves a key-IV pair (K, IV) and (K', IV'). The idea is to have the inner state of the (K, IV) instance after $n \geq 1$ steps be a *starting* inner state. Then, the corresponding (K', IV') initializes to this starting state and the equality of the inner states is preserved until the end of the procedure. The similarity between the two keystreams is detected and this provides a basis for the key-recovery attack. Since LFSR-based word-oriented stream ciphers usually do not use counters which are the usual countermeasure against this kind of slide attacks, one way to protect against sliding is to have the initial inner state populated by the key, IV and constants so that it disallows the next several states to be starting states. For example, in ZUC [50], constants are loaded in a way that makes it difficult to mount a slide attack.

In the following, we point out that Loiss, similar to SNOW 2.0 and SNOW 3G, does not properly defend against sliding. If $C_0 = S_1^{-1}(0)$ and $C_1 = S_2(0)$, a slide by one step can be achieved as follows.

Observation 2 *Let $K = (K_{15}, \ldots, K_0)$ and $IV = (A, \ldots, A, B)$, where*

$$A = (\alpha \oplus \alpha^{-1} \oplus 1)^{-1}(K_0 \oplus \alpha^{-1}K_0 \oplus \alpha^{-1}K_1 \oplus K_2 \oplus \alpha K_5 \oplus \alpha K_8 \oplus K_{11} \oplus K_{13} \oplus C_0)$$

and B is determined by $B \oplus C_1 \oplus S_2(B \oplus C_1) = A$. Also, assume that $K_7 = C_0$ and $K_4 = K_{10} = K_{15} = C_0 \oplus A$. Then, for $K' = (K_0 \oplus B, K_{15}, \ldots, K_1)$ and $IV' = (A, \ldots, A)$, we have

$$z'_0 = z_1 \tag{5.14}$$

Proof: We will show that

$$\begin{aligned} IS^1 &= (s^1_{31}, \ldots, s^1_0, R^1, y^1_{16}, \ldots, y^1_0) \\ &= (s'^0_{31}, \ldots, s'^0_0, R'^0, y'^0_{16}, \ldots, y'^0_0) = IS'^0 \end{aligned} \tag{5.15}$$

As for the BOMM bytes y_i, $15 \leq i \leq 0$, in the (K, IV) instance of the cipher, only y_0 will be updated since $R^0 = 0$. In other words, $y^1_i = A$ for $15 \leq i \leq 1$. Moreover, from the specification of B, it follows that $y^1_0 = A$. Since $IV' = (A, \ldots, A)$, $y'^0_i = A$ for $15 \leq i \leq 0$ as well, i.e., (5.15) holds for the BOMM bytes. As for the equality between R^1 and R'^0, by the initialization procedure, $R'^0 = 0$. To have $R^1 = 0$ as well, it suffices to have each of the four LFSR registers $s^0_{31}, s^0_{26}, s^0_{20}, s^0_7$ equal to $C_0 = S^{-1}(0)$, which is exactly the case due to the values to which bytes K_{15}, K_8, K_4 and K_7 are set. Finally, to establish the equality of the LFSR values in (5.15), the expression defining A are substituted into the way the LFSR is updated during the initialization procedure with the feed-forward, verifying that $s^1_{31} = s'^0_{31} = K_{15} \oplus A$. As for the other LFSR values, $s^1_i = s'^0_i$ holds directly due to the specification of K, IV, K', IV'.

Thus, the initialization procedures of the two cipher instances are slided, i.e., $IS^t = IS'^{t-1}$ for $1 \leq t \leq 64$. At time $t = 64$, in the (K, IV) instance of the cipher, a regular keystream step is applied, whereas in the (K', IV') instance, an initialization step is applied which destroys the slide property by introducing a difference between s^{65}_{31} and s'^{64}_{31}. However, it can be verified that this difference does not affect the two corresponding first keystream words, which proves (5.14). □

It should be noted that, as we verified by solving $B \oplus C_1 \oplus S_2(B \oplus C_1) = A$ for each $A \in F_2^8$, there always exists a byte B specified by this observation.

Due to the requirement on bytes K_7, K_4, K_{10} and K_{15} from the formulation of the observation above, a Loiss key K has a related key pair specified by the observation above with probability 2^{-32}. For the related keys K and K' satisfying the conditions above, the attack can be performed by going through all $A \in F_2^8$ and verifying whether the relation (5.14) is satisfied for $IV = (A, \ldots, A, B)$, and $IV' = (A, \ldots, A)$. If yes, then such an A byte is a candidate for the right-hand side of the equation above specifying A, which depends only on K bytes. Each false candidate out of 2^8 candidates for A will pass the test (5.14) with probability 2^{-8}. That way, around one byte of the key information leaks. Slides by more than one step may also be possible.

5.4 Conclusion

We presented a practical-complexity related-key attack on the Loiss stream cipher. The fact that a slowly changing array (the BOMM) has been added as a part of the FSM in Loiss allowed the difference to be contained (i.e., do not propagate) during a large number of inner state update steps with a relatively high probability. The attack was implemented and our implementation takes less than one hour on a PC with 3GHz Intel Pentium 4 processor to recover 92 bits of the 128-bit key. The possibility of extending the attack to a resynchronization attack in a single-key model was discussed. We also showed that a slide attack is possible for the Loiss stream cipher.

6

On the sliding property of SNOW 3G and SNOW 2.0

In response to concerns about the security of the 3GPP encryption primitive KASUMI [3], [16] (see also [47]), the Security Algorithms Group of Experts (SAGE) proposed a possible replacement for KASUMI which is currently used in 3G systems as a component of the UEA1 confidentiality algorithm. The core primitive of the new confidentiality algorithm, UEA2, is the SNOW 3G stream cipher [51]. The design of SNOW 3G is based on SNOW 2.0 [49], a stream cipher which is chosen for the ISO/IEC standard IS 18033-4 along with Decim [11], MUGI [132] and Rabbit [28]. SNOW 3G passed extensive internal cryptanalytic efforts, surveyed in [52], but the full evaluation has not been released to public. Externally, SNOW 3G was analyzed in [25].

In this chapter, we show that the initialization procedure of the two ciphers admits a sliding property, resulting in several sets of related-key pairs. In case of SNOW 3G, a set of 2^{32} related key pairs is presented, whereas in case of SNOW 2.0, several such sets are found, out of which the largest are of size 2^{64} and 2^{192} for the 128-bit and 256-bit variant of the cipher, respectively. In addition to allowing related-key key recovery attacks against SNOW 2.0 with 256-bit keys, the presented properties reveal non-random behavior which yields related-key distinguishers and also questions the validity of the security proofs of protocols that are based on the assumption that SNOW 3G and SNOW 2.0 behave like perfect random functions of the key-IV.

Biham *et al.* [16] showed that KASUMI does not behave randomly when examined in the related-key model. As stated in [16], this renders the previous security proofs based on the assumption that KASUMI

behaves like a perfect random function [63] as invalid and puts into question the security of the whole 3GPP system. In [34], [118] the sliding properties of stream ciphers were used to find sets of related keys where it was shown that a stream cipher may be slidable, in the sense that there exist key-IV values such that the inner state of the cipher at some time $t > 0$ corresponds to another key-IV value. Such key-IV pairs produce equal keystreams up to a slide by some number of positions and represent related keys.

We show that a similar strategy is also applicable to SNOW 3G and SNOW 2.0 due to the way the key and the IV are written to the inner state before the first initialization step. More precisely, in this chapter, it is shown that it is possible to find key-IV pairs such that after iterating the cipher for several initialization steps, the inner state represents a *starting* inner state for some other key-IV value. Due to the nature of the of the initialization processes of SNOW 3G and SNOW 2.0, such related keys do not generate slid keystreams, but only keystreams that have several equal words. However, this still allows distinguishing the produced keystream from random keystreams. The found sets of keys that have (not necessarily unique) related keys for different SNOW variants are summarized in Table 6.1.

A feature of the related keys presented in this chapter is that, given a key for which a corresponding key pair exists, it is straightforward to derive this related key, as opposed to related keys from [118] where the relation was non-obvious and the keys corresponded to a solution of a complex system of equations. XYZXYZ(related key relation) The simplicity of the related keys makes the potential related-key attack more realistic, since if the encryption scheme is used in protocols with related keys, it is unlikely that the relation will be complex and represent solutions to complex systems of equations. We also show that in the case of SNOW 2.0 with 256-bit key, the presented properties allow related-key attacks with complexity smaller than the exhaustive search. Finally, by using a property of the related keys by which given a (K, IV) value, the related key K' depends on the value of IV, we present a simple time-memory trade-off for the case where the attackers position is weakened with respect to the assumptions on the two related keys.

The rest of the chapter is organized as follows. In Section 6.1, we briefly review the specifications of SNOW 3G and SNOW 2.0. The sets of related-keys are specified in Sections 6.2 and 6.3. The attacks against SNOW 2.0 with 256-bit key are examined in Section 6.4 and the conclusion is given in Section 6.5.

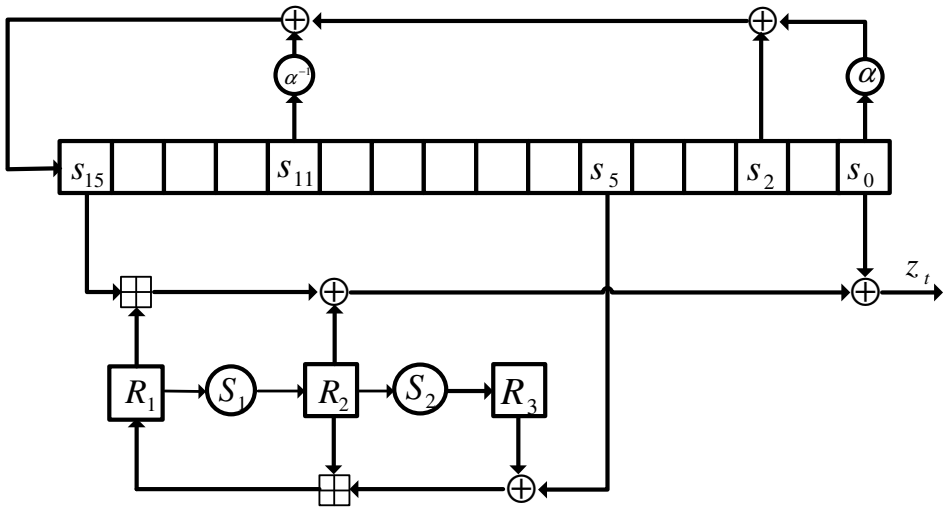

Figure 6.1: The SNOW 3G stream cipher

Snow Variant	Source	# of related key pairs	# of slide steps	Key recovery attack
SNOW 3G	Th. 1	2^{32}	3	-
SNOW 2.0 (128-bit key)	Th. 2	2^{32}	2	-
SNOW 2.0 (128-bit key)	Th. 3	2^{64}	3	-
SNOW 2.0 (256-bit key)	Th. 4	2^{160}	2	$\surd$
SNOW 2.0 (256-bit key)	Th. 5	2^{192}	3	$\surd$
SNOW 2.0 (256-bit key)	Th. 6	2^{192}	4	$\surd$

Table 6.1: Summary of results

6.1 Specifications of SNOW 3G and SNOW 2.0

Both SNOW 3G and SNOW 2.0 contain two main components: a Linear Feedback Shift Register (LFSR) and a Finite State Machine (FSM). The inner state of SNOW 3G (see Fig. 6.1) can be represented by $(s_0^t, \ldots s_{15}^t, R_1^t, R_2^t, R_3^t)$, where the s values represent 32-bit LFSR registers, the R values represent the 32-bit FSM registers and t denotes the number of iterations that have been executed so far. In SNOW 2.0, the FSM contains only two 32-bit registers and the inner state can be represented by $(s_0^t, \ldots s_{15}^t, R_1^t, R_2^t)$.

Unlike SNOW 3G which supports only 128-bit keys, SNOW 2.0 can be used with 128-bit and 256-bit keys. The size of the IV in both ciphers is 128 bits. In what follows, we briefly review the FSM and the LFSR update steps for the two ciphers.

SNOW 3G: The FSM update step is given by

$$\begin{aligned} R_3^{t+1} &= S_2(R_2^t), \;\; R_2^{t+1} = S_1(R_1^t) \\ R_1^{t+1} &= R_2^t \boxplus (R_3^t \oplus s_5^t) \end{aligned} \tag{6.1}$$

where S_1 and S_2 are two different 32×32 S-boxes, made of four parallel 8-bit S-boxes followed by a multiplication by a 4×4 matrix over GF(2^8) and $\boxplus$ denotes addition modulo 2^{32}.

The LFSR update is given by

$$s_{15}^{t+1} = \begin{cases} \alpha^{-1} \cdot s_{11}^t \oplus s_2^t \oplus \alpha \cdot s_0^t \oplus F^t, & t < 32 \\ \alpha^{-1} \cdot s_{11}^t \oplus s_2^t \oplus \alpha \cdot s_0^t & t \geq 32 \end{cases} \tag{6.2}$$

where α is a root of the $GF(2^8)[x]$ polynomial $x^4 + \beta^{23}x^3 + \beta^{245}x^2 + \beta^{48}x + \beta^{239}$, β is a root of the $GF(2)[x]$ polynomial $x^8 + x^7 + x^5 + x^3 + 1$, α^{-1} is the multiplicative inverse of α and F_t is the FSM output which is given by

$$F^t = (s_{15}^t \boxplus R_1^t) \oplus R_2^t.$$

Let $\mathbf{1}$ denote the all-one 32-bit word. The cipher operates as follows: the secret inner state is populated by $K = (K_0, \ldots K_4)$ and $IV = (IV_0, \ldots IV_4)$ according to

$$\begin{aligned} & s_{15}^0 = K_3 \oplus IV_0, \;\; s_{14}^0 = K_2, \;\; s_{13}^0 = K_1, \;\; s_{12}^0 = K_0 \oplus IV_1 \\ & s_{11}^0 = K_3 \oplus \mathbf{1}, \;\; s_{10}^0 = K_2 \oplus \mathbf{1} \oplus IV_2, \\ & s_9^0 = K_1 \oplus \mathbf{1} \oplus IV_3, \;\; s_8^0 = K_0 \oplus \mathbf{1} \\ & s_7^0 = K_3, \;\; s_6^0 = K_2, \;\; s_5^0 = K_1, \;\; s_4^0 = K_0 \\ & s_3^0 = K_3 \oplus \mathbf{1}, \;\; s_2^0 = K_2 \oplus \mathbf{1}, \;\; s_1^0 = K_1 \oplus \mathbf{1}, \;\; s_0^0 = K_0 \oplus \mathbf{1} \end{aligned} \tag{6.3}$$

and the FSM registers are reset to zero, i.e., $R_1^0 = R_2^0 = R_3^0 = 0$. The cipher is then iterated by executing

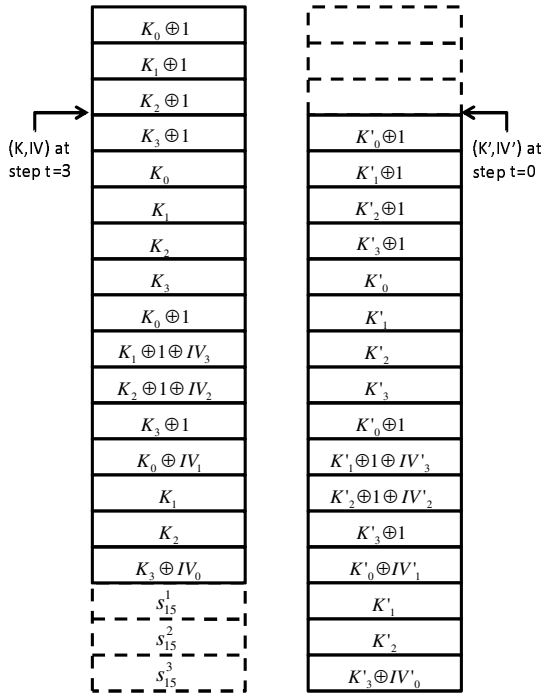

Figure 6.2: (K, IV) and (K', IV') LFSR at times 3 and 0, respectively. For example, row 4 contains $K_3 \oplus \mathbf{1} = s_0^3$ and $K'_0 \oplus \mathbf{1} = s_0'^0$

(6.1) and (6.2) for 33 times without generating any output. Note that for $t < 32$, according to (6.2), the FSM output F_t participates in the LFSR update, contrary to step $t = 32$. Finally, the keystream words $(z^0, z^1, \ldots)$ are produced by

$$z^{t-33} = s_0^t \oplus F^t, t \geq 33. \tag{6.4}$$

In each such step, after generating the keystream word, the FSM and subsequently the LFSR are updated by (6.1) and (6.2).

SNOW 2.0: The FSM update function is defined by

$$R_1^{t+1} = s_5 \boxplus R_2^t, \quad R_2^{t+1} = S(R_1^t) \tag{6.5}$$

where S is a permutation of $\mathbb{Z}_{2^{32}}$ based on the round function of Rijndael [44]. The LFSR update function, and the FSM output F_t are defined in the same way as for SNOW 3G.

For the 128-bit version of SNOW 2.0 with $K = (K_3, K_2, K_1, K_0)$ and $IV = (IV_3, IV_2, IV_1, IV_0)$, the starting inner state is populated according to (6.3). For SNOW 2.0 with 256-bit key, $K = (K_7, \ldots K_0)$, the LFSR is populated by

$$\begin{aligned} &s_{15} = K_7 \oplus IV_0,\ s_{14} = K_6,\ s_{13} = K_5,\ s_{12} = K_4 \oplus IV_1 \\ &s_{11} = K_3,\ s_{10} = K_2 \oplus IV_2,\ s_9 = K_1 \oplus IV_3,\ s_8 = K_0 \\ &s_7 = K_7 \oplus \mathbf{1},\ s_6 = K_6 \oplus \mathbf{1}, \qquad \ldots, \qquad s_0 = K_0 \oplus \mathbf{1} \end{aligned} \tag{6.6}$$

The initialization process and the keystream generation are done the same way as in SNOW 3G.

The following notation will be used throughout the rest of the chapter. For both SNOW 3G and SNOW 2.0, two instances of the cipher will be considered: one is initialized by (K, IV) and the other one is initialized by (K', IV'). Adding "′" as a suffix to the word will distinguish whether it relates to the (K, IV) or the (K', IV') instance of the cipher. For example, z'_i, s'^t_j, R'^t_k denote the keystream and the inner state of the (K', IV') instance of the cipher. Let IS_t denote the complete inner state of the (K, IV) instance of cipher at time $t \geq 0$. For example IS'_0 represents the inner state of the cipher initialized by (K', IV'), after applying equations (6.3) and before executing any initialization steps.

The inner state at $t = 0$, i.e., the state right after applying (6.3) or (6.6) will be referred to as the *starting* inner state. The iteration in which the cipher goes from time t to time $t + 1$ is denoted by *step* t. Step t will be referred to as an *initialization step* if $0 \leq t \leq 31$. If $t \geq 32$, the step will be called a *keystream generation* step. The operators $\boxplus$ and $\boxminus$ denote addition and subtraction modulo 2^{32}, respectively.

6.2 Related-key pairs for SNOW 3G

In this section, we show that it is possible to initialize SNOW 3G by (K, IV) so that its inner state at time $t = 3$ represents a valid *starting* inner state corresponding to another (K', IV'). More precisely, we show that there exists a set of 2^{32} (K, IV) values such that for each such value, a unique (K', IV') exists so that $IS_3 = IS'_0$.

The initial equality $IS_3 = IS'_0$ is preserved until step 32, i.e., $IS_t = IS'_{t-3}$, $3 \leq t \leq 32$. At that point, a difference occurs due to the fact that in the (K, IV) instance an initialization step is applied to update the inner state whereas in the (K', IV') instance, a keystream generation step is applied according to

(6.2). Nevertheless, due to the high degree of similarity among the corresponding inner states at the point where the keystream words are produced, several such words will be equal, contrary to how a perfect stream cipher should behave.

Define $C_1 = S_1^{-1}(S_2^{-1}(0))$, $C_2 = (S_1^{-1}(0) \boxminus S_1(0)) \oplus S_2(0)$ and $C_3 = (\boxminus S_1(S_1^{-1}(S_2^{-1}(0)))) \oplus S_2(S_1(0))$ and let a_0, b_0, b_1, b_0' be 32-bit words. The following theorem specifies a set of 2^{32} related keys for SNOW 3G.

Theorem 1 *Let* $K = (a_0, C_1, C_2, C_3)$ *and* $K' = (C_3, a_0 \oplus \mathbf{1}, C_1 \oplus \mathbf{1}, C_2 \oplus \mathbf{1})$. *Then, there exist unique* $IV = (b_0, b_1, 0, 0)$ *and* $IV' = (b_0', b_0, 0, b_1)$ *such that* $IS_t = IS'_{t-3}$, $3 \leq t \leq 32$ *and*

$$z_3 = z_0', z_4 = z_1', z_8 = z_5', z_9 = z_6'. \tag{6.7}$$

Proof: First, we show that there exist unique IV and IV' of the form above so that K and K' satisfy $IS_3 = IS_0'$, i.e.,

$$(s_0^3, \ldots s_{15}^3, R_1^3, R_2^3, R_3^3) = (s_0'^0, \ldots s_{15}'^0, R_1'^0, R_2'^0, R_3'^0) \tag{6.8}$$

Unfolding the FSM registers at $t = 3$ yields

$$R_1^3 = s_7^0 \oplus S_2(S_1(0)) \boxplus S_1(s_5^0),$$
$$R_2^3 = S_1(s_6^0 \oplus S_2(0) \boxplus S_1(0)) \ , \ R_3^3 = S_2(S_1(s_5^0)).$$

Substituting the values s_5^0, s_6^0 and s_7^0 according to $s_5^0 = K_1 = C_1$, $s_6^0 = K_2 = C_2$ and $s_7^0 = K_3 = C_3$ (which follows by (6.2) and by the theorem formulation) shows that $R_1^3 = 0$, $R_2^3 = 0$ and $R_3^3 = 0$. Since $R_1'^0 = 0$, $R_2'^0 = 0$ and $R_3'^0 = 0$ by the SNOW 3G specification, the equality of the FSM words is established.

As for the LFSR values of equality (6.8), the problem is depicted in Fig. 6.2. It suffices to equate the expressions shown inside the rows using the keys specified by the theorem, skipping the first 3 rows. For example, row 4 corresponds to $s_0^3 = s_0'^0$. This is trivially satisfied by the K and K' specified by the theorem by setting $K_3 \oplus \mathbf{1} = C_3 \oplus \mathbf{1} = K_0' \oplus \mathbf{1}$, without imposing any constraint on IV, IV'. It is straightforward to verify that the same holds for rows 5, 6, 7, 8, 9, 12, 15. However, equating rows 10, 11, 13, 14 and 16 yields

$$IV_3 = IV_2 = 0, \ IV_1 = IV_3', \ IV_2' = 0, \ IV_0 = IV_1' \tag{6.9}$$

Finally, equating rows 17, 18, 19 and substituting values for s_{15}^1, s_{15}^2, s_{15}^3 we have

$$\alpha(K_0 \oplus \mathbf{1}) \oplus K_2 \oplus \mathbf{1} \oplus \alpha^{-1}(K_3 \oplus \mathbf{1}) \oplus K_3 \oplus IV_0 = K_0 \oplus \mathbf{1} \tag{6.10}$$

$$\begin{aligned} &\alpha(K_1 \oplus \mathbf{1}) \oplus K_3 \oplus \mathbf{1} \oplus \alpha^{-1}(K_0 \oplus IV_1) \oplus \\ &((K_0 \oplus \mathbf{1}) \boxplus K_1) \oplus S_1(0) = K_1 \oplus \mathbf{1} \end{aligned} \tag{6.11}$$

$$\begin{aligned} &\alpha(K_2 \oplus \mathbf{1}) \oplus K_0 \oplus \alpha^{-1}(K_1) \oplus \\ &((K_1 \oplus \mathbf{1}) \boxplus (K_2 \oplus S_2(0) \boxplus S_1(0)) \oplus \\ &S_1(K_1) = K_2 \oplus \mathbf{1} \oplus IV_0'. \end{aligned} \tag{6.12}$$

It is clear that equations (6.10)-(6.12) can be solved explicitly in IV_0, IV_1 and IV_0'. In other words, by letting $K = (a_0, C_1, C_2, C_3)$ and $K' = (C_3, a_0 \oplus \mathbf{1}, C_1 \oplus \mathbf{1}, C_2 \oplus \mathbf{1})$ as specified by the theorem and fixing a_0, these three equations yield a unique IV_0, IV_1 and IV_0', which take the place of b_0, b_1 and b_0', respectively, showing that for K, K', there exist unique IV, IV' of the form specified by the theorem, satisfying (6.8).

To complete the proof it suffices to show that (6.8) implies (6.7). From (6.8), using (6.2), it follows that $IS_t = IS_{t-3}'$ for $3 < t \leq 32$. Again, according to (6.2), it follows that the difference in times $t = 33, 34, 35$ is present in registers $\{s_{15}\}$, $\{s_{15}, s_{14}\}$, $\{s_{15}, s_{14}, s_{13}\}$, respectively. As for times $t = 36, 37$, the difference in the inner states stays only in $\{s_{14}, s_{13}, s_{12}\}$, $\{s_{13}, s_{12}, s_{11}\}$, respectively. Then, using (6.4), it follows that $z_3 = z_0'$, $z_4 = z_1'$. By following the difference propagation, it is straightforward to see that at $t = 41, 42$ the active registers are $\{s_{14}, s_{13}, s_{12}, s_9, s_8, s_7\}$ and $\{s_{13}, s_{12}, s_{11}, s_8, s_7, s_6\}$, respectively, which, using (6.4), completes the proof of (6.7). □

In the previous Theorem, related keys due to the slide of 3 steps are described. An attempt to change the number of sliding steps is unlikely to yield new interesting sets of related keys for SNOW 3G. Namely, slide pairs on the distance of 2 steps do not exist due to the fact that $R_3^2 = S_2(S_1(0))$ is a constant different than zero, which means that the inner state after 2 initialization steps cannot represent a starting inner state of another slid instance of the cipher. As for the slide by 4 steps, the FSM constraint restricts the key K to 2^{32} possible values. Then, the K candidates are restricted by an additional 64-bit filter due to the LFSR constraint, i.e., by equations $s_{13}^4 = s_{13}'^0$ and $s_{14}^4 = s_{14}'^0$. The two constraints together render the related-keys highly unlikely to exist. Finally, an eventual slide by more than 4 steps does not produce related keys since

the difference between the initialization and the keystream generation steps for longer than 4 steps destroys the equivalence between the inner states which is needed to have some equal words in the corresponding output sequences.

6.3 Related-key pairs for SNOW 2.0

In this section, we show that the strategy from Section 6.2 is also applicable against SNOW 2.0. In particular, for SNOW 2.0 with 128-bit keys, we show that two different related key sets exist due to the slide by 2 and by 3 steps. As for the 256-bit key version of SNOW 2.0, each of the slides by 2, 3 and 4 steps yield related key sets.

6.3.1 SNOW 2.0 with 128-bit keys

The following theorem reveals a set of 2^{32} related key pairs for the 128-bit version of SNOW 2.0, due to the slide by 2 steps. Let $C_1 = S^{-1}(0)$ and $C_2 = \boxminus S(0)$ and let a_0, a_3, b_1 and b'_0 be 32-bit words. Note that, according to the SNOW 2.0 specification, K and IV are are indexed in reverse order.

Theorem 2 *Let a_0 and a_3 satisfy*

$$\alpha(a_0 \oplus \boldsymbol{1}) \oplus C_2 \oplus \alpha^{-1}(a_3 \oplus \boldsymbol{1}) \oplus a_3 = a_0 \tag{6.13}$$

Let $K = (a_3, C_2, C_1, a_0)$ and $K' = (C_1 \oplus \boldsymbol{1}, a_0 \oplus \boldsymbol{1}, a_3, C_2)$. Then, for any $IV = (0, 0, b_1, 0)$, there exists a unique $IV' = (0, b_1, 0, b'_0)$ so that for SNOW 2.0 with 128-bit key, we have $IS_t = IS'_{t-2}$ for $2 \leq t \leq 32$ and

$$\begin{aligned} z_2 = z'_0, z_3 = z'_1, z_4 = z'_2 \\ z_7 = z'_5, z_8 = z'_6, z_9 = z'_7 \end{aligned} \tag{6.14}$$

Proof: Similar to the proof of Theorem 1, it will be shown that the K, K', IV and IV' values obeying the conditions of the theorem imply $IS_2 = IS'_0$. Since $R_1^2 = s_6 \boxplus S(0) = K_2 \boxplus S(0) = \boxminus S(0) \boxplus S(0) = 0 = R_1'^0$ and $R_2^2 = S(s_5) = S(K_1) = S(S^{-1}(0)) = 0 = R_2'^0$, the equality of the FSM registers is established. The

LFSR constraint amounts to showing that

$$s_i^2 = s_i^{'0}, \;\; 0 \leq i \leq 15 \tag{6.15}$$

By substituting $s_i^2 = s_{i+2}^0$ for $i \leq 13$ and substituting s_{i+2}^0 and $s_i^{'0}$ according to (6.3), it easy to verify that for $i \in \{0, 1, 2, 3, 4, 5, 6, 11\}$, (6.15) is satisfied without imposing any constraints on IV and IV'. On the other hand, using the same substitutions, due to (6.15) for $i \in \{7, 8, 9, 10, 12, 13\}$, it follows that

$$\begin{aligned} &IV_3 = 0, \; IV_2 = 0, \; IV_3' = 0, \\ &IV_1 = IV_2', \; IV_1' = 0, \; IV_0 = 0 \end{aligned} \tag{6.16}$$

which leaves both $IV_1 = IV_2'$ and IV_0' unspecified. As for (6.15) for $i = 14$, it is satisfied due to (6.13). From (6.15), for $i = 15$, it follows that for any $IV_1 = b_1$, $IV_0' = b_0'$ is uniquely determined.

From $IS_2 = IS_0'$, by (6.2), it follows that $IS_t = IS_{t-2}'$ for $2 < t \leq 32$. In times $t = 33, 34$, the difference is present in $\{s_{15}\}$, $\{s_{15}, s_{14}\}$ registers, respectively. In times $t = 35, 36, 37$ the difference in the inner states is present only in $\{s_{14}, s_{13}\}$, $\{s_{13}, s_{12}\}$, $\{s_{12}, s_{11}\}$, respectively. Following the propagation further reveals that in times $t = 40, 41, 42$, the difference is only in $\{s_{14}, s_{13}, s_9, s_8\}$, $\{s_{13}, s_{12}, s_8, s_7\}$ and $\{s_{12}, s_{11}, s_7, s_6\}$, respectively. Taking into account (6.4), (6.14) follows. □

The number of K values for which related key-IVs exist is equal to the number of possible a_0, a_3 that satisfy the linear equation (6.13), i.e. 2^{32} values.

The next theorem reveals a larger set of 2^{64} related key pairs for 128-bit keyed SNOW 2.0, due to the slide by 3 steps. Let a_0, a_1 be arbitrary 32-bit words and let $A_3 = \boxminus S(a_1)$. Define the constant $C_1 = S^{-1}(0) \boxminus S(0)$.

Theorem 3 *Let $K = (A_3, C_1, a_1, a_0)$ and $K' = (C_1 \oplus \mathbf{1}, a_1 \oplus \mathbf{1}, a_0 \oplus \mathbf{1}, A_3)$. Then, there exist unique $IV = (0, 0, b_1, b_0)$ and $IV' = (b_1, 0, b_0, b_0')$, for SNOW 2.0 with 128-bit key, such that $IS_t = IS_{t-3}'$ for $3 \leq t \leq 32$ and that*

$$z_3 = z_0', z_4 = z_1', z_8 = z_5', z_9 = z_6'$$

As for the sliding by 4 steps, the FSM constraint imposes a 64-bit constraint on the key K and the equations $s_{13}^4 = s_{13}^{'0}$ and $s_{14}^4 = s_{14}^{'0}$ provide another 64-bit constraint. Since the expected number of such related key pairs is 1, they are less relevant and their treatment is omitted. As for sliding by more than 4 steps, the

difference between the initialization and the keystream generation steps for longer than 4 steps destroys the equivalence between the inner states which consequently prevents having equal words in the corresponding output sequences.

6.3.2 SNOW 2.0 with 256-bit keys

The theorem that follows uses sliding by 2 steps to describe a set of 2^{160} related key pairs for SNOW 2.0 with 256-bit key. Define the constants $C_1 = S^{-1}(0) \oplus \mathbf{1}$, $C_2 = (\boxminus S(0)) \oplus \mathbf{1}$ and let a_0, a_1, a_2, a_3, a_4, a_7, b_1, b_3 and b_0' be 32-bit words.

Theorem 4 *Assume that*

$$\alpha(a_0 \oplus \boldsymbol{1}) \oplus a_2 \oplus \alpha^{-1}(a_3) \oplus a_7 = a_0 \tag{6.17}$$

Let $K = (a_7, C_2, C_1, a_4, a_3, a_2, a_1, a_0)$ *and* $K' = (a_1 \oplus b_3 \oplus \boldsymbol{1}, a_0 \oplus \boldsymbol{1}, a_7, C_2, C_1, a_4, a_3, a_2)$. *If* $IV = (b_3, 0, b_1, 0)$, *there exists a unique* $IV' = (0, b_1, 0, b_0')$ *such that for SNOW 2.0 with a 256-bit key, we have* $IS_t = IS'_{t-2}$ *for* $2 \leq t \leq 32$ *and*

$$\begin{aligned} z_2 = z'_0, z_3 = z'_1, z_4 = z'_2 \\ z_7 = z'_5, z_8 = z'_6, z_9 = z'_7 \end{aligned} \tag{6.18}$$

Due to (6.17), the number of K values for which related key-IVs exist is 2^{160}.

Next, a set of 2^{192} related key pairs due to the slide by 3 steps is described. Let a_0, a_1, a_2, a_3, a_4, a_5, b_2 and b_3 be arbitrary 32-bit words. Let $A_7 = (\boxminus S(a_5 \oplus \mathbf{1})) \oplus \mathbf{1}$ and define the constant $C_1 = (S^{-1}(0) \boxminus S(0)) \oplus \mathbf{1}$.

Theorem 5 *Let* $K = (A_7, C_1, a_5, a_4, a_3, a_2, a_1, a_0)$ *and* $K' = (a_2 \oplus b_2 \oplus \boldsymbol{1}, a_1 \oplus b_3 \oplus \boldsymbol{1}, a_0 \oplus \boldsymbol{1}, A_7, C_1, a_5, a_4, a_3)$. *Then, there exist unique* b_1, b_0 *and* b_0' *such that with* $IV = (b_3, b_2, b_1, b_0)$ *and* $IV' = (b_1, 0, b_0, b_0')$, *for SNOW 2.0 with a 256-key, we have* $IS_t = IS'_{t-3}$ *for* $3 \leq t \leq 32$ *and*

$$z_3 = z'_0, z_4 = z'_1, z_8 = z'_5, z_9 = z'_6 \tag{6.19}$$

Finally, another set of 2^{192} related key pairs is described by using a slide of 4 steps for SNOW 2.0 with 256-bit keys. Let a_1, a_2, a_3, a_4, a_5 and a_6 be arbitrary 32-bit values. Define $A_7 = (S^{-1}(0) \boxminus S(a_5 \oplus \mathbf{1})) \oplus \mathbf{1}$

and $A_0 = \boxminus S((a_6 \oplus \mathbf{1}) \boxplus S(0))$.

Theorem 6 *Let $K = (A_7, a_6, a_5, a_4, a_3, a_2, a_1, A_0)$. Then, there exist unique b_3, b_2, b'_1 and b'_0 such that for $K' = (a_3 \oplus \mathbf{1}, a_2 \oplus b_2 \oplus \mathbf{1}, a_1 \oplus b_3 \oplus \mathbf{1}, A_0 \oplus \mathbf{1}, A_7, a_6, a_5, a_4)$, $IV = (b_3, b_2, 0, 0)$ and $IV' = (0, 0, b'_1, b'_0)$ for SNOW 2.0 with 256-bit key, we have $IS_t = IS'_{t-4}$ for $4 \leq t \leq 32$ and*

$$z_4 = z'_0, z_9 = z'_5$$

6.4 Related-key attacks

Theorems 4, 5 and 6 allow generic attacks against SNOW 2.0 with 256-bit key in which the attacker queries the two instances of the cipher initialized by K and its related K' to find the IV and IV' that give slide pairs. Then, the found IV and IV' are plugged in the equations that are necessary to be satisfied if the slide is to happen. Since the equations are only in secret key bits and are easy to solve given that they establish equivalence between the LFSR register values on the distance of only small number of steps less than the number of slid steps, the key space is restricted.

Moreover, in case of SNOW 2.0 with 256-bit key, variations of the generic attack stated above, by which related-key setting is relaxed up to some point, are possible. Namely, it can be observed that in Theorems 4, 5 and 6, K' depends on the IV, the initialization vector of key K. It follows that, by varying IV in (K, IV), the key K is related to different related keys K', which in turn indicates that, given a cipher instance initialized by K, it is not necessary for the attacker to have access to a single K' cipher instance, but rather to a set of possible K' values. Furthermore, as shown below, in the scenario less favorable for the attacker in which the difference between parts of K and K' is unknown, it is possible to reduce some of the additional attack complexity at the expense of additional memory.

Let $K[i]$ and $K'[j]$ be the two corresponding key subwords in the keys specified by one of the Theorems 4, 5 or 6. In case $K'[j]$ does not depend on the IV, for the attack to work, the difference $K[i] \oplus K'[j]$ has to be equal to the constant specified by the Theorem. For example, in case of Theorem 6, the difference $K[0] \oplus K'[4]$ has to be equal to $\mathbf{1}$. Such a scenario between the two keys is common for the classical related key model. On the other hand, if $K'[j]$ depends on the IV, we distinguish the following two possible scenarios:

(a) $K[i] \oplus K'[j]$ is an arbitrary value known to the attacker

(b) $K[i] \oplus K'[j]$ is an arbitrary value unknown to the attacker

Clearly, scenario (b) is less favorable for the attacker than the scenario (a). In what follow, we examine possible attacks when scenarios (a) and (b) are assumed for the key subwords in question. It should be noted that the number of unknown key bits in the two related keys can be taken to be the smaller of the numbers of unknown bits in the two keys. Since, in what follows, every two related keys have the same number of unknown bits, the number of bits in the key is equal to the number of unknown bits in one (any) of the two keys.

Let the attacker have access to two instances of the cipher, as specified by Theorem 4 and let $K[1]$ and $K'[7]$ be related by scenario (a), i.e. let b_3 be known to the attacker. To find the IV and IV' such that (K, IV) and (K', IV') yield a slide pair, the attacker queries the K instance with $IV = (b_3, 0, 0, 0)$ once and the K' instance around 2^{32} times by varying b'_0 in $IV' = (0, 0, 0, b'_0)$, i.e., until (6.18) is satisfied. Then, due to (6.15) for $i = 15$, after simplifying the equation and substituting $s^1_{15} = a_0 \oplus \mathbf{1}$, we have

$$\alpha(a_1) \oplus a_1 \oplus a_3 \oplus \alpha^{-1}(a_4) \oplus ((a_0 \oplus \mathbf{1}) \boxplus C_1) \oplus S(0) \oplus \alpha(\mathbf{1}) = \alpha^{-1}(b_1) \oplus b_3 \oplus b'_0 \tag{6.20}$$

Since $b_1 = 0$, b_3 and b'_0 are known, the equation above introduces a 32-bit constraint on key bits, reducing the unknown key bits number from 160 to 128. In case we assume the scenario (b) between $K[1]$ and $K'[7]$, the following process can be applied:

- For each b_3, query the K instance of the cipher using $IV = (b_3, 0, 0, 0)$. Save each $(z_2, z_3, z_4, z_7, z_8, z_9)$ as a row of table T.

- Sort table T

- For each b'_0, query the K' instance of the cipher using $IV' = (0, 0, 0, b'_0)$ and search for $(z'_0, z'_1, z'_2, z'_5, z'_6, z'_7)$ value in table T. If found, return the corresponding (b_3, b'_0)

The advantage of the latter attack is that it does not assume any relation between $K[1]$ and $K'[7]$. It requires 2^{32} chosen-IV queries to each of the two oracles, storage of size 2^{32} and the computational effort dominated by a key search over the space of 2^{128} keys.

As for the attack based on Theorem 5, assume a relation of type (a) between $K[1]$ and $K'[6]$ and also between $K[2]$ and $K'[7]$, where K and K' are the keys of the two instances of the cipher available to the attacker. Since values b_3 and b_2 are known, trying all possible guesses for b_1, b_0 and b'_0 yields the IV and IV' that correspond to the slid inner states. The cost of such a procedure is 2^{64} queries to the K instance of the cipher and 2^{96} queries to the K' oracle. Out of 2^{96} (b_1, b_0, b'_0) values, only the triplet that produces slide inner states is expected to pass, since (6.19) represents a 128-bit constraint. Once the b_1, b_0 and b'_0 have been found, equations $s^3_{13} = s'^0_{13}$, $s^3_{14} = s'^0_{14}$ and $s^3_{15} = s'^0_{15}$ that hold for slide pairs can be expanded. After simplifying the equations and substituting $s^1_{15} = a_0 \oplus \mathbf{1}$ and $s^2_{15} = a_1 \oplus b_3 \oplus \mathbf{1}$, we have

$$\alpha(a_0) \oplus a_2 \oplus \alpha^{-1}(a_3) \oplus (\boxminus S(a_5 \oplus \mathbf{1})) \oplus a_0 \oplus \alpha(\mathbf{1}) \oplus \mathbf{1} = b_0 \tag{6.21}$$

$$\alpha(a_1) \oplus a_3 \oplus \alpha^{-1}(a_4) \oplus ((a_0 \oplus \mathbf{1}) \boxplus (a_5 \oplus \mathbf{1})) \oplus a_1 \oplus \alpha(\mathbf{1}) \oplus S(0) = \alpha^{-1}(b_1) \oplus b_3 \tag{6.22}$$

$$\begin{aligned} \alpha(a_2) \oplus a_4 \oplus \alpha^{-1}(a_5) \oplus ((a_1 \oplus b_3 \oplus \mathbf{1}) \boxplus (C_1 \oplus \mathbf{1}) \boxplus S(0)) \oplus \\ \oplus S(a_5 \oplus \mathbf{1}) \oplus a_2 \oplus \alpha(\mathbf{1}) = b'_0 \oplus b_2 \end{aligned} \tag{6.23}$$

By guessing a_0, a_1 and a_5 the system is linearized in $GF(2^{32})$ and can be rewritten as

$$a_2 \oplus \alpha^{-1}(a_3) = L_1, \;\; a_3 \oplus \alpha^{-1}(a_4) = L_2, \;\; (\alpha \oplus 1)(a_2) \oplus a_4 = L_3$$

where L_1, L_2 and L_3 are known constants. These three equations above are independent and easy to solve in a_2, a_3, a_4. Consequently, the number of unknown key bits is reduced from 192 to 96. To summarize, to attack 192 bits of the secret key in the related key scenario, we require 2^{64} chosen-IV queries to the first instance and 2^{96} chosen-IV queries to the second instance of the cipher and finally a brute force over 2^{96} values to find the two secret keys. Note that given the key of form K, it is sufficient for the attacker to have access to any of the 2^{64} possible keys related to K', as long as the difference between $K[1]$ and $K'[6]$ and also between $K[2]$ and $K'[7]$ is known, i.e. scenario (a) is assumed for both pairs for key subwords. If instead of (a), scenario (b) is assumed for one of the two key subword pairs in question, say for $K[1]$ and $K'[6]$, the attack proceeds as follows:

- For each b_1, b_0

  - Create a table T with rows containing (z_3, z_4, z_8, z_9) generated by (K, IV) where b_3 is varying

in $IV = (b_3, b_2, b_1, b_0)$ and b_2 is known and fixed

- Sort table T
- For each b'_0 search (z'_0, z'_1, z'_5, z'_6) generated using K' and $IV' = (b_1, 0, b_0, b'_0)$ in T. If found, return values for b_3, b_1, b_0 and b'_1
- Otherwise: delete table T

On average one incorrect candidate for b_3, b_1, b_0 and b'_1 will be returned by the procedure above since (6.19) is a 128-bit constraint. The procedure requires sorting 2^{64} tables, each table containing 2^{32} rows, storage size of 2^{32}, 2^{96} chosen-IV queries to both instances of the cipher and finally, an exhaustive search over 2^{96} possible key values. If both of the key subwords pairs in question are assumed to follow relation (b), around 2^{32} false candidates for b_3, b_2, b_1, b_0 and b'_1 out of possible $2^{32\times5}$ values are expected to pass the 128-bit constraint (6.19), which augments the computational effort of exhaustive search to $2^{96} \times 2^{32}$. Since for each b_1, b_0 a table containing (z_3, z_4, z_8, z_9) is formed, there is an additional cost of sorting 2^{64} tables, each table containing 2^{64} rows and a storage requirement of 2^{64}. The number of the chosen IV queries is 2^{128} and 2^{96} to the K and K' instances of the cipher, respectively.

Compared to Theorems 4 and 5, Theorem 6 is less favorable for attacks since, for keys K of the form specified by the theorem, the related key K' exists only for unique b_3 and b_2 which depend on the key K. Moreover, the b_3 and b_2 values participate in the expressions for the key subwords $K'[5]$ and $K'[6]$, respectively. In other words, given an instance with a key K, there exists no simple transformation, such as rotation or exclusive-or with a constant, to obtain K'. Thus, given an instance with an unknown key K, the attacker does not know which transformation has to be applied on K to obtain K'. Instead of assuming that, nonetheless, the attacker has access to two instances with related K and K', we present the attack in the following more relevant scenario. Let the attacker know the left-hand side values in equations $s^4_{13} = s'^0_{13}$ and $s^4_{14} = s'^0_{14}$ that determine the correct b_3 and b_2:

$$\alpha(a_1) \oplus a_3 \oplus \alpha^{-1}(a_4) \oplus (((\boxminus S((a_6 \oplus \mathbf{1}) \boxplus S(0))) \oplus \mathbf{1} \oplus b'_1) \boxplus (a_5 \oplus \mathbf{1})) \oplus S(0) \oplus a_1 \oplus \alpha(\mathbf{1}) = b_3$$

$$\alpha(a_2) \oplus a_4 \oplus \alpha^{-1}(a_5) \oplus ((a_1 \oplus \mathbf{1} \oplus b_3) \boxplus (a_6 \oplus \mathbf{1}) \boxplus S(0)) \oplus S(a_5 \oplus \mathbf{1}) \oplus a_2 \oplus \alpha(\mathbf{1}) = b_2$$

The assumption lowers the number of starting unknown key bits from 192 to 128. For a perfect stream cipher, recovering 128 unknown bits of the keys should not be possible in less than 2^{128} operations. By having the knowledge about the key, the attacker also has the values of correct b_2 and b_3. Now the b'_1 and b'_0 values that produce a slide pair are found by applying 2^{64} queries to the K' oracle and comparing with the corresponding output with the output of the K instance of the cipher, used with the $IV = (b_3, b_2, 0, 0)$. After the correct b'_1 and b'_0 have been found, the equations $s^4_{12} = s'^0_{12}$ and $s^4_{15} = s'^0_{15}$ can be used to restrict the key space:

$$\alpha(\boxminus S((a_6 \oplus \mathbf{1}) \boxplus S(0))) \oplus a_2 \oplus \alpha^{-1}(a_3) \oplus (S^{-1}(0) \boxminus S(a_5 \oplus \mathbf{1})) \oplus$$
$$\oplus(\boxminus S((a_6 \oplus \mathbf{1}) \boxplus S(0))) \oplus \mathbf{1} \oplus \alpha(\mathbf{1}) = b'_1$$
$$\alpha(a_3) \oplus a_5 \oplus \alpha^{-1}(a_6) \oplus (a_2 \oplus b_2 \oplus \mathbf{1} \boxplus (S^{-1}(0) \boxminus S(a_5 \oplus \mathbf{1})) \boxplus S(a_5 \oplus \mathbf{1})) \oplus$$
$$\oplus S((a_6 \oplus \mathbf{1}) \boxplus S(0)) \oplus a_3 \oplus \alpha(\mathbf{1}) = b'_0$$

The key space is reduced to $128 - 64 = 64$ bits. Since it is expected that one false b'_0 and b'_1 will pass the test, the exhaustive search over 2^{65} keys and 2^{64} queries to the second oracle suffice to attack 128 unknown key.

In the case of related key sets due to Theorem 1 and 2 for SNOW 3G and SNOW 2.0 with 128-bits, the attacks are unrelevant since the number of initial unkown key bits is only 2^{32}. The attack against keys specified by Theorem 3 is also less relevant since the exhaustive search over the initial unknown 64 bits is more effective than the attack, since it would require around 2^{96} chosen-IV queries.

Finally, it should be noted that equations that reduce the key space considered in this section contain operations tat are not linear in $GF(2^{32})$. For example, (6.20) contains operation $\boxplus$ and (6.21) contains an S-box S application. So, in the attack based on Theorem 5, another key K'' equal to K' on all subwords except on a_5 would allow another equation of the form (6.21), with a'_5 instead of a_5, which would in turn reveal $(\boxminus S(a_5 \oplus \mathbf{1})) \oplus (\boxminus S(a'_5 \oplus \mathbf{1}))$. However, in each case above, exploiting the non-linearity for obtaining more key bit information requires introducing more related keys. For example, changing b_3 in (6.23) requires new related key K', since K' depends on b_3. Moreover, introducing more related keys does not lower the number of required chosen-IV queries. Since in this section the focus has been on extending the flexibility of the related key attack, adding more related keys without improving the practicality of the related key attack

scenarios has been omitted.

6.5 Discussion and conclusions

We presented related key pair sets for SNOW 3G and SNOW 2.0 cipher by using a sliding technique. For several of the presented related key sets, the transformation from the key K to its related key K' is simple and amounts to rotation and bit inversion.

Using the derived related key sets, related-key key recovery attacks against SNOW 2.0 with 256-bit in complexity smaller than the exhaustive search can be mounted. Moreover, the fact that the K' depends on the IV of its related key was used to mount attacks under different assumptions on the related keys. Furthermore, the existence of the related keys exhibits non-random behavior of the ciphers, which questions the validity of the security proofs of protocols (such as the ones used in the 3GPP networks [63]) that are based on the assumption that SNOW 3G and SNOW 2.0 behave like ideal random functions when regarded as functions of the key-IV. For a more detailed discussion on related-key and *known-key* distinguishers, attacks, their security models and notions, the reader is referred to [81], [20].

7

Differential fault analysis of HC-128

The ECRYPT stream cipher project, also known as eSTREAM, is a project that aimed to identify new promising stream ciphers. The first call for stream cipher submissions was made in 2004 and it consisted of profile 1 and profile 2: software oriented ciphers and hardware oriented ciphers. The ciphers were put through a three-phase elimination process, finalizing in 2008, when four software oriented ciphers, including HC-128 and Rabbit, and three hardware oriented ciphers were selected as members of the eSTREAM portfolio. In this and the following chapter, we provide differential fault analysis of HC-128 and Rabbit stream ciphers.

HC-128 [134] is a high speed stream cipher that has passed all the three phases of the ECRYPT eSTREAM competition and is currently a member of the eSTREAM software portfolio. The cipher design is suitable for modern super-scalar processors. It uses a 128-bit secret key and 128-bit initialization vector. At each step, it produces a 32-bit keystream output word. The inner state of the cipher is relatively large and amounts to 32768 bits, consisting of two arrays, P and Q, of 512 32-bit words, each. HC-256 [133] is another cipher similar in structure to HC-128 but uses a 256-bit key and 256-bit IV.

In this chapter, we present a differential fault analysis attack on HC-128. The fault model in which we analyze the cipher is the one in which the attacker is able to fault a random word of the inner state of the cipher but cannot control its exact location nor its new faulted value. To perform the attack, we exploit the fact that some of the inner state words in HC-128 may be utilized several times without being updated. Our attack requires about 7968 faults and recovers the complete internal state of HC-128 by solving a set of 32 systems of linear equations over Z_2 in 1024 variables.

Along with the HC-128 proposal [134], an initial security analysis pointed out to a small bias in the

least significant bit of the output words which allows a distinguisher based on 2^{151} outputs. Contrary to the claims of the cipher designer [134], in [95] it was shown that the distinguisher can be extended to other bits as well, due to the bias occurring in the operation of addition of three n-bit integers, which is utilized in HC-128. However, the initial security claim [134] that there exists no distinguisher for HC-128 that uses less than 2^{64} bits [134] has not been even nearly contradicted. In [138], Zenner presented a cache timing analysis of HC-256 but this attack is not directly applicable to HC-128.

Our attack presented in this chapter requires around half the number of fault injections when compared to the attack [58] on RC4 in the equivalent fault model. In general, fault analysis attacks [30] fall under the category of implementation dependent attacks, which include side channel attacks such as timing analysis and power analysis. In fault analysis attacks, some kind of physical influence such as ionizing radiation is applied to the cryptographic device, resulting in a corruption of the internal memory or the computation process. The examination of the results under such faults often reveals some information about the cipher key or the secret inner state. The first fault analysis attack targeted the RSA cryptosystem in 1996 [30] and subsequently, fault analysis attacks were expanded to block ciphers (e.g., [19], [48]) and stream ciphers (e.g., [58], [74]). The threat of fault analysis attacks became more realistic after cheap and low-tech methods were found to induce faults.

Throughout our attack, we introduce a new technique which exploits what can be called the *reuse* of inner state words in different iterations of the cipher. Unlike in the fault analysis model which assumes that every fault inverts exactly one bit of the inner state, the fault model assumed in this chapter allows only the assumption that the fault will be localized in one of the 32-bit inner state words and no assumption on the distribution of the newly induced value, which impedes the differential analysis. The *reuse* of inner state words allows us to overcome this difficulty as follows. After faulting the inner state value, at one of the next iterations of the cipher, say at iteration r, the faulty value participates in the output. Based on the output at iteration r, some information about the difference between the original and the faulty value can be learned. If the same inner state value is *reused* at iteration $r + t$ without being updated in the meantime, the faulty value enters the output transformation with a partially known difference and the differential analysis can be applied to deduce information about the other values that participated in the output. It follows that the *reuse* of inner state values without updating it may facilitate fault analysis in weaker fault analysis models.

7.1 HC-128 specifications and definitions

The following notation is used throughout the chapter:

$+$ and $\boxminus$: addition mod 2^{32} and subtraction mod 512.

$\oplus$: bit-wise XOR.

$\ll$, $\gg$: left and right shift, respectively, defined on 32 bit values.

$\lll$, $\ggg$: left and right rotation, respectively, defined on 32 bit values.

x^b: The b^{th} bit of a word x.

$x^{c..b}$, where $c > b$: The word $x^c|x^{c-1}|..|x^b$.

$s'_i\langle P[f]\rangle, s'_i\langle Q[f]\rangle$: The faulty keystream, where the fault is inserted while the cipher is in state $i = 268$ and occurs at $P[f]$, $Q[f]$, respectively.

The HC-128 Keystream Generation Algorithm

1: $i = 0$
2: repeat until enough keystream bits are generated
3: $\quad j = i \bmod 512$
4: $\quad$ if $(i \bmod 1024) < 512$
5: $\quad\quad P[j] = P[j] + g_1(P[j \boxminus 3], P[j \boxminus 10], P[j \boxminus 511])$
6: $\quad\quad s_i = h_1(P[j \boxminus 12]) \oplus P[j]$
7: $\quad$ else
8: $\quad\quad Q[j] = Q[j] + g_2(Q[j \boxminus 3], Q[j \boxminus 10], Q[j \boxminus 511])$
9: $\quad\quad s_i = h_2(Q[j \boxminus 12]) \oplus Q[j]$
10: $\quad i = i + 1$

Figure 7.1: The HC-128 Keystream Generation Algorithm

The secret inner state of HC-128 consists of the tables P and Q, each containing 512 32-bit words. The execution of the cipher is governed by two public counters i and j. The functions g_1, g_2, h_1 and h_2 in Fig. 7.1, are defined as follows:

$$g_1(x, y, z) = ((x \ggg 10) \oplus (z \ggg 23)) + (y \ggg 8),$$

$$g_2(x, y, z) = ((x \lll 10) \oplus (z \lll 23)) + (y \lll 8),$$

$$h_1(x) = Q[x^{7..0}] + Q[256 + x^{23..16}], \;\; h_2(x) = P[x^{7..0}] + P[256 + x^{23..16}].$$

The key and IV initialization procedures are omitted since they are not relevant to our attack. We say that

HC-128 is *in state* i, if i steps have been executed, counting from the initial inner state. We will denote the iteration in which the cipher goes from state i to $i+1$ by *step* i.

Definition 2 *Let $P_s[j]$ denote the $P[j]$ value after it has been updated for s times by the HC-128 KGA. Similarly, let $Q_s[j]$ denote the $Q[j]$ value after it has been updated for s times, $j = 0, \ldots 511$.*

Definition 2 allows representing P and Q values at different cipher states as follows. If $s \in \{1, 2, \ldots\}$, $j \in \{0, \ldots 511\}$ and HC-128 is in state i, then

$$P[j] = \begin{cases} P_0[j], & i \in \{0, \ldots j\} \\ P_s[j], & i \in \{1024 \times (s-1) + j + 1, \ldots 1024 \times s + j\} \end{cases}$$

$$Q[j] = \begin{cases} Q_0[j], & i \in \{0, \ldots 512 + j\} \\ Q_s[j], & i \in \{1024 \times (s-1) + 512 + j + 1, \ldots \\ & \qquad 1024 \times s + 512 + j\} \end{cases}$$

To simplify the notation, regardless of whether h_1 or h_2 was called, the input value will be called the h *input value*. Both functions take a 32-bit word on the input. However, only the least significant byte and third least significant byte of the input value are used. Let x denote the input to the corresponding h function called in step i. Define $A_i = x^{7..0}$ and $B_i = 256 + x^{23..16}$.

7.2 The attack overview

The fault model in which we analyze the cipher is the one in which the attacker is able to fault a random word of the inner state tables P and Q but cannot control its exact location nor its new faulted value. We also assume that the attacker is able to reset the cipher arbitrary number of times. To perform the attack, the faults are induced while the cipher is in state 268 instead of state 0. Such a choice reduces the number of required faults to perform the attack. Throughout the rest of the chapter, whenever it is referred to a fault occurrence, it is assumed that the fault occurs when the cipher is in step $i = 268$. The aim of the attack is to recover the tables P_1 and Q_1, i.e. P and Q tables of the cipher in step $i = 1024$. Since the iteration

function of HC-128 is $1 - 1$, the inner state can then be rewind to the initial state $i = 0$. The attack can now be summarized as follows. First, the faults are induced and the corresponding output is collected as follows:

- Repeat the following steps until all of the P, Q words have been faulted at least once
  - Reset the cipher, iterate it for 268 steps and then induce the fault
  - Store the resulting faulty keystream words s'_i, $i = 268, \ldots 1535$

Then, the h input values, as defined in the previous section, are recovered for certain steps as follows:

- Recover the h input values in steps $512, \ldots 1023$ (details are provided in section 7.4.1)
- Recover a subset of the h input values in steps $1024, \ldots 1535$ (the size of the recovered subset is quantified in section 7.4.2)

The inner state is recovered, bit by bit, in 32 phases. In phase $b = 0$, the bits $P_1^0[i], Q_1^0[i]$, $i = 0, \ldots 512$ are recovered. Then, in phases $b = 1, \ldots 30$, assuming the knowledge of $P_1^{b-1..0}[i], Q_1^{b-1..0}[i]$, $i = 0, \ldots 512$, the bits $P_1^b[i], Q_1^b[i]$ are recovered. In each phase, a system of linear equations over Z_2 in $P_1^b[i], Q_1^b[i]$ is generated as follows:

- Generate 512 equations of the form $(P_1^b[A_i] + P_1^b[B_i]) \oplus Q_1^b[i] = s_i^b$, $i = 512, \ldots 1023$ (section 7.4.3)
- Recover a subset of the $P_1^b[0], \ldots P_1^b[255]$ and a subset of $Q_1^b[0], \ldots Q_1^b[255]$ values and add the recovered information to the system (section 7.4.4)
- Generate more equations in $P_1^b[i], Q_1^b[i]$ values by considering the relations between faulty and non-faulty keystreams (section 7.4.5)
- Solve the obtained system of linear equations

Finally, the most significant bits of all the P and Q words are recovered by phase $b = 31$.

7.3 The faulty value position and difference

In this section, two algorithms are provided. The first one is used to recover the XOR difference between certain faulty and non-faulty inner state values after the fault has been induced and the cipher is

iterated for certain number of steps. The algorithm is useful since the XOR differences between the non-faulty and the faulty inner state values is used to perform differential cryptanlaysis when the corresponding inner state values are *reused* in future cipher iterations. The second algorithm is used to recover the position of the induced fault. Before describing these two algorithms, an analysis of how the fault propagates as the cipher iterates is provided. Namely, we show that the position of the fault in the P or the Q tables uniquely determines the way by which the difference propagates through the corresponding table. This is due to the fact that, in HC-128, the update steps 5 and 8 in Fig. 7.1 use indices which are independent of the current state. Furthermore, although the indices used in the keystream output generation steps 6 and 9 depend on the inner state information, this does not impede the recovery of initial fault position, as will be shown below. To illustrate the above argument, assume that the fault occurred at $Q[f]$ while the cipher is in state $i = 268$. Since, according to line 5 of Fig. 7.1, the faulty value $Q'[f]$ is surely not referenced is during steps $i = 0, \ldots 511$, it follows that $P_1'[l] = P_1[l]$, $l = 0, \ldots 511$. Also, according to the update line 8 of Fig. 7.1, by which values $Q[j]$, $Q[j \boxminus 3]$, $Q[j \boxminus 10]$ and $Q[j \boxminus 511]$ are referenced, the first time in which $Q'[f]$ will be referenced is during the state in which $Q[f-1]$ is updated, i.e., in step $i = 512 + f - 1$. Thus, $Q_1[f-1] \neq Q_1'[f-1]$. More generally, define

$$\Delta Q_1[j] = \begin{cases} 0, & \text{if } Q_1[j] = Q_1'[j] \\ \\ 1, & \text{if } Q_1[j] \neq Q_1'[j]. \end{cases} \tag{7.1}$$

Applying the same logic to follow the propagation until state 1024, for $1 \leq f \leq 501$, it is straightforward to check that

$$(\Delta Q_1[j])_{j=0}^{512} = \underbrace{00\ldots0}_{j=0,\ldots f-2}\ 110110110\ \underbrace{111\ldots11}_{j=f+8,\ldots 511}$$

The difference propagation in the inner state is also partially projected to the keystream. For instance, if the fault occurs at $Q[f]$, then $s_j = s_j'$ holds for $512 \leq j < 512 + f - 1$. The first difference occurs at $i = 512 + j$, $j = f - 1$, after the value $Q[f-1]$ is affected and then referenced for the output in the same

step. We define

$$\Delta s_i = \begin{cases} 0 & \text{if } s_i = s_i' \\ 1 & \text{if } s_i \neq s_i' \end{cases} \tag{7.2}$$

to track the difference propagation in the keystream output. In the presented reasoning, we implicitly assume that any difference in the right-hand side values of lines 5,6,8 or 9 of Fig. 7.1 always causes a difference in the corresponding left-hand sides. For $100,000$ times, the inner state of HC-128 has been randomly initialized, iterated for 268 times and then faulted at random word. In all the $100,000$ experiments, the correctness of our assumption was verified. The following Lemmas provide the complete difference propagation patterns for both the inner state and the keystream. The proofs are omitted since they are straightforward.

Lemma 4 *If the fault occurred in the P table, its position f uniquely determines the sequence $(\Delta P_1[j])_{j=0}^{512}$. Similarly, if the fault occurred in the Q table, its position f uniquely determines the sequence $(\Delta Q_1[j])_{j=0}^{512}$. The corresponding sequences, depending on the fault positions, are given in Table 7.1.*

Lemma 5 *If the fault occurred in the P table, the fault position uniquely determines $(\Delta s_i)_{i=256}^{511}|(\Delta s_i)_{i=1024}^{1279}$. Similarly, if the fault occurred in the Q table, the fault position uniquely determines sequence $(\Delta s_i)_{i=512}^{1023}$. The corresponding sequences, depending on the fault position, are provided in Table 7.2.*

7.3.1 Recovering the differences between faulty and non-faulty words

After a fault is introduced, other P and Q values are affected as the cipher iterates. In this section, we show how to derive the difference between these affected faulty values and their original counterparts. For illustration, assume that the fault occurred at $Q[f]$. In step $i = 512 + f - 1$, the faulty and non-faulty keystream words will be produced by

$$s_{512+f-1} = h_2(Q[f-13]) \oplus Q[f-1], \;\; s'_{512+f-1} = h_2(Q'[f-13]) \oplus Q'[f-1].$$

Fault at $P[f]$	$(\Delta P_1[j])_{j=0}^{512}$
$f = 0$	$1\ \underbrace{0\ldots0}_{j=1\ldots510}\ 1$
$f \in \{1, \ldots 257\}$	$\underbrace{0\ldots0}_{j=0\ldots f-1}\ 1\ \underbrace{0\ldots0}_{j=f+1\ldots511}$
$f \in \{258, \ldots 264\}$	$\underbrace{0\ldots0}_{j=0\ldots f-1}\ 10000000001001001001101101 10\ \underbrace{1\ldots1}_{j=f+28\ldots511}$
$f \in \{265, 266, 267, 268\}$	$\underbrace{0\ldots0}_{j=0\ldots f-1}\ 1001001001101101 10\ \underbrace{1\ldots1}_{j=f+18\ldots511}$
$f \in \{269, \ldots 511\}$	$\underbrace{0\ldots0}_{j=0\ldots f-2}\ 11011011 0\ \underbrace{1\ldots1}_{j=f+8\ldots511}$
Fault at $Q[f]$	$(\Delta Q_1[j])_{j=0}^{512}$
$f = 0$	$1001001001101101 10\ \underbrace{1\ldots1}_{j=18\ldots511}$
$f \in \{1, \ldots 501\}$	$\underbrace{0\ldots0}_{j=0\ldots f-2}\ 11011011 0\ \underbrace{1\ldots1}_{j=f+8\ldots511}$
$f \in \{502, \ldots 508\}$	$\underbrace{0\ldots0}_{j=0\ldots f-503}\ 1001001001101101 10\ \underbrace{1\ldots1}_{j=f-484\ldots511}$
$f \in \{509, 510, 511\}$	$\underbrace{0\ldots0}_{j=0\ldots f-510}\ 100100110110110 0\ \underbrace{1\ldots1}_{j=f-493\ldots511}$

Table 7.1: The effect of faults induced during state 268 on the P and Q tables

However, since $Q'[f-13] = Q[f-13]$, it follows that $s_{512+f-1} \oplus s'_{512+f-1} = Q[f-1] \oplus Q'[f-1]$, which allows the recovery of $Q_1[f-1] \oplus Q'_1[f-1]$. For a fault position f, define the set $S(f)$ as follows:

$$l \in S(f) \Leftrightarrow 0 \leq l \leq 511 \quad \text{and} \quad l \in \{f-1, f, f+2, f+3, f+5, f+6, f+8, f+9, f+10, f+13, f+16, f+19\} \tag{7.3}$$

where "+" and "-" denote addition and subtraction in the set of integers Z. In other words, given a fault at position f in the P or Q tables, the set $S(f)$ defines the set of positions for which the difference from the original counterpart words can be recovered as given by the following two Lemmas.

Lemma 6 *Let HC-128 be in step* 268 *when a fault occurs in* $P[f]$, $269 \leq f \leq 511$. *Then, for* $l \in S(f)$, *we have*

$$P_1[l] \oplus P'_1[l] = s_l \oplus s'_l \tag{7.4}$$

Fault at $Q[f]$	$(\Delta s_i)_{i=512}^{1023}$
$f = 0$	$100100100110110110\ \underbrace{1\ldots1}_{i=18\ldots511}$
$f \in \{1,\ldots 499\}$	$\underbrace{0\ldots0}_{i=0\ldots f-2}\ 110110110\ \underbrace{1\ldots1}_{i=f+8\ldots511}$
$f = \{500, 501\}$	$\underbrace{0\ldots0}_{i=0..f-501}\ 1\ \underbrace{0\ldots0}_{i=f-499\ldots498}\ 1101101101111$
$f \in \{502,\ldots 508\}$	$\underbrace{0\ldots0}_{i=0\ldots f-503}\ 101100100110110110\ \underbrace{1\ldots1}_{i=f-484\ldots511}$
$f = \{509, 510, 511\}$	$\underbrace{0\ldots0}_{0..f-510}\ 100100110110110\ \underbrace{1\ldots1}_{i=f-494\ldots511}$
Fault at $P[f]$	$(\Delta s_i)_{i=256}^{511} \vert (\Delta s_i)_{i=1024}^{1279}$
$f \in \{0,\ldots 247\}$	$\underbrace{0\ldots0}_{i=0\ldots254+f}\ 110110110\ \underbrace{1\ldots1}_{i=263+f\ldots511}$
$f \in \{248,\ldots 255\}$	$\underbrace{0\ldots0}_{i=0\ldots254+f}\ \underbrace{110110110}_{i=255+f\ldots511}$
$f = 256$	$\underbrace{0\ldots0}_{i=0\ldots11}1\ \underbrace{0\ldots0}_{i=13\ldots510}\ 1$
$f = 257$	$\underbrace{0\ldots0}_{i=0\ldots12}1\ \underbrace{0\ldots0}_{i=14\ldots511}$
$f \in \{258,\ldots 264\}$	$\underbrace{0\ldots0}_{i=0\ldots f-247}\ 101100100110110110\ \underbrace{1\ldots1}_{i=f-228\ldots511}$
$f \in \{265,\ldots 267\}$	$\underbrace{0\ldots0}_{i=0\ldots f-254}\ 100100110110110\ \underbrace{1\ldots1}_{i=f-238\ldots511}$
$f = 268$	$\underbrace{0\ldots0}_{i=0\ldots11}100100100110110110\ \underbrace{1\ldots1}_{i=30\ldots511}$
$f \in \{269,\ldots 511\}$	$\underbrace{0\ldots0}_{i=0\ldots f-258}\ 110110110\ \underbrace{1\ldots1}_{i=f-248\ldots511}$

Table 7.2: The effect of faults induced during state 268 on the keystream

Proof: The distribution of corrupted values in P_1 when $f \geq 269$ is provided in Table 7.1. If $l = f - 1$, then

$$s_{f-1} = h_1(P_1[f-13]) \oplus P_1[f-1],\ \ s'_{f-1} = h_1(P'_1[f-13]) \oplus P'_1[f-1]$$

According to Table 7.1, $P_1[f-13] = P'_1[f-13]$ and since there is no corrupted values in the Q table, (7.4) follows. Similar proof follows for the other $l \in S(f)$ values, $269 \leq f \leq 511$. ■

Lemma 7 *Let HC-128 be in state* 268*, when a fault occurs in word* $Q[f]$*,* $0 \leq f \leq 501$*. Then, for* $l \in S(f)$*, we have*

$$Q_1[l] \oplus Q'_1[l] = s_{512+l} \oplus s'_{512+l} \tag{7.5}$$

The proof of Lemma 7 is analogous to the proof of Lemma 6. Note that the upper bound on f in Lemma 7 allows a simplified treatment of recoverable differences. Namely, if the fault is on $Q[f]$ for $f > 501$, the propagation starts as early as in step $i = 512$ and the set of recoverable differences differs from $S(f)$.

Given the fault position $P[f]$ or $Q[f]$, the above two Lemmas establish that for $l \in S(f)$, $P[l] \oplus P'[l]$ or $Q[l] \oplus Q'[l]$ can be recovered. A converse question can also be posed: Given a position, say $Q[l]$, which fault positions in the Q table will allow the recovery of $Q_1[l] \oplus Q'_1[l]$? For that purpose, it is convenient to define the set $S_Q^{-1}(l)$ for $0 \leq l \leq 511$ as follows

$$\begin{aligned} f \in S_Q^{-1}(l) \Leftrightarrow 0 \leq f \leq 501 \quad \text{and} \quad f \in \{&l+1, l, l-2, l-3, l-5, l-6, \\ &l-8, l-9, l-10, l-13, l-16, l-19\} \end{aligned} \tag{7.6}$$

Now, given a position $Q[l]$, the set $S_Q^{-1}(l)$ provides all fault positions such that $Q_1[l] \oplus Q'_1[l] = s_{512+l} \oplus s'_{512+l}$ according to Lemma 7. Similarly, given a position $268 \leq l \leq 511$ in the P table, the set $S_P^{-1}(l)$ defined by

$$\begin{aligned} f \in S_P^{-1}(l) \Leftrightarrow 269 \leq f \leq 511 \quad \text{and} \quad f \in \{&l+1, l, l-2, l-3, l-5, l-6, \\ &l-8, l-9, l-10, l-13, l-16, l-19\} \end{aligned} \tag{7.7}$$

provides the fault positions f such that $P_1[l] \oplus P'_1[l] = s_l \oplus s'_l$ can be recovered according to Lemma 6.

7.3.2 Recovering the position of the fault

In this section, we provide an algorithm to deduce the position where the fault occurred. Since, according to Lemmas 4 and 5, the fault position uniquely determines the corresponding sequences, the following functions can be defined:

$$\phi^P : f \mapsto (\Delta s_i)_{i=256}^{511} | (\Delta s_i)_{i=1024}^{1279}, \quad \phi^Q : f \mapsto (\Delta s_i)_{i=512}^{1023}$$

Algorithm 1: Fault Position Recovery

INPUT: $(\Delta s_i)_{i=256}^{1279} = (s_i \oplus s_i')_{i=256}^{1279}$
OUTPUT: The position where the fault occurred
1: If both $(\Delta s_i)_{512}^{1023} \in \Delta^P$ and $(\Delta s_i)_{i=256}^{511}|(\Delta s_i)_{i=1024}^{1279} \in \Delta^Q$, return *undefined*
2: If $(\Delta s_i)_{i=512}^{1023} \in \Delta^P$, return $\phi_P^{-1}((\Delta s_i)_{i=512}^{1023})$
3: If $(\Delta s_i)_{i=256}^{511}|(\Delta s_i)_{i=1024}^{1279} \in \Delta^Q$, return $\phi_Q^{-1}((\Delta s_i)_{i=256}^{511}|(\Delta s_i)_{i=1024}^{1279})$

The functions are explicitly given in Table 7.2. By checking that no two right-hand side sequences in both parts of the Table 7.2 are equal, it follows that

Lemma 8 *The functions ϕ^P and ϕ^Q are 1-1.*

Let $\Delta^P = \phi^P(\{0, \ldots 511\})$ and $\Delta^Q = \phi^Q(\{0, \ldots 511\})$. If the fault does not cause $(\Delta s_i)_{512}^{1023} \in \Delta^P$ and $(\Delta s_i)_{i=256}^{511}|(\Delta s_i)_{i=1024}^{1279} \in \Delta^Q$ at the same time, which, as will be shown, happens with negligible probability, then Algorithm 1 returns the fault position.

From line 1 of Algorithm 1, if there is conflicting information on whether the fault occurred in the P or the Q table, the algorithm returns *undefined*. To estimate the probability of this unwanted event, let $F_{P[f]}$ and $F_{Q[f]}$ denote the event that the fault occurs at position $P[f]$ and $Q[f]$, respectively. Let U denote the event that Algorithm 1 returns $undefined$. Then we have

$$\text{Prob}[U] = \sum_{f=0}^{511} \text{Prob}[U \cap F_{P[f]}] + \sum_{f=0}^{511} \text{Prob}[U \cap F_{Q[f]}] =$$
$$\frac{1}{1024}(\sum_{f=0}^{511} \text{Prob}[U|F_{P[f]}] + \sum_{f=0}^{511} \text{Prob}[U|F_{Q[f]}]) \tag{7.8}$$

where $\text{Prob}[U \cap F_{P[f]}] = \text{Prob}[F_{P[f]}]\text{Prob}[U|F_{P[f]}]$ and also $\text{Prob}[U \cap F_{Q[f]}] = \text{Prob}[F_{Q[f]}]\text{Prob}[U|F_{Q[f]}]$. To expand the probability $\text{Prob}[U|F_{P[f]}]$, let n_0 and n_1 denote the number of faulty values among the values $P[0], \ldots P[255]$ and $P[256] \ldots P[511]$, respectively, at state 512, given that the fault occurred at $P[f]$. Also, let $p = \frac{n_0+n_1}{256} - \frac{n_0 n_1}{256^2}$. If $n(\delta_i')$ is the number of 1 values in a 512-element sequence $\delta_i' \in \Delta_Q$, then

$$\text{Prob}[U|F_{P[f]}] = \sum_{\delta_i' \in \Delta^Q} \text{Prob}[(\Delta s_i)_{i=512}^{1023} = \delta_i'|F_{P[f]}] = \sum_{\delta_i' \in \Delta^Q} p^{n(\delta_i')}(1-p)^{512-n(\delta_i')}$$

As for the probability Prob$[U|F_{Q[f]}]$, let n_0 and n_1 denote the number of faulty words among $Q[0], \ldots Q[255]$ and $Q[256], \ldots Q[511]$, respectively, at state 268, given that the fault occurred at $Q[f]$. Let n_2 and n_3 denote the number of faulty words among $Q[0], \ldots Q[255]$ and among $Q[256], \ldots Q[511]$, respectively, at state 1024, given that the fault occurred at $Q[f]$. Let $p_0 = \frac{n_0+n_1}{256} - \frac{n_0 n_1}{256^2}$ and $p_1 = \frac{n_2+n_3}{256} - \frac{n_2 n_3}{256^2}$. If $m_0(\delta'_i)$ and $m_1(\delta'_i)$, denote the number of 1 values among $\delta'_{12}, \ldots \delta'_{255}$ and $\delta'_{256}, \ldots \delta'_{511}$, respectively, where $\delta'_i \in \Delta_P$, then

$$\text{Prob}[U|F_{Q[f]}] = \sum_{\delta'_i \in \Delta^P} \text{Prob}[(\Delta s_i)_{i=256}^{511} | (\Delta s_i)_{i=1024}^{1279} = \delta'_i | F_{Q[f]}] =$$
$$= \sum_{\delta'_i \in \Delta^P} p_0^{m_0(\delta'_i)} (1-p_0)^{244-m_0(\delta'_i)} p_1^{m_1(\delta'_i)} (1-p_1)^{256-m_1(\delta'_i)}$$

Calculating the sets Δ_P and Δ_Q and substituting the corresponding values using Table 7.2 allows the computation of the sums in Eq. (7.8) as $\frac{1}{1024}\sum_{f=0}^{511} \text{Prob}[U|F_{P[f]}] = 2^{-66.293}$ and $\frac{1}{1024}\sum_{f=0}^{511} \text{Prob}[U|F_{Q[f]}] = 2^{-30.406}$. Thus, the probability that Algorithm 1 returns $undefined$ as fault position is is Prob$[U] = 2^{-30.406}$.

7.4 Using DFA to generate equations

As described in section 7.2, the attack is performed by introducing faults until every P and Q word is faulted. Let T be the number of fault injections required to fault each of the 1024 words in the P and Q tables at least once. The expected number of required faults, $E(T)$, is given by $E(T) = 7698.4$ (see the coupons' collector problem in [105].) After inducing that number of faults, the average number of faults at a particular word $P[i]$ or $Q[i]$ will be $7698.4/1024 \approx 7.52$. As stated in section 7.2, the attack proceeds in 32 phases. Each phase b relies on the knowledge of $P_1^{b-1..0}[i]$, $Q_1^{b-1..0}[i]$, $i = 0, \ldots 511$, recovered in previous phases. Only the first phase, $b = 0$, does not require any previous bit knowledge. In each phase, a linear system of equations over Z_2 in $P_1^b[i]$, $Q_1^b[i]$, $i = 0, \ldots 511$ is generated and solved. Phase $b = 31$ proceeds with minor modifications compared to phases $0 \leq b \leq 30$, as explained below.

7.4.1 The recovery of h input values for steps $512, \ldots 1023$

In every HC-128 step, one of the two h functions is called, i.e., either h_1 or h_2. The input for the h functions is a 32-bit value, out of which only 16 bits, A_i, B_i, play a role in the computation. In this section, we describe a method to recover all of the A_i, B_i values, for $i = 512, \ldots 1023$.

To recover A_i, assume that the fault occurred at $P[f]$ while the cipher was in state 268. As can be seen from Table 7.1, if $1 \leq f \leq 255$, then as the cipher iterates through steps $i = 512, \ldots 1023$, no other P values gets corrupted. Also, the Q table does not get corrupted. Thus, in case $1 \leq f \leq 255$, the non-faulty and the faulty keystream words in step $512 \leq i \leq 1023$ are

$$s_i \;\; = \;\; (P_1[A_i] + P_1[B_i]) \oplus Q_1[j],\; s'_i\langle P[f]\rangle = (P'_1[A_i] + P_1[B_i]) \oplus Q_1[j]$$

Since $P'_1[A_i] \neq P_1[A_i]$ implies that $A_i = f$, then we have

$$s_i \neq s'_i\langle P_1[f]\rangle \Rightarrow A_i = f \tag{7.9}$$

In case $f = 0$, the fault does propagate to $P[511]$ and if $s_i \neq s'_i\langle P[0]\rangle$, then it is unclear whether $A_i = 0$ or $B_i = 511$, or both equalities hold. However, if there exists no faulty keystream for $1 \leq f \leq 255$ such that (7.9) is true, then $A_i = 0$. As for B_i, assume that a fault is inserted at word $P[f]$, $256 \leq f \leq 268$, while the cipher is in state 268. From Table 7.1, it is clear that at state 1024, none of the $P[0], \ldots P[f-1]$ values will be corrupted and the value $P[f]$ will necessarily be corrupted. Similarly, if the fault is inserted at $P[f]$ where $269 \leq f \leq 511$, none of the values $P[0], \ldots P[f-2]$ get corrupted and the value $P[f-1]$ will necessarily be corrupted. Thus, if f_{max} denotes the maximal f such that $s_i \neq s_i\langle P[f]\rangle$, then

$$B_i = \begin{cases} f_{max} & \text{if } f_{max} \in \{256, \ldots 268\} \\ f_{max} - 1 & \text{if } f_{max} \in \{269, \ldots 510\} \end{cases}$$

Finally, if $f_{max} = 511$, it is not clear whether $B_i = 510$ or $B_i = 511$. To differentiate between these two cases, it should be verified whether $s_i \neq s'_i$ also holds for any f which does not corrupt $P[511]$, for instance for $f = 507$. The recovery of A_i, B_i, for all $512 \leq i \leq 1023$ is given by Algorithm 2.

Algorithm 2: Recovery of A_i and B_i, for some $i = 512, \ldots 1023$

INPUT: Step $i \in \{512, \ldots 1023\}$
OUTPUT: A_i, B_i

1: If exists $1 \leq f \leq 255$ such that $s_i \neq s'_i\langle P[f]\rangle$: $A_i = f$
2: else $A_i = 0$
3: Find f_{max}, the maximum f such that $s_i \neq s_i\langle P[f]\rangle$
4: If $256 \leq f_{max} \leq 268$: $B_i = f_{max}$
5: else if $269 \leq f_{max} \leq 510$: $B'_i = f_{max} - 1$
6: else if $s_i = s_i\langle P[507]\rangle$: $B_i = 510$
7: else $B_i = 511$
9: Return A_i, B_i

Given the definition of h_2 and by noting that $j = i \bmod 512$, the recovered A_i and B_i values are in fact

$$A_i = \begin{cases} Q_0^{7..0}[j \boxminus 12] & \text{if } i \in \{512..523\} \\ \\ Q_1^{7..0}[j \boxminus 12] & \text{if } i \in \{524..1023\} \end{cases} \tag{7.10}$$

$$B_i = \begin{cases} Q_0^{23..16}[j \boxminus 12] + 256 & \text{if } i \in \{512..523\} \\ \\ Q_1^{23..16}[j \boxminus 12] + 256 & \text{if } i \in \{524..1023\} \end{cases} \tag{7.11}$$

7.4.2 The recovery of the h input values for steps $1024, \ldots 1535$

In this subsection, A_i, B_i values for a subset of $i = 1024, \ldots 1535$ are recovered. While B_i values will be recovered by a method similar to the one from the previous subsection, the same method is not applicable for A_i recovery and we will utilize the *reuse* of inner state words to recover the A_i values.

As for the recovery of B_i for $i = 1024, \ldots 1535$, from Table 7.1 it can be observed that if for some $1 \leq f \leq 501$, $Q[f]$ is faulted at step 268, the value $Q[f + 7]$ will remain unchanged and the values $Q[f + 8], \ldots Q[511]$ will surely be corrupted. Thus, if f_{min} denotes the minimal $249 \leq f \leq 501$ such that $s_i = s'_i\langle Q[f]\rangle$, then $B_i = f_{min} + 7$. Also, since $Q[509]$, $Q[510]$ and $Q[511]$ will get corrupted regardless of the fault position in Q, it is not possible to distinguish which of values 509, 510 or 511 B_i was equal to.

Thus, if for given step i, $B_i < 509$ holds, B_i will be recovered. Moreover, if $B_i < 500$, $Q_1^{7..0}[B_i]$ will be recovered by (7.11). Assuming that $B_i < 500$ for step i, then A_i can be recovered as follows. Consider

the faulty keystream $s'_i\langle Q[f]\rangle$ where $f \in S_Q^{-1}(B_i)$. According to Lemma 7 and (7.6)

$$Q_1[B_i] \oplus Q'_1[B_i] = s_{512+B_i} \oplus s'_{512+B_i}$$

Thus, $Q_1'^{7..0}[B_i]$ can be recovered by $Q_1'^{7..0}[B_i]$= $Q_1^{7..0}[B_i] \oplus s_{512+B_i}^{7..0} \oplus s_{512+B_i}'^{7..0}$. After being used in step $512 + B_i$, the value $Q[B_i]$ is *reused* in step i as follows

$$s_i \;\; = \;\; (Q_1[A_i] + Q_1[B_i]) \oplus P_2[j],\; s'_i\langle Q[f]\rangle = (Q'_1[A_i] + Q'_1[B_i]) \oplus P'_2[j]$$

If $257 \leq f \leq 501$, $Q_1[A_i] = Q'_1[A_i]$ holds according to Table 7.1. Also, the P table remains uncorrupted and thus $P_2[j] = P'_2[j]$. Thus, focusing on the least significant byte and XORing the previous two values yields

$$s_i^{7..0} \oplus s_i'^{7..0}\langle Q[f]\rangle = (Q_1^{7..0}[A_i] + Q_1^{7..0}[B_i]) \oplus (Q_1^{7..0}[A_i] + Q_1'^{7..0}[B_i]) \tag{7.12}$$

Since $s_i^{7..0} \oplus s_i'^{7..0}$, $Q_1^{7..0}[B_i]$ and $Q_1'^{7..0}[B_i]$ are known, (7.12) represents a test that allows eliminating some wrong candidates for $Q_1^{7..0}[A_i]$ value. One test of the form (7.12) will be generated for each faulty instance for which the fault position is $Q[f]$, where $f \in S_Q^{-1}(B_i)$. Consequently, an $0 \leq A_i \leq 255$ can be discarded if the corresponding $Q_1^{7..0}[A_i]$ recovered by (7.11) does not satisfy (7.12).

Algorithm 3: Recovery of A_i and B_i, for some $i = 1024, \ldots 1535$

INPUT: Step $i \in \{1024, \ldots 1535\}$
OUTPUT: A_i or $undef$, B_i or $undef$

1: Calculate $F = \{257 \leq f \leq 501 | s_i = s'_i\langle Q[f]\rangle\}$
2: If $|F| = 0$: $B_i = undef$
3: Else $B_i = min(F) + 7$
4: If $B_i > 500$ $A_i = undef$; Return A_i, B_i
5: Else: let $Cand(A_i) = \{0, 1, ..255\}$
6: For each $f \in S_Q^{-1}(B_i)$
7: Deduce $Q_1'^{7..0}[B_i] = Q_1^{7..0}[B_i] \oplus s_{512+B_i}^{7..0} \oplus s_{512+B_i}'^{7..0}$
8: For $A_i = 0, \ldots 255$
9: If $s_i^{7..0} \oplus s_i'^{7..0} \neq (Q_1^{7..0}[A_i] + Q_1^{7..0}[B_i]) \oplus (Q_1^{7..0}[A_i] + Q_1'^{7..0}[B_i])$
10: Eliminate A_i from $Cand(A_i)$
11: If for every σ_i,$\sigma'_i \in \{0, 1\}$, $s_i^{23..16} \oplus s_i'^{23..16} \neq d$, where
12: $d = (Q_1^{23..16}[A_i] + Q_1^{23..16}[B_i] + \sigma_i) \oplus (Q_1^{23..16}[A_i] + Q_1'^{23..16}[B_i] + \sigma'_i)$
13: Eliminate A_i from $Cand(A_i)$
14: If $|cand(A_i)| = 1$, let A_i be the unique $cand(A_i)$ member
15: Else: $A_i = undef$
16: Return A_i, B_i

The test (7.12) can be reformulated so that the third least significant byte is used as follows

$$s_i^{23..16} \oplus s_i'^{23..16} =$$

$$(Q_1^{23..16}[A_i] + Q_1^{23..16}[B_i] + \sigma_i) \oplus (Q_1^{23..16}[A_i] + Q_1'^{23..16}[B_i] + \sigma_i') \tag{7.13}$$

where σ_i is a carry corrector defined to be 1 if $Q_1^{15..0}[A_i] + Q_1^{15..0}[B_i] \geq 2^{16}$ and 0 otherwise. Another carry corrector, σ_i', is defined analogously. The value $Q_1'^{23..16}[B_i]$ is obtained in the same way as the value $Q_1'^{7..0}[B_i]$ above. If $0 \leq A_i \leq 255$ and the corresponding $Q_1^{23..16}[A_i]$ are substituted in (7.13) and none of $\sigma_i, \sigma_i' \in \{0, 1\}$ satisfy the test, then A_i is discarded.

In what follows, we estimate the expected number of steps for which both the A_i and B_i values are recovered by the presented method. Let $1024 \leq i \leq 1535$ be a step of HC-128. If, for example, for step i, $257 \leq B_i \leq 492$, then the B_i value will surely be recovered as provided by the above method. Furthermore, for such a particular value B_i, $|S_Q^{-1}(B_i)| = 12$ will hold. Since for each $f \in S_Q^{-1}(B_i)$ around 7.52 faults occur at $Q[f]$, as shown at the beginning of section 7.4, around $7.52 \times 12 = 90.24$ tests given by Eq. (7.12) and the same number of tests given by Eq. (7.13) will be applied to the set of candidates for A_i. According to our experimental results, such a number of tests is sufficient to discard all the false candidates for A_i. In particular, an experiment in which Algorithm 3 was executed for all 512 steps $i \in \{1024, \ldots 1535\}$ for $10,000$ times, with random HC-128 instantiations, was conducted. On average, in 472.7 out of the 512 steps, both A_i and B_i values were recovered.

7.4.3 Equations of the form $P_1^b[A_i] \oplus P_1^b[B_i] \oplus Q_1^b[j] = s_i^b \oplus c_{i,b}$

After the steps given by subsections 7.4.1 and 7.4.2 have been executed, the attack proceeds in 32 phases, each consisting of 3 parts, as presented by the attack overview in section 7.2. In this subsection, the first part of b-th attack phase is presented.

The first part of b-th phase, in which starting 512 equations are generated, proceeds as follows. In steps $i \in \{512, \ldots 1023\}$, the keystream output word is generated as $(P_1[A_i] + P_1[B_i]) \oplus Q_1[j] = s_i$ where $j = i \bmod 512$. Since A_i and B_i for $i \in \{512, \ldots 1023\}$ have been recovered in subsection (7.4.1), focusing

on the b-th bit yields 512 bits equations of the form

$$P_1^b[A_i] \oplus P_1^b[B_i] \oplus Q_1^b[j] = s_i^b \oplus c_{i,b}, i = 512, \ldots 1023 \tag{7.14}$$

where $c_{i,b}$ is a known carry corrector which is equal to 1 if there is carry in $(P_1^{b-1..0}[A_i] + P_1^{b-1..0}[B_i])$ and 0 otherwise. In case $b \in \{0, \ldots 7\}$ or $b \in \{16, \ldots 23\}$, relying on the knowledge obtained by (7.10) and (7.11), the system can be extended by adding information $Q_1^b[w] = a_w, w = 0, \ldots 499$, regarded as equations. However, for $b \notin \{0, \ldots 7, 16, \ldots 23\}$ such equations are unavailable. Hence, a method to systematically add more equations to the system (7.14) that works for all $b = 0, \ldots 31$, i.e., that makes the corresponding system of rank 1024, is necessary. In order to provide a generic treatment for all b values, in what follows, equations derived from information given by (7.10) and (7.11) will not be utilized.

7.4.4 Recovering bits $P_1^b[0], \ldots P_1^b[255]$ and $Q_1^b[0], \ldots Q_1^b[255]$

In the second part of the b-th phase of the attack, the system of equations given by (7.14) is expanded. Note that in steps $512 \leq i \leq 1023$, the output is generated by $s_i = (P_1[A_i] + P_1[B_i]) \oplus Q_1[j]$, whereas in steps $1024 \leq i \leq 1535$, the output is generated by $s_i = (Q_1[A_i] + Q_1[B_i]) \oplus P_2[j]$. The idea is to corrupt $P_1[B_i]$ and $Q_1[B_i]$ in the previous two relations and recover $P_1[A_i]$ and $Q_1[A_i]$ by observing how these values react to addition of different values. The difference of the corrupted values is controlled by utilizing the *reuse* of $P_1[B_i]$ and $Q_1[B_i]$ over different states of the cipher. The analysis results in the recovery of a subset of the $P_1^b[0], \ldots P_1^b[255]$ and also a subset of the $Q_1^b[0], \ldots Q_1^b[255]$ values.

As for recovering $P_1^b[0], \ldots P_1^b[255]$, let $512 \leq i \leq 1023$ and $268 \leq B_i \leq 511$. Consider a fault at position $P[f]$, so that $f \in S_P^{-1}(B_i)$. Using Lemma 6 and (7.7), define $\delta = s_{B_i} \oplus s'_{B_i} = P_1[B_i] \oplus P'_1[B_i]$. Assume that for the faulty cipher instance in question, $\delta^b = 1$. Consider the difference

$$\Delta = s_i \oplus s'_i = (P_1[A_i] + P_1[B_i]) \oplus (P_1[A_i] + P'_1[B_i]),$$

and denote by c_b and c'_b the carry from the $b-1$ to b-th bit in the sums $P_1[A_i] + P_1[B_i]$ and $P_1[A_i] + P'_1[B_i]$,

respectively. If $c_b = c'_b$, then the bit $P^b[A_i]$ is recovered as follows

$$P_1^b[A_i] = \begin{cases} \delta^{b+1} \oplus \Delta^{b+1}, & \text{if } c_b = c'_b = 0 \\ \\ \delta^{b+1} \oplus \Delta^{b+1} \oplus 1, & \text{if } c_b = c'_b = 1 \end{cases} \tag{7.15}$$

If $c_b \neq c'_b$, the bit $P_1^b[B_i]$ is not uniquely determined and will not be recovered.

Algorithm 4: Recovery of $P_1^b[A_i]$, for some $i = 512, \ldots 1023$

INPUT: Step $i \in \{512, \ldots 1023\}$
OUTPUT: Bit $P_1^b[A_i]$, or *undef*

1: For every faulty keystream, where the fault occurred at $P[f]$, $f \in S_P^{-1}(B_i)$
2: Calculate $\delta = s_{B_i} \oplus s'_{B_i}$ and $\Delta = s_i \oplus s'_i$
3: If $P_1^{b-1..0}[x] + P_1^{b-1..0}[B_i] < 2^b$, set $c_b = 0$, else set $c_b = 1$
4: If $P_1^{b-1..0}[x] + P_1'^{b-1..0}[B_i] < 2^b$, set $c'_b = 0$, else set $c'_b = 1$
5: If $c_b = c'_b$:
6: Return $P_1^b[A_i]$ calculated according to (7.15)
7: Return *undef*

To recover $P_1^b[A_i]$, the explained procedure is repeatedly applied using each fault occurring at $P[f]$, $f \in S_P^{-1}(B_i)$. Let $p_1 = Prob[\delta^b = 1]$ and $p_2 = Prob[c_b = c'_b]$, then the probability of success can be lower bounded as follows:

$$Prob[P_1^b[A_i] \text{ recovery succeeds}] \geq \sum_{B_i=256}^{511} \frac{1}{256} \left(1 - (1 - p_1 p_2)^{|S_P^{-1}(B_i)|}\right) \tag{7.16}$$

The values $|S_P^{-1}(B_i)|$ are given by Table 7.4 and $Prob[\delta^b = 1] = \frac{1}{2}$. As for $Prob[c_b = c'_b]$, it can be modelled as the probability that there exists a carry at bit b in two random sums [127]. It achieves a minimum for $b = 31$ and thus the lower bound for the success probability over possible bit positions is given by $Prob$[Recovery of $P_1^b[A_i]$ succeeds] ≥ 0.908.

Now, the probability that for some particular $k \in \{0, \ldots 255\}$, the value $P_1^b[k]$ will not be recovered in some particular step i is then less than $1 - \frac{1}{256} \times 0.908$. Let $Z_k = 1$ if $P_1^b[k]$ has not been recovered after applying the algorithm for all steps $i = 512, \ldots 1023$. Otherwise, let $Z_k = 0$. The number of $P_1^b[k]$,

$0 \leq k \leq 255$ values not recovered can then be estimated as

$$E(\sum_{k=0}^{255} Z_k) = \sum_{k=0}^{255} E(Z_k) \leq 256 \times (1 - \frac{1}{256} \times 0.908)^{512} = 41.511 \tag{7.17}$$

Thus, the method presented in this section when applied on steps $i = 512, \ldots 1023$, is expected to recover more than $256 - 41.511 = 214.49$ of the $P_1^b[0], \ldots P_1^b[255]$ values. The exact procedure is presented by Algorithm 4.

As for recovering $Q_1^b[0], \ldots Q_1^b[255]$, an analogous technique, applied on steps $i = 1024, \ldots 1535$, is used. The exact procedure is presented by Algorithm 5. The expected number of recovered values is calculated analogously to (7.16), whereas it needs to be taken into account that A_i and B_i, $i \in \{1024, \ldots 1535\}$, need to be successfully recovered by subsection 7.4.2. The $|S_Q^{-1}(B_i)|$ values, given at Table 7.3, are more favorable than the corresponding $|S_P^{-1}(B_i)|$ values in (7.16). The expected number of $Q_1^b[0], \ldots Q_1^b[255]$ values to be recovered is 218.01.

Algorithm 5: Recovery of $Q_1^b[A_i]$, for some $i = 1024, \ldots 1535$

INPUT: Step $i \in \{1024, \ldots 1535\}$
OUTPUT: Bit $Q_1^b[A_i]$, or *undef*

1: If A_i or B_i are unknown, return *undef*
2: For every faulty keystream, where the fault occurred at $Q[f]$, $Q \in S_Q^{-1}(B_i)$
3: Calculate $\delta = s_{512+B_i} \oplus s'_{512+B_i}$ and $\Delta = s_i \oplus s'_i$
4: If $Q_1^{b-1..0}[x] + Q_1^{b-1..0}[B_i] < 2^b$, set $c_b = 0$, else set $c_b = 1$
5: If $Q_1^{b-1..0}[x] + Q_1'^{b-1..0}[B_i] < 2^b$, set $c'_b = 0$, else set $c'_b = 1$
6: If $c_b = c'_b = 0$: Return: $\delta^{b+1} \oplus \Delta^{b+1}$
6: If $c_b = c'_b = 1$: Return $\delta^{b+1} \oplus \Delta^{b+1} \oplus 1$
8: Return *undef*

l	0	1	2	3	4	5	6	7	8	9	
$\lvert S_Q^{-1}(l)\rvert$	2	2	3	4	4	5	6	6	7	8	
l	10	11	12	13	14	15	16	17	18	$19, \ldots 500$	
$\lvert S_Q^{-1}(l)\rvert$	9	9	9	10	10	10	11	11	11	12	
l	501	502	503	504	505	506	507	508	509	510	511
$\lvert S_Q^{-1}(l)\rvert$	11	10	10	9	8	8	7	6	6	5	4

Table 7.3: The number of fault positions which allow the recovery of $Q_1[l] \oplus Q'_1[l]$

l	268	269	270	271	272	273	274	275	276	277	278
$\lvert S_P^{-1}(l)\rvert$	1	2	2	3	4	4	5	6	6	7	8
l	279	280	281	282	283	284	285	286	287	$288, \ldots 510$	511
$\lvert S_P^{-1}(l)\rvert$	9	9	9	10	10	10	11	11	11	12	11

Table 7.4: The number of fault positions which allow the recovery of $P_1[l] \oplus P'_1[l]$

7.4.5 Utilizing equations in faulty bits

In this subsection, the system constructed in subsections 7.4.3 and 7.4.4 is expanded further for the purpose of attaining the full rank of the system. Consider the faulty output word in steps $512, \ldots 1023$, $s'_i = h_2(Q'_1[j \boxminus 12]) \oplus Q'_1[j]$. Evidently, regarding the previous relation as an equation is useless since it includes faulty inner state bits. Below, a method to transform the faulty inner state bits participating in the equation to original inner state bits is provided. Again, the *reuse* of inner state words is utilized.

Let the fault position be $Q[f]$, where $f \in S_Q^{-1}(l)$ and $244 \leq l \leq 499$. The non-faulty and the faulty instances of the cipher in step $i_0 = 512 + l + 12$ are

$$s_{i_0} \;=\; h_2(Q_1[l]) \oplus Q_1[l+12],\; s'_{i_0} = h_2(Q'_1[l]) \oplus Q'_1[l+12] \tag{7.18}$$

Note that $Q_1^{7..0}[l] = A_{i_0}$ and $Q_1^{23..16}[l] = B_{i_0} - 256$ are known according to subsection 7.4.1 and that the difference $Q'_1[l] \oplus Q_1[l]$ can be calculated as $\delta = Q'_1[l] \oplus Q_1[l] = s_{512+l} \oplus s'_{512+l}$, according to Lemma 7. Thus, A'_{i_0} and B'_{i_0} can be recovered as

$$A'_{i_0} \;=\; Q_1^{7..0}[l] \oplus \delta^{7..0},\; B'_{i_0} = Q^{23..16}[l] \oplus \delta^{23..16} + 256 \tag{7.19}$$

So, the second equation in line (7.18), considering only bit b, can be rewritten as

$$s'^{b}_{i_0} \oplus c_{i_0,b} = P_1^b[A'_{i_0}] \oplus P_1^b[B'_{i_0}] \oplus Q'^{b}_1[l+12] \tag{7.20}$$

where A'_{i_0} and B'_{i_0} are known and $c_{i_0,b}$ is an indicator of the carry in $P_1^{b-1..0}[A'_{i_0}] + P_1^{b-1..0}[B'_{i_0}]$ which is also known due to the assumption that bits $b-1, \ldots 0$ of all the P and Q words are known. Finally, to add equation (7.20) to the system constructed in the previous sections, the variable $Q'^{b}_1[l+12]$ needs to be

eliminated. Once again, to reexpress $Q_1^{'b}[l+12]$, the idea is to wait for this value to be *reused* once more during steps $1024, \ldots 1535$.

Due to the assumed lower bound $244 \leq l$, it follows that $l + 12 \geq 256$. Hence, it is possible for the B_i index in some step $1024 \leq i \leq 1535$ to take the value $l + 12$ which was used in step i_0. If such a step exists, denote it by i_1. Also, assume that $A_{i_1} < f - 1$, so that $Q_1[A_{i_1}] = Q'_1[A_{i_1}]$. Finally, assume that both A_{i_1} and B_{i_1} have been successfully recovered by the procedure given in subsection 7.4.2. Then, if $j_1 = i_1 \bmod 512$, the non-faulty and faulty keystream words are $s_{i_1} = (Q_1[A_{i_1}] + Q_1[B_{i_1}]) \oplus P_2[j_1]$ and $s'_{i_1} = (Q_1[A_{i_1}] + Q'_1[B_{i_1}]) \oplus P_2[j_1]$ and the difference can be computed as

$$s_{i_1} \oplus s'_{i_1} = (Q_1[A_{i_1}] + Q_1[B_{i_1}]) \oplus (Q_1[A_{i_1}] + Q'_1[B_{i_1}]) \tag{7.21}$$

Extracting bit b from (7.21) and cancelling out $Q_1[A_{i_1}]$ yields

$$s^b_{i_1} \oplus s^{'b}_{i_1} = Q^b_1[B_{i_1}] \oplus c_{i_1,b} \oplus Q^{'b}_1[B_{i_1}] \oplus c'_{i_1,b} \tag{7.22}$$

where $c_{i_1,b}$ and $c'_{i_1,b}$ are carry indicators for $Q_1^{b-1..0}[A_{i_1}] + Q_1^{b-1..0}[B_{i_1}]$ and $Q_1^{b-1..0}[A_{i_1}] + Q_1^{'b-1..0}[B_{i_1}]$, respectively. The carry indicator $c_{i_1,b}$ is calculated trivially and as for $c'_{i_1,b}$, it is necessary to find $Q_1^{'b-1..0}[B_{i_1}]$. For that, it suffices to focus on the bits $b-1, \ldots 0$ in equation (7.21), since all values except $Q_1^{'b-1..0}[B_{i_1}]$ are known and the required value can be calculated as

$$Q_1^{'b-1..0}[B_{i_1}] = ((s_{i_1}^{b-1..0} \oplus s_{i_1}^{'b-1..0}) \oplus (Q_1^{b-1..0}[A_{i_1}] + Q_1^{b-1..0}[B_{i_1}])) - Q_1^{b-1..0}[A_{i_1}] \tag{7.23}$$

After finding $c_{i_1,b}$ and $c'_{i_1,b}$, from (7.22) and since $B_{i_1} = l + 12$, $Q'_1[l + 12]$ can be expressed in terms of Q_1 bits as

$$Q^{'b}_1[l+12] = s^b_{i_1} \oplus s^{'b}_{i_1} \oplus Q^b_1[B_{i_1}] \oplus c_{i_1,b} \oplus c'_{i_1,b} \tag{7.24}$$

Substituting (7.24) in (7.20) yields

$$s^{'b}_{i_0} \oplus c_{i_0,b} = P^b_1[A'_{i_0}] \oplus P^b_1[B'_{i_0}] \oplus s^b_{i_1} \oplus s^{'b}_{i_1} \oplus Q^b_1[B_{i_1}] \oplus c_{i_1,b} \oplus c'_{i_1,b} \tag{7.25}$$

which is added to the system of equations without introducing any new variables. The described procedure

is summarized by Algorithm 6.

Algorithm 6: Add equations by expressing the faulty with non-faulty bits

INPUT: Faulty keystream for a fault occuring at $Q[f]$, $f \in S_P^{-1}(l)$, $244 \leq l \leq 499$
OUTPUT: An equation of form (7.25)

1: Let $\delta = s_{512+l} \oplus s'_{512+l}$ and $i_0 = 512 + l + 12$
2: Calculate A'_{i_0} and B'_{i_0} according to (7.19)
3: If $P_1^{b-1..0}[A'_{i_0}] + P_1^{b-1..0}[B'_{i_0}] < 2^b$ set $c_{i_0,b} = 0$, else $c_{i_0,b} = 1$
4: For $1024 \leq i_1 \leq 1535$ such that A_{i_0} and B_{i_0} are known
5: If $B_{i_0} = l + 12$ and $A_{i_0} < f - 1$
6: If $Q_1^{b-1..0}[A_{i_1}] + Q_1^{b-1..0}[B_{i_1}] < 2^b$, let $c_{i_1,b} = 0$, else $c_{i_1,b} = 1$
7: Calculate $Q_1'^{b-1..0}[B_{i_1}]$ according to (7.23)
8: If $Q_1^{b-1..0}[A_{i_1}] + Q_1'^{b-1..0}[B_{i_1}] < 2^b$, let $c'_{i_1,b} = 0$, else $c'_{i_1,b} = 1$
9: Return equation (7.25)

Let N denote the number of equations generated by repeating the procedure above for all $f \in S_Q^{-1}(l)$ and $244 \leq l \leq 499$. To estimate $E(N)$, let $\rho(l + 12)$ be the step number i, $1024 \leq i \leq 1535$, for which $B_i = l + 12$, if such a step exists. Also, let I denote the indicator function, returning 1 if the condition in question is true and returning 0 otherwise. Finally, let $FLT_Q[f]$ be the number of faults that occurr at position $Q[f]$. Then

$$N = \sum_{l=244}^{499} \sum_{f \in S_Q^{-1}(l)} FLT_Q[f] \times I[\rho(l+12) \text{ exists}] \times I[A_{\rho(l+12)} < f - 1] \times I[A_{\rho(l+12)}, B_{\rho(l+12)} \text{ known}]$$

Recall that $E(FLT_Q[f]) = 7.52$. Also, $E(I[\rho(l+12) \text{ exists}]) \approx 1-(\frac{255}{256})^{512}$. If $f > 257$, $E(I[A_{\rho(l+12)} < f-1]) = 1$ and otherwise $\frac{f-2}{256}$. Finally, according to subsection 7.4.2, $E(I[A_{\rho(l+12)}, B_{\rho(l+12)} \text{ known}]) \geq \frac{472.7}{512}$. Substituting the values above and using additivity of $E(\cdot)$ yields that $E(N) \geq 18380.1$.

7.5 Attack complexity and experimental results

Adding the number of equations generated by the algorithms presented in subsections 7.4.3, 7.4.4 and 7.4.5 gives a lower bound of $512 + 214.49 + 218.01 + 18380.1 = 19324.6$ equations expected to be in the final system for bits $b \in \{0, \ldots 30\}$. The correctness of the system and the uniqueness of the solution have been verified experimentally as follows. For 100 times HC-128 was randomly initialized and the faults have been simulated as specified by the attack. The procedures specified by by subsections 7.4.3, 7.4.4

and 7.4.5 have been executed and the rank of the resulting system of equations for bits $b \in \{0, \ldots 30\}$ was verified to be 1024 in all the 100 times. As for bit $b = 31$, the procedures from subsection 7.4.4 are not applicable, leaving out the system to be generated only by subsections 7.4.3 and 7.4.5, yielding about $512 + 18380.1 = 18892.1$ equations. Again, throughout the 100 experiments, the rank of resulting system for bit $b = 31$ was 1022 each time. Thus, to yield a complete HC-128 inner state, the missing two bits need to be guessed. The correctness of the guessed bits is easily verified by running the cipher and comparing the resulting key stream with the observed one. As for the attack complexity, around 7698.4 faults at random inner state words are required, as given by the beginning of section 7.4. The most expensive computational factor in the attack is solving the linear system of equations in 1024 bit variables for 32 times.

7.6 Conclusion

In this chapter, a DFA attack on HC-128 was presented. The adopted attack model assumes that the attacker is able to fault a random word of the inner state of the cipher but cannot control its exact location nor its new faulted value. The attack operates by constructing 32 systems of linear equations over Z_2, each of 1024 bit variables representing the inner state values. It also utilizes what we called the *reuse* of inner state words in different states of the cipher in order to facilitate the differential fault analysis.

8

Differential fault analysis of Rabbit

After passing all three phases of the eStream candidate elimination process, Rabbit [27] (also see RFC 4503) has become a member of profile 1 eSTREAM portfolio. While originally designed with high software performance in mind, Rabbit turns out to be also very fast and compact in hardware. Fully optimized software implementations achieve an encryption speed of up to 3.7 clock cycles per byte (CPB) on a Pentium 3, and of 9.7 CPB on an ARM7. It uses a 128-bit secret key, 64-bit IV and generates 128 pseudo-random bits as keystream output at each iteration. The size of the secret internal state amounts to 513 bits, consisting of two sets of 8 32-bit words and one additional 1-bit value.

In this chapter, we present a practical fault analysis attack on Rabbit. The fault model in which we analyze the cipher is the one in which the attacker is assumed to be able to fault a random bit of the internal state of the cipher but cannot control the exact location of injected faults. Our attack requires around $128 - 256$ faults, precomputed table of size $2^{41.6}$ bytes and recovers the complete internal state of Rabbit in about 2^{38} steps.

The security of Rabbit has been thoroughly investigated in the series of white papers published by the crypto lab at Cryptico A/S. These papers include analysis of the key setup function [39], analysis of IV-setup [42], mod n cryptanalysis [40], algebraic cryptanalysis [38] and periodic properties [41]. Also, a distinguishing attack requiring 2^{247} 128-bit samples was reported in [9]. The bias utilized in this attack was resulting from the bias in the Rabbit core function where it was shown that images of the Rabbit core function, g, have significantly less zeros than ones at each offset and this was used to show that there exists a bias in the least significant bit of certain keystream subblocks. This work was extended in [94], where the probability distribution of several keystream bits together was calculated by means of Fast Fourier Transform,

using the techniques described in [97]. The complexity of the latter attack is 2^{158}. The authors also presented an attack in which the $2^{51.5}$ instantiations of the cipher are analyzed based on the first three keystream output blocks of each instantiation. The additional assumption is that certain differences expressed in terms of XOR among these $2^{51.5}$ internal states are known. This attack recovers all $2^{51.5}$ keys and requires 2^{32} precomputation steps, 2^{32} memory, and $2^{97.5}$ steps. According to the authors, the attack is given under an unusual cryptanalytic assumption. This attack was considered the first known key recovery attack on Rabbit.

The fault model is the one in which an attacker is assumed to be able to cause a bit-flip at a random location in the internal state of the cipher. However, the exact position of the flipped bit is unknown to the attacker. The attacker is also assumed to be able reinitialize the cipher sufficient amount of times, iterate and obtain keystream words. The main idea of the attack is to gain information on the input value of the g function based on its input-output differences obtained during fault analysis.

8.1 Fault analysis

Cryptanalytic attacks can be broadly classified into two classes. In the first class of attacks, the attacker tries to exploit any weakness in the underlying mathematical structure of the cipher. This type includes, for example, differential cryptanalysis, linear cryptanalysis and algebraic attacks. The second class of attacks are implementation dependent attacks, which include side channel attacks, such as timing analysis [83] and power analysis [84], and fault analysis attacks. In fault analysis attacks [30], the attacker applies some kind of physical influence on the internal state of the cryptosystem, such as ionizing radiation which flips random bits in devices' memory. By examining the results of cryptographic operations under such faults, it is often possible to deduce information about the secret key or the secret internal state of the cipher.

Fault attacks were first introduced by Boneh *et al.* [30] in 1996 where they described attacks that targeted the RSA public key cryptosystem by exploiting a faulty Chinese remainder theorem computation to factor the modulus n. Subsequently, fault analysis attacks were extended to symmetric systems such as DES [19] and later to AES [48] and other primitives. Fault analysis attacks became a more realistic serious threat after cheap and low-tech methods of applying faults were presented (e.g., [6, 126]).

Hoch and Shamir [58] showed that fault analysis attacks present a powerful tool against stream ciphers as well. Stream ciphers based on LFSRs, LILI-128 and SOBER-t32 as well as RC4 were shown to be

insecure in a fault analysis model in which the attacker does not have the ability to choose the exact location of the induced fault. In the case of RC4, the key recovery attack requires 2^{16} faults and 2^{26} keystream words. In [17], RC4 was assessed using a different fault model in which the attacker may specify the location at which the fault is induced but can not specify the value of injected faults. The attack requires 2^{16} induced faults. Another more advanced fault analysis attack on RC4 which requires 2^{10} faults was also introduced in the same paper.

Hojsík and Rudolf [59] presented an attack on another eSTREAM cipher, Trivium [33]. The attack recovers the secret internal state using 42 fault injections. The fault model used is the one in which the attacker is able to flip a random bit in the internal state of Trivium without being able to exactly control its location. This work was subsequently improved in [60], providing an attack that recovers Trivium inner state with only 3.2 fault injections on average. The authors used different cipher representation and were able to reduce high-degree equations to linear ones, concluding that a change in the way by which the cipher is represented may result in a better attack.

In this chapter, we use the same model as the one used in fault analysis of Trivium [59, 60]. The attacker is assumed to be able to flip a random bit in the internal state of the cipher without being able to exactly control its location. In other words, the exact location of induced fault is assumed to be unknown to the attacker.

The rest of the chapter is organized as follows. The Rabbit specifications that are relevant to our attack are briefly reviewed in the next section. The main idea behind our attack is presented in section 8.3.1. The procedure used to determine the location of induced faults is described in section 8.3.2 and the complete attack is described in section 8.3.3. Finally, the attack success probability and its associated complexity are analyzed in section 8.4.

8.2 Specification of Rabbit stream cipher

Internal state of Rabbit consists of 513 bits. It includes: eight 32-bit values: $x_{0,t}, \cdots x_{7,t}$, eight 32-bit counters, $c_{0,t}, \ldots c_{7,t}$, and one additional bit $\phi_{7,t}$, used in the counter update. When the cipher steps from time t to time $t+1$, the counter is updated independently of x values, by adding known a_i values, corrected

with carries ϕ as follows:

$$c_{0,t+1} = c_{0,t} + a_0 + \phi_{7,t}$$

$$c_{j,t+1} = c_{j,t} + a_j + \phi_{j-1,t+1}, 1 \leq j \leq 7$$

where

$$\phi_{j,t+1} = \begin{cases} 1 - 1_{\mathbb{Z}/2^{32}\mathbb{Z}}(c_{0,t} + a_0 + \phi_{7,t}) & \text{if } j = 0 \\ 1 - 1_{\mathbb{Z}/2^{32}\mathbb{Z}}(c_{j,t} + a_j + \phi_{j-1,t+1}) & \text{if } j > 0 \end{cases}$$

and $a_0 = a_3 = a_6 = 4D34D34D$, $a_1 = a_4 = a_7 = D34D34D3$, $a_2 = a_5 = 34D34D34D$. Function $\mathbf{1}_{\mathbb{Z}/2^{32}\mathbb{Z}}$ is defined by

$$\mathbf{1}_{\mathbb{Z}/2^{32}\mathbb{Z}}(x) = \begin{cases} 0 & \text{if } x \geq 2^{32} \\ 1 & \text{if } x < 2^{32} \end{cases}$$

The x values are updated by

$$\begin{aligned} x_{0,t+1} &= g_{0,t} + (g_{7,t} \lll 16) + (g_{6,t} \lll 16) \\ x_{1,t+1} &= g_{1,t} + (g_{0,t} \lll 8) + g_{7,t} \\ x_{2,t+1} &= g_{2,t} + (g_{1,t} \lll 16) + (g_{0,t} \lll 16) \\ x_{3,t+1} &= g_{3,t} + (g_{2,t} \lll 8) + g_{1,t} \\ x_{4,t+1} &= g_{4,t} + (g_{3,t} \lll 16) + (g_{2,t} \lll 16) \\ x_{5,t+1} &= g_{5,t} + (g_{4,t} \lll 8) + g_{3,t} \\ x_{6,t+1} &= g_{6,t} + (g_{5,t} \lll 16) + (g_{4,t} \lll 16) \\ x_{7,t+1} &= g_{7,t} + (g_{6,t} \lll 8) + g_{5,t} \end{aligned} \tag{8.1}$$

where

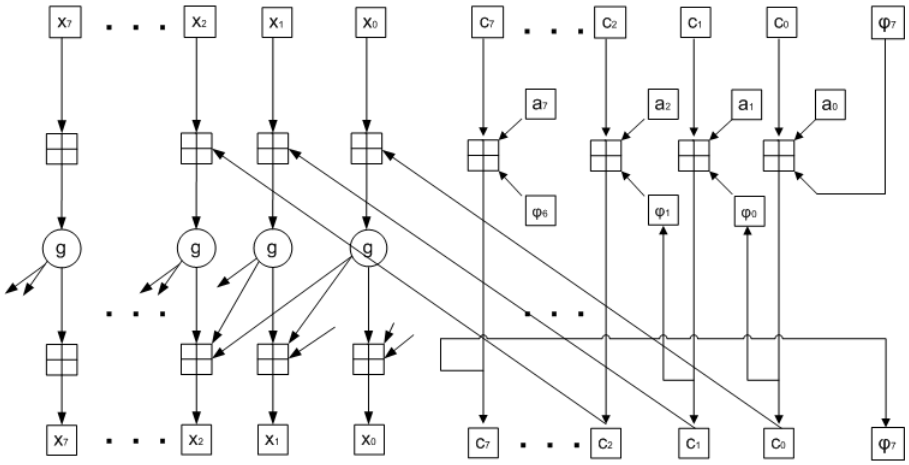

Figure 8.1: Simplified view of the state update function of Rabbit, rotations omitted

$$g_{j,t} = (x_{j,t} + c_{j,t+1})^2 \oplus [(x_{j,t} + c_{j,t+1})^2 \gg 32] \tag{8.2}$$

The 128-bit keystream output block $s_{t+1}^{[127..0]}$, is constructed as follows:

$$\begin{aligned}
s_{t+1}^{[15..0]} &= x_{0,t+1}^{[15..0]} \oplus x_{5,t+1}^{[31..16]}, & s_{t+1}^{[31..16]} &= x_{0,t+1}^{[31..16]} \oplus x_{3,t+1}^{[15..0]} \\
s_{t+1}^{[47..32]} &= x_{2,t+1}^{[15..0]} \oplus x_{7,t+1}^{[31..16]}, & s_{t+1}^{[63..48]} &= x_{2,t+1}^{[31..16]} \oplus x_{5,t+1}^{[15..0]} \\
s_{t+1}^{[79..64]} &= x_{4,t+1}^{[15..0]} \oplus x_{1,t+1}^{[31..16]}, & s_{t+1}^{[95..80]} &= x_{4,t+1}^{[31..16]} \oplus x_{7,t+1}^{[15..0]} \\
s_{t+1}^{[111..96]} &= x_{6,t+1}^{[15..0]} \oplus x_{3,t+1}^{[31..16]}, & s_{t+1}^{[127..112]} &= x_{6,t+1}^{[31..16]} \oplus x_{1,t+1}^{[15..0]}
\end{aligned} \tag{8.3}$$

Figure 8.1 shows a simplified view the Rabbit state update function. The description of the key setup scheme of Rabbit is omitted since it does not play a role in the attack outlined in this chapter.

8.3 Differential fault analysis attack

Throughout the rest of this chapter, faulty words will be denoted same as non-faulty ones, except that a "′" sign will be added. This way, faulty Rabbit internal state words at time t will be denoted by $x'_{i,t}$, $c'_{j,t}$, $\phi'_{7,t}$. The whole Rabbit internal state at time t, consisting of $[(x_{i,t})_{i=0...7}, (c_{i,t})_{i=0...7}, \phi_{7,t}]$, will be denoted by S_t. Accordingly, its faulty counterpart will be denoted by S'_t. We will also use "$+$" to denote addition mod 32, unless otherwise stated.

Faulty keystream output at step t will be denoted by s'_t. The i-th 16-bit segment of word s will be denoted by $s^{(i)}$. For example $s_t^{(1)}$ denotes $s^{[31..16]}$, i.e., bits 16 to 31 of word s_t.

According to our fault analysis model, the attacker has the power to flip a bit within the internal state of the cipher, that is $x_{i,t}, c_{i,t}$, $i = 0, \ldots 7$, $\phi_{7,t}$ but the attacker can not control or know the exact location of the induced fault (both at the bit and at the word level).

8.3.1 The Main Idea

Before stating the complete attack procedure, we provide a motivational example that illustrates the idea behind the attack. Let states of Rabbit at step t, S_t and S'_t, differ only in i-th bit of word $x_{0,t}$. Consequently, $x'_{0,t} + c'_{0,t+1} = x_{0,t} + c_{0,t+1} + \sigma 2^i$, for some unknown $\sigma \in \{-1, +1\}$ and $i \in \{0, \ldots 31\}$. Then, with high probability, $g'_{0,t} \neq g_{0,t}$ and $g'_{i,t} = g_{i,t}$ for $i = 1..7$. This implies that $x'_{i,t+1} \neq x_{i,t+1}$, for $i = 0, 1, 2$ and $x'_{i,t+1} = x_{i,t+1}$ for $i = 3..7$. In particular, since $x'^{[31..16]}_{5,t+1} = x^{[31..16]}_{5,t+1}$ and $x'^{[15..0]}_{3,t+1} = x'^{[15..0]}_{3,t+1}$, then using the first line in Eq. (8.3), the following holds

$$s'^{[15..0]}_{t+1} = x'^{[15..0]}_{0,t+1} \oplus x^{[31..16]}_{5,t+1} \qquad s^{[15..0]}_{t+1} = x^{[15..0]}_{0,t+1} \oplus x^{[31..16]}_{5,t+1}$$
$$s'^{[31..16]}_{t+1} = x'^{[31..16]}_{0,t+1} \oplus x^{[15..0]}_{3,t+1} \qquad s^{[31..16]}_{t+1} = x^{[31..16]}_{0,t+1} \oplus x^{[15..0]}_{3,t+1}$$

Thus guessing $x^{[31..16]}_{5,t+1}$ and $x^{[15..0]}_{3,t+1}$ makes a candidate for values $x_{0,t+1}$ and $x'_{0,t+1}$ and consequently, using Eq. (8.1), a candidate for

$$x_{0,t+1} - x'_{0,t+1} =$$

$$(g_{0,t} + g_{7,t} \lll 16 + g_{6,t} \lll 16) - (g'_{0,t} + g'_{7,t} \lll 16 + g'_{6,t} \lll 16) = g'_{0,t} - g'_{0,t}$$

Since inputs to $g_{0,t}$ and $g'_{0,t}$ differ by $\pm 2^i$ for some unknown $i = 0, \ldots 31$, this constraint can be described by a set of g function additive differentials $\{(\pm 2^i, \delta) | i = 0, \ldots 31\}$.

Suppose now the attacker obtains two more faulted keystream words s''_{t+1} and s'''_{t+1}, derived from states S''_t and S'''_t differing from S_t on bits j and k of word $x_{0,t}$, where $k \neq j, k \neq i$, $j \neq i$. Since in all three cases, values $x^{[31..16]}_{5,t+1}$ and $x^{[15..0]}_{3,t+1}$ do not change, using s_{t+1}, s'_{t+1}, s''_{t+1} and s'''_{t+1} three sets of differentials $\{(\pm 2^i, \delta_1) | i = 0, \ldots 31\}$, $\{(\pm 2^i, \delta_2) | i = 0, \ldots 31\}$ and $\{(\pm 2^i, \delta_3) | i = 0, \ldots 31\}$ are obtained using the same

guess in the way described above. As will be shown later, the probability that there exists an input x for the g function such that it satisfies all three sets of differentials at once, i.e., such that there exist mutually different i_1, i_2 and i_3 such that

$$g(x) - g(x \pm 2^{i_1}) = \delta_1,$$
$$g(x) - g(x \pm 2^{i_2}) = \delta_2,$$
$$g(x) - g(x \pm 2^{i_3}) = \delta_3$$

is small if the guess above is not correct. Thus, the attacker is able to discard wrong guesses for $x_{5,t+1}^{[31..16]}$ and $x_{3,t+1}^{[15..0]}$. Also, if the guess is a correct one, the attacker obtains candidates for g input value $x_{0,t} + c_{0,t+1}$. In the following we provide a full internal state recovery algorithm.

8.3.2 Determining the position of the fault

In the attack proposed in this chapter, the first step after inducing a fault is to make restrictions on the position where the fault took place. The induced bit flipping can happen at one of the bits of words $x_{0,t}, \ldots x_{7,t}$, $c_{0,t}, \ldots c_{7,t}$ as well as at the 1-bit value $\phi_{7,t}$.

In the following we provide a tool for deducing important information on the location at which the fault occurred. Based on difference among faulty and non-faulty keystreams, information on the difference among internal states S_t and S'_t is deduced. More precisely, only keystream words s_t and s'_t will be used and according to the fault model, it will be assumed that internal states S_t and S'_t differ exactly on one bit.

To express these differences in a convenient way, we introduce the function d_{ST}, describing differences on the internal states and the function d_{KS}, describing differences among faulty and non-faulty keystream words. Let

$$d_{ST}(S, S') = \begin{cases} 0, & \text{if a fault occured either at } x_{0,t},\ c_{0,t} \text{ or } \phi_{7,t} \\ 1 \leq i \leq 7, & \text{if a fault accured either at } x_{i,t} \text{ or } c_{i,t} \end{cases}$$

The function d_{ST} is defined for every pair of states (S, S') that differ exactly on one bit. If s and s' are two 128-bit keystream words at some step, then we define

$$d_{KS}(s,s') = \begin{cases} 0, & s^{(5)} = s'^{(5)}, s^{(6)} = s'^{(6)} \text{ and } s^{(i)} \neq s'^{(i)} \text{ for } i \neq 5,6 \\ 1, & s^{(0)} = s'^{(0)}, s^{(5)} = s'^{(5)} \text{ and } s^{(i)} \neq s'^{(i)} \text{ for } i \neq 0,5 \\ 2, & s^{(0)} = s'^{(0)}, s^{(7)} = s'^{(7)} \text{ and } s^{(i)} \neq s'^{(i)} \text{ for } i \neq 0,7 \\ 3, & s^{(2)} = s'^{(2)}, s^{(7)} = s'^{(7)} \text{ and } s^{(i)} \neq s'^{(i)} \text{ for } i \neq 2,7 \\ 4, & s^{(1)} = s'^{(1)}, s^{(2)} = s'^{(2)} \text{ and } s^{(i)} \neq s'^{(i)} \text{ for } i \neq 1,2 \\ 5, & s^{(1)} = s'^{(1)}, s^{(4)} = s'^{(4)} \text{ and } s^{(i)} \neq s'^{(i)} \text{ for } i \neq 1,4 \\ 6, & s^{(3)} = s'^{(3)}, s^{(4)} = s'^{(4)} \text{ and } s^{(i)} \neq s'^{(i)} \text{ for } i \neq 3,4 \\ 7, & s^{(3)} = s'^{(3)}, s^{(6)} = s'^{(6)} \text{ and } s^{(i)} \neq s'^{(i)} \text{ for } i \neq 3,6 \end{cases}$$

If a pair of 128 bit (s, s') words does not satisfy any of the conditions proposed by the right-hand side of the equation above, function $d_{KS}(s, s')$ is *undefined*.

To understand the motivation behind the above definition, assume that the injected fault affected the input to the function $g_{0,t}$. From Figure 8.1, it is clear that such fault *directly* affects the computation of $x_{0,t+1}$, $x_{1,t+1}$ and $x_{2,t+1}$. From Eq. (8.3), it follows that these three terms also *directly* affect the computation of all words on the output stream except $s'^{[95..80]}_{t+1} = s'^{(5)}$ and $s'^{[111..96]}_{t+1} = s'^{(6)}$ which explains the first line in the above definition. A similar argument applies to to rest of entries in the definition of $d_{KS}(s, s')$.

The criterion for determining the position of the fault $d_{ST}(S_t, S'_t)$ based on the first keystream word can now be simply stated as follows:

- If $d_{KS}(s_{t+1}, s'_{t+1})$ is defined, put $d_{ST}(S_t, S'_t) = d_{KS}(s_{t+1}, s'_{t+1})$

- Otherwise, leave $d_{ST}(S_t, S'_t)$ undefined

During the attack, when after a fault $d_{ST}(S_t, S'_t)$ value is undefined, the fault will be discarded and the attacker proceeds by inducing another fault.

The successfulness of this criterion can be measured by two types of errors, p_{incorr} and p_{undef}. Error p_{incorr} is defined as the probability that the criterion returns a wrong $d_{ST}(S, S')$ value, while error p_{undef} is be defined as the probability that the criterion will leave $d_{ST}(S, S')$ undefined. The probability that the

criterion returns correct $d_{ST}(S, S')$ value will be denoted as p_{corr}.

Estimating p_{corr}, p_{incorr} and p_{undef} The estimates for the above defined probabilities are derived analytically. Firstly, we estimate the probability that a fault in one of the $c_{i,t}$, $i = 0..6$ values propagates to $c_{i+1,t+1}$ and not only to $c_{i,t+1}$ during the update step, via carry transfer mechanism implemented by auxiliary $\phi_{i,t+1}$ value. Suppose the fault occurred at position $c_{i,t}$, the probability that $c_{i,t} + a_i + \phi_{i-1,t+1}$ will have a carry at 32-nd bit place is approximately equal to

$$p_{cr} \approx \frac{1}{32} \sum_{i=1}^{32} \frac{1}{2^i} = 0.03125.$$

This probability is given by the event that that addition of $\pm 2^i$, $i = 0..31$ to a random 32-bit number x changes value of $1_{\mathbb{Z}/2^{32}\mathbb{Z}}(x)$. While the a_i values in the actual cipher are fixed ($a_i \in \{4D34D34D,$ $D34D34D3$,$34D34D34D\}$), our experimental results confirmed the accuracy of the above approximation.

Then, values p_{corr}, p_{incorr}, p_{undef} are estimated in what follows. Suppose the a random fault was induced in the internal state of the cipher. Then, the position of the bit-flip can be at

- $x_{i,t}, i \in \{0, \dots 7\}$ with probability $\frac{256}{513}$. In this case, our criterion will return a correct $d_{ST}(S, S')$ value if $g(x_{i,t} + c_{i,t+1}) \neq g(x'_{i,t} + c_{i,t+1})$, i.e., with probability $\frac{2^{32}-1}{2^{32}}$. In case that is not true, the criterion leaves $d_{ST}(S, S')$ undefined.

- $\phi_{7,t}$ with probability $\frac{1}{513}$. In this case, $c'_{0,t+1} \neq c_{0,t+1}$ with probability 1. Let $z \in \{0, \dots 7\}$ such that $c'_{j,t+1} \neq c_{j,t+1}$ for $j = 0, ..z$ and $c'_{j,t+1} = c_{j,t+1}$ for $j = z + 1, ..7$. If

  - $z = 0$, which happens with probability $\frac{2^{32}-1}{2^{32}}$, and $g(x_{0,t}+c_{0,t+1}) \neq g(x'_{0,t}+c_{0,t+1})$, for which the probability is $\frac{2^{32}-1}{2^{32}}$, then our criterion returns a correct value. If, however, in this case $g(x_{0,t} + c_{0,t+1}) = g(x'_{0,t} + c_{0,t+1})$ which happens with probability $\frac{1}{2^{32}}$, the criterion leaves $d_{ST}(S, S')$ undefined.

  - $z = 1$ which happens with probability $\frac{1}{2^{32}}$. In this case, we consider only the case $g(x_{i,t} + c_{i,t+1}) \neq g(x'_{i,t} + c_{i,t+1})$, $i = 0, 1$, probability being $(\frac{2^{32}-1}{2^{32}})^3$ and in this case again our criterion leaves $d_{ST}(S, S')$ undefined. Other possibilities within the case $z = 1$ are highly improbable and hence do not have any practical implications on the success probability of our attack.

  - $z \geq 2$ occurs with probability $(\frac{1}{2^{32}})^2 \times \frac{2^{32}-1}{2^{32}}$. We do not go into further consideration since these

events are highly improbable.

- $c_{i,t}, i \in \{0, \ldots 7\}$, with probability $\frac{256}{513}$. Again, let $z \in \{0, \ldots 7\}$ such that $c'_{j,t+1} \neq c_{j,t+1}$ for $j = i, \ldots i+z$ and $c'_{j,t+1} = c_{j,t+1}$ for $j = i+z+1, ..7$. If

  - $z = 0$, which happens with probability $1 - p_{cr}$ if $i \leq 6$ and with probability 1 if $i = 7$, the same analysis as with a fault on $x_{i,t}$ values applies. Namely, the correct $d_{ST}(S, S')$ value will be returned with probability $\frac{2^{32}-1}{2^{32}}$ and otherwise criterion value in question will be left undefined

  - $z = 1$, which happens with probability $p_{cr} \times \frac{2^{32}-1}{2^{32}}$ if $i \leq 5$, with probability p_{cr} if $i = 6$ and with probability 0 if $i = 7$, the following analysis applies. If values $g(x_{j,t} + c_{j,t+1})$ and $g(x_{j+1,t} + c_{j+1,t+1})$ are both equal to, or both different than $g(x'_{j,t} + c_{j,t+1})$ and $g(x'_{j+1,t} + c_{j+1,t+1})$, respectively, the criterion leaves $d_{ST}(S, S')$ undefined and the probability for this to happen is $(\frac{2^{32}-1}{2^{32}})^2 + (\frac{1}{2^{32}})^2$. However, if $g(x_{j,t} + c_{j,t+1}) = g(x_{j,t} + c'_{j,t+1})$ and $g(x_{j+1,t} + c_{j+1,t+1}) \neq g(x_{j+1,t} + c'_{j+1,t+1})$ the criterion returns the wrong value as an answer. The probability for this to happen is $\frac{1}{2^{32}} \times \frac{2^{32}-1}{2^{32}}$. In case $g(x_{j,t} + c_{j,t+1}) \neq g(x_{j,t} + c'_{j,t+1})$ and $g(x_{j+1,t} + c_{j+1,t+1}) = g(x_{j+1,t} + c'_{j+1,t+1})$, which occurs with the same probability, the criterion returns the right answer.

  - $z = 2$, which happens with probability $p_{cr} \times \frac{1}{2^{32}} \times \frac{2^{32}-1}{2^{32}}$ if $i \leq 4$, with probability $p_{cr} \times \frac{1}{2^{32}}$ if $i = 5$ and with probability 0 if $i \geq 6$, the following consideration applies. In the case where all three g values are changed, $d_{ST}(S, S')$ value is left undefined and the probability for this case is $(\frac{2^{32}-1}{2^{32}})^3$. Other cases are highly improbable and we do not consider them.

  - $z \geq 3$ occurs with probability $p_{cr} \times \frac{1}{2^{32}} \times \frac{2^{32}-1}{2^{32}}$. We do not go into consideration of further cases since their corresponding probabilities are negligible.

Using the probabilities from the discussion above, but ignoring parts that are less than $\frac{1}{2^{32}}$, provides a practically accurate estimate for the probability that the criterion will return a correct $d_{ST}(S, S')$ value as follows:

$$p_{corr} \approx \frac{256}{513}\frac{2^{32}-1}{2^{32}} + \frac{1}{513}(\frac{2^{32}-1}{2^{32}})^2 + \frac{7 \times 32}{513}(1 - p_{cr})\frac{2^{32}-1}{2^{32}} + \frac{32}{513}(1 \times \frac{2^{32}-1}{2^{32}}) = 0.98635$$

Again, ignoring terms that are less than $\frac{1}{2^{32}}$, probability of $d_{ST}(S, S')$ being left undefined is given by

$$p_{undef} \approx \frac{6 \times 32}{513} \times p_{cr} \times \frac{2^{32}-1}{2^{32}}((\frac{2^{32}-1}{2^{32}})^2 + (\frac{1}{2^{32}})^2) + \frac{32}{513} \times p_{cr} \times ((\frac{2^{32}-1}{2^{32}})^2 + (\frac{1}{2^{32}})^2) = 0.013645$$

Finally, ignoring terms less than $\frac{1}{2^{32}}$ yields probability for the criterion to return a false $d_{ST}(S, S')$ value is $p_{incorr} \approx 0$.

To confirm the previous theoretical estimates $p_{corr} \approx 0.98635$, $p_{undef} \approx 0.013645$ and $p_{incorr} \approx 0$, the following experiment was conducted. A Rabbit internal state was randomly initialized and a random fault was induced. The criterion was applied and it was noted which of the three options happened: correct position of the fault returned, incorrect position of the fault returned or position of the fault left undefined. After repeating the experiment for 10^6 times, the probabilities were obtained as $p_{corr} = 0.98408$, $p_{undef} = 0.015924$, and $p_{incorr} = 0$. Hence, given, say 100 faults, correct position will be determined for around 98 faults and 2 faults will be discarded. For no faults the incorrect position will be returned. Thus, it can be concluded that the proposed criterion represents reliable means for determining the position of faults.

8.3.3 The complete attack

Before stating the complete attack we introduce following definitions. Throughout the following three definitions, let $k \in \{0, 1, 2\}$ and $\sigma \in \{-1, +1\}$ be fixed values and let x be restricted to set $Z_{2^{32}}$. By $(\sigma 2^i, \delta)_k$ we denote a g function additive differential where the input difference is $\sigma 2^i$ and the output difference δ, taken after rotating g function output for $8 \times k$ bits.

Definition 3 *An x value will be considered to* satisfy differential $(\sigma 2^i, \delta)_k$ *if*

$$[g(x) \lll (8k)] - [g(x + \sigma 2^i) \lll (8k)] = \delta$$

Definition 4 *A set of differentials*

$$(\pm 2^i, \delta)_k|_{i=0}^{31} := \{(\sigma 2^i, \delta)_k | i = 0..31, \sigma = -1, +1\}$$

will be called a generalized differential. *An x value will be considered to* satisfy *generalized differential $(\pm 2^i, \delta)_k|_{i=0}^{31}$ if it satisfies any of the differentials contained in the set.*

Definition 5 *A set of generalized differentials*

$$\Delta = \{(\pm 2^i, \delta_1)_k|_{i=0}^{31}, (\pm 2^i, \delta_2)_k|_{i=0}^{31}, \ldots (\pm 2^i, \delta_n)_k|_{i=0}^{31})\}$$

will be considered satisfiable *if at least one x value satisfies them all, i.e., if there exists an x value as well as distinct values $d_1, \ldots d_n$, chosen from the set $\{\pm 2^i | i = 0..31\}$, such that*

$$[g(x) \lll 8k] - [g(x + d_j) \lll 8k] = \delta_j, j = 1, \ldots n$$

For such an x, we shall say that it satisfies *set Δ.*

The following procedure, `flt_init(t)` induces a sufficient number of faults at the internal state of the cipher at step t and arranges faulty keystream words to appropriate sets, using the mechanism described in Section 8.3.2.

- Let $FLTS_i = \emptyset$, $i = 0..7$.
- While $|FLTS_i| < 3$ for any $i = 0..7$
  - Reinitialize the cipher, forward to step t and induce a fault. Obtain s'_{t+1}. If $d_{KS}(s'_{t+1}, s_{t+1})$ is defined, let $i = d_{KS}(s'_{t+1}, s_{t+1})$ and add s'_{t+1} to $FLTS_i$.

The procedure that follows, `derive_inf(i,k)`, utilizes information in $FLTS_i$ to deduce the set of possible values for $x_{i,t} + c_{i,t+1}$. Parameter k can take values 0,1 and 2 and it determines the way $x_{i,t} + c_{i,t+1}$ value will be recovered. Namely, as will be seen from the algorithm, there are three different ways to derive candidates for this value and the logic of these three ways is encoded through values of $\alpha_{i,k}, \beta_{i,k}$, $k = 0, 1, 2$ in Table 8.1. The values for $\alpha_{i,k}, \beta_{i,k}$ have been derived utilizing Eq. (8.3). For example, running `derive_inf(0,0)`, returns the set of candidates for $x_{0,t} + c_{0,t+1}$ by working on values $s_{t+1}^{(0)}$ and $s_{t+1}^{(1)}$, i.e., guessing values $x_{3,t+1}^{[15..0]}$ and $x_{5,t+1}^{[31..16]}$, creating the set of generalized differentials using $g_{0,t} - g'_{0,t} = x_{0,t+1} - x'_{0,t+1}$ and finally finding g-input values that satisfy it. On the other hand running `derive_inf(0,1)` aims to recover the same value $x_{0,t} + c_{0,t+1}$, but in a different way. Namely, in this

case, the procedure operates on values $s_{t+1}^{(4)}$ and $s_{t+1}^{(7)}$, i.e., guesses $x_{4,t+1}^{[15..0]}$ and $x_{6,t+1}^{[31..16]}$, derives the generalized differential set by $g_{0,t} \lll 8 - g'_{0,t} \lll 8 = x_{2,t+1} - x'_{2,t+1}$ and then searches for g-input values that satisfy the set. The objective of obtaining the same value in three different ways is to take the intersection afterwards and hence minimize redundant candidates. Also, in the first case, a difference with no rotation was obtained and in the second, a difference after 8-bit rotations was found. The table is encoded so that whenever $k = 0$, $k = 1$ and $k = 2$, the number of rotations in the obtained difference will be 0, 8 and 16, respectively. This justifies the same value k present as index both for α, β and for generalized differentials themselves from $\Delta_i^k(A)$ sets in the procedure below.

The complete procedure `derive_inf(i,k)` follows:

- Let $Sat(\Delta_i^k) = \emptyset$
- For $A = 0, \ldots 2^{32} - 1$
  - Form the set of generalized differentials as follows:

$$\Delta_i^k(A) = \{ \quad (\pm 2^l, ([s^{(\alpha_{i,k})} || s^{(\beta_{i,k})}] \oplus A) - ([s'^{(\alpha_{i,k})} || s'^{(\beta_{i,k})}] \oplus A))_k |_{l=0}^{31} \\ | \ s' \in FLTS_i \quad \}$$

  - Let $Sat(\Delta_i^k) = Sat(\Delta_i^k) \cup Sat(\Delta_i^k(A))$, where $Sat(\Delta_i^k(A))$ is the set of x values that satisfy $\Delta_i^k(A)$

where $\alpha_{i,k}$, $\beta_{i,k}$, $i = 0..7$, $k = 0, 1, 2$ are defined by Table 8.1. The Derivation of $Sat(\Delta_i^k(A))$ sets is done using precomputation, as explained in Section 8.4.2. To recover g input values at step t, i.e., values $x_{i,t} + c_{i,t+1}$, the procedure `g_inp(t)` can be invoked, as follows:

- `flt_init(t)`
- For $i = 0, \ldots 7$
  - Call `derive_inf(i,0)`, `derive_inf(i,1)` and `derive_inf(i,2)` to find $Sat(\Delta_i^0)$, $Sat(\Delta_i^1)$ and $Sat(\Delta_i^2)$
  - $Cand(x_{i,t} + c_{i,t+1}) = Sat(\Delta_i^0) \cap Sat(\Delta_i^1) \cap Sat(\Delta_i^2)$

i	0	1	2	3	4	5	6	7
$\alpha_{i,0}$	1	4	3	6	5	0	7	2
$\beta_{i,0}$	0	7	2	1	4	3	6	5
$\alpha_{i,1}$	4	3	6	5	0	7	2	1
$\beta_{i,1}$	7	2	1	4	3	6	5	0
$\alpha_{i,2}$	3	6	5	0	7	2	1	4
$\beta_{i,2}$	2	1	4	3	6	5	0	7

Table 8.1: α and β index values used during the attack

In the next section it will be shown that the probability that there will be more than one candidate for $x_{i,t} + c_{i,t+1}$, i.e., that there will be more than 1 elements in the set $\texttt{Cand}(x_{i,t} + c_{i,t+1})$, is small.

Finally, the complete internal state at time $t = 1$ can be recovered by invoking the previous procedure for $t = 0$ which yields values $x_{i,0} + c_{i,1}$, $i = 0..7$. This in turn yields $g_{i,0}$, $i = 0..7$ values, which yield $x_{i,1}$, $i = 0..7$, by Eq. 8.1. Invoking the previous procedure once again for $t = 1$ yields values $x_{i,1} + c_{i,2}$, $i = 0..7$. Subtracting according values reveals $c_{i,2}$, $i = 0..7$. Now $c_{i,1}$ values can be recovered by reversing the counter one step backward, according to the specification of counter update step. Whether $\phi_{7,1} = 0$ or $\phi_{7,1} = 1$ is found by mere trying both options and comparing the resulting keystream words.

8.4 Attack success probability and complexity

8.4.1 Success Probability

In this section we show that the procedure from previous section determines the internal state uniquely. More precisely, it will be shown that $|Cand(x_{i,t} + c_{i,t+1})| = 1$, for any $i = 0 \ldots 7$ and $t \geq 0$ with high probability. This will be done by modelling g as a random function and then showing that if differences $([s^{(\alpha_{i,k})}||s^{(\beta_{i,k})}] \oplus A)$ - $([s'^{(\alpha_{i,k})}||s'^{(\beta_{i,k})}] \oplus A)$ are chosen uniformly randomly, i.e., not corresponding to the actual values produced by the attack procedure, this set of candidates will have 0 elements with high probability. Then, this probability can be taken as probability that $|Cand(x_{i,t} + c_{i,t+1})| = 1$ since following the procedure with actual differences and using the real g function guarantees existence of one correct candidate for $x_{i,t} + c_{i,t+1}$. According to the way complete internal state is recovered from g input values at times

$t = 0$ and $t = 1$, as described by the last paragraph of the previous section, it is clear that from uniqueness and correctness of g input values, uniqueness and correctness of the recovered internal state at step $t = 1$ follows.

Since during the algorithm, $Cand(x_{i,t}+c_{i,t+1})$ is derived as the intersection of $Sat(\Delta_i^0)$, $Sat(\Delta_i^1)$ and $Sat(\Delta_i^2)$, the probability distribution of the number of elements in these three sets is first examined. Assume g is a randomly chosen function and differences $(s^{(\alpha_{i,k})}|s^{(\beta_{i,k})})\oplus A-(s'^{(\alpha_{i,k})}|s'^{(\beta_{i,k})})\oplus A$ are chosen randomly uniformly. Sets $Sat(\Delta_i^k)$, $k = 0, 1, 2$ are formed as follows, as described by `derive_inf(i,k)`:

$$Sat(\Delta_i^k) = Sat(\Delta_i^k(0)) \cup \ldots \cup Sat(\Delta_i^k(2^{32} - 1))$$

For a given A, consider a generalized differential from $\Delta_i^k(A)$. The probability that random 32-bit x value will satisfy it is $63/2^{32}$. Since each set of generalized differentials $\Delta_i^k(A)$ contains at least three generalized differentials

$$P[\text{ x satisfies } \Delta_i^k(A)] \leq (63/2^{32})^3 = 2^{-78.068}$$

The probability that among 2^{32} possible x values there exists at least one that will satisfy $\Delta_i^k(A)$, i.e., the probability that $\Delta_i^k(A)$ is satisfiable, is

$$P[\text{ Some } x \in \{0, ..2^{32} - 1\} \text{ satisfies } \Delta_i^k(A)] \leq 1 - (1 - 2^{-78.068})^{2^{32}} \approx 2^{-46}$$

Finally, the probability that for at least one A there will exist an x that will satisfy $\Delta_i^k(A)$, i.e., the probability that $Sat(\Delta_i^k)$ is nonempty in a random model, is

$$P[\ Sat(\Delta_i^k) \text{ is empty }] \geq (1 - 2^{-46})^{2^{32}} = 1 - 2^{-14} \tag{8.4}$$

The final set of candidates for $x_{i,t} + c_{i,t+1}$ in procedure g_inp is derived as an intersection of $Sat(\Delta_i^0)$, $Sat(\Delta_i^1)$ and $Sat(\Delta_i^2)$. The probability that, in a random model, the intersection of these three sets is non-empty is

$$P[\text{ Randomly modelled } Cand(x_{i,t} + c_{i,t+1}) \text{ nonempty }] \leq (2^{-14})^3 = 2^{-42} \tag{8.5}$$

This can finally be taken as an upper bound for the probability that there will be an element other than the correct one in $Cand(x_{i,t} + c_{i+1,t})$. Since $Cand(x_{i,t} + c_{i,t+1})$ is calculated for $i = 0, \ldots 7$ at times $t = 0, 1$ during the attack procedure, it can be concluded that there will be no redundant candidates for the internal state after the procedure is completed.

8.4.2 Attack complexity

The attack complexity can be measured by the number of faults required, computational complexity as well as storage complexity. First, we examine the number of faults necessary to undertake an attack.

As described above, the input for the attack is a non-faulty keystream word s_{t+1} as well as certain number of faulty keystream words s'_{t+1}. Also, the set of faulty states from which s'_{t+1} values are produced needs to satisfy certain properties. More precisely, as specified by the `flt_init` procedure, at each of the following groups of bits

$$\begin{gathered} x_{0,t}, c_{0,t}, \phi_{7,t} \\ x_{1,t}, c_{1,t} \\ x_{2,t}, c_{2,t} \\ \vdots \\ x_{7,t}, c_{7,t} \end{gathered} \tag{8.6}$$

the attacker has to produce at least three different faults and obtain three corresponding s'_{t+1} values. It follows that the minimal number of required faults that will need to be induced is $3 \times 8 = 24$. However, since an attacker does not have the possibility to choose locations of faults he induces, the number of necessary faults will be higher.

Let n denote the overall number of induced faults. Let $p(n)$ denote the probability that that there will be at least 3 faults at each one of the 8 groups of bits above. Let A_i be the event that after inducing n random faults there will be at most 2 faults at $x_{i,t}, c_{i,t}$, or $x_{i,t}, c_{i,t}, \phi_{7,t}$ if $i = 0$. Then, $A_i = B_i^0 \cup B_i^1 \cup B_i^2$ where B_i^j, $j = 0, 1, 2$, $i = 0..7$ is an event that at $x_{i,t}, c_{i,t}$ or $x_{i,t}, c_{i,t}, \phi_{7,t}$ if $i = 0$ there will be 0,1 or 2 different faults.

Then, $p(n)$ can be approximated as follows:

$$
\begin{aligned}
& p(n) = \\
& 1 - P[A_0 \cup \ldots \cup A_7] = \\
& 1 - P[(B_0^0 \cup B_0^1 \cup B_0^2) \cup \ldots \cup (B_7^0 \cup B_7^1 \cup B_7^2)] \approx \\
& 1 - (\sum_{i=0}^{7} \sum_{j=0}^{2} P[B_i^j]) - (\sum_{i_1=0}^{7} \sum_{j_1=0}^{2} \sum_{i_2=i_1+1}^{7} \sum_{j_2=0}^{2} P[B_{i_1}^{j_1} \cap B_{i_2}^{j_2}])
\end{aligned}
$$

where the fact that $P[B_i^{j_1} \cap B_i^{j_2}] = 0$ for $j_1 \neq j_2$ has been used. For $i = 0 \ldots 7$

$$
\begin{aligned}
P[B_i^0] &= (\frac{7}{8})^n, i = 0 \ldots 7 \\
P[B_i^1] &= \sum_{k_1+k_2=n-1} (\frac{7}{8})^{k_1} (\frac{1}{8}) (\frac{7 \times 32 + 1}{8 \times 32})^{k_2} \\
P[B_i^2] &= \sum_{k_1+k_2+k_3=n-2} (\frac{7}{8})^{k_1} (\frac{1}{8}) (\frac{7 \times 32 + 1}{8 \times 32})^{k_2} (\frac{31}{8 \times 32}) (\frac{7 \times 32 + 2}{8 \times 32})^{k_3}
\end{aligned}
$$

The second order probabilities are provided as follows:

$$
\begin{aligned}
& P[B_{i_1}^0 \cap B_{i_2}^0] = (\frac{6}{8})^n \\
& P[B_{i_1}^0 \cap B_{i_2}^1] = \sum_{k_1+k_2=n-1} (\frac{6}{8})^{k_1} \frac{1}{8} (\frac{6 \times 32 + 1}{8 \times 32})^{k_2} \\
& P[B_{i_1}^0 \cap B_{i_2}^2] = \sum_{k_1+k_2+k_3=n-2} (\frac{6}{8})^{k_1} \frac{1}{8} (\frac{6 \times 32 + 1}{8 \times 32})^{k_2} \frac{31}{8 \times 32} (\frac{6 \times 32 + 2}{8 \times 32})^{k_3} \\
& P[B_{i_1}^1 \cap B_{i_2}^2] = \\
& 3 \times \sum_{k_1+k_2+k_3+k_4=n-3} (\frac{6}{8})^{k_1} \frac{1}{8} (\frac{6 \times 32 + 1}{8 \times 32})^{k_2} \frac{1}{8} (\frac{6 \times 32 + 2}{8 \times 32})^{k_3} \frac{31}{8 \times 32} (\frac{6 \times 32 + 3}{8 \times 32})^{k_4} \\
& P[B_{i_1}^2 \cap B_{i_2}^2] = \\
& 6 \times \sum_{k_1+k_2+k_3+k_4+k_5=n-4} (\frac{6}{8})^{k_1} \frac{1}{8} (\frac{6 \times 32 + 1}{8 \times 32})^{k_2} \frac{1}{8} (\frac{6 \times 32 + 2}{8 \times 32})^{k_3} \\
& \qquad \frac{31}{8 \times 32} (\frac{6 \times 32 + 3}{8 \times 32})^{k_4} \frac{31}{8 \times 32} (\frac{6 \times 32 + 4}{8 \times 32})^{k_5}
\end{aligned}
$$

Substituting the according values of n yields $p(64) = 0.900$, $p(96) = 0.997$ and $p(128) = 0.999$. The quality of the approximation above has been verified by the following experiment. For 10^5 times, a data structure equivalent to Rabbit internal state was initialized with zeros and n faults were simulated by writing

1 to a uniformly random chosen bit location. After each iteration, if there was at least three 1-bits at each of the groups of bits in question, a counter was incremented. At the end of the experiment, the probability was obtained by dividing the counter by 10^5. Obtained ratios for $n = 64, 96, 128$ were 0.900, 0.996, 0.999 respectively. Consequently, throughout the rest of the chapter, we assume that 64-128 faults are practically sufficient to grantee that there will be at least 3 faults at each one of the 8 groups of bits defined in Eq. (8.6).

Since during the attack, as described in Section 8.3.3, procedure `flt_init` is called two times, the number of necessary faults is around $128 - 256$.

As for computational and storage complexity, the `flt_init` procedure can make use of precomputation. In particular, 32 tables $T_0^+, \ldots T_{31}^+$ can be created, such that cell $T_i^+[j]$ contains all the x values such that $j = g(x) - g(x + 2^i)$. Another 32 tables $T_0^-, \ldots T_{31}^-$ can be created, such that cell $T_i^-[j]$ contains all the x values such that $j = g(x) - g(x - 2^i)$. Analogous sets of tables can be created for $[g(x) \lll 8] - g(x \pm 2^i) \lll 8]$ and $[g(x) \lll 16] - g(x \pm 2^i) \lll 16]$. Thus, the storage complexity is given by $3 \times 64 \times 2^{32} = 2^{39.6}$ words, i.e., $2^{41.6}$ bytes, and now the computational complexity for a query for x such that it satisfies a generalized differential is $O(1)$. Since around $2 \times 8 \times 3 \times 2^{32}$ such queries are made, the computational complexity of the attack is about 2^{38} steps.

To summarize, the proposed attack requires around $128 - 256$ faults, precomputed table of size $2^{41.6}$ bytes, and recovers the cipher internal state in about 2^{38} steps.

9

Summary and future research directions

9.1 Summary of contributions

This section briefly summarizes the contributions of this thesis. In the first two chapters, the motivation and the background for this work was presented. In addition, basic approaches to constructing symmetric key primitives and the corresponding cryptanalytic methods were explained.

In Chapter 3, a new heuristic for searching for compatible differential paths was presented. Our work shows that more hash function rounds can be reached if one applies automated reasoning to resolve the middle part of the boomerang structure. We have applied the proposed search heuristic to HAS-160 hash function. The heuristic has been explain by providing examples for the three types of propagations used during the search (single-path propagations, quartet propagations and quartet addition propagations). Using the 1-bit constraints along with these propagations yielded an acceptable rate of false positives and the second order collision was successfully found. A particular colliding quartet found by the heuristic has been provided.

In Chapter 4, a reduced-round SM3 compression function was studied in terms of resistance to second order collision attacks. We provided an example of a second order collision for the function reduced to 32 out of 64 steps. In particular, the interaction between the top and the bottom differential was studied in detail and by using a long carry propagation in one of the differentials, the contradictions were avoided. We also pointed out a property that does not exist in the function that SM3 is built on, SHA-2. In particular, a slide-rotational property of the SM3-XOR function is exposed. SM3-XOR is the SM3 hash function where modular addition is replaced by XOR. An example of a slide-rotational pair for SM3-XOR compression

function is given.

In Chapter 5, we presented a practical-complexity related-key attack on the Loiss stream cipher that recovers the full secret key. The attack was implemented and our implementation takes less than one hour on a PC with 3GHz Intel Pentium 4 processor to recover 92 bits of the 128-bit key. The possibility of extending the attack to a resynchronization attack in a single-key model was discussed. The main problem in the cipher is the innovation that has been added as a part of the Finite State Machine in the cipher structure. The new component is a slowly changing array reminiscent of the RC4 stream cipher. This component allowed differences to be contained (i.e., do not propagate) during a large number of inner state update steps with a relatively high probability thus allowing certain form of differential cryptanalysis to be efficient against all of the initial procedure steps. We also showed that a slide attack is possible for the Loiss stream cipher.

In Chapter 6, we studied the security of the two LFSR-based software oriented stream ciphers. In particular, we presented related key pair sets for SNOW 3G and SNOW 2.0 cipher by using the sliding technique. For several of the presented related key sets, the transformation from the key K to its related key K' is simple and amounts to rotation and bit inversion. Using the derived related key sets, related-key key recovery attacks against SNOW 2.0 with 256-bit in complexity smaller than the exhaustive search can be mounted. Moreover, the fact that the K' depends on the IV of its related key was used to mount attacks under different assumptions on the related keys. Furthermore, the existence of the related keys exhibits non-random behavior of the ciphers, which questions the validity of the security proofs of protocols (such as the ones used in the 3GPP networks [63]) that are based on the assumption that SNOW 3G and SNOW 2.0 behave like ideal random functions when regarded as functions of the key-IV.

In Chapter 7, a differential fault analysis attack was studied in the context of HC-128 cipher, which is based on a slow-changing array with comparatively large inner state. The adopted attack model assumes that the attacker is able to fault a random word of the inner state of the cipher but cannot control its exact location nor its new faulted value. The attack operates by constructing 32 systems of linear equations over Z_2, each of 1024 bit variables representing the inner state values. It also utilizes what we called the *reuse* of inner state words in different states of the cipher in order to facilitate the differential fault analysis.

In Chapter 8, differential fault analysis attack was devised for the Rabbit stream cipher which is an eStream final portfolio member. Unlike in the Chapter 7, the adopted fault model assumes that the attacker is able to fault a random bit of the inner state. Given this scenario and the different architecture of the Rabbit

stream cipher, the proposed attack requires around $128 - 256$ faults, precomputed table of size $2^{41.6}$ bytes, and recovers the cipher internal state in about 2^{38} steps.

9.2 Future work

In what follows, we list some of the possible topics of interest for future extension of our work:

- In the context of our work on second order analysis of hash functions provided in Chapters 3 and 4, it would be interesting to apply the proposed compatible differential search heuristic against other hash functions such as SHA-2 or SM3. A successful application of the heuristic would increase the number of steps for which the second order collision can be constructed. One possible direction would be to attempt to extend the attacks provided in [22] and [10]. The impact of the high rate of contradictory paths reported in the first order collision attack [100] should be investigated and re-measured in the context of the second order analysis of these hashes.

- When it comes to cryptanalysis of stream ciphers, it would be interesting to work on improving the attacks provided in Chapter 6 on the SNOW family. In particular, techniques for recovering the inner state given a known-LFSR difference would be of use in this context [24]. This is relevant since that after a slide pair of the inner states has been obtained, the difference in the LFSR typically has low Hamming weight, e.g., it is localized in one 32-bit word, as is the case in some of the related keys provided in 6. The same idea may also be evaluated in the context of our slide attack on Loiss stream cipher provided in Chapter 5.

- In the fault analysis of stream ciphers, certain constructions appear to require more faults to recover the inner state than others. One such construction is the construction based on slow-changing arrays and particularly the one with comparatively large state, e.g., HC-128 (analyzed in Chapter 7) or RC4 (see the respective fault attack [17]). The obstacle in applying the typical differential analysis approach is that the faulty inner state words pass only through a small portion of the inner state before control over the differences in the state is lost. The differential analysis in such a case allows only a small portion of inner state information per fault to be extracted. Investigating methods to overcome this problem in random-position fault models for the goal of reducing the overall number of random faults is a possible research direction.

Bibliography

[1] *Telecommunications Technology Association. Hash Function Standard Part 2, Hash Function Algorithm Standard (HAS-160), TTAS.KO-12.0011/R1, 2008.*

[2] Specification of SMS4, Block Cipher for WLAN Products - SMS4 (in Chinese), Declassified in: September 2006. `http://www.oscca.gov.cn/UpFile/20062101642319799O.pdf`.

[3] 3RD GENERATION PARTNERSHIP PROJECT. Technical Specification Group Services and System Aspects, 3G security, V3.1.1: 'Specification of the 3GPP Confidentiality and Integrity Algorithms: Document 2: KASUMI Specification', 2011.

[4] ACIIÇMEZ, O., SCHINDLER, W., AND ÇETIN KAYA KOÇ. Cache based remote timing attack on the AES. In *CT-RSA* (2007), M. Abe, Ed., vol. 4377 of *Lecture Notes in Computer Science*, Springer, pp. 271–286.

[5] AKKAR, M.-L., BEVAN, R., DISCHAMP, P., AND MOYART, D. Power analysis, what is now possible... In *ASIACRYPT* (2000), T. Okamoto, Ed., vol. 1976 of *Lecture Notes in Computer Science*, Springer, pp. 489–502.

[6] ANDERSON, R. J., AND KUHN, M. G. Low cost attacks on tamper resistant devices. In *Security Protocols Workshop* (1997), B. Christianson, B. Crispo, T. M. A. Lomas, and M. Roe, Eds., vol. 1361 of *Lecture Notes in Computer Science*, Springer, pp. 125–136.

[7] ASHUR, T., AND DUNKELMAN, O. Linear analysis of reduced-round CubeHash. In *ACNS* (2011), J. Lopez and G. Tsudik, Eds., vol. 6715 of *Lecture Notes in Computer Science*, pp. 462–478.

[8] AUMASSON, J.-P. Zero-sum distinguishers, Rump session talk at CHES, 2009. http://131002.net/data/talks/zerosum_rump.pdf.

[9] AUMASSON, J.-P., AND PHAN, R. C.-W. On the cryptanalysis of the hash function fugue: Partitioning and inside-out distinguishers. *Inf. Process. Lett. 111*, 11 (2011), 512–515.

[10] BAI, D., YU, H., WANG, G., AND WANG, X. Improved boomerang attacks on SM3. In *ACISP* (2013), C. Boyd and L. Simpson, Eds., vol. 7959 of *Lecture Notes in Computer Science*, Springer, pp. 251–266.

[11] BERBAIN, C., BILLET, O., CANTEAUT, A., COURTOIS, N., DEBRAIZE, B., GILBERT, H., GOUBIN, L., GOUGET, A., GRANBOULAN, L., LAURADOUX, C., MINIER, M., PORNIN, T., AND SIBERT, H. Decimv2. In *The eSTREAM Finalists*, M. J. B. Robshaw and O. Billet, Eds., vol. 4986 of *Lecture Notes in Computer Science*. Springer, 2008, pp. 140–151.

[12] BERBAIN, C., BILLET, O., CANTEAUT, A., COURTOIS, N., GILBERT, H., GOUBIN, L., GOUGET, A., GRANBOULAN, L., LAURADOUX, C., MINIER, M., PORNIN, T., AND SIBERT, H. Sosemanuk, a fast software-oriented stream cipher. In *The eSTREAM Finalists*, M. J. B. Robshaw and O. Billet, Eds., vol. 4986 of *Lecture Notes in Computer Science*. Springer, 2008, pp. 98–118.

[13] BERTONI, G., ZACCARIA, V., BREVEGLIERI, L., MONCHIERO, M., AND PALERMO, G. AES power attack based on induced cache miss and countermeasure. In *ITCC (1)* (2005), IEEE Computer Society, pp. 586–591.

[14] BIHAM, E., BIRYUKOV, A., AND SHAMIR, A. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In *EUROCRYPT* (1999), J. Stern, Ed., vol. 1592 of *Lecture Notes in Computer Science*, Springer, pp. 12–23.

[15] BIHAM, E., BIRYUKOV, A., AND SHAMIR, A. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. *J. Cryptology 18*, 4 (2005), 291–311.

[16] BIHAM, E., DUNKELMAN, O., AND KELLER, N. A related-key rectangle attack on the full KASUMI. In *ASIACRYPT* (2005), B. K. Roy, Ed., vol. 3788 of *Lecture Notes in Computer Science*, Springer, pp. 443–461.

[17] BIHAM, E., GRANBOULAN, L., AND NGUYEN, P. Q. Impossible fault analysis of RC4 and differential fault analysis of RC4. In *FSE* (2005), H. Gilbert and H. Handschuh, Eds., vol. 3557 of *Lecture Notes in Computer Science*, Springer, pp. 359–367.

[18] BIHAM, E., AND SHAMIR, A. Differential cryptanalysis of DES-like cryptosystems. In *CRYPTO* (1990), A. Menezes and S. A. Vanstone, Eds., vol. 537 of *Lecture Notes in Computer Science*, Springer, pp. 2–21.

[19] BIHAM, E., AND SHAMIR, A. Differential fault analysis of secret key cryptosystems. In *CRYPTO* (1997), B. S. K. Jr., Ed., vol. 1294 of *Lecture Notes in Computer Science*, Springer, pp. 513–525.

[20] BIRYUKOV, A., KHOVRATOVICH, D., AND NIKOLIC, I. Distinguisher and related-key attack on the full AES-256. In *CRYPTO* (2009), S. Halevi, Ed., vol. 5677 of *Lecture Notes in Computer Science*, Springer, pp. 231–249.

[21] BIRYUKOV, A., KIRCANSKI, A., AND YOUSSEF, A. M. Cryptanalysis of the Loiss Stream Cipher. In *Selected Areas in Cryptography* (2012), L. R. Knudsen and H. Wu, Eds., vol. 7707 of *Lecture Notes in Computer Science*, Springer, pp. 119–134.

[22] BIRYUKOV, A., LAMBERGER, M., MENDEL, F., AND NIKOLIC, I. Second-order differential collisions for reduced SHA-256. In *ASIACRYPT* (2011), D. H. Lee and X. Wang, Eds., vol. 7073 of *Lecture Notes in Computer Science*, Springer, pp. 270–287.

[23] BIRYUKOV, A., NIKOLIĆ, I., AND ROY, A. Boomerang attacks on BLAKE-32. In *FSE* (2011), A. Joux, Ed., vol. 6733 of *Lecture Notes in Computer Science*, Springer, pp. 218–237.

[24] BIRYUKOV, A., PRIEMUTH-SCHMID, D., AND ZHANG, B. Analysis of SNOW 3G resynchronization mechanism. In *SECRYPT* (2010), S. K. Katsikas and P. Samarati, Eds., SciTePress, pp. 327–333.

[25] BIRYUKOV, A., PRIEMUTH-SCHMID, D., AND ZHANG, B. Multiset collision attacks on reduced-round SNOW 3G and SNOW 3G $^{(+)}$. In *ACNS* (2010), J. Zhou and M. Yung, Eds., vol. 6123 of *Lecture Notes in Computer Science*, pp. 139–153.

[26] BIRYUKOV, A., AND WAGNER, D. Slide attacks. In *FSE* (1999), L. R. Knudsen, Ed., vol. 1636 of *Lecture Notes in Computer Science*, Springer, pp. 245–259.

[27] BOESGAARD, M., VESTERAGER, M., PEDERSEN, T., CHRISTIANSEN, J., AND SCAVENIUS, O. Rabbit: A new high-performance stream cipher. In *FSE* (2003), T. Johansson, Ed., vol. 2887 of *Lecture Notes in Computer Science*, Springer, pp. 307–329.

[28] BOESGAARD, M., VESTERAGER, M., AND ZENNER, E. The Rabbit stream cipher. In *The eSTREAM Finalists*, M. J. B. Robshaw and O. Billet, Eds., vol. 4986 of *Lecture Notes in Computer Science*. Springer, 2008, pp. 69–83.

[29] BOGDANOV, A., KHOVRATOVICH, D., AND RECHBERGER, C. Biclique cryptanalysis of the full AES. In *ASIACRYPT* (2011), D. H. Lee and X. Wang, Eds., vol. 7073 of *Lecture Notes in Computer Science*, Springer, pp. 344–371.

[30] BONEH, D., DEMILLO, R. A., AND LIPTON, R. J. On the importance of checking cryptographic protocols for faults (extended abstract). In *EUROCRYPT* (1997), W. Fumy, Ed., vol. 1233 of *Lecture Notes in Computer Science*, Springer, pp. 37–51.

[31] BONNEAU, J., AND MIRONOV, I. Cache-collision timing attacks against AES. In *CHES* (2006), L. Goubin and M. Matsui, Eds., vol. 4249 of *Lecture Notes in Computer Science*, Springer, pp. 201–215.

[32] C, J. D. G. Linear cryptanalysis of stream ciphers. In *FSE* (1994), B. Preneel, Ed., vol. 1008 of *Lecture Notes in Computer Science*, Springer, pp. 154–169.

[33] CANNIÈRE, C. D. Trivium: A stream cipher construction inspired by block cipher design principles. In *ISC* (2006), S. K. Katsikas, J. Lopez, M. Backes, S. Gritzalis, and B. Preneel, Eds., vol. 4176 of *Lecture Notes in Computer Science*, Springer, pp. 171–186.

[34] CANNIÈRE, C. D., KÜÇÜK, Ö., AND PRENEEL, B. Analysis of Grain's initialization algorithm. In *AFRICACRYPT* (2008), S. Vaudenay, Ed., vol. 5023 of *Lecture Notes in Computer Science*, Springer, pp. 276–289.

[35] CANNIÈRE, C. D., AND RECHBERGER, C. Finding SHA-1 characteristics: General results and applications. In *ASIACRYPT* (2006), X. Lai and K. Chen, Eds., vol. 4284 of *Lecture Notes in Computer Science*, Springer, pp. 1–20.

[36] CHO, H.-S., PARK, S., SUNG, S. H., AND YUN, A. Collision search attack for 53-step HAS-160. In *ICISC* (2006), M. S. Rhee and B. Lee, Eds., vol. 4296 of *Lecture Notes in Computer Science*, Springer, pp. 286–295.

[37] CORON, J.-S., PATARIN, J., AND SEURIN, Y. The random oracle model and the ideal cipher model are equivalent. In *CRYPTO* (2008), D. Wagner, Ed., vol. 5157 of *Lecture Notes in Computer Science*, Springer, pp. 1–20.

[38] CRYPTICO A/S. Algebraic Analysis of Rabbit, 2003, White Paper. Available from `www.cryptico.com`.

[39] CRYPTICO A/S. Analysis of the key setup function in Rabbit, 2003, White Paper. Available from `www.cryptico.com`.

[40] CRYPTICO A/S. Mod n analysis of Rabbit, 2003, White Paper. Available from `www.cryptico.com`.

[41] CRYPTICO A/S. Periodic properties of Rabbit, 2003, White Paper. Available from `www.cryptico.com`.

[42] CRYPTICO A/S. Security Analysis of the IV-setup for Rabbit, 2003, White Paper. Available from `www.cryptico.com`.

[43] DAEMEN, J., KNUDSEN, L. R., AND RIJMEN, V. The block cipher Square. In *FSE* (1997), E. Biham, Ed., vol. 1267 of *Lecture Notes in Computer Science*, Springer, pp. 149–165.

[44] DAEMEN, J., AND RIJMEN, V. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[45] DHEM, J.-F., KOEUNE, F., LEROUX, P.-A., MESTRÉ, P., QUISQUATER, J.-J., AND WILLEMS, J.-L. A practical implementation of the timing attack. In *CARDIS* (1998), J.-J. Quisquater and B. Schneier, Eds., vol. 1820 of *Lecture Notes in Computer Science*, Springer, pp. 167–182.

[46] DUNKELMAN, O., INDESTEEGE, S., AND KELLER, N. A differential-linear attack on 12-round Serpent. In *INDOCRYPT* (2008), D. R. Chowdhury, V. Rijmen, and A. Das, Eds., vol. 5365 of *Lecture Notes in Computer Science*, Springer, pp. 308–321.

[47] DUNKELMAN, O., KELLER, N., AND SHAMIR, A. A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3G telephony. In *CRYPTO* (2010), T. Rabin, Ed., vol. 6223 of *Lecture Notes in Computer Science*, Springer, pp. 393–410.

[48] DUSART, P., LETOURNEUX, G., AND VIVOLO, O. Differential fault analysis on A.E.S. In *ACNS* (2003), J. Zhou, M. Yung, and Y. Han, Eds., vol. 2846 of *Lecture Notes in Computer Science*, Springer, pp. 293–306.

[49] EKDAHL, P., AND JOHANSSON, T. A new version of the stream cipher SNOW. In *Selected Areas in Cryptography* (2002), K. Nyberg and H. M. Heys, Eds., vol. 2595 of *Lecture Notes in Computer Science*, Springer, pp. 47–61.

[50] ETSI/SAGE. Document 2: Specification of the 3GPP Confidentiality and Integrity Algorthithms 128-EEA3 & 128-EUA3: ZUC specification', Version 1.4, 2010. Available at: http://gsmworld.com/our-work/programmes-and-initiatives/fraud-and-securit

[51] ETSI/SAGE. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2&UIA2. Document 2: SNOW 3G Specification, version 1.1' (September 2006) http://www.3gpp.org/ftp.

[52] ETSI/SAGE. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2&UIA2. Document 5: Design and Evaluation Report: version 1.1' (September 2006) http://www.3gpp.org/ftp.

[53] FENG, D., FENG, X., ZHANG, W., FAN, X., AND WU, C. Loiss: A byte-oriented stream cipher. In *IWCC* (2011), Y. M. Chee, Z. Guo, S. Ling, F. Shao, Y. Tang, H. Wang, and C. Xing, Eds., vol. 6639 of *Lecture Notes in Computer Science*, Springer, pp. 109–125.

[54] FOUQUE, P.-A., LEURENT, G., AND NGUYEN, P. Q. Automatic search of differential path in MD4. *IACR Cryptology ePrint Archive 2007* (2007), 206.

[55] GILBERT, H., AND PEYRIN, T. Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In *FSE* (2010), S. Hong and T. Iwata, Eds., vol. 6147 of *Lecture Notes in Computer Science*, Springer, pp. 365–383.

[56] GORSKI, M., LUCKS, S., AND PEYRIN, T. Slide attacks on a class of hash functions. In *ASIACRYPT* (2008), J. Pieprzyk, Ed., vol. 5350 of *Lecture Notes in Computer Science*, Springer, pp. 143–160.

[57] HEVIA, A., AND KIWI, M. A. Strength of two data encryption standard implementations under timing attacks. *ACM Trans. Inf. Syst. Secur. 2*, 4 (1999), 416–437.

[58] HOCH, J. J., AND SHAMIR, A. Fault analysis of stream ciphers. In *CHES* (2004), M. Joye and J.-J. Quisquater, Eds., vol. 3156 of *Lecture Notes in Computer Science*, Springer, pp. 240–253.

[59] HOJSÍK, M., AND RUDOLF, B. Differential fault analysis of Trivium. In *FSE* (2008), K. Nyberg, Ed., vol. 5086 of *Lecture Notes in Computer Science*, Springer, pp. 158–172.

[60] HOJSÍK, M., AND RUDOLF, B. Floating fault analysis of Trivium. In *INDOCRYPT* (2008), D. R. Chowdhury, V. Rijmen, and A. Das, Eds., vol. 5365 of *Lecture Notes in Computer Science*, Springer, pp. 239–250.

[61] HONG, D., KOO, B., AND SASAKI, Y. Improved preimage attack for 68-step HAS-160. In *ICISC* (2009), D. Lee and S. Hong, Eds., vol. 5984 of *Lecture Notes in Computer Science*, Springer, pp. 332–348.

[62] INTERNET ENGINEERING TASK FORCE. RFC: SM3 hash function, October, 2011. `tools.ietf.org/html/shen-sm3-hash-00`.

[63] IWATA, T., AND KOHNO, T. New security proofs for the 3GPP confidentiality and integrity algorithms. In *FSE* (2004), B. K. Roy and W. Meier, Eds., vol. 3017 of *Lecture Notes in Computer Science*, Springer, pp. 427–445.

[64] JOUX, A. Multicollisions in iterated hash functions. application to cascaded constructions. In *CRYPTO* (2004), M. K. Franklin, Ed., vol. 3152 of *Lecture Notes in Computer Science*, Springer, pp. 306–316.

[65] KELSEY, J., KOHNO, T., AND SCHNEIER, B. Amplified boomerang attacks against reduced-round MARS and Serpent. In *FSE* (2000), B. Schneier, Ed., vol. 1978 of *Lecture Notes in Computer Science*, Springer, pp. 75–93.

[66] KELSEY, J., AND SCHNEIER, B. Second Preimages on n-bit Hash Functions for Much Less than 2^n Work. *IACR Cryptology ePrint Archive 2004* (2004), 304.

[67] KHOVRATOVICH, D. Bicliques for permutations: Collision and preimage attacks in stronger settings. In *ASIACRYPT* (2012), X. Wang and K. Sako, Eds., vol. 7658 of *Lecture Notes in Computer Science*, Springer, pp. 544–561.

[68] KHOVRATOVICH, D., AND NIKOLIĆ, I. Rotational cryptanalysis of ARX. In *FSE* (2010), S. Hong and T. Iwata, Eds., vol. 6147 of *Lecture Notes in Computer Science*, Springer, pp. 333–346.

[69] KHOVRATOVICH, D., NIKOLIĆ, I., AND RECHBERGER, C. Rotational Rebound Attacks on Reduced Skein. In *ASIACRYPT* (2010), M. Abe, Ed., vol. 6477 of *Lecture Notes in Computer Science*, Springer, pp. 1–19.

[70] KIM, J., BIRYUKOV, A., PRENEEL, B., AND HONG, S. On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1. In *SCN* (2006), R. D. Prisco and M. Yung, Eds., vol. 4116 of *Lecture Notes in Computer Science*, Springer, pp. 242–256.

[71] KIRCANSKI, A., ALTAWY, R., AND YOUSSEF, A. M. A heuristic for finding compatible differential paths with application to HAS-160. *Accepted in ASIACRYPT 2013, LNCS, Springer*. Available at: http://eprint.iacr.org/2013/359.

[72] KIRCANSKI, A., SHEN, Y., WANG, G., AND YOUSSEF, A. M. Boomerang and slide-rotational analysis of the SM3 hash function. In *Selected Areas in Cryptography* (2012), L. R. Knudsen and H. Wu, Eds., vol. 7707 of *Lecture Notes in Computer Science*, Springer, pp. 304–320.

[73] KIRCANSKI, A., AND YOUSSEF, A. On the Sliding Property of SNOW 3G and SNOW 3.0. IET Information Security, Vol. 4, Issue 5, pg. 199-206, December 2011.

[74] KIRCANSKI, A., AND YOUSSEF, A. M. Differential Fault Analysis of Rabbit. In *Selected Areas in Cryptography* (2009), M. J. J. Jr., V. Rijmen, and R. Safavi-Naini, Eds., vol. 5867 of *Lecture Notes in Computer Science*, Springer, pp. 197–214.

[75] KIRCANSKI, A., AND YOUSSEF, A. M. Differential Fault Analysis of HC-128. In *AFRICACRYPT* (2010), D. J. Bernstein and T. Lange, Eds., vol. 6055 of *Lecture Notes in Computer Science*, Springer, pp. 261–278.

[76] KIRCANSKI, A., AND YOUSSEF, A. M. On the structural weakness of the GGHN stream cipher. *Cryptography and Communications 2*, 1 (2010), 1–17.

[77] KIRCANSKI, A., AND YOUSSEF, A. M. On the sliding property of SNOW 3G and SNOW 2.0. *IET Information Security 5*, 4 (2011), 199–206.

[78] KIRCANSKI, A., AND YOUSSEF, A. M. On the Sosemanuk Related Key-IV Sets. In *LATINCRYPT* (2012), A. Hevia and G. Neven, Eds., vol. 7533 of *Lecture Notes in Computer Science*, Springer, pp. 271–287.

[79] KNELLWOLF, S., MEIER, W., AND NAYA-PLASENCIA, M. Conditional differential cryptanalysis of trivium and KATAN. In *Selected Areas in Cryptography* (2011), A. Miri and S. Vaudenay, Eds., vol. 7118 of *Lecture Notes in Computer Science*, Springer, pp. 200–212.

[80] KNUDSEN, L. R. Truncated and higher order differentials. In *FSE* (1994), B. Preneel, Ed., vol. 1008 of *Lecture Notes in Computer Science*, Springer, pp. 196–211.

[81] KNUDSEN, L. R., AND RIJMEN, V. Known-key distinguishers for some block ciphers. In *ASIACRYPT* (2007), K. Kurosawa, Ed., vol. 4833 of *Lecture Notes in Computer Science*, Springer, pp. 315–324.

[82] KNUDSEN, L. R., ROBSHAW, M. J. B., AND WAGNER, D. Truncated differentials and skipjack. In *CRYPTO* (1999), M. J. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 165–180.

[83] KOCHER, P. C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO* (1996), N. Koblitz, Ed., vol. 1109 of *Lecture Notes in Computer Science*, Springer, pp. 104–113.

[84] KOCHER, P. C., JAFFE, J., AND JUN, B. Differential power analysis. In *CRYPTO* (1999), M. J. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 388–397.

[85] LAI, X. Higher order derivatives and differential cryptanalysis. In Blahut, R., Costello Jr., D., Maurer, U., Mittelholzer, T. (eds.). *Communications and Cryptography* (1992), 227–233. Kluwer.

[86] LAMBERGER, M., AND MENDEL, F. Higher-order differential attack on reduced SHA-256. *IACR Cryptology ePrint Archive 2011* (2011), 37.

[87] LANGFORD, S. K., AND HELLMAN, M. E. Differential-linear cryptanalysis. In *CRYPTO* (1994), Y. Desmedt, Ed., vol. 839 of *Lecture Notes in Computer Science*, Springer, pp. 17–25.

[88] LEANDER, G., ZENNER, E., AND HAWKES, P. Cache timing analysis of LFSR-based stream ciphers. In *IMA Int. Conf.* (2009), M. G. Parker, Ed., vol. 5921 of *Lecture Notes in Computer Science*, Springer, pp. 433–445.

[89] LEURENT, G. Analysis of differential attacks in ARX constructions. In *ASIACRYPT* (2012), X. Wang and K. Sako, Eds., vol. 7658 of *Lecture Notes in Computer Science*, Springer, pp. 226–243.

[90] LEURENT, G. Construction of differential characteristics in ARX designs - application to Skein. *IACR Cryptology ePrint Archive 2012* (2012), 668.

[91] LEURENT, G. Cryptanalysis of WIDEA. *IACR Cryptology ePrint Archive 2012* (2012), 707.

[92] LEURENT, G., AND ROY, A. Boomerang attacks on hash function using auxiliary differentials. In *CT-RSA* (2012), O. Dunkelman, Ed., vol. 7178 of *Lecture Notes in Computer Science*, Springer, pp. 215–230.

[93] LIN, D., AND JIE, G. Cryptanalysis of Loiss stream cipher. *To appear in: The Computer Journal* (2012). Oxford University Press. Available online: `http://comjnl.oxfordjournals.org/content/early/2012/05/21/comjnl.bxs047.short?rss=1`.

[94] LU, Y., AND DESMEDT, Y. Improved distinguishing attack on Rabbit. In *ISC* (2010), M. Burmester, G. Tsudik, S. S. Magliveras, and I. Ilic, Eds., vol. 6531 of *Lecture Notes in Computer Science*, Springer, pp. 17–23.

[95] MAITRA, S., PAUL, G., RAIZADA, S., SEN, S., AND SENGUPTA, R. Some observations on HC-128. *Des. Codes Cryptography 59*, 1-3 (2011), 231–245.

[96] MATSUI, M. Linear cryptoanalysis method for DES cipher. In *EUROCRYPT* (1993), T. Helleseth, Ed., vol. 765 of *Lecture Notes in Computer Science*, Springer, pp. 386–397.

[97] MAXIMOV, A., AND JOHANSSON, T. Fast computation of large distributions and its cryptographic applications. In *ASIACRYPT* (2005), B. K. Roy, Ed., vol. 3788 of *Lecture Notes in Computer Science*, Springer, pp. 313–332.

[98] MENDEL, F., AND NAD, T. Boomerang distinguisher for the SIMD-512 compression function. In *INDOCRYPT* (2011), D. J. Bernstein and S. Chatterjee, Eds., vol. 7107 of *Lecture Notes in Computer Science*, Springer, pp. 255–269.

[99] MENDEL, F., NAD, T., AND SCHLÄFFER, M. Cryptanalysis of round-reduced HAS-160. In *ICISC* (2011), H. Kim, Ed., vol. 7259 of *Lecture Notes in Computer Science*, Springer, pp. 33–47.

[100] MENDEL, F., NAD, T., AND SCHLÄFFER, M. Finding SHA-2 characteristics: Searching through a minefield of contradictions. In *ASIACRYPT* (2011), D. H. Lee and X. Wang, Eds., vol. 7073 of *Lecture Notes in Computer Science*, Springer, pp. 288–307.

[101] MENDEL, F., NAD, T., AND SCHLÄFFER, M. Collision attacks on the reduced dual-stream hash function RIPEMD-128. In *FSE* (2012), A. Canteaut, Ed., vol. 7549 of *Lecture Notes in Computer Science*, Springer, pp. 226–243.

[102] MENDEL, F., NAD, T., AND SCHLÄFFER, M. Finding collisions for round-reduced SM3. In *CT-RSA* (2013), E. Dawson, Ed., vol. 7779 of *Lecture Notes in Computer Science*, Springer, pp. 174–188.

[103] MENDEL, F., AND RIJMEN, V. Colliding message pair for 53-step HAS-160. In *ICISC* (2007), K.-H. Nam and G. Rhee, Eds., vol. 4817 of *Lecture Notes in Computer Science*, Springer, pp. 324–334.

[104] MENEZES, A., J, VAN OORSCHOT, P., C. AND VANSTONE, S., A. Handbook of Applied Cryptographic Research, CRC Press, 1996.

[105] MITZENMACHER, M., UPFAL, E. Probability and computing, Cambrige University Press, ISBN-10: 0521835402.

[106] MOUHA, N., CANNIÈRE, C. D., INDESTEEGE, S., AND PRENEEL, B. Finding collisions for a 45-step simplified HAS-V. In *WISA* (2009), H. Y. Youm and M. Yung, Eds., vol. 5932 of *Lecture Notes in Computer Science*, Springer, pp. 206–225.

[107] MURPHY, S. The return of the cryptographic boomerang. *IEEE Transactions on Information Theory 57*, 4 (2011), 2517–2521.

[108] NAD, T. The CodingTool Library, 2010. Gratz University of Technology, Available at: http://www.iaik.tugraz.at/content/research/krypto/codingtool/.

[109] NATIONAL BUREAN OF STANDARDS. Data Encryption Standard, U.S. Department of Commerce, Federal Information Processing Standards 46 (1977).

[110] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. Advanced Encryption Standard (AES) (FIPS PUB 197), 2001.

[111] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. FIPS PUB 180-1: Secure Hash Standard. Federal Information Processing Standards Publication 180-1, U.S. Department of Commerce, April 1995. http://www.itl.nist.gov/fipspubs.

[112] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. FIPS PUB 180-2: Secure Hash Standard. Federal Information Processing Standards Publication 180-2, U.S. Department of Commerce, August 2002. http://www.itl.nist.gov/fipspubs.

[113] OSVIK, D. A., SHAMIR, A., AND TROMER, E. Cache attacks and countermeasures: The case of AES. In *CT-RSA* (2006), D. Pointcheval, Ed., vol. 3860 of *Lecture Notes in Computer Science*, Springer, pp. 1–20.

[114] OSWALD, E. Enhancing simple power-analysis attacks on elliptic curve cryptosystems. In *CHES* (2002), B. S. K. Jr., Çetin Kaya Koç, and C. Paar, Eds., vol. 2523 of *Lecture Notes in Computer Science*, Springer, pp. 82–97.

[115] PAUL, O. S., CROWLEY, P., AND (M, Q. Truncated differential cryptanalysis of five rounds. In *Salsa20, in Workshop Record of SASC 2006: Stream Ciphers Revisted, eSTREAM technical report 2005/073 (2005). URL: http://www.ecrypt.eu.org/stream/papers.html* (2006).

[116] PEYRIN, T. Analyse de fonctions de hachage cryptographes. Ph.D. Thesis, University of Versailles, 2008., http://www.iacr.org/phds/?p=detail&entry=500.

[117] PEYRIN, T. Improved differential attacks for ECHO and Grøstl. In *CRYPTO* (2010), T. Rabin, Ed., vol. 6223 of *Lecture Notes in Computer Science*, Springer, pp. 370–392.

[118] PRIEMUTH-SCHMID, D., AND BIRYUKOV, A. Slid pairs in Salsa20 and Trivium. In *INDOCRYPT* (2008), D. R. Chowdhury, V. Rijmen, and A. Das, Eds., vol. 5365 of *Lecture Notes in Computer Science*, Springer, pp. 1–14.

[119] RON RIVEST. The MD5 Message Digest Algorithm, RFC 1321, Internet Activities Board, Intenet Privacy Task Force, April 1992. `www.ietf.org/rfc/rfc1321.txt`.

[120] SALEHANI, Y. E., KIRCANSKI, A., AND YOUSSEF, A. M. Differential Fault Analysis of Sosemanuk. In *AFRICACRYPT* (2011), A. Nitaj and D. Pointcheval, Eds., vol. 6737 of *Lecture Notes in Computer Science*, Springer, pp. 316–331.

[121] SASAKI, Y. Boomerang distinguishers on MD4-family: First practical results on full 5-pass HAVAL. In *Selected Areas in Cryptography* (2011), A. Miri and S. Vaudenay, Eds., vol. 7118 of *Lecture Notes in Computer Science*, Springer, pp. 1–18.

[122] SASAKI, Y., AND AOKI, K. A preimage attack for 52-step HAS-160. In *ICISC* (2008), P. J. Lee and J. H. Cheon, Eds., vol. 5461 of *Lecture Notes in Computer Science*, Springer, pp. 302–317.

[123] SASAKI, Y., AND WANG, L. Distinguishers beyond three rounds of the RIPEMD-128/-160 compression functions. In *ACNS* (2012), F. Bao, P. Samarati, and J. Zhou, Eds., vol. 7341 of *Lecture Notes in Computer Science*, Springer, pp. 275–292.

[124] SASAKI, Y., WANG, L., TAKASAKI, Y., SAKIYAMA, K., AND OHTA, K. Boomerang distinguishers for full HAS-160 compression function. In *IWSEC* (2012), G. Hanaoka and T. Yamauchi, Eds., vol. 7631 of *Lecture Notes in Computer Science*, Springer, pp. 156–169.

[125] SCHLÄFFER, M., AND OSWALD, E. Searching for differential paths in MD4. In *FSE* (2006), M. J. B. Robshaw, Ed., vol. 4047 of *Lecture Notes in Computer Science*, Springer, pp. 242–261.

[126] SKOROBOGATOV, S. P., AND ANDERSON, R. J. Optical fault induction attacks. In *CHES* (2002), B. S. K. Jr., Çetin Kaya Koç, and C. Paar, Eds., vol. 2523 of *Lecture Notes in Computer Science*, Springer, pp. 2–12.

[127] STAFFELBACH, O., AND MEIER, W. Cryptographic significance of the carry for ciphers based on integer addition. In *CRYPTO* (1990), A. Menezes and S. A. Vanstone, Eds., vol. 537 of *Lecture Notes in Computer Science*, Springer, pp. 601–614.

[128] STEVENS, M., LENSTRA, A. K., AND DE WEGER, B. Chosen-prefix collisions for MD5 and colliding x.509 certificates for different identities. In *EUROCRYPT* (2007), M. Naor, Ed., vol. 4515 of *Lecture Notes in Computer Science*, Springer, pp. 1–22.

[129] WAGNER, D. The boomerang attack. In *FSE* (1999), L. R. Knudsen, Ed., vol. 1636 of *Lecture Notes in Computer Science*, Springer, pp. 156–170.

[130] WANG, X., YIN, Y. L., AND YU, H. Finding collisions in the full SHA-1. In *CRYPTO* (2005), V. Shoup, Ed., vol. 3621 of *Lecture Notes in Computer Science*, Springer, pp. 17–36.

[131] WANG, X., AND YU, H. How to break MD5 and other hash functions. In *EUROCRYPT* (2005), R. Cramer, Ed., vol. 3494 of *Lecture Notes in Computer Science*, Springer, pp. 19–35.

[132] WATANABE, D., FURUYA, S., YOSHIDA, H., TAKARAGI, K., AND PRENEEL, B. A new keystream generator MUGI. *IEICE Transactions 87-A*, 1 (2004), 37–45.

[133] WU, H. A new stream cipher HC-256. In *FSE* (2004), B. K. Roy and W. Meier, Eds., vol. 3017 of *Lecture Notes in Computer Science*, Springer, pp. 226–244.

[134] WU, H. The stream cipher HC-128. In *The eSTREAM Finalists*, M. J. B. Robshaw and O. Billet, Eds., vol. 4986 of *Lecture Notes in Computer Science*. Springer, 2008, pp. 39–47.

[135] YOSHIDA, H., AND BIRYUKOV, A. Analysis of a SHA-256 variant. In *Selected Areas in Cryptography* (2005), B. Preneel and S. E. Tavares, Eds., vol. 3897 of *Lecture Notes in Computer Science*, Springer, pp. 245–260.

[136] YU, H., CHEN, J., AND WANG, X. The boomerang attacks on the round-reduced Skein-512. In *Selected Areas in Cryptography* (2012), L. R. Knudsen and H. Wu, Eds., vol. 7707 of *Lecture Notes in Computer Science*, Springer, pp. 287–303.

[137] YUN, A., SUNG, S. H., PARK, S., CHANG, D., HONG, S., AND CHO, H.-S. Finding collision on 45-step HAS-160. In *ICISC* (2005), D. Won and S. Kim, Eds., vol. 3935 of *Lecture Notes in Computer Science*, Springer, pp. 146–155.

[138] ZENNER, E. A cache timing analysis of HC-256. In *Selected Areas in Cryptography* (2008), R. M. Avanzi, L. Keliher, and F. Sica, Eds., vol. 5381 of *Lecture Notes in Computer Science*, Springer, pp. 199–213.