Cryptanalysis and Design of Iterated Hash Functions

by

Norbert Pramstaller

A PhD Thesis
Presented to the Faculty of Computer Science in Partial Fulfillment of the Requirements for the PhD Degree

Assessors

Prof. Dr. Ir. Vincent Rijmen (TU Graz, Austria)
Prof. Dr. Ir. Bart Preneel (KU Leuven, Belgium)

November 2007

Institute for Applied Information Processing and Communications (IAIK)
Faculty of Computer Science
Graz University of Technology, Austria

This work is dedicated to my family:
Hermann, Luisa,
Sonja, Sylvia,
Artur, Christian,
Julia, Jana, Sarah, and Hannah.

Success is not the key to happiness.
Happiness is the key to success.
If you love what you are doing,
you will be successful.
Albert Schweitzer (1875-1965)

Abstract

Cryptographic hash functions play a fundamental role in cryptology. In the past years, much progress has been made in the cryptanalysis of hash functions. As a consequence, most of the hash functions in use today show weaknesses or have been broken. Motivated by these recent developments, we focus in this thesis on the analysis and design of cryptographic hash functions.

We review generic attack methods on iterated hash functions that show limitations of the popular Merkle-Damgård design principle. The collision attack on the widely used SHA-1 has attracted most attention in academia, industry, and governmental organizations. Firstly, because it was for a long time believed that SHA-1 is secure, and secondly, because different parts of the attack on SHA-1 were initially not well understood. Therefore, we review different strategies that led to the collision attack on SHA-1 and explain and refine several of them. Amongst others, we show how to search for linear characteristics by exploiting methods from coding theory, and we introduce an analytical way to determine the probability of such characteristics thoroughly.

The analysis of hash functions mostly focuses on collision attacks and only few results with respect to (second) preimage attacks have been published. We discuss two new hash function proposals, for which we can construct second preimages efficiently. Motivated by these results, we also look at the implications of both collision and second preimage attacks on constructions such as the hash-based message authentication codes NMAC and HMAC. It turns out that the cryptanalytic results on hash functions do affect such constructions. The implications are mainly of theoretical nature, since the attack complexities are far from being practical. Nevertheless, they show the importance of considering the security of hash-based constructions too.

The progress in the cryptanalysis of hash functions also motivated researchers to look at new designs and new design strategies for hash functions. Several new proposals have been published and the majority of them has been broken shortly after publication. For some selected design principles for hash functions, we discuss pros and cons and argue about possible strategies in designing new hash functions. Also a new block-cipher-based hash function proposal is presented. We discuss open problems and further research directions in the analysis and design of cryptographic hash functions.

Zusammenfassung

Kryptografische Hash-Funktionen sind ein fundamentaler Bestandteil der modernen Kryptografie. In den letzten Jahren hat die Kryptanalyse von Hash-Funktionen einen enormen Fortschritt erlebt. Ein teils überraschendes Resultat ist, dass der Großteil der Hash-Funktionen entweder gebrochen wurde oder es wurden Schwachstellen der Algorithmen aufgezeigt. Aufgrund dieser Resultate, beschäftigt sich diese Dissertation mit der Analyse und dem Entwurf kryptografischer Hash-Funktionen.

Wir beschreiben generische Attacken auf Hash-Funktionen, wodurch die technischen Grenzen des weit verbreiteten Merkle-Damgård-Designprinzips hervorgehoben werden. Die Kollisionsattacke auf SHA-1, die am meisten eingesetzte Hash-Funktion, hat in Forschung, Industrie und in staatlichen Organisationen erhebliches Aufsehen erregt. Wir beschreiben die verschiedenen Schritte dieser Attacke und betrachten einige davon im Detail. Wir erläutern das Finden von linearen Charakteristiken basierend auf Methoden der Kodierungstheorie sowie auch die detaillierte Berechnung der Wahrscheinlichkeit für das Auftreten solcher Charakteristiken.

Die Kryptanalyse hat sich bis heute fast ausschließlich auf Kollisionsattacken konzentriert und nur wenige Resultate in Bezug auf Second-Preimage-Attacken wurden publiziert. Wir besprechen zwei neue Designs von Hash-Funktionen und zeigen effiziente Second-Preimage-Attacken. Motiviert durch die Resultate in Bezug auf Kollisionsattacken und Second-Preimage-Attacken, analysieren wir auch den Einfluss dieser Resultate auf Applikationen, wie zum Beispiel die Message-Authentication-Codes NMAC und HMAC.

Der Fortschritt in der Kryptanalyse von Hash-Funktionen hat dazu geführt, dass sich heute viele Forschungsgruppen neben der Analyse auch mit dem Design von neuen Hash-Funktionen beschäftigen. Mehrere neue Designs wurden in den letzten Jahren präsentiert, allerdings wurde der Großteil dieser Hash-Funktionen bereits kurz nach ihrer Veröffentlichung gebrochen. Wir beschreiben ausgewählte Designvorschläge und präsentieren, basierend auf den abgeleiteten Vor- und Nachteilen, mögliche Richtlinien für das Design neuer Hash-Funktionen. Auch stellen wir eine neue Hash-Funktion vor, die auf der bekannten Blockchiffre Advanced Encryption Standard (AES) beruht. Als Abschluss dieser Dissertation geben wir einen Ausblick auf offene Fragen und neue Forschungsschwerpunkte in Bezug auf die Analyse und das Design von kryptografischen Hash-Funktionen.

Acknowledgements

I would like to start the acknowledgements with the most important person during my PhD studies: Professor Vincent Rijmen. "Vincent, without you I would have never been able to reach this important goal in my life. Thank you!" I was lucky having the chance to work together with, and to learn from, one of the most famous and brilliant researchers in cryptography. Besides the scientific guidance throughout the past years, I would like to thank Vincent for being not only my professor but also for being a very good friend. Thank you, for having had time also outside university and thank you for listening to me with respect to personal, life-related matters. To cut a long story short: thank you for everything!

I would like to thank Professor Bart Preneel for being my external reviewer and examiner. His valuable comments and suggestions helped to improve the quality of this thesis.

Special thanks go to the IAIK Krypto Group. Shortly after Vincent launched this new group at IAIK, it was already clear that we will become more a 'research family' than an ordinary research group. To date, we are a well-rehearsed successful team and we really enjoy doing research together. Therefore, thank you Mario Lamberger, Florian Mendel, Christian Rechberger, Martin Schläffer, and Michaela Tretter-Dragovic. I would also like to thank Christophe De Cannière who joined our group for one year and also Jorge Munilla, who was guest researcher for three months. In particular, I would like to thank Mario Lamberger and Florian Mendel who invested their valuable time for proofreading my thesis and for helpful discussions on it. Thank you Mario for shedding light on mathematical subjects and for remaining patient while explaining them to me.

Before I will thank my other colleagues and friends, I would like to devote some words to my family: my parents Hermann and Luisa, my sisters Sonja and Sylvia, my brothers in law Artur Mair and Christian Gasser, and my four nieces Julia, Jana, Sarah, and Hannah. Having such a family makes it almost impossible to not succeed with your plans and to remain at the right path in life and work. Thank you for your support and for being there for me whenever I needed you. Without you, I would not be who I am and I would have never reached my goal. My family is really the best one to have!

Thank you to all the people working at IAIK. In particular, I would like to thank Manfred Aigner, Stefan Mangard, and Elisabeth Oswald. Doing research together with them was a great experience and I was able to learn a lot from them. Special thanks go also to Professor Karl Christian Posch. He actually brought me into IAIK and guided me through my Master studies. Manfred,

Stefan, Elisabeth, and Karl Christian also supported me in making my decision to jump at the chance for moving from hardware to the amazing subject of cryptography. Today, I am fully convinced of having made the right decision.

I would also like to thank my coauthors of articles that have been published during my studies: Sandra Dominikus, Florian Mendel, Stefan Mangard, Mario Lamberger, Elisabeth Oswald, Christian Rechberger, Vincent Rijmen, and Johannes Wolkerstorfer (TU Graz), Krystian Matusiewicz and Josef Pieprzyk (Macquarie University Australia), Frank K. Gürkaynak, Simon Häne, Hubert Kaeslin, Norbert Felber, and Wolfgang Fichtner (ETH Zurich), Steve Babbage (Vodafon Group R&D Newbury), Carlos Cid (Royal Holloway, University of London), and Håvard Raddum (University of Bergen). It was a pleasure and an honor to work together with you.

Before I go over to thank my friends, I would still like to mention the European project ECRYPT. This project (and of course my Professor and my department) made it possible to attend two summer schools in Samos (a Greek island!)—the first at the beginning, and the second close to the end of my PhD studies. Starting a PhD with such an event is the best that can happen. From the first step on you know important people in your field and you can easily become a member of an amazing research community.

It is important for me to thank all my friends for their support and simply for being my friends. I would like to start with Thomas Edlinger, Martina Ziesler, Michaela Jahn, Isabelle Van Nieuwenhuyse, Arne Tauber, Michaela Tretter-Dragovic, Stefan Mangard, Mario Lamberger, Barbara Angermeier, Christian Mudrak, Alois and Anneliese Jahn, and Monika Edlinger. It is really good to have such friends. Whenever I needed their help, they were there for me and we had a very nice time together during the past years—thank you! Special thanks go also to my Italian friends: Hanspeter Harpf, Christof Preindl, Lukas Rungger, Federica Battisti, Michela Cancellaro, Anna Maria Vegni, Germana Nagler, Gernot Mussner, and Günter Comploj. Thanks go also to my friends in Belgium: Willem Zuttermann, Wouter Dufoort, Steven Holderik, Christophe Capoen, Isolde Verbeke, Isabelle and Marno Verhoye, Isablle Van Nieuwenhuyse, Jan and Martene Van Nieuwenhuyse, Brecht Wyseur, and Joseph Lano. They gave me the chance to see Belgium from a local's point of view and they introduced me to Belgian mentality and Belgian life—a wonderful experience I will never forget.

I emphasize that this is not a comprehensive list of all those people that have been part of my path and brought me to this point in my life. For those I forgot: thank you too and my apologizes for not having you mentioned here! When I was writing the acknowledgements, I first wanted to devote few sentences to each of you. I immediately noticed that this would have led to another long chapter in this thesis. Therefore, I decided to thank you all personally when I see you the next time. I conclude my acknowledgements with saying once again: thank you all!

Norbert Pramstaller
Graz, November 2007

Contents

List of Tables

List of Figures

List of Notations

List of Abbreviations

MAC	Message Authentication Code
AES	Advanced Encryption Standard
DES	Data Encryption Standard
DBL	Double-block-length hash construction
SBL	Single-block-length hash construction
MD	Merkle-Damgård
DM	Davies-Meyer mode of operation
MP	Miyaguchi-Preneel mode of operation
MMO	Matyas-Meyer-Oseas mode of operation
ECB	Electronic Codebook mode
CBC	Cipher Block Chaining mode
CTR	Counter mode
HMAC	Hash-based MAC
NMAC	Nested MAC construction
CBC-MAC	MAC based on block cipher operating in CBC mode

List of Mathematical Symbols

$\mathbf{M}$	matrix
$\mathbf{M}^T$	transpose of matrix $\mathbf{M}$
$\mathbf{M}_{k \times n}$	matrix $\mathbf{M}$ with k rows and n columns
$\mathbf{I}_k$	$k \times k$ identity matrix
$rank(\mathbf{M})$	the rank of matrix $\mathbf{M}$
$\mathbf{v}$	row vector $\mathbf{v} = \{v_1, \ldots, v_n\}$
$\mathbf{v}^T$	transpose of $\mathbf{v}$ represented as column vector
$GF(q)$	finite field (Galois field) with q elements
$a \oplus b$	exclusive-or (XOR) of a and b
$a \boxplus b$	modular addition of a and b
$a + b$	integer addition
$a \| b$	concatenation of two strings (vectors)
$\|a\|$	bit length of variable a
$GCD(a, b)$	greatest common divisor of a and b

There are two things
to aim at in life:
first to get what you want;
and, after that, to enjoy it.
Only the wisest of mankind
achieve the second.

Logan Pearsall Smith
(1865-1946)

1 Introduction

In the last decades, information technology has become more and more part of our daily lives and can be found in the majority of applications we are using to date. Just to mention few of them: the mobile systems GSM and UMTS, e-banking, e-government, and wireless internet. These ubiquitous techniques do not only support and ease our daily lives but are also open to abuse. Therefore, securing information systems is as important as the systems themselves. *Cryptology* is the science that studies this problem. A popular and common reference treating all different aspects of this field is the book by Menezes *et al.* [116].

The ISO 7498-2 standard [47] distinguishes between five types of security services: data confidentiality, data integrity, authentication, access control, and non-repudiation. Standardized are also 13 types of security mechanisms, which are implemented by using cryptographic primitives. At the highest level, we distinguish between *asymmetric primitives* and *symmetric primitives*. Most asymmetric primitives can be used to provide data confidentiality by means of asymmetric encryption and data integrity services by digital signature schemes.

1.1 Symmetric Primitives

Symmetric primitives can be used to provide confidentiality and data integrity services. There are four types of symmetric primitives: block ciphers, stream ciphers, hash functions, and message authentication codes. Confidentiality is usually provided by means of encryption, which is typically implemented by using block ciphers or stream ciphers. Data integrity mechanisms are usually implemented by cryptographic hash functions or message authentication codes.

1.1.1 Cryptographic Hash Functions

Since we focus on cryptographic hash functions in this thesis, we informally describe this primitive. A cryptographic hash function is an algorithm that maps a message string of arbitrary length to a string of fixed length, called hash value. Hash functions are used as an elementary building block in a large variety of cryptographic systems, but their primary use is to compute fixed-length 'fingerprints' of messages of arbitrary length in such a way that it is difficult to find two

messages that lead to the same fingerprint. An important class of applications where this property is useful are for instance digital signature schemes. Instead of signing the entire message, only the typically much shorter fingerprint of this message needs to be signed. This significantly improves the performance of such schemes.

1.1.2 Cryptanalysis

The security of symmetric primitives can in general not be proven and the trust basically relies on a thorough security analysis over the years. Analyzing the security of these primitives is called *cryptanalysis*. In the past 20 years, a lot of effort has been invested in the cryptanalysis of block ciphers, stream ciphers, and message authentication codes. The analysis of cryptographic hash functions has drawn similar attention in the cryptographic community. For instance, Preneel analyzed numerous hash functions in his PhD thesis [143] and presented attacks for the majority of them.

Let us consider for instance the hash function MD4 proposed by Rivest in 1990. Shortly after MD4 was published, weaknesses have been presented by den Boer and Bosselaers in [45]. Rivest, then proposed a strengthened variant of MD4 in 1991, namely MD5. However, also for MD5 weaknesses have been shown by den Boer and Bosselaers in [46]. Dobbertin presented cryptanalytic results for both MD4 [49, 51] and MD5 [50]. NIST has published several hash functions known as the SHA family. The first hash function was SHA-0 (formerly called the Secure Hash Algorithm) proposed in 1993. NIST replaced SHA-0 by SHA-1 in 1995. In 1998, Chabaud and Joux [31] analyzed the hash function SHA-0. Their results motivated more researchers to look at hash functions. In 2004, a Chinese research team led by Prof. Wang announced at the CRYPTO 2004 rump session that they have broken, amongst others, the most widely employed hash functions MD5 and SHA-1 [172, 173]. These announcements were surprising and unsettling for both academia and industry. After these breakthrough results, many researchers devoted their time to the cryptanalysis of hash functions and also new ideas for designing hash functions have been presented. The main focus was on the hash function as a primitive and only recently also other primitives such as message authentication codes derived from these hash functions have been analyzed. The progress in the cryptanalysis of hash functions does also impact such constructions. However, most attacks on hash-based constructions are still of theoretic nature since the attack complexities are far beyond being practical. Nevertheless, counterexamples demonstrate the importance of considering also the impact on hash-based constructions and applications. For instance, Stevens *et al.* have presented two colliding X.509 certificates for different identities in [159] based on the collision attacks on MD5.

It is surprising how much progress in the cryptanalysis of hash functions has been made during the last couple of years. Nevertheless, the analysis and design of cryptographic hash functions is still an open research problem, and we can expect several new directions in this field in the forthcoming years.

1.2 Outline of this Thesis and Main Contribution

In this thesis, we focus on the analysis and design of cryptographic hash functions. In the following, we give an outline of each chapter including the main contributions.

In Chapter 2, we treat the preliminaries that are needed for the discussed topics in the remainder of this thesis. We will describe symmetric primitives such as block ciphers, hash functions, and message authentication codes and look at the differential properties of the Boolean functions used in the hash function SHA-1. An introduction to finite fields and linear codes concludes this chapter.

A general overview of cryptanalytic techniques for hash functions is presented in Chapter 3. We focus on collision, second preimage, and preimage attacks on iterated hash functions. We discuss birthday attacks and shortcut attacks. We review generic attacks that exploit the structure of the Merkle-Damgård design principle. This overview of attack strategies will be further extended in the according chapters.

In Chapter 4, we focus on collision attacks on SHA-1. We review existing results in a chronological order and explain the main strategy that led to the first collision attack on SHA-1. We introduce how one can exploit coding theory for finding high-probability linear characteristics, which is a crucial part of the collision attack. Furthermore, we refine the probability estimates of these linear characteristics by including side-effects such as carries in modular additions. As a result, we present an analytical method to determine the probability of linear characteristics for SHA-1. The results of this chapter have been published in [114, 138].

Chapter 5, is devoted to second preimage attacks on hash functions. We discuss and analyze a new hash function design strategy called SMASH. As a result of our analysis, we show how one can construct collisions. This collision attack is then further generalized resulting in a second preimage attack. We also discuss a new hash function construction that uses a block cipher as underlying primitive. We show that the choice of the underlying block cipher is crucial for the security of this scheme. For a block cipher following the FX construction, we can construct second preimages efficiently. The results of this chapter have been published in [95, 136, 141].

In Chapter 6, we focus on the impact of recent cryptanalytic results on hash functions on hash-based message authentication codes. We first review and discuss results of the impact of collision attacks and go then over to the impact of second preimage attacks. Firstly, we introduce a new notation for iterated hash functions that is similar to the notion of second preimages: *b-block bypass*. We show that for the hash functions discussed in Chapter 5 this new notion applies. Secondly, we look at the implications on hash-based message authentication codes if such hash functions are employed. The results of this chapter have been published in [136].

We discuss the design of cryptographic hash functions in Chapter 7. We

look at proposed modifications for SHA-1 and discuss a new design principle. We also propose a new block-cipher-based hash function and give a security analysis. This new proposal has been presented in [110]. We conclude this chapter with a summary and a discussion on the design of cryptographic hash functions.

In Chapter 8, we present a summary and conclude this thesis by discussing open problems and future research directions.

As indicated above, the main results in this thesis have been published in [95, 110, 114, 136, 138, 141]. Other results that focus on hash functions, stream ciphers, and the hardware implementation of cryptographic primitives, published during my studies, can be found in [2, 103, 107, 111, 112, 113, 130, 135, 137, 139, 140, 142].

2 Preliminaries

In this chapter, we treat the preliminaries that are needed for the topics we will discuss in the remainder of this thesis. We will start with giving the notation that we follow throughout the thesis. Then, we present the basics of cryptographic primitives such as block ciphers, hash functions, and message authentication codes. Furthermore, we will discuss the differential properties of Boolean functions and modular additions that are used in the hash function SHA-1. We also give a short introduction to finite fields and linear codes.

2.1 Notation

For the concatenation of two variables we write $a \| b$. Addition modulo 2 (XOR) is denoted by $a \oplus b$, modular addition is denoted by $a \boxplus b$. The operator '+' denotes addition of arbitrary integers. The bit length of a variable a is denoted by $|a|$. We use two different kinds of indexing. Superscripts denote different (independent) objects. For instance, different messages are denoted by m^i and m^j with $i \neq j$. To denote parts of these objects we use subscripts. For instance a message m consisting of t blocks is denoted by $m = m_1 \| \ldots \| m_t$. Note that for messages, we always assume that each message block m_i is of the same length. We use the concatenation operator to denote messages consisting of a number of blocks, since it is convenient for iterated hash functions.

Variables that belong to the internal structure of a cryptographic primitive are denoted by capital letters. Subscripts of these variables denote the appearance of the variable in a certain iteration, *e.g.* A_i means the value of variable A in iteration i. If we want to index a certain bit of this variable, we use an additional subscript: $A_{i,j}$ denotes the bit in position j of variable A in iteration i. For representing an n-bit binary variable, we use the big-endian notation, *i.e.* on the right we write the least significant bit (LSB), and on the left we write the most significant bit (MSB). For instance, storing the decimal value 1027 $(= 2^{10} + 2^1 + 2^0)$ into a 32-bit register is represented in hexadecimal notation as `00000403`. Rotation of a variable A is denoted by $\lll$ (left rotation) or $\ggg$ (right rotation). Rotation can also be denoted by accordingly setting the subscript. For instance, a rotation by 5 positions to the left (right) of an n-bit variable A in

iteration i is denoted by $A_{i,(j+5) \bmod n}$ ($A_{i,(j-5) \bmod n}$) with $0 \leq j < n$. For the sake of readability, we do not explicitly write the modulo operator—it is always clear from the context that rotations are performed modulo the bit-length of the variable.

2.2 Block Ciphers

Block ciphers are the most versatile symmetric cryptographic primitives since they can be used to provide data confidentiality, data integrity, and authentication services. These are three out of the five security services standardized by ISO 7498-2 [47]. Another reason for their versatility is that other symmetric primitives can be constructed from block ciphers as illustrated in Figure 2.1.

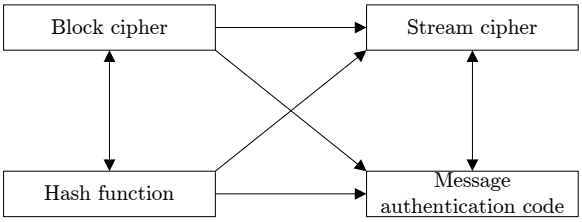

Figure 2.1: The four symmetric cryptographic primitives: block ciphers, stream ciphers, hash functions, and message authentication codes. The arrows denote which primitive can be constructed from the other.

A specific n-bit block cipher with a k-bit key can be considered formally as a family of 2^k permutations on n bits. Every key defines a different permutation $\{0,1\}^n \rightarrow \{0,1\}^n$. Therefore, what we expect from a 'good' block cipher is that it should behave as a good *pseudo-random permutation* [10]. We denote by $E_k(m)$ the encryption of the n-bit input block m under the key k. The inverse operation, decryption, is denoted by $E_k^{-1}(m)$. If we refer to a specific block cipher then we use the abbreviation of the block cipher instead of the letter E, for instance, we write $\mathrm{AES}_k(m)$ if we mean encryption of the message m with AES under the key k.

The two most widely used block ciphers to date are the Data Encryption Standard DES [121] (and variants such as 3-DES [116]) and its successor the Advanced Encryption Standard AES [122]. In Table 2.1, we list the block length and key lengths specified for DES and AES.

The obvious drawback of DES is its short-key size. However, many other weaknesses are known. One of the first weak properties of DES that has been observed by Hellman *et al.* [67] is the *complementation property*: $y = DES_k(m) \Leftrightarrow \overline{y} = DES_{\overline{k}}(\overline{m})$, where the bar denotes the bitwise complement. The complementation property reduces the key space by a factor of 2. Also other weaknesses are known. For instance, for DES there exist four *weak keys* (encryption equals decryption) and six additional pairs of *semi-weak keys* (see for instance Moore

Table 2.1: Block length and key length for DES and AES.

	block length in bits	key length in bits
DES	64	56
AES-128	128	128
AES-192	128	192
AES-256	128	256

and Simmons [119]). These properties are due to the interaction of the key schedule and cipher structure. DES also succumbs to key-recovery attacks that are faster than brute-force key search by applying differential (*e.g.* Biham and Shamir [20]) and linear cryptanalysis (*e.g.* Matsui [104]). For AES no such, or any equivalent properties are known to date. AES is also resistant against differential and linear cryptanalysis. Therefore, it is widely believed that AES behaves as a good pseudo-random permutation, a good block cipher.

In [123], several modes of operation for block ciphers have been standardized. The simplest mode of operation is the *electronic codebook* mode (ECB). There the input message (plaintext) is split into n-bit blocks and then each block m_i is encrypted under the key k resulting in the ciphertext $c_i = E_k(m_i)$. It is easy to see that this mode has some shortcomings. For instance, a repeated input message block leads to a repeated ciphertext: $m_i = m_j \Leftrightarrow c_i = c_j$ for $i \neq j$. To overcome the drawback with repeating patterns, one can use the *cipher block chaining* mode (CBC). There feedback between the current and previous iteration is introduced to avoid that a repeating input block results in a repeating pattern in the ciphertext: $c_i = E_k(m_i \oplus c_{i-1})$. For processing the first input block an n-bit initial value c_0 is defined. This value does not need to be secret but should not be predictable by an attacker. Note that after encrypting $2^{n/2}$ message blocks under the same key, the CBC mode starts leaking information similar to the ECB mode. Another mode of operation is the *counter* mode (CTR). In this mode of operation, an incrementing counter value ctr_i of n bits is used in each iteration as the input message block for the block cipher, and the message block is added (XORed) to the output of the block cipher resulting in the ciphertext $c_i = E_k(\mathrm{ctr}_i) \oplus m_i$. Also the counter mode starts leaking information after encrypting $2^{n/2}$ message blocks under the same key.

2.3 Cryptographic Hash Functions

A cryptographic hash function H is an algorithm that maps a message string m of arbitrary length to a string $h = H(m)$ of fixed length n, called hash value. Hash functions are used as an elementary building block in a large variety of cryptographic systems, but their primary use is to compute n-bit 'fingerprints' of messages of arbitrary length in such a way that it is difficult to find two messages that lead to the same fingerprint. An important class of applications where this

property is useful are for instance digital signature schemes [116, Chapter 11]: if $H(m)$ can be considered to be a unique representation of m, then, instead of the complete message, it suffices to sign this compact hash value.

It is clear, however, that if more than 2^n different messages are hashed, at least one pair of messages will have to share the same hash value. Hence, the purpose of a hash function is not to prevent the existence of such colliding messages (they are unavoidable), but to ensure that it would take an infeasible effort to find them. We make this more precise in the next sections.

2.3.1 Iterated Hash Functions

Most hash functions in use today are designed following the Merkle-Damgård design principle [41, 118]. The idea is to split the input message m into ℓ-bit blocks, which are then processed one after another by iterating a compression function f. Messages whose length is not a multiple of ℓ bits need to be padded first by applying an unambiguous padding method.

More formally, let $H : \{0,1\}^* \to \{0,1\}^n$ be an iterated hash function, based on a *compression function* $f : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$, and let $m = m_1 \| \ldots \| m_t$ be a (padded) message consisting of t blocks of ℓ bits each. The hash value is then computed as shown in (2.1).

$$\begin{aligned} h_0 &= iv \\ h_i &= f(h_{i-1}, m_i) \quad \text{for } i = 1, \ldots, t \\ h_{t+1} &= g(h_t) \end{aligned} \tag{2.1}$$

The n-bit variable h_i is called the intermediate chaining value. We call h_{t+1} the final hash value and h_0 the initial value, which is initialized using a predefined n-bit value iv. The function g is called the output transformation. In most hash function designs, the output transformation is simply the identity mapping, *i.e.* we have $h_{t+1} = h_t$. To denote the application of the hash function H to a t-block message, we write $h = H(m) = H(iv, m) = H(iv, m_1 \| \ldots \| m_t)$.

The purpose of the Merkle-Damgård construction is to extend the collision resistance of a compression function to a hash function which accepts messages of arbitrary length. In order to achieve this, messages are preprocessed using a technique called Merkle-Damgård (MD) strengthening. MD strengthening specifies an unambiguous padding method which includes the binary representation of the message length and fixes a predefined value of iv. This also prevents trivial attacks such as long-message attacks and fixed-point attacks. We give a more detailed description of these attacks in Section 3.2.2.

We still mention a specific property of iterated hash functions, namely the *extension property*. Assume we are given two colliding t-block messages m and m^*. Then we can add any binary string e to both messages, $m \| e$ and $m^* \| e$, without destroying the collision: both $H(m \| e)$ and $H(m^* \| e)$ are identical. If we consider these two extended messages, $m \| e$ and $m^* \| e$, then we say that an *internal collision* has occurred after processing the two t-block messages m and m^*.

2.3.2 Security Requirements

The security properties that hash functions are expected to provide, are informally summarized in the following three requirements:

- *Collision resistance:* it is practically infeasible to find two messages m and m^*, with $m^* \neq m$, such that $H(m) = H(m^*)$.
- *Second preimage resistance:* for a given message m, it is practically infeasible to find a second message $m^* \neq m$ such that $H(m) = H(m^*)$.
- *Preimage resistance:* for a given hash value h, it is practically infeasible to find a message m such that $H(m) = h$.

The second requirement is a weaker variant of the first, but suffices, for instance, in applications where the adversary has no control whatsoever over the data that will be hashed. The third requirement is important in public-key signature schemes, in order to prevent adversaries from constructing valid (albeit meaningless) messages for arbitrary signatures.

The resistance of a hash function to collision and (second) preimage attacks depends in the first place on the length n of the hash value. Regardless of how a hash function is designed, an adversary will always be able to find preimages or second preimages after trying out about 2^n different messages. Finding collisions requires a much smaller number of trials: about $2^{n/2}$, as we will see in Section 3.1.1. As a result, hash functions producing less than 160 bits of output are currently considered inherently insecure. Moreover, if the internal structure of a particular hash function allows collisions or preimages to be found more efficiently than what could be expected based on its hash length, then the function is considered to be broken.

2.3.3 Different Types of Collisions

As described in the previous section, one of the security requirements of a cryptographic hash function is collision resistance. Finding two different messages that have the same hash value with a complexity below the $2^{n/2}$ constitutes a collision attack. Nevertheless, there exist also certificational weaknesses of hash functions. Certificational, since the property itself does not render a hash function insecure. However, as we will see later on in this thesis, this kind of certificational weaknesses can, in the worst case, be exploited for attacking hash functions (see for instance also [143]). We give an informal definition of two different types of collisions.

Definition 2.1. *(*Pseudo-Collision*) A pseudo-collision for the hash function H has one of the following properties:*

$$H(iv, m) = H(iv^*, m^*) \text{ or}$$
$$H(iv, m) = H(iv^*, m) \ ,$$

where $m \neq m^*$ and $iv \neq iv^*$.

Definition 2.2. *(*Near-Collision*) A near-collision for the hash function H has the following property:*

$$H(iv, m) \oplus H(iv, m^*) = \delta \ ,$$

where $m \neq m^*$, and δ is a non-zero value with low Hamming weight.

So far, we did not specify the non-zero values δ and $iv' = iv \oplus iv^*$. If we allow a δ with an arbitrary (high) Hamming weight, then it is clear that any two different messages will with overwhelming probability lead to a near-collision as in Definition 2.2. Therefore, we require that δ has a low Hamming weight. We will further elaborate on this in Section 3.1.1 and in Chapter 4. These definitions can also be adapted to second preimages and preimages as has been shown in [143].

2.3.4 Dedicated Hash Functions

Dedicated hash functions are hash functions that have been designed only for hashing purposes. In general, they are designed from scratch without using an existing cryptographic primitive such as a block cipher (*e.g.* AES). Some of the most popular hash functions are for instance MD4 [153], MD5 [154], SHA-1 and the SHA-2 family of hash functions [128], RIPEMD-160 [52], and Tiger [1]. All of these hash functions are iterated hash functions that follow the Merkle-Damgård design principle (see Section 2.3.1). The most commonly used hash functions in practice are MD5 and SHA-1.

A common component of these designs is that they all use a compression function f that maps an ℓ-bit input message block together with an n-bit chaining value h_{i-1} to an n-bit chaining value h_i. In most hash functions the compression function can be considered as a *weak* block cipher working in a certain mode of operation.

2.3.5 Block-Cipher-Based Hash Functions

An alternative to dedicated hash functions are functions that are based on existing block ciphers. The advantage of block-cipher-based constructions is that the underlying primitive has been analyzed thoroughly over years. For instance, no weaknesses have been found to date for the popular AES. Nevertheless, designing a hash function by using a block cipher leads to a completely new situation for the security analysis, since there is no secret key involved; all values are known by an adversary. A new approach to analyze the security of block ciphers with this in mind, has been introduced recently by Knudsen and Rijmen in [91].

We know that a block cipher can be inverted easily and therefore, we require a certain mode of operation. By using a block cipher that has been thoroughly analyzed, we basically only have to investigate the mode of operation.

Single-Block-Length Constructions

For a single-block-length (SBL) construction, one takes an existing block cipher and chooses a mode of operation to define the compression function. In [144], Preneel *et al.* discuss the general model of a block-cipher-based compression function. The block cipher E has two inputs (plaintext and key) and one output (ciphertext). For these inputs one can select the input message m_i, the intermediate chaining variable h_{i-1}, the value $m_i \oplus h_{i-1}$, or any constant c. Furthermore, one of these four values can be XORed to the ciphertext of the block cipher. Based on this general model, there are in total $4^3 = 64$ possible modes of operation. As a result of the security analysis in [144], only 12 out of the 64 possible modes of operation are secure. In [24], Black *et al.* have proven the security of these modes in the *ideal cipher model* (we briefly discuss security proofs in this model later on in this section). The three most popular modes are (see also Figure 2.2):

$$\begin{aligned} h_i &= E_{m_1}(h_{i-1}) \oplus h_i && \text{Davies-Meyer (DM)} \\ h_i &= E_{h_{i-1}}(m_i) \oplus m_i \oplus h_{i-1} && \text{Miyaguchi-Preneel (MP)} \\ h_i &= E_{h_{i-1}}(m_i) \oplus m_i && \text{Matyas-Meyer-Oseas (MMO)} \end{aligned} \tag{2.2}$$

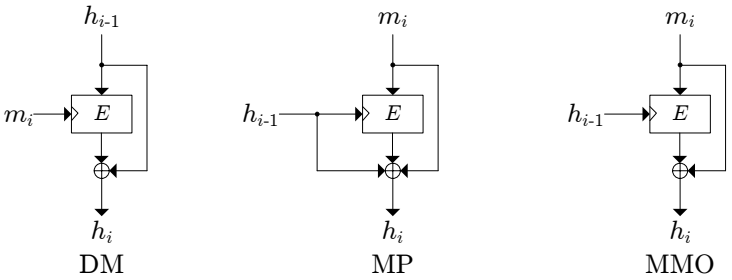

Figure 2.2: The Davies-Meyer, Miyaguchi-Preneel, and the Matyas-Meyer-Oseas mode of operation. The hatch denotes the key input of the block cipher E.

For instance, we could build a hash function by choosing AES-128 operating in one of the modes of operation in (2.2). This results in an iterated hash function with a 128-bit hash value. From the size of the hash value, we observe an immediate consequence of single-block-length constructions, namely we can only provide a hash value that equals the block length of the underlying block cipher. As discussed in Section 2.3.2, we can only expect a collision resistance of $2^{n/2}$. For our hash function with AES-128 this results in 2^{64}. Such a security margin is clearly no longer sufficient anymore for nowadays use.

Double-Block-Length Constructions

A common approach to overcome the limited hash-value size of single-block-length constructions is to use two or more block ciphers within one iteration and concatenate their outputs. Such schemes are referred to as *double-block-length* hash functions if the output of two block ciphers is concatenated resulting in a $2n$-bit hash value (we abbreviate double-block-length by DBL for the remainder of this section). Examples for DBL constructions are for instance MDC-2 and MDC-4 [116, pp. 341-343], or the recent DBL proposal of Hirose [68] (see also Section 5.2). In [116], MDC-2 is instantiated with DES [121] as underlying block cipher and in [166] instantiated with AES-128.

As opposed to single-block-length hash functions, the required security margins for DBL hash functions based on a block cipher with an n-bit output, are that the complexity for a collision attack is substantially higher than $2^{n/2}$ and the complexity for a (second) preimage attack is higher than 2^n (see [88]).

For block-cipher-based hash functions with an n-bit block cipher it is common to define the rate r of the scheme. The rate is interpreted as a measure for the performance of a double-block-length hash function. The rate is defined as follows:

Definition 2.3. *The rate r of a block-cipher-based hash function is defined as*

$$r = \frac{|m_i|}{\#encryptions \cdot n} \ .$$

Based on Definition 2.3, we have $r = 1/2$ for MDC-2 and for MDC-4 we get $r = 1/4$ (see Menezes *et al.* [116, page 339]). From this, we can derive that MDC-2 is twice as fast as MDC-4 but the rate does not provide any information about the security margins. For the block-cipher-based hash functions defined in (2.2) we have $r = 1$ if a block cipher with $|k| = n$ is employed. A block cipher with $|k| \neq n$ (*e.g.* DES) only affects the DM mode of operation: $r = |m|/n$. Both MMO and MP have $r = 1$.

We want to stress that the rate of a hash function gives only a rough estimate on the performance, since it only considers the number of calls to the underlying block cipher per message block. However, it can be that even if there are for instance two calls to the block cipher, parts of it, *e.g.* the key schedule of AES (see [122] for further details), only need to be computed once. This is clearly better than two full computations of AES. Nevertheless, in most cases the given definition of the rate is accurate enough for a first estimate of the performance and for a rough comparison of different proposals.

Proofs of Security in the Ideal Cipher Model

The first security analysis of the 64 possible SBL modes of operation by Preneel *et al.* in [144] was performed in a synthetic way. They analyzed the security of these schemes with respect to existing attacks and filtered out 52 constructions, since they are considered to be insecure by this analysis. Black *et al.* proved

these constructions secure in the ideal cipher model [24]. In the following, we will give a basic informal overview of the assumptions these proofs are based on. The discussion is based on Black's work presented in [22].

To prove the security of cryptographic primitives, we have to define a model on which we base our proofs. Mostly, we use the so-called *standard model*. In this model, we do not use any special mathematical objects such as random oracles [9]. This model does not turn out to be useful if we do not define additional hardness assumptions. For instance, it is widely believed that factoring the product of two large primes is difficult. Another assumption in this model is for instance that we assume from a good n-bit block cipher that it should behave as a pseudo-random permutation (PRP) on n bits. Informally, this means that for an n-bit block cipher with a secret randomly-chosen key it should be infeasible to distinguish its output from a randomly chosen permutation on n bits. For constructing a block-cipher-based hash functions the PRP assumption is not sufficient. We illustrate this by an example given by Black in [22]. Assume, we are given an n-bit block cipher E with a key k of n bits. Furthermore, we assume that this block cipher is a good PRP. Now, we define the block cipher $\widetilde{E}$ as follows:

$$\widetilde{E}_k(m) = \begin{cases} k & \text{if } m = k \\ E_k(k) & \text{if } m = E_k^{-1}(k) \\ E_k(m) & \text{otherwise} \end{cases} \tag{2.3}$$

The only difference between E and $\widetilde{E}$ is the invariant that $\widetilde{E}_k(k) = k$ for any key k. Since we assumed that E is a good PRP, $\widetilde{E}$ is also a good PRP: for a randomly chosen key, $\widetilde{E}_k(\cdot)$ is computationally indistinguishable from a randomly chosen n-bit permutation. Now we build a SBL hash function with the block cipher $\widetilde{E}$ working in the MMO mode of operation. Since, $\widetilde{E}$ is a good PRP it fulfills the requirements and we expect to have a strong hash function. However, for this construction it is easy to construct collisions. Namely, the 2-block message $m = m_1 \| m_2$, where m_1 is an arbitrary n-bit value and

$$m_2 = m_1 \oplus \widetilde{E}_{iv}(m_1)$$

results in $h_2 = 0$ for any value of m_1 and any value of iv (see also Figure 2.3). This follows from the fact that $h_1 = \widetilde{E}_{iv}(m_1) \oplus m_1$. Then by choosing $m_2 = m_1 \oplus \widetilde{E}_{iv}(m_1)$, we have $\widetilde{E}_{m_2}(m_2) = m_2$ based on the definition of $\widetilde{E}$ in (2.3). This leads to $h_2 = \widetilde{E}_{m_2}(m_2) \oplus m_2 = m_2 \oplus m_2 = 0$.

Even if this demonstrative example shows that the PRP assumption is not enough for constructing a block-cipher-based hash function, we stress that the block cipher $\widetilde{E}$ was chosen such that the collision attack works. Also, there is no reason why a block cipher should possess an unusual property such as the invariant $\widetilde{E}_k(k) = k$.

In the *ideal cipher model* (also referred to as *black box model*) a block cipher E is considered as being chosen uniformly from the set of all possible n-bit block ciphers with a k-bit key. For every single key there are $2^n!$ possible permutations. Furthermore, since any permutation may be assigned to a single key, there are in

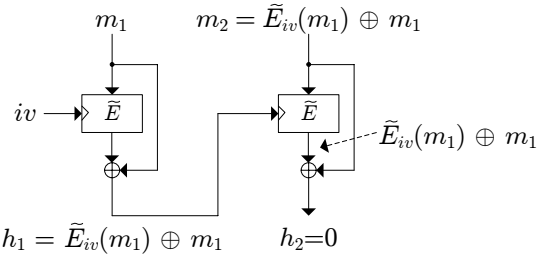

Figure 2.3: Constructing collisions for a MMO hash construction with underlying block cipher $\widetilde{E}$ defined in (2.3).

total $(2^n!)^{2^k}$ possible block ciphers. In this model, an adversary is given access to the ideal cipher oracle, which he can query for encryption and decryption. If we consider a block-cipher-based hash function, then the security with respect to for instance collision resistance is proved by showing that the advantage of an adversary in finding a colliding message pair is not better than what we can expect from the birthday paradox. For the 12 provably secure SBL block-cipher-based hash functions it has been shown by Black *et al.* in [24] that the advantage of an adversary in finding a colliding message pair is upper bounded by $q(q+1)/2^n$, where q is the number of queries to the ideal cipher oracle for encryption and decryption. In other words, the adversary cannot do better than finding a collision based on a birthday attack.

The question is what we can expect from a cryptographic scheme that has been proven secure in the ideal cipher model, once we employ a practical block cipher (this is called *instantiation*). In the cryptographic community there are different opinions about the value of such proofs. In [22], Black has demonstrated that even if a block-cipher-based hash construction is provable secure in the ideal cipher model, it can be broken once a concrete block cipher is employed. Such a scheme it is then called *uninstantiable*. He showed that for the 12 provably secure SBL schemes, he can construct variants that are also provably secure in the ideal cipher model but they are uninstantiable. It is important to note that these variants possess quite unusual properties compared to known block-cipher-based constructions.

2.4 Message Authentication Codes

In this section, we briefly introduce message authentication codes, or short MACs. We will not discuss attack scenarios and the security of MACs, since we will work further on this in Chapter 6. MACs are used for symmetric message authentication, first to protect the integrity of the message and to provide origin authentication. To achieve this goal, a secret key k is involved (shared between legitimate parties) for computing the *tag* of a given message.

A message authentication code can be considered as a family of hash functions

H_k parameterized by the secret key k. Therefore, the way how the message is processed to compute the tag is similar to that of an iterated hash function. As shown in Figure 2.1, we have three possibilities to construct a MAC from existing primitives: block ciphers, hash functions, and stream ciphers. We note that only few dedicated MAC algorithms exist. For instance the Message Authenticator Algorithm MAA designed in 1988 by Davies and Clayden [43].

We demonstrate the basic working principle by considering the most popular block-cipher-based MAC construction, namely the CBC-MAC and several variants of it, which are standardized in [57]. The basic CBC-MAC is based on a block cipher working in the CBC mode of operation (see Section 2.2). As opposed to a hash function only the last ciphertext is output resulting in the tag. Assuming we have a padded message m consisting of t blocks of n bits each (n is the block length of the underlying cipher), we can summarize the tag computation as follows:

$$\begin{aligned} c_0 &= iv \\ c_i &= E_k(m_i \oplus c_{i-1}) \qquad \text{for } i = 1, \ldots, t \\ \text{tag} &= g(c_t) \end{aligned} \tag{2.4}$$

The function $g(\cdot)$ is an output transformation and is in the simplest case a truncation of n bits to $\widetilde{n} < n$ bits. The security of CBC-MAC has been proven for fixed-length messages by Bellare *et al.* in [7] and for variable-length messages by Petrank and Rackoff in [132]. For the proof of security with respect to variable-length messages a variant of CBC-MAC has been defined, namely EMAC. For EMAC two keys k' and k'' are derived from the key k. The message is then processed by CBC-MAC as in (2.4) with key k'. Finally, the output of (2.4) is encrypted by applying the block cipher with key k''.

Note that for variable-length messages an output transformation is required for CBC-MAC in order to avoid the fact that it is easy to construct different messages leading to identical tags. This would result in a *forgery attack* on the MAC (see Chapter 6 for further details). We demonstrate the importance of the output transformation for the basic CBC-MAC by a simple example (see also Menezes *et al.* [116, page 354]). Assume that we are given two single-block messages m^1 and m^2, with $m^1 \neq m^2$, and according tags, denoted by tag_{m^1} and tag_{m^2}. We choose the two-block message $m^3 = m^1 \| y$, where y is an arbitrary n-bit value, and request the according tag tag_{m^3}. Assume the tags have been computed as shown in (2.4), where $g(\cdot)$ is the identity mapping:

$$\begin{aligned} \text{tag}_{m^1} &= E_k(m^1 \oplus iv) \\ \text{tag}_{m^2} &= E_k(m^2 \oplus iv) \\ \text{tag}_{m^3} &= E_k(y \oplus E_k(m^1 \oplus iv)) \end{aligned}$$

Based on this information, we can now construct a fourth 2-block message m^4 that has the identical tag as m^3. We choose the message $m^4 = m^2 \| y'$, where

$y' = y \oplus \text{tag}_{m^1} \oplus \text{tag}_{m^2}$. This results in

$$\begin{aligned} \text{tag}_{m^4} &= E_k(y' \oplus E_k(m^2 \oplus iv)) = E_k(y \oplus \text{tag}_{m^1} \oplus \text{tag}_{m^2} \oplus \text{tag}_{m^2}) \\ &= E_k(y \oplus \text{tag}_{m^1}) = \text{tag}_{m^3} \ . \end{aligned}$$

It is easy to see that truncating the output to $\widetilde{n} < n$ bits destroys this property. However, as has been shown by Knudsen in [86] truncating the output is still not enough for avoiding forgery attacks. By using stronger output transformations these attacks can be thwarted. For instance, the proposal MacDES by Knudsen and Preneel in [89] includes two additional applications of the block cipher DES for processing the first message block (2 encryptions with the same key as for processing the entire input message) and an additional encryption with a different key before the tag is output. For this proposal, truncating the output is not required but can be done optionally. An overview of the security analysis of several CBC-MAC variants has been presented for instance by Brincat and Mitchell in [25]. To summarize, the problems with these CBC-MAC constructions arise due to the employment of the *weak* block cipher DES with only 64 output bits. If AES is used as underlying block cipher, then the basic CBC-MAC construction with a simple output truncation to less than 128 bits can be used—there is no need to implement any other variants.

An alternative to block-cipher-based MAC constructions are MACs based on hash functions. For instance Preneel and van Oorschot presented MDxMAC in [145], which is a hash-based MAC construction for the MD family of hash functions such as MD4 [153] and MD5 [154]. They also presented a thorough security analysis of iterated MACs in [146]. The most well known hash-based constructions to date are NMAC and HMAC presented by Bellare *et al.* in [6]. We will describe both MAC constructions and discuss their security with respect to recent cryptanalytic results in more detail in Chapter 6.

An example of a MAC based on stream ciphers was presented by Lai *et al.* in [94]. There exist several other proposals for MACs. To mention few of them: UMAC proposed by Black *et al.* in [23], Poly1305-AES MAC proposed by Bernstein in [12], and the MAC construction ALRED proposed by Daemen and Rijmen in [38].

2.5 On Differential Properties of Boolean Functions and Modular Addition

In this section, we introduce signed-bit differences as a generalization of XOR differences. We will look at modular additions and the differential properties of Boolean functions with respect to signed-bit differences which have been introduced by Wang *et al.* in [169]. Regarding the Boolean functions, we will focus only on those we need for the analysis of the hash function SHA-1 in Chapter 4. For a more general treatment of signed-bit differences we refer to the PhD thesis of Daum [42].

Table 2.2: Addition of signed-bit differences.

A'_j	B'_j	C'_j	S'_j	C'_{j+1}
0	0	0	0	0
0	0	v	$(-1)^{A_j \oplus B_j} v$	$-v(A_j \oplus B_j)$
0	v	0	$(-1)^{A_j \oplus C_j} v$	$-v(A_j \oplus C_j)$
v	0	0	$(-1)^{B_j \oplus C_j} v$	$-v(B_j \oplus C_j)$
0	u	v	0	$\frac{1}{2}(u+v)$
u	0	v	0	$\frac{1}{2}(u+v)$
u	v	0	0	$\frac{1}{2}(u+v)$
v	v	v	$(-1)^{A_j \oplus B_j \oplus 1} v$	$(-1)^{A_j \oplus B_j} v$
v	v	$-v$	$(-1)^{A_j \oplus B_j \oplus 1} v$	$(-1)^{A_j \oplus B_j} v$
v	$-v$	v	$(-1)^{A_j \oplus C_j \oplus 1} v$	$(-1)^{A_j \oplus C_j} v$
$-v$	v	v	$(-1)^{B_j \oplus C_j \oplus 1} v$	$(-1)^{B_j \oplus C_j} v$

2.5.1 Signed-Bit Differences

We define the sign of a difference in bit position j as

$$w'_j = w_j - w^*_j, \quad \text{where } w_j, w^*_j \in \{0,1\} \text{ and } w'_j \in \{-1,0,+1\} \ .$$

In particular, if $w'_j = 0$ the difference is zero. The signed-bit difference is defined as $W'_j = w'_j 2^j$. A useful property of signed-bit differences is that the difference also includes information about the values of w_j and w^*_j.

$$W'_j = \begin{cases} +2^j & \text{if } w_j = 1 \text{ and } w^*_j = 0 \\ 0 & \text{if } w_j = w^*_j \\ -2^j & \text{if } w_j = 0 \text{ and } w^*_j = 1 \end{cases}$$

Let us now consider the addition of two signed-bit differences. The addition $S = A + B$ is defined as

$$\begin{aligned} S_j &= A_j \oplus B_j \oplus C_j \\ C_{j+1} &= (A_j \wedge B_j) \oplus (A_j \wedge C_j) \oplus (B_j \wedge C_j) \text{ with } C_0 = 0 \ , \end{aligned}$$

where C_{j+1} is the resulting carry of the addition in bit position j. Table 2.2 lists all possible cases for the output and carry difference of a signed-bit addition with $v, u \in \{-1, +1\}$.

To perform the addition of two signed-bit differences, we can use Table 2.2 for computing the resulting difference. We know that the output difference is $C'_{j+1} 2^{j+1} + S'_j 2^j$. For instance, if there are two non-zero differences at the input

with opposite signs, then both C'_{j+1} and S'_j are zero and hence, the output difference is zero. If the differences have the same sign, for instance -2^j and -2^j, the output difference is -2^{j+1}, since $C'_{j+1} = -1$ and $S'_j = 0$.

2.5.2 Differential Properties of Boolean Functions in SHA-1

For our analysis of local collisions in Section 4.4, we need the differential properties with respect to signed-bit differences of the Boolean functions defined for SHA-1:

$$f_{\text{IF}}(B, C, D) = (B \wedge C) \oplus (\neg B \wedge D)$$
$$f_{\text{MAJ}}(B, C, D) = (B \wedge C) \oplus (B \wedge D) \oplus (C \wedge D)$$
$$f_{\text{XOR}}(B, C, D) = B \oplus C \oplus D$$

In Table 2.3, we list the cases that occur in a local collision (see Section 4.4, Figure 4.7) where $v \in \{-1, +1\}$. As can be seen in Table 2.3, for f_{XOR} the sign of the input difference is flipped with probability 1/2 depending on the input values and assuming that the input values have been chosen randomly. For f_{MAJ} the sign is preserved but the difference propagates with probability 1/2. For instance, if the input difference $D'_j = v2^j$ then the output difference of f_{XOR} is given by $(-1)^{B_j \oplus C_j} v2^j$ and for f_{MAJ} it is given by $(B_j \oplus C_j)v2^j$. In Section 6.2.2, we also need the differential properties of f_{IF}. We give some cases in Table 2.4.

Table 2.3: Differential properties of f_{XOR} and f_{MAJ} for signed-bit differences.

B'_j	C'_j	D'_j	$f_{\text{XOR}}(B'_j, C'_j, D'_j)$	$f_{\text{MAJ}}(B'_j, C'_j, D'_j)$
0	0	v	$(-1)^{B_j \oplus C_j} v$	$(B_j \oplus C_j)v$
0	v	0	$(-1)^{B_j \oplus D_j} v$	$(B_j \oplus D_j)v$
v	0	0	$(-1)^{C_j \oplus D_j} v$	$(C_j \oplus D_j)v$

Table 2.4: Differential properties of f_{IF} for signed-bit differences.

B'_j	C'_j	D'_j	$f_{\text{IF}}(B'_j, C'_j, D'_j)$
0	0	v	$(B_j \oplus 1)v$
0	v	0	$B_j v$
v	0	0	$(C_j \oplus D_j)(-1)^{D_j} v$
0	v	v	v
v	v	0	$(D_j \oplus 1)v$
v	0	v	$C_j v$
0	$-v$	v	$(-1)^{B_j} v$
v	$-v$	0	$-D_j v$
v	0	$-v$	$-C_j v$
v	v	v	v
v	v	$-v$	0
v	$-v$	v	0
$-v$	v	v	v

2.6 Finite Fields

In this section, we will introduce the basics on finite fields. We will only introduce the necessary definitions that are required for the analysis of the hash function design strategy SMASH in Chapter 3. We will start by defining groups, rings, and fields. Then, we go over to polynomials over fields. This introduction is based on the work of Lidl and Niederreiter in [98]. We refer to their work for a more comprehensive discussion on the theory of finite fields and applications.

2.6.1 Groups, Rings, and Fields

Definition 2.4. *An Abelian group $(G, +)$ is a set G together with a binary operation $+$ on G such that the following properties hold:*

1. *closure:* $\forall a, b \in G : a + b \in G$
2. *associativity:* $\forall a, b, c \in G : a + (b + c) = (a + b) + c$
3. *identity element:* $\exists 0 \in G, \forall a \in G : a + 0 = 0 + a = a$
4. *inverse element:* $\forall a \in G, \exists b \in G : a + b = b + a = 0$
5. *commutativity:* $\forall a, b \in G : a + b = b + a$

An example of an Abelian group is $(\mathbb{Z}, +)$ with '+' being the ordinary integer addition. For this group, the identity element is 0 and the inverse element of $a \in \mathbb{Z}$ is $-a$. The additive notation is common for Abelian groups. For $(a+(-b))$,

we simply write $a - b$. If we consider the multiplicative notation then we say that we have a multiplicative group $(G, \cdot)$ with the binary operation '$\cdot$'. The inverse element of $a \in G$ is then denoted by a^{-1} and the identity element is 1. Furthermore, for the n-fold composite of $a \in G$ we write $a^n = a \cdot a \cdots a$ in the multiplicative notation and in the additive notation we write $na = a+a+\cdots+a$. We adopt the convention that $a^0 = 1$ in multiplicative notation and $0a = 0$ in additive notation.

Definition 2.5. *A multiplicative group $(G, \cdot)$ is said to be* cyclic *if there exists an element $a \in G$ such that $\forall b \in G$, there is some integer j with $b = a^j$.*

We call such an element the *generator* of the cyclic group and we write $G = \langle a \rangle$. Note that a cyclic group can have more than one generator. For instance, in the additive group $(\mathbb{Z}, +)$ two generators are 1 and -1. Another example for a cyclic group are the integers modulo n. For arbitrary integers a, b and a positive integer n, we say that $a \equiv b \bmod n$ if the difference $a-b$ is a multiple of n, *i.e.* $a = b+kn$ for some integer k. The set of all residue classes modulo n is denoted by $\mathbb{Z}_n$. It is also called $\mathbb{Z}$ modulo n and is represented by the set $\{0, 1, \ldots, n-1\}$. This cyclic group $(\mathbb{Z}_n, +)$ is generated by the set $\{\ldots, -2n+1, -n+1, 1, n+1, 2n+1, \ldots\} \equiv 1 \bmod n$. This example also leads to the definition of a *finite* group.

Definition 2.6. *A group is called finite if it contains finitely many elements. The number of elements is called the* order *of the group and denoted by $|G|$.*

Definition 2.7. *For $a \in (G, \cdot)$, we call the least positive integer k, for which $a^k = 1$, the order of the element a and we denote it by $ord(a) = k$.*

Definition 2.8. *A* ring *$(R, +, \cdot)$ is a set R together with two binary operations '$+$' and '$\cdot$', such that*

1. *$(R, +)$ is an Abelian group*
2. *the operation '$\cdot$' is associative, i.e. $\forall a, b, c \in R : (a \cdot b) \cdot c = a \cdot (b \cdot c)$*
3. *the distributive laws hold:*
   $\forall a, b, c \in R : a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$

Furthermore, we say $(R, +, \cdot)$ is a commutative ring *if the operation '$\cdot$' is commutative. If there exists a multiplicative identity then we call the ring a* ring with identity.

In the following we use 0 to denote the identity element of the Abelian group $(R, +)$ and the additive inverse of a is denoted by $-a$. For a ring with identity, we denote the multiplicative identity by 1. A well-known example of a commutative ring is the ring of integers with ordinary addition and multiplication: $(\mathbb{Z}, +, \cdot)$. Another example is the ring $(\mathbb{Z}_n, +, \cdot)$, the ring of integers modulo n.

Definition 2.9. *An algebraic structure $(F, +, \cdot)$ is called a field if the following properties hold:*

1. *$(F, +, \cdot)$ is a commutative ring*
2. *for all elements in F there exists an inverse element in F with respect to the operation '$\cdot$', except for the identity element 0 in $(F, +)$*

Furthermore, we call $(F, +, \cdot)$ *a* finite field *if the set* F *contains only finitely many elements.*

In the following, we use 1 to denote the multiplicative identity of the field F.

Definition 2.10. *Given a field* F*. A vector space* V *is an Abelian group together with an external operation* $F \times V \to V$*, which satisfies the following properties:* $\forall \lambda, \mu \in F$ *and* $\forall \mathbf{u}, \mathbf{v} \in V$ *it holds that*

1. $\lambda(\mu\mathbf{u}) = (\lambda\mu)\mathbf{u}$
2. $(\lambda + \mu)\mathbf{u} = \lambda\mathbf{u} + \lambda\mathbf{u}$
3. $1\mathbf{u} = \mathbf{u}$
4. $\lambda(\mathbf{u} + \mathbf{v}) = \lambda\mathbf{u} + \lambda\mathbf{v}$

The elements of V are called vectors and the elements of F are called scalars. An example of a vector space is $F \times F \times \cdots \times F = F^n$. We say that F^n is an n-dimensional vector space over F, and every vector can be written as $\mathbf{v} = (v_1, \ldots, v_n)$, where $v_i \in F$. Addition of two vectors $\mathbf{u}, \mathbf{v} \in F^n$ is defined as

$$\mathbf{u} + \mathbf{v} = (u_1 + v_1, u_2 + v_2, \ldots, u_n + v_n)$$

and scalar multiplication is performed as

$$\lambda\mathbf{v} = (\lambda v_1, \ldots, \lambda v_n) \ ,$$

where all component-wise operations are carried out in the underlying field F.

2.6.2 Polynomials over Fields

Let F be a field. A *polynomial* over F is an expression of the form

$$f(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + \cdots a_n x^n \ ,$$

where n is a non-negative integer, the a_i's are called *coefficients* and $a_i \in F$ for $0 \leq i \leq n$. The variable x does not belong to F and is therefore called the *indeterminate*. For two polynomials over F

$$f(x) = \sum_{i=0}^{n} a_i x^i \text{ and } g(x) = \sum_{i=0}^{n} b_i x^i$$

we define the *polynomial addition* as the addition of the coefficients, *i.e.*

$$f(x) + g(x) = \sum_{i=0}^{n} (a_i + b_i) x^i \ ,$$

where without loss of generality, we assume that both polynomials involve the same powers of x. The product of two polynomials over F

$$f(x) = \sum_{i=0}^{n} a_i x^i \text{ and } g(x) = \sum_{j=0}^{m} b_j x^j$$

is given by

$$f(x)g(x) = \sum_{k=0}^{n+m} c_k x^k, \text{ with } c_k = \sum_{\substack{i+j=k \\ 0\leq i\leq n, 0\leq j\leq m}} a_i b_j \ .$$

With these operations it can be shown that the set of polynomials over F forms a ring.

Definition 2.11. *The ring formed by the set of polynomials over F with the operations given above, is called the* polynomial ring *over F and denoted by $F[x]$.*

Let $f(x) = \sum_{i=0}^{n} a_i x^i \in F[x]$ and suppose $a_n \neq 0$. Then we call a_n the *leading coefficient* of $f(x)$, and a_0 is the *constant term*. The polynomial where all coefficients are zero is called the *zero polynomial* and we simply write 0 to denote it. Furthermore, we call n the *degree* of $f(x)$ denoted by $n = deg(f(x))$. By convention we set $deg(0) = -\infty$. A polynomial with degree 0 is a *constant polynomial*. If the leading coefficient of $f(x)$ is $a_n = 1$ then we call $f(x)$ a *monic polynomial*. For polynomial rings over fields there is a division with remainder.

Theorem 2.1. *Let $g(x) \neq 0$ be a polynomial in $F[x]$. Then for any $f(x) \in F[x]$ there exist polynomials $q(x)$,$r(x) \in F[x]$ such that*

$$f(x) = q(x)g(x) + r(x), \textit{where } deg(r(x)) < deg(g(x)) \ ,$$

For computing the division with remainder, we can use the Euclidean algorithm (see for instance [116]).

Definition 2.12. *A polynomial $p(x) \in F[x]$ is said to be irreducible over F if $p(x)$ has positive degree and $p(x) = b(x)c(x)$ with $b(x), c(x) \in F[x]$ implies that either $b(x)$ or $c(x)$ is a constant polynomial.*

In other words, an irreducible polynomial over $F[x]$ allows only trivial factorizations.

Definition 2.13. *An element $b \in F$ is called a root of the polynomial $f(x) \in F[x]$ if $f(b) = 0$.*

Now we introduce subfields and extension fields. Let F be a field. Any subset K of F that is itself a field is called a *subfield*. In this case, we say F is an *extension field* of K. Furthermore, if $K \neq F$ we say that K is a *proper subfield* of F.

In cryptography, the most important fields are finite fields. We know that the residue classes modulo n form the ring $(\mathbb{Z}_n, +, \cdot)$. If $n = p$, where p is prime, then the residue classes modulo p form a finite field $(\mathbb{Z}_p, +, \cdot)$. For the remainder of this thesis, we will denote the finite field with q elements by $GF(q)$, where GF stands for Galois field. The simplest finite field is the binary field $GF(2)$ containing only two elements, namely the elements 0 and 1. It can be shown,

cf. [98], that every finite field consists of $q = p^n$ elements and can be realized as an extension field of $GF(p)$, for p a prime number. We illustrate this fact by an example.

Example 2.1. We construct the finite field $GF(2^2)$ as an extension field of $GF(2)$, *i.e.* $p = 2$ and $n = 2$. This implies that the elements 0 and 1 are included in $GF(4)$. The finite field $GF(4)$ contains 4 elements. We add the new element α. For this new element, we can now derive the Cayley tables given in Table 2.5.

Table 2.5: Cayley tables for addition and multiplication in $GF(2^2)$.

$+$	0	1	α
0	0	1	α
1	0	0	$1+\alpha$
α	α	$1+\alpha$	0

$\cdot$	0	1	α
0	0	0	0
1	0	1	α
α	0	α	α^2

As can be seen in Table 2.5, we have two new elements, namely $1 + \alpha$ and α^2. Together with the elements $0, 1, \alpha$ we have five elements $\{0, 1, \alpha, \alpha + 1, \alpha^2\}$. However, we know that $GF(4)$ only consists of four elements. Therefore, two of these five elements have to be equal. Assume $\alpha^2 = \alpha$. This leads to $\alpha = 0$ or $\alpha = 1$. Since this does not add any new element, we try $\alpha + 1 = \alpha$, *i.e.* $0 = 1$, which is impossible since 0 and 1 are distinct in $GF(2)$. The last equality is $\alpha^2 = \alpha + 1$. If we use this equality then addition and multiplication is defined properly in $GF(4)$. Thus the elements of $GF(4)$ are $\{0, 1, \alpha, \alpha+1\}$, where alpha is the root of the monic irreducible polynomial $x^2 + x + 1$, *i.e.* $\alpha^2 + \alpha + 1 = 0$. Whenever α^2 turns up in a multiplication it is replaced by $\alpha + 1$.

Theorem 2.2. *Every finite field $GF(p^n)$ with p a prime number can be realized as the set of all polynomials over $GF(p)$ with degree $\leq n - 1$. Addition is performed by adding the coefficients modulo p. Multiplication in the finite field is polynomial multiplication followed by a reduction modulo a monic irreducible polynomial of degree n.*

For efficient implementations of finite field arithmetic, we refer to Hankerson *et al.* [64]. For the finite extension field $GF(p^n)$, we call p the *characteristic* of $GF(p^n)$. Furthermore, we denote by $(GF(p^n)^*, \cdot)$ the multiplicative group of the non-zero elements in $GF(p^n)$. An important fact about finite fields is that $(GF(p^n)^*, \cdot)$ is a cyclic group. A generator of this group is called a primitive element in $GF(p^n)$.

We can now also define the inverse operation of multiplication with respect to the irreducible polynomial $f(x)$ for the multiplicative group $(GF(p^n)^*, \cdot)$. We can use the extended Euclidean algorithm (see Menezes *et al.* [116, Algorithm 2.221, page 82]) for finding the inverse of the polynomial $a(x)$ as follows. The extended Euclidean algorithm returns two polynomials $b(x)$ and $c(x)$ such that

$$a(x)b(x) + f(x)c(x) = GCD(a(x), f(x)) \ , \tag{2.5}$$

where GCD denotes the *greatest common divisor* of $a(x)$ and $f(x)$. Since $f(x)$ is monic and irreducible, we know that $GCD(a(x), f(x)) = 1$. If we reduce (2.5) modulo the irreducible polynomial, we have

$$a(x)b(x) \equiv 1 \bmod f(x)$$

which means that $b(x)$ is the inverse element of $a(x)$.

Definition 2.14. *A primitive polynomial over $GF(p)$ of degree n is a monic polynomial that is irreducible over $GF(p)$ and has a root $\alpha \in GF(p^n)$ that generates the multiplicative group of $GF(p^n)$.*

Example 2.2. For the extension field $GF(4)$ the polynomial $x+1$ is a generator of $GF(4)^*$. This can be shown easily by computing the powers of the element $(1+x)$ modulo the irreducible polynomial. Furthermore, $(x+1)$ is a root of the irreducible polynomial $f(x) = x^2 + x + 1$ defining $GF(4)$. This is since $f(x+1) = (x+1)^2 + (x+1) + 1 = (x^2+1) + x$. We know that $x^2 = x+1$ and therefore we get $f(x+1) = (x+1+1) + x = 0$. Since $f(x)$ is monic, we can conclude that $f(x)$ is a primitive polynomial over $GF(2)$.

2.7 Linear Codes

In this section, we introduce the basic concepts of coding theory, whereby we will focus only on linear codes. This basic introduction to coding theory is based on the work by van Lint in [163]. After this introduction, we explain algorithms that can be used to search for codewords of low Hamming weight. This introduction serves as a starting point for Section 4.3.

2.7.1 Basic Definitions

Definition 2.15. *Let $A = \{0, \ldots, q-1\}$ be an alphabet consisting of q elements. We call a sequence of n symbols in A a word of length n. The set of all words of length n is denoted by A^n.*

It is clear that if the alphabet A consists of q symbols, then there exist exactly q^n words of length n. Now we give the definition of a code C.

Definition 2.16. *An $[n,k]$-code C over the alphabet $A = \{0, 1, \ldots, q-1\}$ with q elements is a subset of A^n consisting of exactly q^k codewords. The number n is called the length of the code denoted by $length(C)$, and k is called the dimension of C denoted by $dim(C)$.*

Definition 2.17. *(Hamming Distance and Weight) For $u = (u_1, \ldots, u_n) \in A^n$ and $v = (v_1, \ldots, v_n) \in A^n$ the Hamming distance of u, v is defined as*

$$d(u,v) := \#\{i : u_i \neq v_i\} \text{ with } i \in \{1, \ldots, n\} .$$

The Hamming weight of u, denoted by $wt(u)$, is the number of nonzero entries in u, i.e. $wt(u) = d(\mathbf{0}, u)$, where $\mathbf{0}$ denotes the zero word, *i.e. $\mathbf{0} = \{0, \ldots, 0\}$.*

The Hamming distance is a very important measure for codes. Some of its properties are given by the following lemma.

Lemma 2.1. *The Hamming distance $d(u, v)$ is a metric on A^n, i.e. it satisfies*

$$\begin{aligned} &d(u,v) \geq 0, \text{ and } d(u,v) = 0 \text{ if and only if } u = v \\ &d(u,v) = d(v,u) \text{ for all } u, v \\ &d(u,v) \leq d(u,w) + d(w,v) \text{ for all } u, v, w \ . \end{aligned}$$

Before we go over to linear codes, we still have to define a very important property of codes, namely the minimum distance.

Definition 2.18. *Let C be a $[n, k]$-code over A. We call*

$$d := \min_{x,y \in C, x \neq y} d(x, y)$$

the minimum distance of C. Then we call C also an $[n, k, d]$-code.

Definition 2.19. *(Linear Code) Let the alphabet A be the finite field $GF(q)$ with q elements. Then we call an $[n, k]$-code C over $GF(q)$ a linear code, if C is a subspace of the vector space $GF(q)^n$, whereby multiplication by $\alpha \in GF(q)$ and addition of $a = (a_1 \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ is defined as follows*

$$\begin{aligned} \alpha a &= (\alpha a_1, \ldots, \alpha a_n) \\ a + b &= (a_1 + b_1, \ldots, a_n + b_n) \ . \end{aligned}$$

Note that if $q = 2$, we speak of a linear binary code. For a linear code, we can now characterize the minimum distance by the following theorem.

Theorem 2.3. *For an $[n, k]$-linear code over $GF(2)$ the minimum distance equals the minimum weight of all codewords different from the zero vector.*

Proof. For $u, v \in C$ it holds that $d(u, v) = wt(u - v)$. Due to the linearity, we know that also $u - v \in C$, *i.e.* $u - v$ is also a codeword in C. Therefore, it follows that

$$d = \min_{\substack{c_1, c_2 \in C \\ c_1 \neq c_2}} d(c_1, c_2) = \min_{\substack{c_1, c_2 \in C \\ c_1 \neq c_2}} wt(c_1 - c_2) = \min_{\substack{c \in C \\ c \neq 0}} wt(c) \ ,$$

since C is a group with respect to addition. □

So far, we defined basic properties of (linear) codes. Now we introduce two important matrices in coding theory: the *generator matrix* and the *check matrix* of a linear code. These matrices define how a message word gets encoded into a codeword that can be transmitted over a noisy channel, and how it can be verified whether or not a received message is a codeword. Encoding is done via a (linear) encoder. Basically, an encoder adds to each message word a certain amount of redundancy resulting in a codeword. With this added redundance it is possible to verify whether or not a received codeword is valid, meaning that it can be decoded to get the message according to the transmitted codeword.

Lemma 2.2. *Let* $\mathbf{G}$ *be a* $k \times n$ *matrix with* $k \leq n$ *and* $rank(\mathbf{G}) = k$*, and let* C *be the set of vectors* $x\mathbf{G}$*, with* $x \in GF(q)^k$*. Then* C *is an* $[n, k]$*-linear code.*

Proof. Let $v, w \in C$. For $x, y \in GF(q)^k$ and $\lambda \in GF(q)$ it holds that

$$v + w = x\mathbf{G} + y\mathbf{G} = (x + y)\mathbf{G} \in C \text{ since } x + y \in GF(q)^k$$
$$\lambda v = \lambda(x\mathbf{G}) = (\lambda x)\mathbf{G} \in C \text{ since } \lambda x \in GF(q)^k.$$

Therefore, the set C is a subspace of $GF(q)^n$ and hence a linear code following Definition 2.19. □

Definition 2.20. *Let* C *be a linear* $[n, k]$*-code over* $GF(q)$*. Furthermore, let* $\mathbf{G}$ *be a* $k \times n$ *matrix such that the codewords are determined by* $x\mathbf{G}$ *for any word* x *of length* k*. Then* $\mathbf{G}$ *is called a generator matrix of the code* C*.*

Note that we call the matrix $\mathbf{G}$ a *systematic* generator matrix if the first k columns of $\mathbf{G}$ are the $k \times k$ identity matrix $\mathbf{I}_k$. We simply write $\mathbf{I}$ if the dimension is clear from the context. One of the main reasons to work with linear codes is the fact that it is easy to verify whether or not a received word is a codeword. To demonstrate this, we define the check matrix of a linear code C.

Definition 2.21. *A check matrix for a* $[n, k]$*-linear code* C *over* $GF(q)$ *is an* $(n - k) \times n$ *matrix* $\mathbf{H}$*, with the property that for all* $v \in GF(q^n)$ *the following holds:*

$$\mathbf{H}v^T = 0 \Leftrightarrow v \in C$$

As for the generator matrix, we call the check matrix $\mathbf{H}$ systematic if the rightmost $(n-k)$ columns are the identity matrix $\mathbf{J}_{n-k}$. The following theorem states an important relationship between the generator matrix and the check matrix of a code C.

Theorem 2.4. *Let* C *be a* $[n, k]$*-linear code over* $GF(q)$*. Then the following holds.* C *possesses a systematic generator matrix* $\mathbf{G}$ *if and only if it possesses a systematic check matrix* $\mathbf{H}$*. In particular, if* $\mathbf{G} = \begin{bmatrix} \mathbf{I} & \mathbf{A} \end{bmatrix}$ *and* $\mathbf{H} = \begin{bmatrix} \mathbf{B} & \mathbf{J} \end{bmatrix}$ *then* $\mathbf{A} = -\mathbf{B}^T$*.*

The presented definitions and theorems are the basics on coding theory we need for the next section and for the discussion in Section 4.3. This introduction is only a very small fraction of coding theory and error correcting codes. Therefore, we refer the interested reader to [101, 134, 147, 163] for a more detailed treatment of coding theory and error correcting codes.

2.7.2 Searching for Low-Weight Codewords

In Definition 2.18, we defined the minimum distance of an $[n, k]$-code C. For linear codes, the minimum distance equals the minimum Hamming weight of all codewords different from the zero codeword, *cf.* Theorem 2.3. Finding codewords with a certain Hamming weight in linear codes, plays an important role in

cryptography. For instance, one of the first cryptosystems that has used error-correcting codes has been proposed by McEliece in [108]. It can be shown that finding a codeword with minimum Hamming weight renders the cryptosystem insecure. However, finding a codeword with a given Hamming weight in a linear code is an NP complete problem (see Berlekamp *et al.* [11]). Nevertheless, there exist probabilistic algorithms for finding codewords in linear codes with low Hamming weight. We describe the basic idea of these algorithms and summarize work factor estimates given in [30]. We will apply these algorithms to search for linear characteristics with low Hamming weight in Chapter 4.

Probabilistic Algorithms

We will briefly discuss some probabilistic algorithms presented by Leon [96] and modified by Chabaud [30], Stern [158], and by Canteaut and Chabaud [29]. The basic common approach of these algorithms is to take a (randomly permuted) subset of a given code C, which is referred to as the punctured code and denoted by $C^\bullet$. Since we only take a subset of C, we have $length(C^\bullet) < length(C)$. Then, we search for codewords with a small Hamming weight in the smaller code $C^\bullet$, instead of searching in C. A found codeword with small Hamming weight in the punctured code is a good candidate for a low Hamming weight codeword in the initial code C.

Modified Leon. A modified variant of Leon's algorithm [96] was presented by Chabaud [30]. It is applied to the generator matrix $\mathbf{G}_{k\times n}$ of a code C and defines the parameters p and s. The first step is to randomly permute the columns of the generator matrix $\mathbf{G}$. Then, we apply a Gaussian elimination such that we have a matrix of the form

$$\mathbf{G}^*{}_{k\times n} = \begin{bmatrix}\mathbf{I} & \mathbf{Z} & \mathbf{B}\end{bmatrix} ,$$

where $\mathbf{Z}$ is a $(k \times (s-k))$ matrix. Then, we define the punctured code $C^\bullet$ with generator matrix $\mathbf{G}^\bullet = \begin{bmatrix}\mathbf{I}_k & \mathbf{Z}_{k\times(s-k)}\end{bmatrix}$. The length of the punctured code is determined by s, where $s > dim(C^\bullet) = k$. Now, we search for all linear combinations of at most p rows of $\mathbf{G}^\bullet$ having a Hamming weight of less than or equal p. For these codewords, we compute the Hamming weight of the codeword in C. The parameter p is usually 2 or 3. Values for the parameter s are $k+13, \ldots, k+20$ (see for instance [30]).

Stern. Stern's algorithm [158] is applied to the check matrix $\mathbf{H}_{(n-k)\times n}$. The parameters of the algorithm are ℓ and p. As before, we first randomly permute the columns of $\mathbf{H}$ and perform a Gaussian elimination to get a matrix of the form

$$\mathbf{H}^* = \begin{bmatrix}\mathbf{I}_\ell & \mathbf{0}_{\ell\times(n-k-\ell)} & \mathbf{Z}_{\ell\times k} \\ \mathbf{0}_{(n-k-\ell)\times\ell} & \mathbf{I}_{n-k-\ell} & \mathbf{B}_{(n-k-\ell)\times k}\end{bmatrix}$$

Now we search for codewords in the punctured code $C^\bullet$ with check matrix $H^\bullet = \begin{bmatrix}\mathbf{Z}_{\ell\times k}\end{bmatrix}$. We proceed as follows. We further split the columns of $\mathbf{H}^\bullet$

into two sets, namely $\mathbf{H}^\bullet = \begin{bmatrix}\mathbf{H}_1^\bullet & \mathbf{H}_2^\bullet\end{bmatrix}$. Then the linear combinations of at most p columns are computed for both $\mathbf{H}_1^\bullet$ and $\mathbf{H}_2^\bullet$ and their weight is stored. Then searching for a collision of both weights, allows to search for codewords of weight $2p$. For these candidates, we compute the resulting Hamming weight in C. Usually, the parameter p is 2 or 3 and ℓ is at most 20 (see for instance [158]).

Canteaut-Chabaud. Canteaut and Chabaud [29] have presented a modification of these algorithms. Instead of performing a Gaussian elimination after the random permutation in each iteration, Canteaut and Chabaud use a more efficient updating algorithm. More precisely, only two randomly selected columns are interchanged in each iteration, that is, only one step of a Gaussian elimination has to be performed. Even if this reduces the probability of finding a 'good' subset of the code, this approach leads to considerable improvements as they have shown for several codes in [29].

Work-Factor Estimates

In [30], Chabaud presented work factor estimates for the above-mentioned low-weight-search algorithms. With these estimates one can determine the effort to find an existing codeword with a weight wt in a linear code of length n and dimension k. The work factor estimate for Leon's and Stern's algorithm can be computed as follows (see [30]), where the parameters p, s, and l depend on the punctured code $C^\bullet$.

$$W_{\mathrm{LEON}(p,s)}(n,k,wt) = \frac{k\frac{k}{2}n + \sum_{i=1}^{p}\binom{k}{i}(i-1)\left[(s-k) + n\frac{\sum_{j=0}^{p-i}\binom{s-k}{j}}{2^{s-k}}\right]}{\sum_{i=1}^{p}\frac{\binom{n-w}{s-i}\binom{w}{i}}{\binom{n}{s}}}$$

$$W_{\mathrm{STERN}(p,l)}(n,k,wt) = \frac{\left[(n-k)\frac{(n-k)}{2}n + 2l\binom{k/2}{p}(p-1) + (n-k-l)(2p-1)\frac{\binom{k/2}{p}2}{2^l}\right]a}{\binom{w}{p}\binom{n-w}{k/2-p}\binom{w-p}{p}\binom{n-w-(k/2-p)}{k/2-p}\binom{n-w-(k-2p)}{(n-k-l)-(w-2p)}}, \quad (2.6)$$

with

$$a = \left[\binom{n}{k/2}\binom{n-k/2}{k/2}\binom{n-k}{n-k-l}\right].$$

In the following, we give an example for computing the work factor estimate for finding a codeword of a certain weight in a linear code as we will define it in Section 4.3. As can be seen in Example 2.3, for the chosen parameters l, p, s, the work factor of Stern's algorithm is about 256 times higher than the work factor estimate for Leon's algorithm.

Example 2.3. Consider a binary linear code C with $dim(C) = k = 672$ and length $n = 11520$. Furthermore, assume that a codeword c of weight $wt(c) = 237$ exists. To find such a codeword in C, we get the following estimates:

- Leon's algorithm with $p = 3$ and $s = dim(C) + 12$:

$$W_{\mathrm{LEON}(p,s)}(n, k, wt) = 2^{41}$$

- Stern's algorithm with $p = 3$ and $l = 20$:

$$W_{\mathrm{STERN}(p,l)}(n, k, wt) = 2^{49}$$

2.8 Summary

In this chapter, we have introduced the notation we will follow throughout the remainder of this thesis. Furthermore, we have discussed the basics of symmetric cryptographic primitives (block ciphers, hash functions, and message authentication codes), that are needed for the discussions in the forthcoming chapters. We have defined signed-bit differences which will be applied in the analysis of SHA-1 in Section 4.4. The introduction to finite fields covers all aspects needed for the analysis of the SMASH design strategy in Section 5.1. The section on coding theory, including the description of algorithms to search for codewords with low Hamming weight serves as a starting point in Section 4.3.

3

Analysis Methods for Hash Functions

In this chapter, we review analysis methods for cryptographic hash functions that have been introduced and successfully applied in the past. We will focus on collision attacks and (second) preimage attacks. We will give a general overview of attack methods. Further details will be provided in later chapters.

3.1 Collision Attacks

Definition 3.1. *For a given hash function H, a message pair (m, m^*) with $m \neq m^*$ possessing the same hash value, i.e. $H(m) = H(m^*)$, is called a* colliding message pair*. Alternatively, we say that the messages collide, or simply that we have a collision for H.*

As can be seen in Definition 3.1, we do not impose any conditions on the message pair except that we require that they are not equal. Also we consider unordered message pairs since it is clear that if (m, m^*) is a colliding message pair then also (m^*, m) will collide. For our discussions, both pairs are identical.

3.1.1 Birthday Attacks

For an n-bit hash function the possible hash values are represented by 2^n distinct bit strings. Therefore, when hashing $(2^n + 1)$ different messages (this implies that the message length is greater than n), there must be at least two identical hash values, a collision. In fact, due to the birthday paradox, we do not have to compute 2^n+1 hash evaluations to find a colliding message pair. In the following, we will show how to determine the probability for finding two messages with identical hash values (see also Buchmann [26]). From this, we can then derive the complexity of a birthday attack.

Lemma 3.1. *Assume we are given an n-bit hash function. It follows from the birthday paradox that if we compute the hash value for*

$$k \geq \sqrt{1.386} \cdot 2^{n/2}$$

randomly chosen messages then the probability for finding a colliding message pair in the set of k messages is greater than $1/2$*. This is called a birthday attack and is also referred to as* square-root *attack.*

Proof. Let us denote the number of different hash values by $N = 2^n$ and the number of different messages by k. Furthermore, we denote by p the probability that at least two out of k messages have the same hash value. In other words, p is the probability for a colliding message pair in a set of k different messages. Thus, $q = 1 - p$ is the probability that all k messages have different hash values. We define an elementary event as a tuple: $(h_1, \ldots, h_k) \in \{1, \ldots, N\}^k$. If this event occurs, then the i-th message has the hash value h_i. Thus, in total there are N^k elementary events. We assume that all events are equally probable and hence each elementary event has a probability of $1/N^k$. We are interested in the following event $\bar{C}$: the number of vectors $(h_1, \ldots, h_k) \in \{1, \ldots, N\}^k$ with different coordinates. This corresponds to drawing a sample of size k without replacement of the population of N elements. So, for the first element, we have N choices, for the second element $(N-1)$, etc. Thus, in total there are $N(N-1)\cdots(N-k+1)$ choices. Since we assume that all elementary events are equally probable, we can compute the probability q by multiplying the number of elements in $\bar{C}$ by $1/N^k$. Therefore we get

$$\begin{aligned} q &= \frac{1}{N^k} N(N-1)\cdots(N-k+1) = \frac{1}{N^k} \prod_{i=1}^{k} (N-i+1) \\ &= \frac{1}{N^k} \prod_{i=1}^{k} N\left(1 - \frac{i-1}{N}\right) = \prod_{i=1}^{k} \left(1 - \frac{i-1}{N}\right) \\ &= \prod_{i=1}^{k-1} \left(1 - \frac{i}{N}\right) . \end{aligned}$$

Since it holds that $1 + x \leq e^x$ for all real numbers x, we can write

$$q \leq \prod_{i=1}^{k-1} e^{-\frac{i}{N}} = e^{-\sum_{i=1}^{k-1} \frac{i}{N}} = e^{\frac{-k(k-1)}{2N}} .$$

Since we want that $q \leq 1/2$, we set

$$q \leq e^{\frac{-k(k-1)}{2N}} \leq 1/2 . \tag{3.1}$$

Inequality (3.1) can be rewritten to get

$$k(k-1) \geq 2N \ln 2$$

resulting in

$$k^2 - k - 2N \ln 2 \geq 0 . \tag{3.2}$$

In order to satisfy (3.2), k is chosen as follows

$$k \geq \frac{1+\sqrt{1+8N\ln 2}}{2} > \frac{\sqrt{1+8N\ln 2}}{2} > \sqrt{\frac{8\ln 2}{4}N} \approx \sqrt{1.386}\cdot 2^{n/2} \ .$$

This implies that $q \leq 1/2$ and hence $p \geq 1/2$. In other words, the probability p to find a colliding message pair is greater than $1/2$, if we compute the hash value for $k \geq \sqrt{1.386}\cdot 2^{n/2}$ randomly chosen messages. □

The birthday attack is a generic attack which considers a hash function as a black box and hence, can be applied successfully to any hash function. A birthday attack can be implemented as has been shown by Yuval in [178] (see also [116, Algorithm 9.92]):

1. Randomly pick a message m and compute $H(m)$
2. Update the list L
   - if $H(m)$ is already in the list L, a colliding message pair has been found
   - else save the pair $(H(m), m)$ in the list L and go back to step 1

After performing about $\sqrt{1.386}\cdot 2^{n/2}$ hash evaluations, we know from the birthday paradox that we can find a matching entry (a collision) with a probability greater than $1/2$. Yuval's approach has a memory requirement of about $2^{n/2}$ to store the list L. A memoryless variant of Yuval's birthday attack has been presented by Quisquater and Delescaille in [148]. An efficient parallel variant of this algorithm has been proposed by van Oorschot and Wiener in [164, 165].

Note that in a birthday attack, an attacker has full control over the messages. Hence, this method enables an attacker to construct *meaningful collisions*. We speak of a meaningful collision if the colliding messages have meaningful content such as readable text and do not just look like random strings.

As described in Section 2.3.3, there exist variants of a collision, namely pseudo-collisions and near-collisions. For a near-collision, we can also apply the birthday paradox.

Lemma 3.2. *Assume we are given an n-bit hash function H and we want to find a near-collision such that $H(m) \oplus H(m^*) \in \Delta$, where Δ denotes the set of all n-bit vectors with a Hamming weight of d, i.e. $\Delta = \{\delta \in \{0,1\}^n : wt(\delta) = d\}$. It follows from the birthday paradox that after computing the hash value for*

$$k \geq \sqrt{1.386 \cdot \frac{2^n}{\binom{n}{d}}}$$

randomly chosen messages, the probability to find two messages m and m^* in this set of k messages such that $H(m) \oplus H(m^*) \in \Delta$ is greater than $1/2$.

Proof. The proof proceeds along the same lines as the proof of Lemma 3.1, except that we are interested in the probability p that two messages m and m^* lead to a near collision. Thus, $q = 1 - p$ is the probability that all hash values $(h_1, \ldots, h_k) \in \{1, \ldots, N\}^k$ satisfy $h_i \oplus h_j \notin \Delta$ for $i \neq j$. For h_1 we have N possibilities, for h_2 we have $N - \binom{n}{d}$ possibilities, where $\binom{n}{d}$ is the cardinality of Δ. So we end up with

$$\begin{aligned} q &= \frac{1}{N^k} N \left(N - \binom{n}{d}\right) \left(N - 2\binom{n}{d}\right) \cdots \left(N - (k-1)\binom{n}{d}\right) \\ &= \prod_{i=1}^{k-1} \left(1 - \frac{i\binom{n}{d}}{N}\right) . \end{aligned}$$

This leads to

$$k \geq \sqrt{1.386 \cdot \frac{2^n}{\binom{n}{d}}}$$

□

It follows from Lemma 3.2 that finding a near-collision is less complex than finding a collision. Note that in the remainder of this thesis, we will omit the factor $\sqrt{1.386}$ if we talk about the complexity of a birthday attack; we simply say that a birthday attack has a complexity of about $2^{n/2}$ hash function evaluations.

In a birthday attack, we are searching for a pair of messages (m, m^*) with $m \neq m^*$ such that $H(m) \oplus H(m^*) = 0$. Wagner introduced a k-dimensional generalization of the birthday problem in [167]. He presented how to find k messages, for $k \geq 2$ (if $k = 2$, we have the standard birthday problem), such that their XOR sum equals zero. He showed for instance that it is possible to find four different messages m^1, m^2, m^3, m^4 such that $H(m^1) \oplus H(m^2) \oplus H(m^3) \oplus H(m^4) = 0$ with about $2^{n/3}$ hash evaluations and $2^{n/3}$ storage. For further details and for applications of the generalized birthday attack, we refer to [167].

3.1.2 Shortcut Attacks

So far, we discussed the birthday attack, where the hash function is considered as a black box. If, for a concrete hash function, the internal structure can be exploited for collision attacks with a complexity below the birthday bound ($2^{n/2}$), we speak of *shortcut attacks*. One method for shortcut attacks on hash functions is to apply techniques from differential cryptanalysis. Differential cryptanalysis was invented in 1989 by Biham and Shamir, which were the first that gave a general description of this method for the cryptanalysis of DES [19]. Similar techniques had been used already for instance by Murphy in [120]. Also the designers of DES claim to have known about differential cryptanalysis already back in the 1974 when the algorithm was designed [34]. Differential cryptanalysis is a general method in the cryptanalysis of block ciphers and stream ciphers, and also applicable to hash functions. The main idea of differential cryptanalysis in hash functions is to force certain bits to be equal in both messages m and m^*.

The more equalities we impose, the more likely these equalities will propagate, eventually leading to equal hash values (a collision).

As opposed to block ciphers, hash functions allow some additional techniques to improve on this idea. Firstly, an attacker can impose additional conditions on the values of certain bits in both messages, in order to increase the probability that the equalities propagate. For most block ciphers, this is not promising since message blocks are usually XORed with an unknown whitening key before being processed. Secondly, also conditions on the bits of internal variables in the hash function can be set. Enumerating messages which fulfill such conditions is relatively easy as long as these conditions are imposed in the first steps, where the state variables do not depend on all message words (nor on any secret key, as in block ciphers).

Based on the notion of collisions it is natural to apply techniques from differential cryptanalysis for the analysis of cryptographic hash functions. The attacker tries to find a pair of messages m and $m^* \neq m$ that possesses the same hash value. Since $m^* \neq m$, we can define the (XOR) difference between both messages: $m' = m \oplus m^*$ and analyze how this difference propagates throughout (parts) of the hash function. Clearly, we are interested in a difference m' that propagates with a high probability. Therefore, the aim is the same as in the cryptanalysis of block ciphers, namely find input differences defining a characteristic with high probability. Nevertheless, a high probability characteristic for a hash function needs to result in a zero output difference such that m and m^* collide. This is not the case for a block cipher. The probability of a characteristic is the fraction of message pairs that follow the characteristic. If a message pair follows the characteristic, it is called a *right pair*. In our context, we can equivalently call it a colliding pair.

3.1.3 Multicollisions

Multicollision attacks have been introduced by Joux in [74]. It is a generic attack, which exploits the iterative structure of a hash function. We start with giving the definition of an r-collision, *cf.* [74].

Definition 3.2. *An r-tuple of messages $(m^1, \ldots, m^r)$ leading all to the same hash value, i.e. $H(m^1) = \cdots = H(m^r)$, is called an r-collision. In particular, if $r = 2$, we say 2-collision, which is an ordinary collision.*

The main result of [74] is that constructing 2^t-collisions is only slightly more expensive than constructing a 2-collision. To explain this in more detail, we define a *collision finding machine C* (see also [74]).

Definition 3.3. *A collision finding machine C for the compression function f, is an algorithm that for a given chaining value h_i returns two message blocks m_i^1 and m_i^2, with $m_i^2 \neq m_i^1$, such that $f(h_i, m_i^1) = f(h_i, m_i^2)$.*

Note that the collision finding machine C may simply implement a birthday attack or any other shortcut collision attack on the underlying compression function f. For the remaining discussion, we assume that it implements a birthday

attack, *i.e.* collisions can be found with a complexity of about $2^{n/2}$ evaluations of the compression function. With this collision finding machine, we now illustrate how one can find a 4-collision. We call C with the initial value iv and obtain two message blocks m_1^1, m_1^2 that collide for the compression function f, *i.e.* $f(iv, m_1^1) = f(iv, m_1^2)$. We call C again but now with the chaining value resulting from the first call, namely $f(iv, m_1^1)$. This results in two colliding message blocks m_2^1, m_2^2. Therefore, we have the following four messages which all possess the same hash value (a graphical illustration is given in Figure 3.1):

$$m^1 = m_1^1 \| m_2^1 \qquad m^2 = m_1^1 \| m_2^2$$
$$m^3 = m_1^2 \| m_2^2 \qquad m^4 = m_1^2 \| m_2^2$$

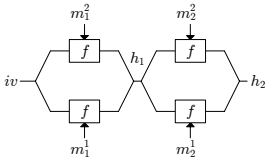

Figure 3.1: Graphical illustration of a 4-collision.

Therefore, the complexity to find a 4-collision is only twice the complexity of finding a normal collision, *i.e.* $2 \cdot 2^{n/2}$. It is easy to see that calling the collision finding machine t times results in a 2^t-collision. The complexity for doing so is about $t \cdot 2^{n/2}$ evaluations of the compression function. Note that since all 2^t colliding messages are of the same length (t blocks), appending the final block including the length encoding does not have any impact.

3.1.4 Expandable Messages

In [80], Kelsey and Schneier introduced *expandable messages*. An expandable message is a kind of multicollision with the difference that it is a set of colliding messages of different length, prior to applying the final message block including the length encoding. As defined in [80], an (a, b)-expandable message can consist of between a and b message blocks all leading to the same hash value. Before we describe how one can construct expandable messages, we give the definition of a *fixed point* in the compression function as has been described by Preneel *et al.* in [144].

Definition 3.4. *We say that we can construct a fixed point for the compression function f, if for a given chaining value h_i, we can find a pair (h_{i-1}, m_i) such that $h_i = f(h_{i-1}, m_i) = h_{i-1}$.*

Most of the compression functions in practical hash functions use the Davies-Meyer construction, *i.e.* the chaining value h_{i-1} is added to the output of the block cipher $E_{m_i}(h_{i-1})$: $f(h_{i-1}, m_i) = E_{m_i}(h_{i-1}) \oplus h_{i-1}$ (see also Section 2.3.5). For this kind of compression functions it is easy to find a fixed point (see also

Preneel *et al.* [144]): select an arbitrary message block m_i, and compute the inverse of the block cipher for the ciphertext 0 to get $h_{i-1} = E_{m_i}^{-1}(0)$. Then we know that

$$h_i = f(h_{i-1}, m_i) = E_{m_i}(h_{i-1}) \oplus h_{i-1} = 0 \oplus h_{i-1} = h_{i-1} \ . \tag{3.3}$$

Note that for constructing the fixed point in this way, we have no control over the chaining value h_{i-1}. Now we can describe Dean's method [44] for constructing expandable messages based on fixed points. Constructing an expandable message based on fixed points consists basically of two steps: (1) for a given iv, we would like to find a message that leads to chaining value h_i, and (2) for exactly this value, we want to find a fixed point. We proceed as follows. Generate about $2^{n/2}$ fixed points as in (3.3) and store the according pairs (h_i, m_i) in the list L_{FP}. Then compute for about $2^{n/2}$ arbitrary messages and the initial value iv the chaining value h_i and store the pair (h_i, m_i) in the list L_{iv}. Due to the birthday paradox, we find a matching entry for h_i in both lists with a good probability. Then we have constructed an expandable message and the length of this expandable message is only restricted by the maximum message length that can be hashed with the considered hash function. Assuming that we have a hash function that can process up to 2^k message blocks, we can construct a $(1, 2^k)$-expandable message with a complexity of about $2 \cdot 2^{n/2} = 2^{n/2+1}$ evaluations of the compression function. To construct a message consisting of a specific number of blocks, say t, we simply append $(t-1)$ equal message blocks that determine the fixed point to the first message block. This adds only negligible work.

Kelsey and Schneier presented an alternative technique for constructing expandable messages. Their method is inspired by the multicollision attack of Joux in [74]. In [74], a huge set of colliding messages of equal length is constructed. Kelsey and Schneier do the same except that they construct a set of messages of different length. One fact they exploit is that the complexity for finding a collision for two messages of different length has the same order of magnitude as a birthday attack. To find a collision between two messages m^1 and m^2, where m^1 consists of one block and m^2 consists of α blocks, we can proceed as follows. Firstly, compute about $2^{n/2}$ chaining values for randomly chosen message blocks m_i^1 and store the pairs (h_i, m_i^1) in the list L_{SB}. Then take an arbitrary message block u and compute the chaining value $h_{\alpha-1}$ for $\alpha - 1$ times the block u. Compute about $2^{n/2}$ hash values for the compression function with chaining value $h_{\alpha-1}$ and store the pair (h_j, m_j) in the list $L_{\alpha\mathrm{B}}$. Due to the birthday paradox, we find a matching entry in both lists with good probability. If so, we have found two colliding messages, where one consists of a single and the other of α blocks, namely

$$H(iv, m^1) = H(iv, (\overbrace{\underbrace{u \| \ldots \| u}_{\alpha-1 \text{ blocks}} \| m_j}^{m^2})) \ ,$$

assuming that no final block including the length encoding of the messages is applied. The complexity for constructing the colliding messages of different length

is about $(\alpha-1)+2^{n/2+1}$ evaluations of the compression function. For the further discussion we denote this method by $(m^1, m^2, h_{\text{temp}}) = \text{FindCollision}(\alpha, h_i)$ (see also [80]).

The described methods can be used to construct $(k, k+2^k-1)$-expandable messages. We construct a list C storing two entries at each index: the left entry is the single-block message m^1, and the second entry is the $(2^{k-1}+1)$-block message m^2. Algorithm 3.1 shows how the list C is constructed. In Figure 3.2, we give a graphical illustration of the list C.

Algorithm 3.1 Construct a $(k, k+2^k-1)$-expandable message [80, Page 480]

Input: initial value iv and variable k

Output: list C, a $k \times 2$ array, where the first entry $C[k][0]$ corresponds to a single-block message, and the second entry $C[k][1]$ to a $(2^{k-1}+1)$-block message. Both messages result in the same hash value.

set $h_{\text{temp}} = iv$
for $i = 0$ to $k-1$ **do**
  $(m^1, m^2, h_{\text{temp}}) = FindCollision(2^i+1, h_{\text{temp}})$
  $C[k-i-1][0] = m^1$
  $C[k-i-1][1] = m^2$
end for

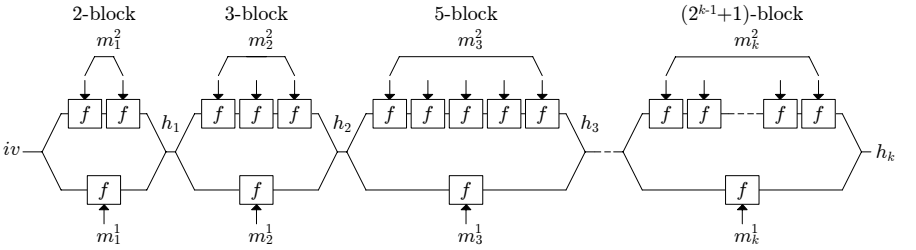

Figure 3.2: A graphical illustration of the $(k, k+2^k-1)$-expandable message stored in list C generated by Algorithm 3.1.

It is easy to see that the list C is a $(k, k+2^k-1)$-expandable message. If we look at Figure 3.2 then concatenating the single-block messages m_i^1 results in a k-block message and concatenating the $(2^{k-1}+1)$-block messages m_i^2 for $0 \le i \le k-1$ results in a message consisting of $\sum_{i=0}^{k-1}(2^i+1) = k + \sum_{i=0}^{k-1} 2^i = k + 2^k - 1$ blocks. Computing the complete list C requires about $2^k + k \cdot 2^{n/2+1}$ evaluations of the compression function.

3.1.5 Collisions for Cascaded Hash Functions

With the observation that constructing multicollisions is only slightly more complex than constructing an ordinary collision (see Section 3.1.3), Joux answered the following open question: does cascading hash functions increase its security

as expected by the size of the cascaded hash value? Assume, we are given two hash functions: H_L with an n_L-bit hash value and H_R with an n_R-bit hash value, where at least one is an iterated hash function. If we cascade these hash functions, *i.e.* we initialize them with their initial value (iv_L and iv_R) and concatenate both hash values, we have in total a $(n_L + n_R)$-bit hash value. The hash value is then computed as $h = H_L(iv_L, m) \| H_R(iv_R, m)$ (see Figure 3.3).

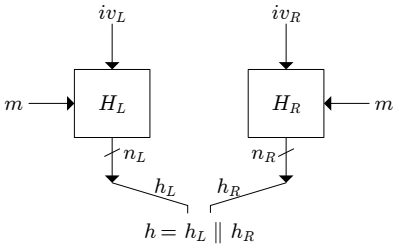

Figure 3.3: Cascading two hash functions H_L and H_R.

At first sight, we would expect a collision resistance of $2^{(n_L+n_R)/2}$. However, Joux showed that the security of a cascaded hash function is in fact only slightly larger than what we can expect from a single hash function. To demonstrate this, we assume that $n_L \leq n_R$ (we refer to [74] for the case $n_L \geq n_R$) and that H_L is an iterated hash function. Attacking this cascaded hash construction works as follows. We set $t = n_R/2$ and construct a $2^{n_R/2}$-collision for H_L as described in Section 3.1.3. This has a complexity of $n_R/2 \cdot 2^{n_L/2} \approx n_R \cdot 2^{n_L/2}$. So, now we have $2^{n_R/2}$ messages all having the same hash value for H_L. Since we assumed that $n_L \leq n_R$, we can simply apply a birthday attack on this set of messages and with reasonable probability, we will find a message pair within this set that also collides under H_R. Increasing the probability can be achieved by making more calls to the collision finding machine. Therefore, the total complexity for finding a collision in the cascade of H_L and H_R can be summarized as $n_R \cdot 2^{n_L/2} + 2^{n_R/2}$, which is significantly less than $2^{(n_L+n_R)/2}$. Note that for this attack to work, the hash function H_R does not have to be an iterated hash function. It can be any other hash function. In general, this attack works as long as one of the functions is an iterated hash function such that we can construct multicollisions. In [74], Joux also presented attacks on cascades of more than two hash functions.

3.2 (Second) Preimage Attacks

Before we discuss different (second) preimage attacks, we give an informal definition of a second preimage and a preimage. For formal definitions, we refer to Rogaway and Shrimpton [155].

Definition 3.5. *For a given hash function H, and for a given message m, a second message $m^* \neq m$ is called a* second preimage *of m, if the hash value $H(m^*)$ equals the hash value $H(m)$.*

This definition does not impose any conditions on the second preimage m^* of m except that both messages must not be equal. Later in this thesis, we will consider second preimages for a message m that have special properties such as the length of m and m^* is identical. Nevertheless, we use this general definition of a second preimage for discussing second preimage attacks.

Definition 3.6. *For a given hash function H, and for a given hash value h, a message m is called a* preimage *of h, if $H(m) = h$.*

Also in this case, no conditions are imposed on the message m.

3.2.1 Shortcut Attacks

In Section 3.1.1, we reviewed the birthday attack for finding colliding message pairs. For (second) preimages there exists a similar approach, namely *meet-in-the-middle* attacks (MITM), which can be considered as shortcut attacks. These attacks are also based on the birthday paradox. As opposed to birthday attacks for finding collisions, for MITM attacks the hash function is not considered as a black box. The attacks are applied to the internal values of a hash function such as the intermediate chaining values. MITM attacks are also referred to as *chaining attacks* in [116]. We will explain their working principle by an example of an iterated hash function. Assume, we are using an n-bit hash function with a compression function that can be easily inverted (for instance a block-cipher-based construction by omitting the mode of operation): $h_i = E_{m_i}(h_{i-1})$. Note that as discussed in Section 2.3, a hash function employing such a compression function is inherently insecure. Assume, we are given a two-block message $m^1 = m_1^1 \| m_2^1$ with according hash value h^1, for which we want to construct a second preimage m^2 of the same length. We can now proceed as follows to find such a message with a complexity of about $2 \cdot 2^{n/2}$ evaluations of the compression function. Firstly, we compute for about $2^{n/2}$ randomly chosen message blocks m^i one iteration forward, *i.e.* $h_i = E_{m^i}(h_0)$ and store the pairs (h_i, m^i) in the list L_1. Then, we compute for about $2^{n/2}$ message blocks m^j and the hash value h^1 one iteration backwards, *i.e.* $h_j = E_{m^j}^{-1}(h^1)$ and store the pairs (h_j, m^j) in the list L_2. Then, based on the birthday paradox, we will have with a good probability a matching entry in both list[1]: $h_i = h_j$. If so, then we have found a second preimage $m^2 = m^i \| m^j$ for the message m^1 after about $2 \cdot 2^{n/2}$ evaluations of the compression function instead of the 2^n expected from the n-bit hash value. Note that for constructing expandable messages in Section 3.1.4, we also applied a MITM approach. We emphasize that this example is just used to explain the working principle of MITM attacks. In general, inverting the compression function cannot be done efficiently and therefore, the complexity for (second) preimage attacks is about 2^n evaluations of the hash function.

Differential cryptanalysis as discussed in Section 3.1.2 can also be exploited for finding second preimages, with the restriction that the attacker cannot choose

[1] As has been shown by Nishimura and Sibuya in [125] the probability for finding a matching value between two sets is larger than the probability of success in the classical birthday problem described in Section 3.1.1. However, the asymptotic complexity is the same for both problems.

a message of his choice. However, applying differential cryptanalysis remains a promising approach. If we consider preimage attacks, it is at first sight not clear how one could exploit differential cryptanalysis for inverting the hash function. Nevertheless, in Section 5.1.4, we show that if a hash function possesses certain differential properties, finding preimages can also be successful by exploiting differential cryptanalysis.

3.2.2 Long-Message Second-Preimage Attacks

Kelsey and Schneier [80], present a generic second preimage attack on iterated hash functions that follow the Merkle-Damgård design principle. The result is that for an n-bit hash function, second preimages can be found in much less than the expected 2^n evaluations of the hash function. However, very long messages are required for this kind of generic attack. In the following, we will describe the basic attack strategy. It can be seen as a generalization of the multicollision attack of Joux in [74]. As noted in [80], the attack strategy was invented in the PhD thesis of Dean [44].

We start with repeating long-message attacks for second preimages [116, Chapter 9, Fact 9.37]. The long-message attack works for iterated hash functions without Merkle-Damgård strengthening, *i.e.* the last message block including the length encoding of the entire message is omitted. Assume, we are given a message consisting of 2^R+1 message blocks for which we want to find a second preimage. From the number of message blocks we know, that there are 2^R chaining values h_i. For constructing a second preimage, we need to find a message block m_{link} such that $f(iv, m_{\text{link}}) = h$ matches one of these 2^R chaining values. Due to the number of chaining values, we can expect to find a match after trying about 2^{n-R} message blocks since for each message block we have a probability of $2^{-(n-R)}$ that it matches one of the chaining values of the (2^R+1)-block message. So the result is a second preimage with fewer blocks. It is easy to see that the larger R the less the complexity for finding a second preimage (note that this was already pointed out in the PhD thesis of Merkle [117]). However, if Merkle-Damgård strengthening is applied, then the attack does not succeed anymore since the last block including the length encoding defeats this attack.

Kelsey and Schneier show how MD strengthening can be bypassed by exploiting expandable messages. Assume that we are given a large target message m consisting of $2^k + k + 1$ message blocks for which we want to construct a second preimage. The first step in the second preimage attack is to construct the list C based on Algorithm 3.1 given in Section 3.1.4, with iv and k as input and store the resulting chaining value, *i.e.* h_k (see also Figure 3.2). Then, we store the intermediate chaining values of the target message in the list $h[j]$, where $k + 1 \leq j \leq 2^k + k + 1$. Note that we do not need to store the first k chaining values since the expandable message in C consists of at least k blocks and hence we cannot construct a message consisting of less than k blocks. For the resulting chaining value of the expandable message h_k, we try to find a message block m_{link} that maps h_k to one of the intermediate chaining values of the target message. More precisely, we choose message blocks such that $f(h_k, m_{\text{link}}) = h[j]$

for $k+1 \leq j \leq 2^k + k + 1$. Once we found the message block m_{link}, we proceed as follows. We know that we found m_{link} for the intermediate chaining value in iteration j. Therefore, we construct an expandable message m_{expand} from list C consisting of $j-1$ blocks. Since the constructed j-block message has the same intermediate chaining value as the target message at iteration j (a collision), we can simply append the remaining $2^k + k + 1 - j$ blocks of the target message to have a second preimage m^* of the same length:

$$m^* = \underbrace{m_{\text{expand}}}_{j-1 \text{ blocks}} \| \underbrace{m_{\text{link}}}_{1 \text{ block}} \| \underbrace{m_{j+1} \| \ldots \| m_{2^k+k+1}}_{2^k+k+1-j \text{ blocks}}$$

Since the target message and the second preimage are of the same length, adding the final block including the length encoding has no impact on the attack.

The complexity for the attack consists of the effort to construct the expandable message (list C) and the effort to find the message block m_{link}. Finding the message block m_{link} has the same complexity as in the original second preimage long-message attack, namely $2^{n-k}+1$. Therefore, the overall complexity is about $(2^k + k \cdot 2^{n/2+1}) + 2^{n-k+1}$ evaluations of the compression function. As can be seen, the dominating term comes from finding the message m_{link}. Therefore, we have again: the longer the target message the lower the complexity for finding a second preimage. We give a demonstrative example by applying the second preimage attack on for instance SHA-1 [128], which produces a 160-bit hash value (*i.e.* $n = 160$). Furthermore, we assume that we are given a target message m consisting of $2^{54} + 54 + 1$ message blocks, *i.e.* $k = 54$. Then finding a second preimage has a complexity of about $54 \cdot 2^{81} + 2^{160-54+1} \approx 2^{107}$ evaluations of the compression function.

3.2.3 (Second) Preimages for Cascaded Hash Functions

Joux showed in [74] how to exploit multicollisions for (second) preimage attacks. As a result, also (second) preimage attacks for cascaded hash functions can be found significantly more efficient than expected from the concatenated hash value if the cascade contains at least one iterated hash function. For the case where $n_L \leq n_R$, (second) preimages can be found with a complexity of about $n_R \cdot 2^{n_L/2} + 2^{n_L} + 2^{n_R}$. The attack works along the same lines as the collision attack discussed in Section 3.1.5. Assume that H_L is an iterated hash function. We set $t = n_R$ and construct a 2^{n_R}-collision for H_L. This has a complexity of about $n_R \cdot 2^{n_L/2}$. Now, we select a last block that maps the last chaining value of the 2^{n_R}-collision to the expected value of H_L. This requires about 2^{n_L} applications of the compression function. Therefore, we have now 2^{n_R} messages that all lead to the expected value of H_L. Since $t = n_R$ and we assumed that $n_L \leq n_R$, we know with reasonable probability that at least one in this set of messages also matches the expected hash value of H_R. The probability for finding a (second) preimage can be increased by increasing the value of t.

3.3 Summary

In this chapter, we have discussed general cryptanalytic methods to analyze the security of hash functions. We have focused on collision, second preimage, and preimage attacks. The most common tool in the cryptanalysis of hash functions is differential cryptanalysis. This technique, initially invented for the analysis of block ciphers, can be successfully exploited for attacks on hash functions.

New general attack methods such as multicollision attacks and second preimage attacks for long messages have been discussed. As a result, multicollision attacks show that the cascading of hash functions does not increase the security margin as one would expect from the resulting size of the hash value. Second preimage attacks for long messages can be considered as a generalization of multicollision attacks. The result is that for very long messages, second preimages can be constructed with much less than 2^n evaluations of the hash function, which is the complexity we expect from an n-bit hash value. Even if these second preimage attacks are not practical, they clearly show some structural limitations of iterated hash functions following the Merkle-Damgård design principle.

We note that another general attack strategy, called *herding attacks* has been presented by Kelsey and Kohno in [79]. Herding attacks are a special form of preimage attacks that work for all iterated hash functions. The original work in [79] has been later on extended to cascades of hash functions by Dunkelman and Preneel in [53].

4

Collision Attacks on SHA-1

In this chapter, we review several collision attack strategies that led to the collision attack on SHA-1 presented by Wang *et al.* in [172]. Firstly, in Section 4.1, we describe the hash function SHA-1 and the linearized variant L-SHA-1, which we use for our analysis. Secondly, we review the attack strategies on SHA-0 and their extensions to SHA-1 in Section 4.2. We discuss in detail the collision attack on SHA-1 presented by Wang *et al.* In Section 4.3, we show how to exploit coding theory to construct a high probability L-characteristic. With our tools, we find the same L-characteristic as found by Wang *et al.* As we will see, this L-characteristic determines the attack complexity of the collision attack on SHA-1. Therefore, Section 4.4 is dedicated to a thorough analysis of its probability. Based on this analysis, we will update the collision attack complexity. Before we summarize this chapter in Section 4.6, we describe a generalization of the attack by Wang *et al.* in Section 4.5. This chapter includes work presented in [114, 138]. The main contribution of this chapter is that we show how to derive a high probability L-characteristic by exploiting coding theory and how we can accurately compute the corresponding probability.

4.1 The Hash Function SHA-1

SHA-1 is an iterated hash function (*cf.* Section 2.3) that computes a 160-bit hash value for messages of length less than 2^{64} bits, *cf.* [128]. As most iterated hash functions, SHA-1 applies MD strengthening as described in Section 2.3. The compression function processes input message blocks of 512 bits and produces a 160-bit chaining value. The compression function of SHA-1 basically consists of two parts: the message expansion and the state update transformation. The chaining variable h_{i-1} (iv in the first iteration) is added to the output of the state update transformation (Davies-Meyer mode of operation). This is graphically illustrated in Figure 4.1. In the following, we will describe both parts in detail.

4.1.1 SHA-1 Message Expansion

In SHA-1, the message expansion is defined as follows. A single 512-bit input message block is represented by 16 words of 32 bits each, denoted by M_i, with

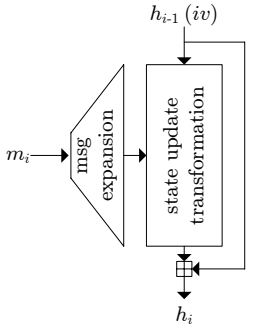

Figure 4.1: The SHA-1 compression function.

$0 \leq i \leq 15$. The message input is linearly expanded into 80 32-bit words W_i defined as follows:

$$W_i = \begin{cases} M_i & \text{for } \; 0 \leq i \leq 15 \\ (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \lll 1 & \text{for } 16 \leq i \leq 79 \end{cases} \tag{4.1}$$

4.1.2 SHA-1 State Update Transformation

The state update transformation starts from a fixed initial value iv for 5 32-bit registers $A, B, \ldots, E$ (also referred to as state variables) and updates these registers in 80 steps ($i = 0, \ldots, 79$) by using the word W_i and the step constant K_i in step i. One step of the state update transformation is defined as (see also Figure 4.2):

$$\begin{aligned} A_{i+1} &= (A_i \lll 5) \boxplus W_i \boxplus f(B_i, C_i, D_i) \boxplus K_i \\ B_{i+1} &= A_i \\ C_{i+1} &= B_i \ggg 2 \\ D_{i+1} &= C_i \\ E_{i+1} &= D_i \end{aligned} \tag{4.2}$$

The function f depends on the step number: steps $i = 0, \ldots, 19$ use the *IF-function* referred to as f_{IF} and steps $i = 40, \ldots, 59$ use the *MAJ-function* referred to as f_{MAJ}. The remaining steps, use a 3-input XOR referred to as f_{XOR}. The Boolean functions are defined as follows:

$$\begin{aligned} f_{\mathrm{IF}}(B, C, D) &= (B \wedge C) \oplus (\neg B \wedge D) \\ f_{\mathrm{MAJ}}(B, C, D) &= (B \wedge C) \oplus (B \wedge D) \oplus (C \wedge D) \\ f_{\mathrm{XOR}}(B, C, D) &= B \oplus C \oplus D \end{aligned}$$

For the sake of completeness, we give the step constants K_i and the initial values, even though they have no impact on our analysis of SHA-1. There are

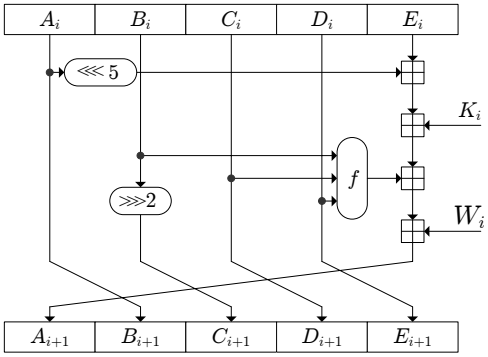

Figure 4.2: One step of the state update transformation of SHA-1.

four different constants (in hexadecimal representation):

$$\begin{aligned} K_i &= \texttt{5A827999} \quad \text{for } i = \ 0, \ldots, 19 \\ K_i &= \texttt{6ED9EBA1} \quad \text{for } i = 20, \ldots, 39 \\ K_i &= \texttt{8F1BBCDC} \quad \text{for } i = 40, \ldots, 59 \\ K_i &= \texttt{CA62C1D6} \quad \text{for } i = 60, \ldots, 79 \end{aligned}$$

The initial values for the 5 32-bit registers (state variables) are specified as:

$$\begin{aligned} A_0 &= \texttt{67452301} \quad B_0 = \texttt{EFCDAB89} \quad C_0 = \texttt{98BADCFE} \\ D_0 &= \texttt{10325476} \quad E_0 = \texttt{C3D2E1F0} \end{aligned}$$

After the last application of the state update transformation, the initial register values are added to the final values of the state variables (feed forward), and the result is either the input to the next iteration or the final hash value.

4.1.3 Linearized Variant L-SHA-1

For the analysis of SHA-1, we define a linearized variant referred to as L-SHA-1 as has been done similarly by Chabaud and Joux for the analysis of SHA-0 [31], and by Rijmen and Oswald for the analysis of SHA-1 [150]. Note that for the remainder of this chapter all vectors and matrices are over $GF(2)$. A single 512-bit input message block is denoted by the row vector $\mathbf{m}$ and the expanded message is denoted by the 2560-bit row vector $\mathbf{w}$. Since the message expansion is linear, it can be described by a 512×2560 matrix $\mathbf{M}$ such that $\mathbf{w} = \mathbf{mM}$. The message expansion starts with a copy of the message, *cf.* (4.1). Hence, there exists a $512 \times 32(80 - 16)$ matrix $\mathbf{F}$ such that $\mathbf{M}$ can be written as:

$$\mathbf{M}_{512\times 2560} = \begin{bmatrix} \mathbf{I} & \mathbf{F} \end{bmatrix} , \tag{4.3}$$

where $\mathbf{I}$ is the 512×512 identity matrix.

The state update transformation is linearized by approximating f_{IF} and f_{MAJ} by f_{XOR}, and by replacing the modular addition by the XOR-operation. One step of the linearized state update transformation is defined as follows:

$$\begin{aligned} A_{i+1} &= (A_i \lll 5) \oplus W_i \oplus f_{\mathrm{XOR}}(B_i, C_i, D_i) \oplus K_i \\ B_{i+1} &= A_i \\ C_{i+1} &= B_i \ggg 2 \\ D_{i+1} &= C_i \\ E_{i+1} &= D_i \end{aligned}$$

Thus, the linear state update transformation can be described by a 2560×160 matrix $\mathbf{S}$, a 160×160 matrix $\mathbf{T}$, and a vector $\mathbf{k}$. The matrix $\mathbf{T}$ describes the transformation of the initial register values and the vector $\mathbf{k}$ represents the transformation of the four step constants. With these matrices the linearized compression function producing the output $\mathbf{o}$ from the input message $\mathbf{m}$ can be described as follows:

$$\mathbf{o} = \mathbf{mMS} \oplus \mathbf{ivT} \oplus \mathbf{k} \tag{4.4}$$

4.2 The Attack Strategy

In this section, we chronologically review the basic strategy for collision attacks on SHA-1. We do not describe the attacks in detail but we highlight the most important facts that led to the collision attack on SHA-1 by Wang *et al.* [172].

We start with a review of the cryptanalysis of SHA-0 presented by Chabaud and Joux in [31] and the improvement of their attack by Biham and Chen in [17]. Independently to the cryptanalysis of SHA-0 by Biham and Chen, Rijmen and Oswald presented the first extension of the attack by Chabaud and Joux to SHA-1 [150]. While Biham *et al.* presented further improvements on SHA-0 and step-reduced variants of SHA-1 in [18], Wang *et al.* announced a practical collision attack on SHA-0 [174] and the first collision attack on the full SHA-1 [172].

4.2.1 Chabaud and Joux

Chabaud and Joux presented in [31] the first differential collision attack on SHA-0, the predecessor of SHA-1. SHA-0 was standardized in 1993 and withdrawn and replaced by SHA-1 without justification in 1995. The only difference to SHA-1 is that the message expansion in (4.1) is defined without left rotation. Chabaud and Joux used a linearized variant of SHA-0 to show how an injected single-bit *disturbance*[1] can be corrected by several consecutive single-bit *corrections*. This is referred to as a *local collision*. Since we intensively look at local collisions in Section 4.4, we formally define a local collision as follows. Note that the following definition is valid for SHA-0 and SHA-1.

[1] In [31] the term *perturbation* is used. However, we will use the synonym *disturbance* throughout this thesis.

Definition 4.1. *(Local Collision) A local collision for SHA-0 and SHA-1 consists of a single-bit disturbance and five consecutive single-bit corrections:*

$$\begin{aligned} \textit{disturbance:} \quad & W'_i = 2^j \\ \textit{corrections:} \quad & W'_{i+1} = 2^{j+5} \\ & W'_{i+2} = 2^j \\ W'_{i+3} = W'_{i+4} = W'_{i+5} = & 2^{j-2} \end{aligned}$$

It follows directly from Definition 4.1 that a disturbance injected after step $i = 74$ cannot be corrected within the remaining steps. In this case the difference in the state variables $A'_{80}, \ldots, E'_{80}$ is non-zero.

Example 4.1. We show an example for a local collision for the linearized variant L-SHA-1 with a disturbance in the least-significant bit. In step i, the differences in the state variables $A'_i, \ldots, E'_i$ equal zero and $W'_i = 2^0$. By injecting the corrections as given in Definition 4.1, we get a collision in step $i + 6$, *i.e.* the differences in state variables $A'_{i+6}, \ldots, E'_{i+6}$ are zero. The according difference pattern through the linearized state update transformation is shown in Table 4.1.

Table 4.1: Difference pattern of a local collision for L-SHA-1 with a disturbance in the least-significant bit. The values are listed in hexadecimal notation. For the sake of readability, zero values are denoted by a dash.

step	W'_i	A'_i	B'_i	C'_i	D'_i	E'_i
i	-------1	--------	--------	--------	--------	--------
$i+1$	------2-	-------1	--------	--------	--------	--------
$i+2$	-------1	--------	-------1	--------	--------	--------
$i+3$	4-------	--------	--------	4-------	--------	--------
$i+4$	4-------	--------	--------	--------	4-------	--------
$i+5$	4-------	--------	--------	--------	--------	4-------
$i+6$	--------	--------	--------	--------	--------	--------

Fact 4.1. *A local collision as in Definition 4.1 holds with probability 1 for L-SHA-0 and L-SHA-1. For both SHA-0 and SHA-1, a local collision has probability < 1, depending on the bit position of the disturbance and on the Boolean functions f_{IF} and f_{MAJ} (cf. Section 4.4).*

Chabaud and Joux showed how to construct a characteristic for the linearized variant of SHA-0. This linear characteristic, referred to as L-characteristic throughout the remainder of this thesis, consists of linear combinations of overlapping local collisions as in Definition 4.1. For the given L-characteristic the attacker tries to construct two message pairs that follow the characteristic for SHA-0. The most naive method is to perform random trials until a message pair is found. The number of trials depends on the probability of the L-characteristic. Chabaud and Joux observed that the probability such that

the L-characteristic holds for the (non-linear) SHA-0 is related to the Hamming weight of the L-characteristic. In general, the lower the Hamming weight the higher the probability. As we will see in Section 4.4, this is not always the case since we encounter certain side-effects such as carry overflows. In general, it is a good approach to first search for L-characteristics with low Hamming weight and then to choose the L-characteristic with the best probability out of this set. With this strategy, Chabaud and Joux presented the first differential collision attack on SHA-0 with a complexity of 2^{61}.

4.2.2 Biham and Chen

Biham and Chen [17] extended the collision attack on SHA-0 by exploiting the following facts:

1. near-collisions are easier to find than collisions
2. there exist *neutral bits* in SHA-0.

Near-Collisions

Biham and Chen have shown that for SHA-0 it is easier to find a high probability L-characteristic that leads to a collision after 82 steps than a high probability L-characteristic that leads to a collision after 80 steps of the state update transformation. In other words, it is easier to find near-collisions than collisions.

The authors found a near-collision for 80 steps of SHA-0 that has a significantly higher probability than a collision for the same number of steps. The fact that near-collisions are easier to find than collisions has been exploited already in the cryptanalysis of MD4 [153] by Dobbertin [49, 51].

Neutral Bits

Additionally, Biham and Chen observed that there exist so-called neutral bits in SHA-0. This observation leads to a significant improvement of the collision attack. A neutral bit can be explained as follows. Assume, we have two messages m and m^* with a certain difference Δ, *i.e.* $m^* = m \oplus \Delta$. Note that this Δ defines the L-characteristic. Furthermore, assume that the given message pair m and m^* leads to a difference δ_i in the state variables of step i for the non-linear SHA-0.

Now, a neutral bit exists, if complementing the j-th bit of both messages still leads to δ_i. In [17] the authors show that there are not only single bits but also pairs of two or more bits that are all simultaneously neutral. With these neutral bits, a set of message pairs is constructed that all lead to the difference δ_i. Furthermore, Biham and Chen observed that a small fraction of the message pairs does not only result in δ_i but also result in $\delta_{i'}$, with $i' > i$, where this difference is given by the L-characteristic. In other words, based on the neutral bits it is possible to construct a message pair, that results in a difference $\delta_{i'}$ ($i' > i$) without decreasing the probability. It is clear that this improves the

overall attack complexity. In [17], the authors show that there exist neutral bits that have no impact on the difference for 15-20 steps of the state update transformation.

Based on a high probability near-collision L-characteristic and the existence of neutral bits, Biham and Chen presented a collision attack on SHA-0 reduced to 65 steps with a complexity of 2^{29}. This approach has further been improved and has also been extended to step-reduced variants of SHA-1 in [18].

4.2.3 Rijmen and Oswald

Rijmen and Oswald presented the first extension of the attack strategy of Chabaud and Joux to SHA-1 [150, 151]. The strategy basically can be summarized as follows:

1. construct a linear variant of SHA-1
2. determine collisions for the linear variant
3. find a collision for the original SHA-1 among the set of collisions for the linearized variant.

For constructing the linearized variant of SHA-1, Rijmen and Oswald used L-SHA-1 as described in Section 4.1.3. Additionally, they defined several linear approximations for the Boolean functions f_{IF} and f_{MAJ} different from the approximation used in L-SHA-1. As a result of their research with different approximations, it turned out that the best results are achieved by using the linearized variant L-SHA-1. In [150, 151] they exploit coding theory to find 'good' collision-producing L-characteristics. Rijmen and Oswald describe the set of collision-producing differences for L-SHA-1 as a linear code and search for codewords (L-characteristics) with a low Hamming weight. They observed that the low-weight codewords have in common that the zeroes and ones appear in 'bands'. In other words, the zeroes/ones appear mainly in adjacent bit positions. Rijmen and Oswald introduced two algorithms that exploit this fact to improve the low-weight codeword search. Based on this strategy, they presented a collision attack on SHA-1 reduced to 53 steps with a complexity of less than 2^{80} hash computations.

4.2.4 Wang *et al.*

In 2005, Wang *et al.* presented the seminal work describing the first collision attack on SHA-1 [171, 172]. Their approach was also applied to practically break SHA-0 [174]. In a similar way other hash functions, such as MD4, MD5, RIPEMD, and HAVAL-128 were broken by Wang *et al.* [168, 169, 173]. In the following, we will review the basic strategy for collision attacks on SHA-1 presented by Wang *et al.* Besides some noticeable new strategies of Wang *et al.* the collision attack exploits techniques that have been described in the previous sections.

As mentioned above, it is easier to find near-collisions than collisions. Wang *et al.* use a 2-block collision for their collision attack on SHA-1. This is schematically depicted in Figure 4.3.

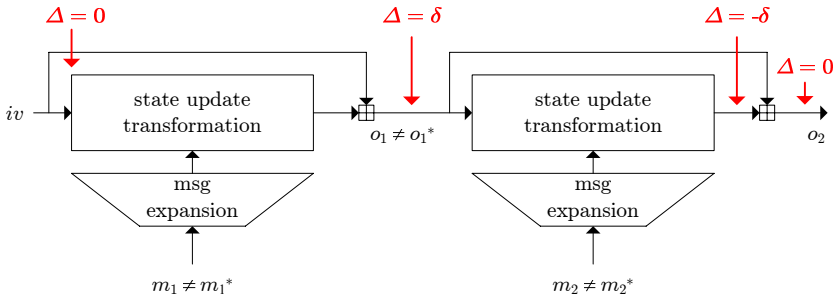

Figure 4.3: The principle of a 2-block collision attack on SHA-1. The differences (colored in red) cancel out after two iterations of the compression function.

Based on Figure 4.3, a 2-block collision can be constructed as follows. In the first iteration we construct two message blocks $m_1 \neq m_1^*$ such that the output difference after the first iteration is $o_1 \oplus o_1^* = \delta \neq 0$. This means that we have a near-collision after the first iteration. For the second iteration, we construct two message pairs $m_2 \neq m_2^*$ that, with the input difference δ, lead to an output difference of $-\delta$ before the feed forward. This difference gets then canceled due to the feed forward, *i.e.* $o_2 \oplus o_2^* = \delta + (-\delta) = 0$. Based on this basic principle, we can now describe in more detail how Wang *et al.* construct these two message blocks with the required differences.

Wang *et al.* split the 80 steps of the state update transformation into two parts: a nonlinear characteristic that we refer to as NL-characteristic, and the already discussed L-characteristic. For the moment, we assume that the first part, referred to as P1, consists of the first 20 steps, and the second part, referred to as P2, consists of the remaining 60 steps of the state update transformation. This is depicted in Figure 4.4.

Firstly, an L-characteristic that results in a *pseudo-near-collision* in P2 is constructed. More precisely, for constructing a pseudo-near-collision in P2, we search for a message difference such that starting from a non-zero difference in the chaining variables it results in a non-zero difference in step 80. Again, a good L-characteristic has a low Hamming weight and therefore, a high probability. How such an L-characteristic can be constructed is discussed in detail in Section 4.3.

Secondly, after the high probability L-characteristic has been constructed, Wang *et al.* search for an NL-characteristic that turns the non-zero difference at the beginning of P2 (δ_i in Figure 4.4) into a zero difference in step $i = 0$. To construct an NL-characteristic in the first 20 steps of the state update transformation, Wang *et al.* exploit the non-linearity imposed by the Boolean function f_{IF} and the modular additions. Unfortunately, Wang *et al.* do not describe in

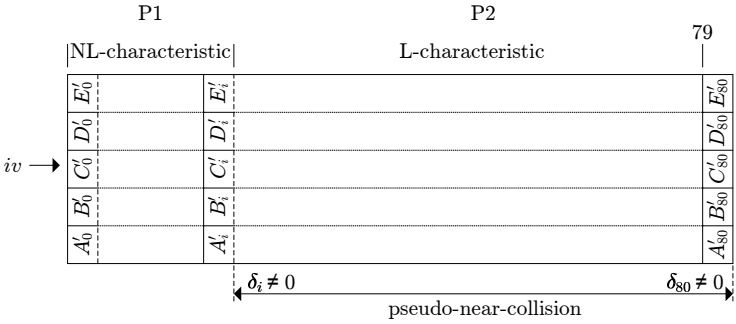

Figure 4.4: The state update transformation split into an NL-characteristic (P1) and an L-characteristic (P2).

detail how to construct this NL-characteristic. But they show based on an example collision for 58 steps of SHA-1 (*cf.* [172]), that the non-linearity can be exploited to map the difference at the beginning of P2 not only to zero but also to a non-zero difference. This fact is used in the second iteration of the collision attack. We can now summarize the 2-block collision attack on SHA-1 as follows (*cf.* Figure 4.5):

1. first iteration:
   (a) construct a pseudo-near-collision L-characteristic for P2
   (b) construct an NL-characteristic that brings the input difference of P2 to a zero difference in step $i = 0$. This NL-characteristic implies conditions on the message
   (c) perform random trials to find two message pairs m_1 and m_1^* (restricted by the conditions of (b)) that follow the remaining steps of the L-characteristic
2. second iteration:
   (a) take the same L-characteristic as for the first iteration, *i.e.* the output difference of P2 in both iterations is the same
   (b) construct a second NL-characteristic that maps the input difference of P2 to the output difference of P2 in the first iteration. This NL-characteristic implies conditions on the message
   (c) perform random trials to find two message pairs m_2 and m_2^* (restricted by the conditions of (b)) that follow the remaining steps of the L-characteristic
3. Due to the feed forward, this results (with a certain probability) in a zero difference after the second iteration and hence a 2-block collision

Until now, we have only required a pseudo-near-collision L-characteristic with a high probability. But what about the probability of the NL-characteristic? In

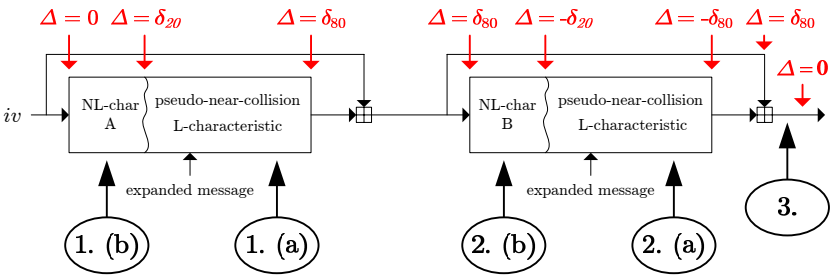

Figure 4.5: The 2-block collision attack strategy by Wang *et al.*

fact, the probability of the NL-characteristic is not important for the collision attack complexity. Indeed, basic and advanced message modification (as described by Wang *et al.* in [172]) result in message pairs that follow the NL-characteristic P1 with a probability close to 1. Therefore, the complexity of a collision attack is only determined by the L-characteristic. Until now, we assumed that the L-characteristic is derived for the last 60 steps of the state update transformation. However, due to the message modification technique it is possible that message pairs can be constructed that follow the NL-characteristic plus some additional steps of the L-characteristic with a probability of (almost) one. In [170, 171], Wang *et al.* claim to be able to use message modification up to step 27. As a consequence of this, we can still use the L-characteristic for the last 60 steps but the overall probability is then determined by fewer than 60 steps. It is clear that the probability of the collision attack increases if more steps can be covered by message modification. Nevertheless, the remaining steps covered by the L-characteristic still determine the attack complexity. Therefore, we discuss in detail how to construct a high probability L-characteristic in the following section.

4.3 Constructing a Linear Characteristic

In this section, we will describe how we can derive an L-characteristic for L-SHA-1. We will first map the problem of finding an L-characteristic with low Hamming weight to the problem of finding low-weight vectors in linear codes. Secondly, we will describe several linear codes and show how we can improve the low-weight search to find an L-characteristic with high probability.

4.3.1 From a Set of Collision-Producing Differences to a Linear Code

With the message expansion described by the matrix $\mathbf{M}_{512\times2560}$ and the linearized state update transformation described by the matrix $\mathbf{S}_{2560\times160}$, the output (hash value) of one L-SHA-1 iteration is $\mathbf{o} = \mathbf{mMS} \oplus \mathbf{ivT} \oplus \mathbf{k}$, *cf.* (4.4)

in Section 4.1.3. Two messages $\mathbf{m_1}$ and $\mathbf{m_1^*} = \mathbf{m_1} \oplus \mathbf{m_1}'$ collide if (see also [150, 151]):

$$\mathbf{o_1^*} \oplus \mathbf{o}_1 = ((\mathbf{m}_1 \oplus \mathbf{m}_1')\mathbf{MS} \oplus \mathbf{ivT} \oplus \mathbf{k}) \oplus (\mathbf{m}_1\mathbf{MS} \oplus \mathbf{ivT} \oplus \mathbf{k}) = \mathbf{m}_1'\mathbf{MS} = \mathbf{0} \quad (4.5)$$

It follows directly from (4.5) that the set of collision-producing differences is a linear code with check matrix $\mathbf{H}_{160\times512} = (\mathbf{MS})^T$. The dimension of the code is $512 - 160 = 352$ and the length of the code is $n = 512$.

Observation 4.1. *The set of collision-producing differences is a linear code. Therefore, finding low-weight differences corresponds to finding low-weight vectors in a linear code.*

Based on Observation 4.1, we can exploit well known and well studied methods of coding theory to search for low-weight vectors in a linear code as described in Section 2.7.2. In the following, we will present several linear codes representing the set of (collision-producing) differences for the linearized model L-SHA-1. We follow a constructive approach to demonstrate the effectiveness of the different codes and the corresponding low-weight L-characteristics. We will start with a code that fulfills the most restrictive condition, namely that the resulting codewords are collision-producing for a 1-block message. Afterwards, we extend the code such that the low-weight search gets improved. We apply the different strategies described in Section 4.2.

Single-Block Collision—Code C_1

Theorem 4.1. *The binary linear code C_1 with dimension $dim(C_1) = 352$ and length $n = 2560$ is defined by the following check matrix:*

$$\mathbf{H}^{C_1}_{2208\times2560} = \begin{bmatrix} \mathbf{S}^T \\ \widetilde{\mathbf{M}} \end{bmatrix} , \quad (4.6)$$

where $\mathbf{S}^T$ is the transpose of the 2560×160 matrix S and $\widetilde{\mathbf{M}} := \begin{bmatrix}\mathbf{F}^T & \mathbf{I}\end{bmatrix}$ with $\mathbf{F}^T$ being the transpose of the 512×2048 matrix $\mathbf{F}$ in (4.3) and the 2048×2048 identity matrix $\mathbf{I}$. The codewords $\mathbf{c}_i \in C_1$ have the following properties:

1. *they are valid expanded-message differences*
2. *they are collision-producing*

Proof. Since $\mathbf{mM} = \mathbf{w} = \mathbf{m}\begin{bmatrix}\mathbf{I}_{512} & \mathbf{F}_{512\times2048}\end{bmatrix}$ and $\mathbf{M}$ is a systematic generator matrix, we can immediately derive the check matrix $\widetilde{\mathbf{M}}$ (*cf.* Section 2.7.1). If a codeword $\mathbf{w}$ fulfills $\mathbf{w}\widetilde{\mathbf{M}}^T = \mathbf{0}$, $\mathbf{w}$ is a valid expanded message. Additionally, we require the codewords to be collision-producing. This condition is determined by the state update transformation matrix $\mathbf{S}$. If $\mathbf{wS} = \mathbf{0}$ then $\mathbf{w}$ is collision-producing. Therefore, we have $\mathbf{S}^T$ as check matrix. Combining these two check matrices leads to the check matrix $\mathbf{H}^{C_1}$ in (4.6). □

When applying a probabilistic algorithm (*cf.* Section 2.7.2) to search for low-weight codewords in code C_1, we find a lowest Hamming weight of 436 for 80 steps. The same weight has been found also by Rijmen and Oswald in [150]. We do not count the weight of the first 20 steps, since these steps are covered by the NL-characteristic (*cf.* Section 4.2.4). The Hamming weights for an increasing number of steps are listed in Table 4.2.

Table 4.2: Lowest Hamming weight found for code C_1.

steps $0, \ldots, 79$	steps $15, \ldots, 79$*	steps $20, \ldots, 79$
436	333	293

*Hamming weight also given in [150]

Multi-Block Collision—Code C_2

Instead of working with a single-block message that leads to a collision, we can also work with multi-block messages that lead to a collision after b iterations (*cf.* Section 4.2.4). For instance take $b = 2$. After the first iteration, we have an output difference $\mathbf{o}'_1 \neq 0$ and after the second iteration we have a collision, *i.e.* $\mathbf{o}'_2 = 0$. The hash computation of two messages is given by

$$\begin{aligned} \mathbf{o}_1 &= \mathbf{m}_1\mathbf{MS} \oplus \mathbf{ivT} \oplus \mathbf{k} \\ \mathbf{o}_2 &= \mathbf{m}_2\mathbf{MS} \oplus \mathbf{o}_1\mathbf{T} \oplus \mathbf{k} \\ &= \mathbf{m}_2\mathbf{MS} \oplus \mathbf{m}_1\mathbf{MST} \oplus \underbrace{\mathbf{ivT}^2 \oplus \mathbf{kT} \oplus \mathbf{k}}_{\text{constant}} . \end{aligned}$$

Based on the same reasoning as for the check matrix $\mathbf{H}^{C_1}$ defined in (4.6), we can construct a check matrix for two-block messages as follows:

$$\mathbf{H}^{C_2}_{4256\times 5120} = \begin{bmatrix} (\mathbf{ST})^T & \mathbf{S}^T \\ \widetilde{\mathbf{M}} & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{M}} \end{bmatrix}$$

This approach can be generalized to construct check matrices for b-block messages that collide after b iterations. The output in iteration b is given by

$$\mathbf{o}_b = \bigoplus_{j=0}^{b-1} \mathbf{m}_{b-j}\mathbf{MST}^j \oplus \underbrace{\mathbf{ivT}^b \oplus \mathbf{k}\bigoplus_{\ell=0}^{b-1} \mathbf{T}^\ell}_{\text{constant}} .$$

To demonstrate the improvements for the low-weight search due to multi-block messages, we additionally define code C_{2a} for 3-block collisions. The output after 3 iterations is given by:

$$\mathbf{o}_3 = \mathbf{m}_3\mathbf{MS} \oplus \mathbf{m}_2\mathbf{MST} \oplus \mathbf{m}_1\mathbf{MST}^2 \oplus \underbrace{\mathbf{ivT}^3 \oplus \mathbf{kT}^2 \oplus \mathbf{kT} \oplus \mathbf{k}}_{\text{constant}}$$

Thus, the set of collision-producing differences for 3-block messages is a linear code with check matrix:

$$\mathbf{H}^{C_{2a}}_{6304\times 7680} = \begin{bmatrix} (\mathbf{ST}^2)^T & (\mathbf{ST})^T & \mathbf{S}^T \\ \widetilde{\mathbf{M}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{M}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \widetilde{\mathbf{M}} \end{bmatrix}$$

Searching for low-weight vectors for a 2-block collision in C_2 with $\mathbf{H}^{C_2}$ and a 3-block collision with the check matrix $\mathbf{H}^{C_{2a}}$, results in the Hamming weights listed in Table 4.3. As can be seen in Table 4.3, the multi-block approach results in a lower Hamming weight for each message block. The complexity for a collision attack is determined by the message block with the highest Hamming weight. Compared to the Hamming weight for a single-block collision in Table 4.2 (weight = 293), we achieve a significant improvement.

Table 4.3: Hamming weight for 2-block and 3-block collisions.

weight of collision-producing differences for steps $20, \ldots, 79$				
two-block collision		three-block collision		
exp. message 1	exp. message 2	exp. message 1	exp. message 2	exp. message 3
152	198	107	130	144

Single-Block Collision and State Update Transformation—Code C_3

As we will describe in Section 4.4, we have to derive conditions involving bits of the expanded message and bits of the state variables, such that an L-characteristic derived for L-SHA-1 holds for SHA-1. For deriving the conditions, we have to consider the differences in the state variables. This means that for the previously derived collision-producing differences (codes C_1, C_2, and C_{2a}), we still have to compute the Hamming weight in the state variables. It is clear that this leads to an increase of the total Hamming weight (see also Table 4.4). Therefore, our next approach is to define a code that also contains the state variables and to look for low-weight vectors in this larger code. This leads to lower Hamming weights for the total, since we search for low-weight vectors in the expanded message words and the state variables concurrently.

Furthermore, we apply the fact that we can use an NL-characteristic that brings a non-zero difference of for instance step 20 to a zero-difference in step $i = 0$ (*cf.* Section 4.2.4). In terms of our linear code, this means that we only require the codewords to be valid expanded messages and no longer to be collision-producing. In other words, we construct a code for which the codewords are differences that lead to a pseudo-collision L-characteristic in the last 60 steps of the L-SHA-1 state update transformation.

We proceed as follows to construct a generator matrix for code C_3 with codewords that describe a pseudo-collision in the last 60 steps. Starting from a collision after step 79 (state variables $A_{80}, \ldots, E_{80}$), we will apply the inverse linearized state update transformation to compute the state variables for step 78,77,...,20. We obtain a generator matrix of the following form:

$$\mathbf{G}^{C_3}_{512\times 11520} = \begin{bmatrix} \mathbf{M} & \mathbf{A} & \mathbf{B} & \mathbf{C} & \mathbf{D} & \mathbf{E} \end{bmatrix} \tag{4.7}$$

The matrix $\mathbf{M}$ is defined in (4.3). Note that since we are looking at the last 60 steps, we only take the last $60 \cdot 32 = 1920$ columns of the matrix $\mathbf{M}$. The 512×1920 matrices $\mathbf{A}, \mathbf{B}, \ldots, \mathbf{E}$ can be constructed by computing the state update transformation backwards starting with a zero difference from step 79, *i.e.* $A'_{80} = B'_{80} = \cdots = E'_{80} = 0$, and ending at step 20. More precisely, we set a single bit in the message (the remaining bits are set to zero) and compute the expanded message words. Using these message words, we then compute the state update backwards. Therefore, for each of the state variables we get a 1×1920 row vector. We repeat this procedure for all 512 bits of the message resulting in the 512×1920 matrices $\mathbf{A}, \ldots, \mathbf{E}$.

The matrix defined in (4.7) is a generator matrix for code C_3 with the following parameters: $dim(C_3) = 512$ and length $n = 11520$. The lowest Hamming weight found for code C_3 is 297. Note that this low-weight vector now also contains the Hamming weight of the state variables $A'_i, \ldots, E'_i$. The Hamming weight for the expanded message is only 127. Compared with the results of the previous sections (code C_1), we achieve a remarkable improvement by also considering the Hamming weight of the state variables and by only requiring that the codewords are valid expanded messages.

Multi-Block Collision and State Update Transformation—Code C_4

As shown in the previous section, we are able to find differences with lower Hamming weight if we use multi-block messages. We exploit this fact to construct code C_4. A multi-block collision with $i = 2$ is shown in Figure 4.6. As can be seen in Figure 4.6, we have a collision after the second iteration due to the feed forward for L-SHA-1, if the state update transformation has the same output difference after each iteration (*cf.* Section 4.2.4). In other words, we want to construct a code with codewords that produce a pseudo-near-collision in the last 60 steps of L-SHA-1. Having such a pseudo-near-collision L-characteristic, we use an NL-characteristic to bring the non-zero difference in the first iteration to zero in step $i = 0$ and another NL-characteristic to bring the non-zero difference in the second iteration to a difference in step $i = 0$ that equals the output difference of the first iteration.

To construct a generator matrix for code C_4, we only have to appropriately extend the generator matrix for code C_3. More precisely, we have to extend the matrix such that the output difference of the state update transformation has a non-zero output difference $\delta_{80} \neq 0$. Therefore, we add 160 rows, denoted by the 160×1920 matrices $\mathbf{0}, \widetilde{\mathbf{A}}, \ldots, \widetilde{\mathbf{E}}$, to the generator matrix in (4.7). For the code

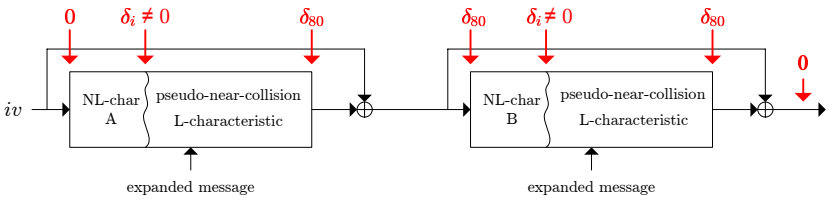

Figure 4.6: 2-block collision for L-SHA-1.

C_4, we get a generator matrix

$$\mathbf{G}^{C_4}_{672\times 11520} = \begin{bmatrix} \mathbf{M} & \mathbf{A} & \mathbf{B} & \mathbf{C} & \mathbf{D} & \mathbf{E} \\ \mathbf{0} & \widetilde{\mathbf{A}} & \widetilde{\mathbf{B}} & \widetilde{\mathbf{C}} & \widetilde{\mathbf{D}} & \widetilde{\mathbf{E}} \end{bmatrix} . \tag{4.8}$$

The matrix in (4.8) is a generator matrix for code C_4 with $dim(C_4) = 672$ and $n = 11520$. Searching for low-weight vectors in C_4 results in a smallest Hamming weight of 237 for the expanded message words and the state variables for the last 60 steps. This low-weight difference (pseudo-near-collision L-characteristic) is shown in Table 4.5.

Summary of Hamming Weights Found

To give an overview of the improvements achieved by constructing different codes, we list the Hamming weights of the found codewords in Table 4.4. As can be seen in Table 4.4, we find the lowest Hamming weight if we use multi-block messages and if we search for pseudo-near-collision L-characteristics in the last steps of L-SHA-1.

Table 4.4: Summary of Hamming weights found for code C_1,C_2,C_3, and C_4.

	Code C_1	**Code C_2**		**Code C_3**	**Code C_4**	
	single-block	two-block		single-block	two-block	
		msg 1	msg 2		msg 1	msg2
weight expanded message	293	152	198	127	108	108
weight state update trans.	563	4730	4817	170	129	129
total Hamming weight	**856**	**4882**	**5015**	**297**	**237**	**237**

4.3.2 Improving Low-Weight Search for L-SHA-1

The low-weight vectors for the previously described codes have not been found by just applying the probabilistic algorithms described in Section 2.7.2. The reason is that the work factors for the different search algorithms are too high. We demonstrate this fact by using code C_4 with $dim(C_4) = k = 672$ and length

$n = 11520$. We want to estimate the work factor for Leon's algorithm and Stern's algorithm for finding vectors with a Hamming weight of 237. The optimal parameters for Stern's algorithm are $p = 3$ and $l = 20$ for C_4. To find a codeword with Hamming weight 237 in C_4 requires approximately 2^{49} elementary operations. Leon's algorithm, with parameters $p = 3$ and $s = dim(C_4) + 12$, requires approximately 2^{41} elementary operations. In the following, we present two different approaches to reduce the work factors for the low-weight search.

Zero-Bands in L-SHA-1

During our research on the different codes, we observed that the found low-weight vectors all have in common that the ones and zeroes occur in bands. More precisely, the ones in the expanded message words and the state variables usually appear in adjacent bit positions (see for instance Table 4.5). This observation has been reported also by Rijmen and Oswald in [150]. This special property of the low-weight differences for L-SHA-1 can be used to improve the low-weight search as follows. By applying Algorithm 4.1 below to the generator matrix, we force certain bits in the codewords to zero. With this approach, we are able to reduce the search space significantly. As described in Section 2.7.2, the basic idea of the probabilistic algorithms is to use a randomly permuted subset of the generator matrix $\mathbf{G}$ to construct the punctured code $C^\bullet$ with generator matrix $\mathbf{G}^\bullet$. This corresponds to a reduction of the search space. If we apply Algorithm 4.1 to $\mathbf{G}$, we actually do something similar but instead of randomly puncturing the code, we have defined a strategy. Algorithm 4.1 shows the pseudo-code.

Algorithm 4.1 Forcing certain bits of the codewords to zero

Input: generator matrix $\mathbf{G}$ for code C, integer r defining the minimum $rank$ of $\mathbf{G}^\bullet$, list L of candidate columns to force to zero
Output: generator matrix $\mathbf{G}^\bullet$ for punctured code $C^\bullet$ with $rank(\mathbf{G}^\bullet) = r$

```
G• = G
while rank(G•) > r and L not empty do
    randomly pick a column y from L (0 ≤ y < length(G•))
    for all rows x (0 ≤ x < rank(G•)) with a one in column y
    add row x to all other rows that have a one in the same column
    remove row x from G•
    remove column y from L
end while
return G•
```

Prior to applying the probabilistic search algorithms, we apply Algorithm 4.1 to reduce the search space of the code. Since we force columns of the codewords to zero, we do not only reduce the dimension of the code but also the length. Computing the estimates for the complexities of this 'restricted code' shows that the expected number of operations to find a vector with a certain Hamming weight decreases remarkably. For instance, applying Algorithm 4.1 to the generator matrix for code C_4 with $r = 50$ leads to the following values for the

punctured code $C_4^\bullet$: $dim(C_4^\bullet) = 50$ and length $n = 2327$. Stern's algorithm with optimal parameter $p = 2$ and $l = 4$ requires approximately 2^{37} elementary operations. For Leon's algorithm, we get a work factor of approximately 2^{25} with $p = 3$ and $s = dim(C_4^\bullet) + 8$. Therefore, the Hamming weights presented for codes C_1, C_2, C_3, C_4 have been found by first applying Algorithm 4.1 and then running the probabilistic search algorithms. With this technique, we are able to reduce the search space significantly and finding the low-weight vectors is possible within minutes on an ordinary computer. Note that by applying Algorithm 4.1 it can be that we get a suboptimal solution. However, as we will see in the next section the lowest weight found with our methods has been proven by Jutla in [76, 78] to be the minimum weight for the message expansion by using a computer-aided proof.

Looking at Disturbances Only

If we look at the low-weight difference for code C_4 in Table 4.5 then we observe the following:

Observation 4.2. *The L-characteristic consists of overlapping local collisions. The state variable A' contains the disturbances of each local collision. This is referred to as the disturbance vector.*

This can be explained as follows. Based on the definition of a local collision (*cf.* Definition 4.1) and the definition of the state update transformation in (4.2), we know that the disturbance that is introduced in step i goes to state variable A'_{i+1}. Then, the disturbance for each local collision gets corrected in the next 5 consecutive steps, *i.e.* the difference in state variables $A'_{i+2}, \ldots, A'_{i+5}$ is zero. Therefore, for each local collision only the disturbance appears in the state variables A'_i. Note that the disturbance of a single local collision does not necessarily have to be injected through the expanded message-words but can also come from an earlier local collision.

For L-SHA-1, the differences of state variables $B'_i, \ldots, E'_i$ are completely determined by the differences of state variable A'_i, *i.e.* the disturbance vector. Additionally, the expanded message can be constructed by the linear combination of the disturbances and corresponding corrections. Therefore, we can construct a code that only describes the pseudo-near-collision for state variable A. If we search for low-weight vectors in this code, then we aim to find low-weight disturbance vectors. As we will see in Section 4.4, based on this disturbance vector we can accurately compute the probability such that the L-characteristic for L-SHA-1 holds for SHA-1.

Remark 4.1. We do not need to look at state variable A but we can also look at the expanded message words. However, since the result is the same in both cases and we can derive a generator matrix for the code describing the disturbance vector in A directly from code C_4, we will use this approach to search for low-weight disturbance vectors.

To search for low-weight disturbance vectors in state variable A' only, we define code C_5 with $dim(C_5) = 672$ and length $n = 1920$ by the following

generator matrix that can be directly derived from G^{C_4}:

$$\mathbf{G}_{672\times 1920}^{C_5} = \begin{bmatrix} \mathbf{A} \\ \widetilde{\mathbf{A}} \end{bmatrix} ,$$

where the 512×1920 matrix $\mathbf{A}$ and the 160×1920 matrix $\widetilde{\mathbf{A}}$ are the same as in (4.8). If we compare this code with the codes C_2, C_3, C_4, we see that the code C_5 is remarkably smaller. This leads to running-time improvements for the probabilistic low-weight search algorithms. Nevertheless, also in this case we can still speed-up the low-weight search by first applying Algorithm 4.1 followed by the probabilistic algorithms. In both cases, we find a smallest Hamming weight of 25 for the disturbance vector. This found low-weight disturbance vector is given in Table 4.6. Note that this vector is the same as the difference in state variable A' in Table 4.5 rotated down by one position. More precisely, we compute A'_{i+1} for $i = 19$ and then for $i = 19, \ldots, 78$ the column A'_{i+1} in Table 4.5 is the same vector as in Table 4.6. In [76, 78] a computer aided proof is used to show that the minimum Hamming weight is 25.

So far, we omitted the fact that all found L-characteristics are rotation invariant with respect to the message expansion and state update transformation. More precisely, we can rotate each L-characteristic (or the according disturbance vector) over all 32 bit positions. Clearly, rotating the L-characteristic over different bit positions does not change the Hamming weight. However, rotating the L-characteristic influences its probability as will be discussed in detail in the next section.

Table 4.5: Low-weight difference found for code C_4. The overall Hamming weight is 237. The differences are listed in hexadecimal notation. For the sake of readability, zero values are denoted by a dash.

step	W'_i	A'_{i+1}	B'_{i+1}	C'_{i+1}	D'_{i+1}	E'_{i+1}
$i = 20$	8-----4-	--------	-------2	--------	A-------	8-------
$i = 21$	2------1	-------3	--------	8-------	--------	A-------
$i = 22$	2-----6-	--------	-------3	--------	8-------	--------
$i = 23$	8------1	-------2	--------	C-------	--------	8-------
$i = 24$	4-----42	-------2	-------2	--------	C-------	--------
$i = 25$	C-----43	-------1	-------2	8-------	--------	C-------
$i = 26$	4-----22	--------	-------1	8-------	8-------	--------
$i = 27$	-------3	-------2	--------	4-------	8-------	8-------
$i = 28$	4-----42	-------2	-------2	--------	4-------	8-------
$i = 29$	C-----43	-------1	-------2	8-------	--------	4-------
$i = 30$	C-----22	--------	-------1	8-------	8-------	--------
$i = 31$	-------1	--------	--------	4-------	8-------	8-------
$i = 32$	4------2	-------2	--------	--------	4-------	8-------
$i = 33$	C-----43	-------3	-------2	--------	--------	4-------
$i = 34$	4-----62	--------	-------3	8-------	--------	--------
$i = 35$	8------1	-------2	--------	C-------	8-------	--------
$i = 36$	4-----42	-------2	-------2	--------	C-------	8-------
$i = 37$	4-----42	--------	-------2	8-------	--------	C-------
$i = 38$	4------2	--------	--------	8-------	8-------	--------
$i = 39$	-------2	-------2	--------	--------	8-------	8-------
$i = 40$	------4-	--------	-------2	--------	--------	8-------
$i = 41$	8------2	--------	--------	8-------	--------	--------
$i = 42$	8-------	--------	--------	--------	8-------	--------
$i = 43$	8------2	-------2	--------	--------	--------	8-------
$i = 44$	8-----4-	--------	-------2	--------	--------	--------
$i = 45$	--------	-------2	--------	8-------	--------	--------
$i = 46$	8-----4-	--------	-------2	--------	8-------	--------
$i = 47$	8-------	-------2	--------	8-------	--------	8-------
$i = 48$	------4-	--------	-------2	--------	8-------	--------
$i = 49$	8-------	-------2	--------	8-------	--------	8-------
$i = 50$	------4-	--------	-------2	--------	8-------	--------
$i = 51$	8------2	--------	--------	8-------	--------	8-------
$i = 52$	--------	--------	--------	--------	8-------	--------
$i = 53$	8-------	--------	--------	--------	--------	8-------
$i = 54$	8-------	--------	--------	--------	--------	--------
$i = 55$	--------	--------	--------	--------	--------	--------
$i = 56$	--------	--------	--------	--------	--------	--------
$i = 57$	--------	--------	--------	--------	--------	--------
$i = 58$	--------	--------	--------	--------	--------	--------
$i = 59$	--------	--------	--------	--------	--------	--------
$i = 60$	--------	--------	--------	--------	--------	--------
$i = 61$	--------	--------	--------	--------	--------	--------
$i = 62$	--------	--------	--------	--------	--------	--------
$i = 63$	--------	--------	--------	--------	--------	--------
$i = 64$	--------	--------	--------	--------	--------	--------
$i = 65$	-------4	-------4	--------	--------	--------	--------
$i = 66$	------8-	--------	-------4	--------	--------	--------
$i = 67$	-------4	--------	--------	-------1	--------	--------
$i = 68$	-------9	-------8	--------	--------	-------1	--------
$i = 69$	-----1-1	--------	-------8	--------	--------	-------1
$i = 70$	-------9	--------	--------	-------2	--------	--------
$i = 71$	------12	------1-	--------	--------	-------2	--------
$i = 72$	-----2-2	--------	------1-	--------	--------	-------2
$i = 73$	------1A	-------8	--------	-------4	--------	--------
$i = 74$	-----124	------2-	-------8	--------	-------4	--------
$i = 75$	-----4-C	--------	------2-	-------2	--------	-------4
$i = 76$	------26	--------	--------	-------8	-------2	--------
$i = 77$	------4A	------4-	--------	--------	-------8	-------2
$i = 78$	-----8-A	--------	------4-	--------	--------	-------8
$i = 79$	------6-	------28	--------	------1-	--------	--------
weight	108	26	25	25	26	27

Table 4.6: Minimum Hamming weight disturbance vector for last 60 steps of L-SHA-1. The values are listed in hexadecimal notation. For the sake of readability, zero values are denoted by a dash.

A'_{20} = -------2	A'_{40} = -------2	A'_{60} = --------
A'_{21} = --------	A'_{41} = --------	A'_{61} = --------
A'_{22} = -------3	A'_{42} = --------	A'_{62} = --------
A'_{23} = --------	A'_{43} = --------	A'_{63} = --------
A'_{24} = -------2	A'_{44} = -------2	A'_{64} = --------
A'_{25} = -------2	A'_{45} = --------	A'_{65} = --------
A'_{26} = -------1	A'_{46} = -------2	A'_{66} = -------4
A'_{27} = --------	A'_{47} = --------	A'_{67} = --------
A'_{28} = -------2	A'_{48} = -------2	A'_{68} = --------
A'_{29} = -------2	A'_{49} = --------	A'_{69} = -------8
A'_{30} = -------1	A'_{50} = -------2	A'_{70} = --------
A'_{31} = --------	A'_{51} = --------	A'_{71} = --------
A'_{32} = --------	A'_{52} = --------	A'_{72} = ------1-
A'_{33} = -------2	A'_{53} = --------	A'_{73} = --------
A'_{34} = -------3	A'_{54} = --------	A'_{74} = -------8
A'_{35} = --------	A'_{55} = --------	A'_{75} = ------2-
A'_{36} = -------2	A'_{56} = --------	A'_{76} = --------
A'_{37} = -------2	A'_{57} = --------	A'_{77} = --------
A'_{38} = --------	A'_{58} = --------	A'_{78} = ------4-
A'_{39} = --------	A'_{59} = --------	A'_{79} = --------

4.4 An Accurate Probability Analysis of Local Collisions in SHA-1

The pseudo-near-collision L-characteristic derived in the previous section determines the complexity of a collision attack on SHA-1. Therefore, we will take now the disturbance vector given in Table 4.6 and derive the probability such that the according pseudo-near-collision L-characteristic holds for the original SHA-1. Since the disturbance vector consists of overlapping local collisions, we start to derive the probability of single local collisions with disturbances in different bit positions. Based on these probabilities, we will then estimate the collision attack complexity for SHA-1. As a result of our thorough analysis of probabilities for local collisions, we will see that the attack complexity is slightly lower than estimated by Wang *et al.* in [172].

4.4.1 Considering the Number of Conditions

To analyze the probability of local collision, we work with signed-bit differences as defined in Section 2.5.1. Table 4.7 shows a local collision with signed-bit differences for f_{XOR} and f_{MAJ}. Note that the first 20 steps of SHA-1 are covered by the NL-characteristic (*cf.* Section 4.2.4). Therefore, we only consider the Boolean functions f_{XOR} and f_{MAJ}.

Table 4.7: Local collision (disturbance-corrections) with signed-bit differences for SHA-1.

step	**difference**				**description**
			f_{XOR}	f_{MAJ}	
i	W'_i	$=$	$+2^j$	$+2^j$	single-bit disturbance at bit position j
$i+1$	W'_{i+1}	$=$	-2^{j+5}	-2^{j+5}	corrections
$i+2$	W'_{i+2}	$=$	$\pm 2^j$	-2^j	
$i+3$	W'_{i+3}	$=$	$\pm 2^{j-2}$	-2^{j-2}	
$i+4$	W'_{i+5}	$=$	$\pm 2^{j-2}$	-2^{j-2}	
$i+5$	W'_{i+8}	$=$	-2^{j-2}	-2^{j-2}	

$\pm$ denotes an arbitrary sign

For the local collision defined in Table 4.7, we can now derive the number of conditions and the corresponding probabilities such that the local collision holds for the original SHA-1. We refer to conditions that contain only bits of the expanded message words as easy conditions since we can easily fulfill them. This follows from the fact that we have full control over the input messages. Conditions that include bits of the state variables are referred to as hard conditions, since in general it is difficult to fulfill them[2]. For the analysis we can

[2]Conditions that include bits of the state variables can be fulfilled by techniques such as basic and advanced message modification. However, this is only possible for slightly more than 20 steps [172]. Since we are looking at the last 60 steps of SHA-1, we assume that it is difficult to fulfill these conditions.

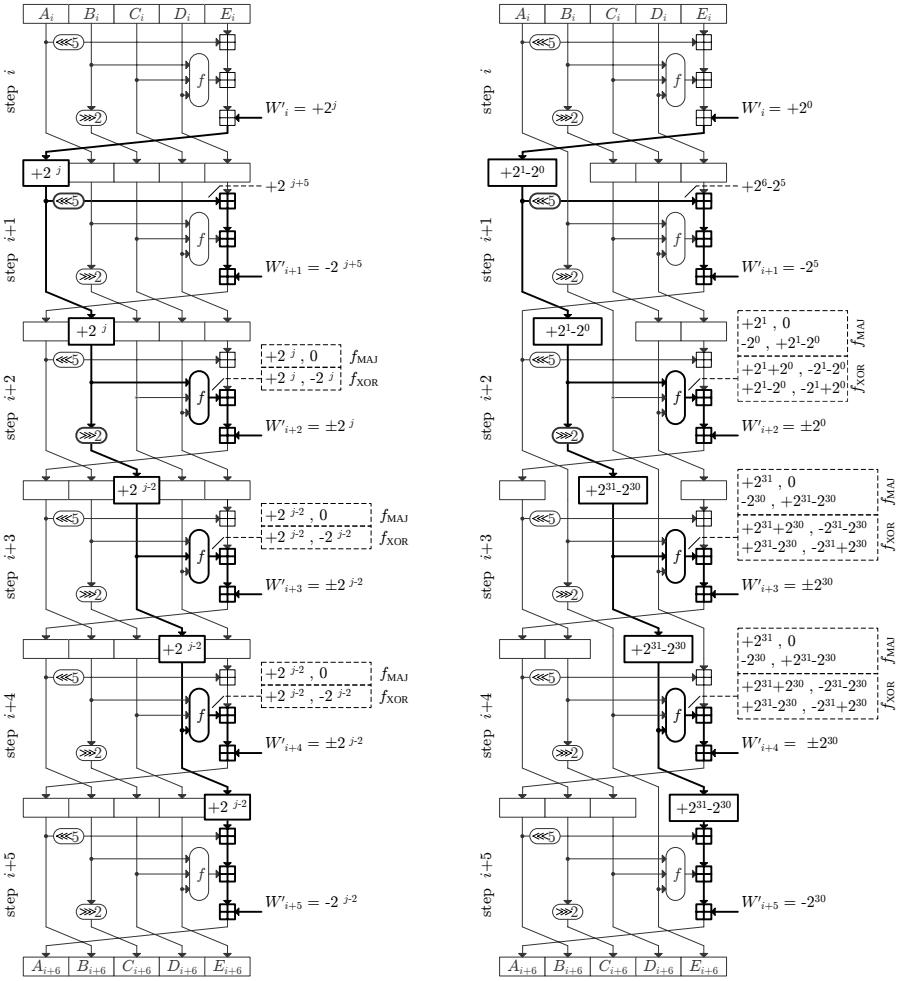

Figure 4.7: On the left, a local collision with disturbance in bit position j. No carry occurs in step i. On the right a local collision with disturbance in bit position $j = 0$. In step i a carry occurs. The differences in the dashed rectangles are the possible output differences of f_{XOR} and f_{MAJ}.

assume without loss of generality that the sign of the disturbance is positive, *i.e.* $W'_i = +2^j$. If the disturbance is -2^j, we get the same results by just flipping all the other signs. The propagation of the disturbance and corrections is shown in the left part of Figure 4.7.

Disturbance in Step i

The disturbance difference is injected in step i. As can be seen in Figure 4.7, we require that the injected difference stays the same in state variable A_{i+1}, *i.e.* $A'_{i+1} = W'_i = +2^j$. This occurs with probability 1/2. Even if the disturbance is injected at bit position $j = 31$ (MSB position), the sign of the difference stays the same in A'_{i+1} with probability 1/2.

Correction in Step $i + 1$

As shown in Figure 4.7, the difference in state variable A is rotated to the left by 5 positions. Therefore, the correction is $W'_{i+1} = -2^{j+5}$. It follows from Table 2.2 (row 6 with $u = +1$ and $v = -1$, or vice versa) in Section 2.5 that if the sign of the correction is the opposite of the sign of the disturbance, then the correction occurs with probability 1 since the sum of the disturbance and the correction is 0 and no carry occurs. We can ensure the negative sign of the correction with the following condition. This condition contains only bits of the expanded message words and hence we can easily fulfill it.

$$W_{i+1,j+5} \oplus W_{i,j} = 1 \qquad (CW_{i+1})$$

Correction in Step $i + 2$

In this step, we have to consider the modular addition and the Boolean functions f_{XOR} and f_{MAJ}. As described in Table 2.3 in Section 2.5, f_{XOR} flips the sign of the input difference with probability 1/2. Therefore, for $B'_{i+2} = +2^j$ the output difference of f_{XOR} can be either $+2^j$ or -2^j depending on C_{i+2} and D_{i+2}. Since we cannot easily influence the values of C_{i+2} and D_{i+2} the probability for the correction is 1/2.

For f_{MAJ}, we get the same probability as for f_{XOR} by defining a condition including bits of the expanded message words only. For the input difference $B'_{i+2} = +2^j$ the possible output difference of f_{MAJ} is either $+2^j$ or 0. Together with the probability of 1/2 for the modular addition, this results in a probability of 1/4. However, if the sign of the correction is negative, then the correction has a probability of 1/2. This can be ensured by fulfilling the following condition

$$W_{i+2,j} \oplus W_{i,j} = 1 \ . \qquad (CW_{i+2})$$

Correction in Step $i + 3$ and $i + 4$

These steps are the same as step $i+2$ except that the difference $+2^j$ is rotated to the right by 2 positions, *i.e.* $+2^{j-2}$. For f_{XOR}, we get a probability of 1/2 in each

step. For f_{MAJ}, we also get the probability 1/2 if the signs of the corrections are negative. This can be ensured by fulfilling the following easy conditions:

$$W_{i+3,j-2} \oplus W_{i,j} = 1 \qquad (CW_{i+3})$$

$$W_{i+4,j-2} \oplus W_{i,j} = 1 \qquad (CW_{i+4})$$

Correction in Step $i+5$

If all corrections have taken place in the previous steps the signed-bit difference is in state variable E. Figure 4.7 shows that E'_{i+5} is the same difference as $A'_{i+1} = +2^j$ rotated by 2 to the right, *i.e.* $E'_{i+5} = +2^{j-2}$. We only have to consider the modular addition. As in step $i+1$, we can fulfill condition

$$W_{i+5,j-2} \oplus W_{i,j} = 1 \qquad (CW_{i+5})$$

such that the correction has negative sign. Hence, the correction in step $i+5$ has probability 1.

Local Collision with Best Probability

With the above-described probabilities for each step of the local collision, we can define a local collision that has the best probability for f_{XOR}. Assume the disturbance is introduced in bit position $j = 1$. In step i, we have a probability of 1/2. Since we can easily fulfill condition CW_{i+1}, we have a probability of 1 in step $i+1$. In step $i+2$ the probability is 1/2. Now, for steps $i+3$ to $i+5$ the disturbance is rotated to bit position $j = 31$. Since a carry in the difference can be ignored (addition mod 2^{32}) we get a total probability of 2^{-2} for a local collision with f_{XOR} and a single-bit disturbance in bit position $j = 1$. For all other bit positions we have a probability less than 1 in steps $i+3$ and $i+4$, since the corrections do not occur in the MSB position. Therefore, the probability 2^{-2} is an upper bound for a local collision with f_{XOR}.

Summary of Probabilities of Local Collisions

Table 4.8 summarizes the probabilities for all possible local collisions with a single-bit disturbance and lists the easy conditions that have to be fulfilled. For the discussion so far, we considered probabilities and easy conditions. Nevertheless, the probabilities for the modular addition and the functions f_{MAJ} and f_{XOR} can also be described in terms of hard conditions. Each single condition is fulfilled with probability 1/2. Consider for instance f_{MAJ}. The input difference $B'_i = +2^j$ leads to the output difference $+2^j(C_i \oplus D_i)$ (see Table 2.3 in Section 2.5). In order to ensure the output difference $+2^j$, we require that $C_i \oplus D_i = 1$. Since we cannot easily influence the values of C_i and D_i, the condition is fulfilled with probability 1/2. The same can be done for the other cases. For a local collision with disturbance in bit position $j = 1$, we have a probability of 2^{-4} for f_{MAJ}. In other words, there are 4 hard conditions.

Table 4.8: Probabilities for local collisions in SHA-1.

	probability		easy conditions	
disturbance	f_{XOR}	f_{MAJ}	f_{XOR}	f_{MAJ}
$j = 0, 2, \ldots, 30$	2^{-4}	2^{-4}	CW_{i+1}, CW_{i+5}	$CW_{i+1}, CW_{i+2}, CW_{i+3}$ CW_{i+4}, CW_{i+5}
$j = 31$	2^{-3}	2^{-4}	CW_{i+1}, CW_{i+5}	$CW_{i+1}, CW_{i+3}, CW_{i+4}, CW_{i+5}$
$j = 1$	2^{-2}	2^{-4}	CW_{i+1}	CW_{i+1}, CW_{i+2}

With the probabilities listed in Table 4.8 the complexity of the attack on SHA-1 can be determined. For the disturbance vector in Table 4.6 or [172, Table 5], we compute the product of all probabilities for each disturbance bit to determine the overall probability and hence the attack complexity. Note that computing the overall probability in this way requires that the probabilities are independent. We performed several measurements that agree with this assumption (see also Remark 4.2).

4.4.2 Accurate Probability Computation

In Section 4.4.1, we determined the probabilities of local collisions with disturbances introduced at different bit positions. For the analysis we did not allow carries in step i, where the disturbance is injected. This restriction can actually be relaxed. In the following, we will analyze the impact of carries in step i on the probability of local collisions. We will show that the probabilities are actually higher for most bit positions of the disturbance.

Single-Bit Disturbance

We start with a disturbance in bit position $j = 0$. As shown in Table 4.8 this results in a probability of 2^{-4}. Now assume that a carry occurs in the difference in step i, *i.e.* the disturbance $W'_i = +2^0$ leads to $A'_{i+1} = +2^1 - 2^0$. This is shown on the right hand side in Figure 4.7.

The carry in step i occurs with probability 1/4. The difference in bit position $j = 1$ can be seen as a new disturbance that leads to a second local collision with a certain probability. To cancel out the difference $A'_{i+1} = +2^1$, we require that the corrections in the consecutive steps also produce a carry in the difference. As described in Section 4.4.1, we fulfill condition CW_{i+1} to ensure that $W'_{i+1} = -2^5$. Therefore, the differences cancel out with probability 1 since $(+2^6 - 2^5) + (-2^5) = 0$. For steps $i+2$ to $i+4$ we first consider f_{XOR}. In step $i+2$, we have a probability of 1/4 because f_{XOR} flips the sign of a bit difference with probability 1/2. Since we have two bit differences this results in a probability of 1/4. The same holds for steps $i + 3$ and $i + 4$. However, since the disturbance is introduced in bit position $j = 0$, the second difference caused by the carry is rotated to bit position $j = 31$ in step $i + 2$. We can ignore carries in this bit position and hence the sign in bit position $j = 31$ has no impact. Therefore, we get a probability of 1/2

for each step. The same analysis can be done for f_{MAJ}. As already mentioned, f_{MAJ} preserves the sign of the input difference but the difference propagates only with probability 1/2. Therefore, we cannot exploit bit position $j = 31$—the probability for steps $i+3$ and $i+4$ is 1/4 each. For step $i+2$ the probability is 1/4 since CW_{i+2} is fulfilled. In step $i+5$, we have a probability of 1 for f_{XOR} and f_{MAJ} based on the same reasoning as for step $i+1$. With the results of this analysis, we can update the probabilities given in Section 4.4.1. The probability for f_{XOR} and f_{MAJ} with a disturbance in bit position $j = 0$ becomes:

$$P_{LC}(f_{\mathrm{XOR}}, j = 0) = 2^{-4} + 2^{-6} = 2^{-3.6781} \quad (4.9)$$

$$P_{LC}(f_{\mathrm{MAJ}}, j = 0) = 2^{-4} + 2^{-8} = 2^{-3.9125} \quad (4.10)$$

Uncorrectable Carries

Let us now consider the case where two carries in the difference occur in step i. In this case the disturbance $W'_i = +2^0$ leads to the following difference in state variable A: $A'_{i+1} = +2^2 - 2^1 - 2^0$. Two carries occur with probability 1/8. If we work with the difference in bit position $j = 2$, we encounter the following problem, which we refer to as *uncorrectable carries*. In step $i+2$ the difference is rotated by two positions to the right, *i.e.* $-2^{31} - 2^{30} + 2^0$. It is not possible to correct the difference $+2^0$ in step $i+3$ anymore since the correction takes place in bit position $j = 30$. For f_{MAJ}, uncorrectable carries for this example take place only in step $i+5$. This is due to the fact that the difference $+2^0$ is blocked by f_{MAJ} with probability 1/2 in steps $i+2$ to $i+4$. However, in step $i+5$ we cannot correct the difference $+2^0$ since the correction takes place in $j = 30$. Therefore, the probabilities given in (4.9) and (4.10) are an upper bound of the probabilities for both functions with a disturbance in $j = 0$.

If we perform the carry analysis for bit position $j = 1$, we also encounter uncorrectable carries as for the disturbance in $j = 0$. Namely, a carry in step i cannot be corrected anymore in step $i+3$ (step $i+5$ for f_{MAJ}, respectively) and therefore, a carry does not increase the probability for a local collision with disturbance in $j = 1$ for both f_{XOR} and f_{MAJ}. Uncorrectable carries can also occur due to the left rotation by 5 in step $i+1$. A disturbance in $j = 26$ that leads to a carry in step i cannot be corrected anymore in step $i+1$ since the correction W'_{i+1} takes place in bit position $j = 31$ but the carry in the difference is rotated to $j = 0$.

Carries that Improve the Probability of Local Collisions

After determining the probabilities for disturbances in bit position $j = 0$ and $j = 1$, we describe now the impact of carry effects for disturbances in bit position $j = 2, \ldots, 31$. Due to uncorrectable carries after bit position $j = 26$, we have to analyze the probability for $j = 2, \ldots, 26$ and $j = 27, \ldots, 31$ separately. We start the explanation for f_{XOR}. For $2 \leq j \leq 26$, we have the same probability in steps i, $i+2$, $i+3$, and $i+4$, namely the probability that no carry occurs and the probabilities for all possible carries. Note that the probability in steps

$i+1$ and $i+5$ is 1 since we fulfill the easy conditions CW_{i+1} and CW_{i+5} (see Section 4.4.1). For $27 \leq j \leq 31$, we have the same except that the probability in step $i+2$ is increased by a factor of 2 if the carry in step i propagates to bit position $j = 31$, *i.e.* in this case the carry difference in step $i+2$ can be ignored. For f_{MAJ}, we also assume that the easy conditions are fulfilled. Then we get the same probabilities as for f_{XOR} with the difference that for $27 \leq j \leq 31$, we cannot exploit the fact that the carry in bit position $j = 31$ can be ignored. This is because for f_{MAJ} the difference only propagates with probability 1/2. In (4.11) and (4.12), we give the formulae to compute the accurate probability for a local collision including all carry effects.

$$P_{LC}(f_{\mathrm{XOR}}, j) = \begin{cases} 2^{-4} + 2^{-6} & \text{for } j = 0 \\ 2^{-2} & \text{for } j = 1 \\ \sum_{k=1}^{27-j} 2^{-4k} & \text{for } j = 2, \ldots, 26 \\ 2 \cdot 2^{-4\cdot(32-j)} + \sum_{k=1}^{31-j} 2^{-4k} & \text{for } j = 27, \ldots, 31 \end{cases} \tag{4.11}$$

$$P_{LC}(f_{\mathrm{MAJ}}, j) = \begin{cases} 2^{-4} + 2^{-8} & \text{for } j = 0 \\ 2^{-4} & \text{for } j = 1 \\ \sum_{k=1}^{27-j} 2^{-4k} & \text{for } j = 2, \ldots, 26 \\ \sum_{k=1}^{32-j} 2^{-4k} & \text{for } j = 27, \ldots, 30 \\ 2^{-4} & \text{for } j = 31 \end{cases} \tag{4.12}$$

Based on these formulae the probability for a local collision with f_{XOR} and f_{MAJ} can be bounded as follows. We know that

$$\sum_{k=1}^{27-j} 2^{-4k} = 2^{-4}\frac{1-2^{-4(28-j)}}{1-2^{-4}} \leq \frac{2^{-4}}{1-2^{-4}} = \frac{1}{2^4-1} ,$$

$$\sum_{k=1}^{32-j} 2^{-4k} = 2^{-4}\frac{1-2^{-4(33-j)}}{1-2^{-4}} \leq \frac{2^{-4}}{1-2^{-4}} = \frac{1}{2^4-1} , \text{ and}$$

$$\begin{aligned} 2 \cdot 2^{-4\cdot(32-j)} + \sum_{k=1}^{31-j} 2^{-4k} &= 2^{-4(32-j)+1} + 2^{-4}\frac{1-2^{-4(32-j)}}{1-2^{-4}} \\ &\leq 2^{-3} + \frac{2^{-4}}{1-2^{-4}} = \frac{1}{2^3} + \frac{1}{2^4-1} . \end{aligned}$$

Therefore, we get the following bounds on the probability:

$$\frac{1}{2^4} \leq \quad P_{LC}(f_{\mathrm{XOR}}, j) \quad \leq \frac{1}{2^4-1} \text{ for } j = 2, \ldots, 26$$

$$\frac{1}{2^4} \leq \quad P_{LC}(f_{\mathrm{XOR}}, j) \quad \leq \frac{1}{2^3} + \frac{1}{2^4-1} \text{ for } j = 27, \ldots, 31$$

$$\frac{1}{2^4} \leq \quad P_{LC}(f_{\mathrm{MAJ}}, j) \quad \leq \frac{1}{2^4-1} \text{ for } j = 2, \ldots, 26 \text{ and } j = 27, \ldots, 30 ,$$

where the lower bound for the probability 2^{-4} is derived by counting conditions as described in Section 4.4.1. For instance, if we compute the probability for a disturbance in bit position $j = 3$ we get for both f_{XOR} and f_{MAJ} a probability of $2^{-3.9068}$ instead of 2^{-4}.

4.4.3 Disturbances in Adjacent Bit Positions

If we have a look at the disturbance vector in Table 2.5 or [172, Table 5], we notice that there occur disturbances in adjacent bit positions, *e.g.* $W'_i = +2^{j+1} + 2^j$ for f_{XOR}. Let us consider the concrete case with

$$\begin{aligned} \text{disturbance:} \quad & W'_i \quad = -2^1 + 2^0 \\ \text{corrections:} \quad & W'_{i+1} = +2^6 - 2^5 \\ & W'_{i+2} = +2^1 - 2^0 \\ & W'_{i+3} = +2^{31} + 2^{30} \\ & W'_{i+4} = +2^{31} + 2^{30} \\ & W'_{i+5} = +2^{31} - 2^{30} \end{aligned}$$

In a straightforward way, we can just treat them as separate disturbances and compute the probability based on (4.11). This results in a probability of

$$P_{LC}(f_{\text{XOR}}, -2^1 + 2^0) = \underbrace{2^{-2}}_{j=1}(\underbrace{2^{-4} + 2^{-6}}_{j=0}) = 2^{-5.678} \ .$$

However, by performing a detailed analysis we show that the probability for this case can be improved to $P_{LC}(f_{\text{XOR}}, -2^1 + 2^0) = 2^{-3.678}$ by defining two additional conditions including bits of the expanded message words only, referred to as CW_i and $CW1_{i+2}$. We assume that the easy conditions described in Section 4.4.1 are fulfilled. If no carry occurs in step i, both disturbances are corrected with probability 2^{-6}. This follows from Section 4.4.1. Now consider the case that a carry occurs in step i. Assume that in step i the disturbances have opposite signs, *e.g.* $W'_i = -2^1 + 2^0$. This can be ensured by fulfilling the new condition

$$W_{i,1} \oplus W_{i,0} = 1 \ . \qquad (CW_i)$$

If a carry occurs in bit position $j = 0$ the difference in state variable A'_{i+1} is -2^0 since the positive sign of the carry cancels the negative difference in $j = 1$ (see Table 2.2 in Section 2.5.2). This occurs with probability 1/2. In step $i + 1$ the probability is 1 since CW_{i+1} is fulfilled. In step $i + 2$, we can increase the probability to 1/2 if the additional condition is fulfilled:

$$W_{i+2,1} \oplus W_{i+2,0} = 1 \qquad (CW1_{i+2})$$

This is based on the same reasoning as for step i. For the remaining steps $i + 3$ and $i + 4$, we get a probability of 1/2 for each step. In step $i + 5$, we have again a probability of 1. Hence, we have a total probability of 2^{-4} for the case that a

carry occurs in step i. Therefore, the total probability for a local collision with f_{XOR} and the disturbance $W_i' = +2^1 - 2^0$ or $W_i' = -2^1 + 2^0$ is:

$$P_{LC}(f_{\text{XOR}}, -2^1 + 2^0) = \underbrace{2^{-4}}_{\text{carry in j=0}} + \underbrace{2^{-6}}_{\text{no carry in step } i} = 2^{-3.6781} \tag{4.13}$$

Wang *et al.* use a probability of 2^{-4} for their estimate of the collision attack complexity. The same analysis can be performed for disturbances in other adjacent bit positions. Again, one has to consider uncorrectable carries. Uncorrectable carries occur if the disturbances are in bit position $j = \{2, 1\}$ and $j = \{27, 26\}$. In these cases, we get the probability of both disturbances without carry. If $j = \{2, 1\}$, we obtain a probability of $2^{-4}2^{-2} = 2^{-6}$ and $j = \{27, 26\}$ results in $2^{-4}2^{-4} = 2^{-8}$. Let us now consider disturbances in adjacent bit positions from $j = 2, \ldots, 25$, *i.e.* $j = \{3, 2\}, \{4, 3\}, \ldots, \{26, 25\}$, and from $j = 27, \ldots, 30$, *i.e.* $j = \{28, 27\}, \{29, 28\}, \{30, 29\}, \{31, 30\}$. The formulae for all cases are given in (4.14), where j refers to the first entry of the tuple.

$$P_{LC}(f_{\text{XOR}}, \{j+1, j\}) = \begin{cases} 2^{-4} + 2^{-6} & \text{for } j = 0 \\ 2^{-4} + 2^{-8} & \text{for } j = 1 \\ \sum_{k=1}^{27-j} 2^{-4k} & \text{for } j = 2, \ldots, 25 \\ 2^{-4} + 2^{-8} & \text{for } j = 26 \\ 2 \cdot 2^{-4(32-j)} + \sum_{k=1}^{31-j} 2^{-4k} & \text{for } j = 27, \ldots, 30 \end{cases} \tag{4.14}$$

4.4.4 Update of Attack Complexity by Wang *et al.*

With the above analysis we covered all cases of disturbances that occur in the disturbance vector of [172]. Since Wang *et al.* count conditions in the last 60 steps of SHA-1 the overall probability can be updated based on (4.11), (4.12), and (4.13). Table 4.9 lists the comparison with [172, Table 9] and Table 4.10 summarizes the conditions that have to be fulfilled.

As can be seen in Table 4.9 the probability is by a factor of approx. 2.7 higher than estimated in [172]. Note that we did not count the disturbances in step $i = 21$ and step $i = 77$ since some of the conditions are fulfilled due to message modification or truncation. This means that the path of the disturbance is fixed and we cannot exploit any carry effects.

Remark 4.2. For the estimate of the probability such that the L-characteristic holds for SHA-1, we assumed that the local collision and corresponding probabilities are independent. We performed several measurements that agree with this assumption. However, in general one would need to look at conditional probabilities (see for instance [111]).

In the case of SHA-1 the improvements that follow from the accurate probability analysis are rather small. In fact, when implementing the final search in a naive way, we have to compute two instances of SHA-1 in order to monitor

Table 4.9: Update on complexity for collision attack on SHA-1.

[172, Table 9]				this section
disturbance bit position	disturbance index	number of conditions	estimated probability	accurate probability
$j = 1$	$23, 24, 27, 28, 32, 35, 36$	$2 \cdot 7 = 14$	2^{-14}	2^{-14}
$j = 0$	$25, 29, 33$	$4 \cdot 3 = 12$	2^{-12}	$2^{-11.0343}$
$j = 1$	$39, 43, 45, 47, 49$	$4 \cdot 5 = 20$	2^{-20}	2^{-20}
$j = 2$	65	$4 \cdot 1 = 4$	2^{-4}	$2^{-3.9068}$
$j = 7$	68	$4 \cdot 1 = 4$	2^{-4}	$2^{-3.9068}$
$j = 4$	71	$4 \cdot 1 = 4$	2^{-4}	$2^{-3.9068}$
$j = 3$	73	$4 \cdot 1 = 4$	2^{-4}	$2^{-3.9068}$
$j = 5$	74	$4 \cdot 1 = 4$	2^{-4}	$2^{-3.9068}$
		total	2^{-66}	$2^{-64.5683}$

the carry effects. This reduces the gained factor from 2.7 to 1.35. Nevertheless, if the disturbance vector has a higher Hamming weight, carry effects can have more impact on the probability. Let us look for instance at SHA1-IME proposed by Jutla and Patthak in [77] (a short description of this proposal is given in Section 7.1.1). They propose a modification of the message expansion of SHA-1 such that the Hamming weight of the disturbance vector is substantially greater than 25. In this case the impact of carry effects can lead to a much higher probability than one would estimate by counting conditions as discussed in Section 4.4.1.

Table 4.10: Conditions that need to be fulfilled in order to increase the probability of the pseudo-near-collision L-characteristic

	easy conditions					
disturbance	CW_i	CW_{i+1}	CW_{i+2}	CW_{i+3}	CW_{i+4}	CW_{i+5}
$W'_{23} = 2^1$	-	$W_{24,6} \oplus W_{23,1} = 1$	-	-	-	$W_{28,31} \oplus W_{23,1} = 1$
$W'_{24} = 2^1$	-	$W_{25,6} \oplus W_{24,1} = 1$	-	-	-	$W_{29,31} \oplus W_{24,1} = 1$
$W'_{25} = 2^0$	-	$W_{26,5} \oplus W_{25,0} = 1$	-	-	-	$W_{30,30} \oplus W_{25,0} = 1$
$W'_{27} = 2^1$	-	$W_{28,6} \oplus W_{27,1} = 1$	-	-	-	$W_{32,31} \oplus W_{27,1} = 1$
$W'_{28} = 2^1$	-	$W_{29,6} \oplus W_{28,1} = 1$	-	-	-	$W_{33,31} \oplus W_{28,1} = 1$
$W'_{29} = 2^0$	-	$W_{30,5} \oplus W_{29,0} = 1$	-	-	-	$W_{34,30} \oplus W_{29,0} = 1$
$W'_{32} = 2^1$	-	$W_{33,6} \oplus W_{32,1} = 1$	-	-	-	$W_{37,31} \oplus W_{32,1} = 1$
$W'_{33} = 2^1 + 2^0$	$W_{33,1} \oplus W_{33,0} = 1$	$W_{34,5} \oplus W_{33,0} = 1$	$W_{35,1} \oplus W_{35,0} = 1$ *	-	-	$W_{38,30} \oplus W_{33,0} = 1$
$W'_{35} = 2^1$	-	$W_{36,6} \oplus W_{35,1} = 1$	-	-	-	$W_{40,31} \oplus W_{35,1} = 1$
$W'_{36} = 2^1$	-	$W_{37,6} \oplus W_{36,1} = 1$	-	-	-	$W_{41,31} \oplus W_{36,1} = 1$
$W'_{39} = 2^1$	-	$W_{40,6} \oplus W_{39,1} = 1$	$W_{41,1} \oplus W_{39,1} = 1$	-	-	$W_{44,31} \oplus W_{39,1} = 1$
$W'_{43} = 2^1$	-	$W_{44,6} \oplus W_{43,1} = 1$	$W_{45,1} \oplus W_{43,1} = 1$	-	-	$W_{48,31} \oplus W_{43,1} = 1$
$W'_{45} = 2^1$	-	$W_{46,6} \oplus W_{45,1} = 1$	$W_{47,1} \oplus W_{45,1} = 1$	-	-	$W_{50,31} \oplus W_{45,1} = 1$
$W'_{47} = 2^1$	-	$W_{48,6} \oplus W_{47,1} = 1$	$W_{49,1} \oplus W_{47,1} = 1$	-	-	$W_{52,31} \oplus W_{47,1} = 1$
$W'_{49} = 2^1$	-	$W_{50,6} \oplus W_{49,1} = 1$	$W_{51,1} \oplus W_{49,1} = 1$	-	-	$W_{54,31} \oplus W_{49,1} = 1$
$W'_{65} = 2^2$	-	$W_{66,7} \oplus W_{65,2} = 1$	-	-	-	$W_{70,0} \oplus W_{65,2} = 1$
$W'_{68} = 2^7$	-	$W_{69,12} \oplus W_{68,7} = 1$	-	-	-	$W_{73,5} \oplus W_{68,7} = 1$
$W'_{71} = 2^4$	-	$W_{72,9} \oplus W_{71,4} = 1$	-	-	-	$W_{76,2} \oplus W_{71,4} = 1$
$W'_{73} = 2^3$	-	$W_{74,8} \oplus W_{73,3} = 1$	-	-	-	$W_{78,1} \oplus W_{73,3} = 1$
$W'_{74} = 2^5$	-	$W_{75,10} \oplus W_{74,5} = 1$	-	-	-	$W_{79,3} \oplus W_{74,5} = 1$

* . . . easy condition $CW1_{i+2}$ as described in Section 4.4.3

4.5 A Generalization of the Collision Attack Strategy of Wang *et al.*

The seminal work of Wang *et al.* in the cryptanalysis of hash functions such as MD4, MD5, and SHA-1, motivated a lot of researchers to investigate how the manual search for the L-characteristic and NL-characteristic can be automated. The first cryptanalytic tool for an automated search of the NL-characteristic for SHA-1 has been presented by De Cannière and Rechberger in [28]. In the following, we will give a short and abstract overview of their automated search method which resulted in the best cryptanalytic results on SHA-1 to date: a 64-step collision has been presented in [28] and further optimizations led to a 70-step collision for SHA-1 by Mendel *et al.* in [27].

This generalization of the method of Wang *et al.* is based on the fact that instead of considering signed-bit differences, the authors of [28] look at pairs of bits. We know from Section 2.5.1 that signed-bit differences also include information about the bit values of the message and not only the difference. This has been generalized in [28] by not only considering single bits but by looking at pairs of bits, resulting in 16 possible conditions. Based on this generalization the working principle of their automated search can be summarized as follows. The first step is to find a high probability L-characteristic as described in detail in this chapter. Based on this L-characteristic the input difference in the message is fixed (linear message expansion) and for instance at step 20 we are given a 'target difference'. This is basically the start point for the automated search. Initially, for the first 20 steps of the state update transformation no conditions are set. Then the algorithm starts to add conditions at randomly chosen bit positions. In the first phase conditions are set in such a way that no difference is allowed, *i.e.* a zero-difference condition is forced. This is done in order to find a sparse NL-characteristic.

The most complex part is then to monitor how the imposed conditions propagate forward and backward in the state update transformation. If a contradiction for the forced conditions occurs, the tool starts from scratch by setting again zero-difference conditions at randomly chosen bit positions. Once all conditions have been fixed an NL-characteristic has been found.

In [28], the authors also introduce a new method for computing the probability of the found NL-characteristic. This method allows to give the accurate complexity for finding a message pair that follows the characteristic and hence leads to colliding hash values. If the remaining search space is still large enough then a greedy approach can be used to still improve the probability of the characteristic and hence, to reduce the overall complexity of the collision search.

We stress that this is a very rough description of the automated search method and additional details and special cases have to be considered for applying the search tool. Therefore, we refer the reader to [27, 28] for a complete description. Based on the results of this new automated collision search, recently, also some speed-up techniques have been presented by Sugita *et al.* in [160], and by Joux and Peyrin in [75].

4.6 Summary

In this chapter, we have given a description of the building blocks of SHA-1 and have reviewed existing collision-attack strategies. We have explained in detail how one can find high-probability L-characteristics by exploiting coding theory. We followed a systematic approach for defining several linear codes and have shown how we can apply probabilistic algorithms for searching codewords with low Hamming weight. Several improvements for a speed-up of these algorithms have been presented by exploiting the internal structure of SHA-1. We have introduced an analytical approach do determine the probability of L-characteristics. These probability estimates also include side-effects such as the impact of carries on the overall probability. We have found the same L-characteristic as Wang *et al.* in [172] and have demonstrated how one can accurately determine its probability. This in turn led to a slight improvement of the probability given by Wang *et al.* At the end of this chapter, we have given a rough description of new results for finding an NL-characteristic. We note that this is the most complex part of the collision search for SHA-1. We emphasize that the starting point for this new approach is a high-probability L-characteristic that can be found as described in this chapter.

5

Cryptanalysis of Selected Hash Function Proposals

In this chapter, we present second preimage attacks on two recent hash function proposals which do not follow the design strategy of the MD-family. Firstly, we start with a collision attack on the design strategy SMASH, which will then be extended to a second preimage attack. Both the collision and the second preimage attack are purely structural. Furthermore, for messages of a certain length the attacks are deterministic.

Secondly, we present a second preimage attack on a double-block-length hash function. This attack works if the construction is instantiated with a block cipher following the FX construction. For different configurations, we are able to construct second preimages deterministically. As with SMASH, this second preimage attack is a purely structural attack too. The material of this chapter was presented in [95, 136, 141].

5.1 Cryptanalysis of SMASH

In this section, we present an attack on the hash function design strategy SMASH. We briefly introduce the design strategy and discuss specific properties that have been exploited for the collision and second preimage attack. We follow the notation of [87], except that we denote finite field addition by '+', and we stick to the convention of [19] to denote a difference by $h' = h + h^*$.

5.1.1 The Design Strategy SMASH

At FSE 2005, Knudsen [87] proposed a new hash function design strategy. It is an iterated hash function with a compression function that is based on a nonlinear bijective n-bit mapping $f : \{0,1\}^n \to \{0,1\}^n$. Let $m = m_1 \| \dots \| m_t$ be the message input after MD strengthening (*cf.* Section 2.3), where each block

m_i consists of n bits. The hash output h_{t+1} is computed as follows:

$$\begin{aligned} h_0 &= f(\text{iv}) + \text{iv} && (5.1)\\ h_i &= f(h_{i-1} + m_i) + h_{i-1} + \theta m_i \qquad \text{for } i = 1, \ldots, t && (5.2)\\ h_{t+1} &= f(h_t) + h_t \ , && (5.3) \end{aligned}$$

where h_i denotes the chaining value after iteration i. The multiplication by θ in (5.2) is defined as an operation in the finite field $GF(2^n)$. Note that if we are talking about the design strategy then θ is an arbitrary field element in $GF(2^n)$ with the only restriction that $\theta \notin \{0, 1\}$ as specified in [87]. If we consider concrete instances of the design strategy, we will properly define the element θ.

As opposed to the design strategy of the MD4 family, SMASH applies the nonlinear bijective mapping f also to the initial value (5.1) and to the final hash computation (5.3). According to Knudsen, *cf.* [87, Section 2.2], this is done in order to avoid pseudo-collision attacks (*cf.* Section 2.3.3), since the attacker does not have full control over the chaining value h_0. Applying f also to the final hash computation should make it impossible to predict the final hash value. The final hash computation can also be seen as the application of (5.2) with $m_i = 0$.

Knudsen proposed two concrete instances of the SMASH design strategy, namely SMASH-256 and SMASH-512. These two instances were intended to be candidate alternatives to SHA-256 and SHA-512 [128]. In the following sections, we will briefly describe both instances. We will give only the necessary details needed for the understanding of the attack.

SMASH-256

The specific instance SMASH-256 is specified by setting $n = 256$. The finite field element θ is defined as a root of the irreducible polynomial

$$q(\theta) = \theta^{256} + \theta^{16} + \theta^3 + \theta + 1 \ ,$$

which defines the arithmetic in $GF(2^{256})$.

Even if the properties of the nonlinear bijective mapping f are irrelevant for our attack, we briefly describe them to give a basic understanding of the SMASH design strategy. The nonlinear mapping f for SMASH-256 is composed of several applications of so-called H-rounds and L-rounds:

$$f = H_1 \circ H_3 \circ H_2 \circ L \circ H_1 \circ H_2 \circ H_3 \circ L \circ H_2 \circ H_1 \circ H_3 \circ L \circ H_3 \circ H_2 \circ H_1$$

Both, the H-rounds and L-rounds take as input a 256-bit value and produce a 256-bit output. The underlying operations are S-Boxes, some linear diffusion layers, and rotations. Each H round has a different rotation constant. The S-Boxes are based on the S-Boxes employed in the block cipher Serpent [16]. Due to the chosen padding method, SMASH-256 can be used to process messages of length less than 2^{128} bits.

SMASH-512

SMASH-512 is specified by setting $n = 512$ and, as in the case of SMASH-256, the finite field element θ is defined as a root of the irreducible polynomial

$$q(\theta) = \theta^{512} + \theta^8 + \theta^5 + \theta^2 + 1 \ ,$$

which defines the arithmetic in $GF(2^{512})$. Due to the chosen padding method one can process messages with a length less than 2^{256} bits. The nonlinear 512-bit mapping f works along the same lines as the mapping for SMASH-256 (*cf.* [87] for further details).

5.1.2 Specific Properties of SMASH

Forward Prediction Property (FPP)

The structure of SMASH exhibits a *forward prediction* property as has been already observed and described in the design document [87]:

Property 5.1. *(Forward prediction property of SMASH) Let h_{i-1} and h^*_{i-1} be two intermediate hash values with difference $h'_{i-1} = h_{i-1} + h^*_{i-1}$. If, for an arbitrary m_i the message block m^*_i is computed as $m^*_i = m_i + h'_{i-1}$, then the output difference is given by*

$$h'_i = h_i + h^*_i = (1 + \theta)h'_{i-1}, \quad 2 \leq i \leq t \ .$$

Obviously, Property 5.1 can be extended to hold over more iterations. That is for $k \leq t - 2$ iterations we have

$$h'_{i+k-1} = (1 + \theta)^k h'_{i-1}, \quad 2 \leq i \leq t - k \ .$$

Pattern Construction Property (PCP)

For the SMASH strategy one can also control the input values to the nonlinear function f in different iterations (except the last iteration). We refer to this property as the *pattern construction* property:

Property 5.2. *(Pattern construction property of SMASH) Let the output of the nonlinear function f in iteration i and $i + k$ be denoted by f_i and f_{i+k}, where $1 \leq i + k < t$. If we choose $m_{i+k} = m_i + h_{i-1} + h_{i+k-1}$ then the output of f is the same in both iterations:*

$$\begin{aligned} f_i &= f(m_i + h_{i-1}) \\ f_{i+k} &= f(m_{i+k} + h_{i+k-1}) \\ &= f(m_i + h_{i-1} + h_{i+k-1} + h_{i+k-1}) = f_i \end{aligned}$$

It is clear that the pattern construction property can be applied in more than two iterations. In general, we can construct messages for which the input to f is the same in all iterations by using Property 5.2. Note that if we construct two inputs to f as described by Property 5.2, then, regarding the nonlinear function f, the same input difference in both iterations leads to the same output difference.

5.1.3 Collision Attack

The properties described in Section 5.1.2 can be exploited for collision attacks on the SMASH design strategy. We start by demonstrating the basic attack strategy by looking at a simplified variant of SMASH. Afterwards, we will extend the strategy for collision attacks on SMASH-256 and SMASH-512.

Collisions for Simplified Variants SMASH-ORD3 and SMASH-ORDy

In order to explain our attack, we first consider a simple instance of SMASH. The instance, referred to as SMASH-ORD3, differs from SMASH in the choice of the finite field element θ. More, precisely, we assume that we can choose a θ such that the element $(1+\theta)$ has order 3, *i.e.* $(1+\theta)^3 = 1$. Such a choice is not explicitly forbidden in [87].

To explain the collision attack on the simplified variant of SMASH, we define the following variables (see also Figure 5.1):

$$\begin{array}{ll} x & \text{an arbitrary } n\text{-bit value} \\ f_1 & = f(m_1 + h_0) \\ f_2 & = f(m_2 + h_1) \\ f_3 & = f(m_3 + h_2) \\ a & = f_1 + f(m_1 + h_0 + x) + \theta x \end{array}$$

The variable x defines an arbitrary n-bit difference. $f_1, \ldots, f_3$ are the output values of f with message block m_i and intermediate chaining variable h_{i-1}, where $i = 1, \ldots, 3$, as input. The difference in h_1 is defined by a. Based on these definitions, we can now construct two 4-block messages $m = m_1 \| m_2 \| m_3 \| m_4$, where m_1, m_2, m_3 can be chosen arbitrarily, and $m^* = m_1^* \| m_2^* \| m_3^* \| m_4^*$ as follows:

$$\begin{array}{lcl} m_4 & = & m_1 + f_1 + f_2 + f_3 + \theta(m_1 + m_2 + m_3) \\ m_1^* & = & m_1 + x \\ m_2^* & = & m_2 + a \\ m_3^* & = & m_3 + (1+\theta)a \\ m_4^* & = & m_4 + (1+\theta)^2 a + x \end{array} \tag{5.4}$$

Note that a depends on both m_1 and x. In particular, if $x = 0$ then $a = 0$, *i.e.* $m = m^*$. As we will describe in the following, the two 4-block messages m and m^* defined in (5.4) lead to the same hash value, a collision. The attack is graphically illustrated in Figure 5.1.

The attack is an extension of the forward prediction property (Property 5.1). It can be verified that the value a is the difference in h_1. We cannot predict the value of a, but we can of course easily compute it once we have chosen an arbitrary m_1 and x. Note that this also determines the difference $f_1' = a + \theta x$.

The basic idea of the attack is to control the propagation of the difference such that the input differences to the function f in the second iteration and in the last but one iteration (*iteration 2* and *iteration 3* in Figure 5.1) equal zero. In this case the nonlinearity of f does not have any impact on the difference propagation. For the last message block (*iteration 4*) we ensure that the difference $m_4' = m_4 +$

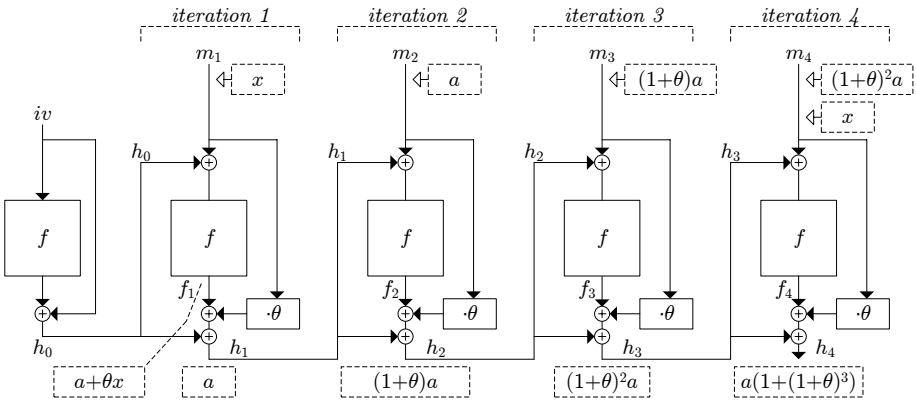

Figure 5.1: The attack on SMASH-ORD3. The dashed rectangles denote differences.

m_4^* equals the difference $m_1' = m_1 + m_1^*$ and that $f_1' = f_4'$. This can be achieved by exploiting the pattern construction property as described in Property 5.2. By choosing the difference in the second message block equal to $m_2' = a$, we make sure that the input difference to f equals zero (differences cancel out). Hence, we ensure that the difference $h_2' = (1+\theta)a$. Similarly, by choosing the difference in the third message block equal to $m_3' = (1+\theta)a$, we ensure that the difference $h_3' = (1+\theta)^2 a$. These two steps exploit the forward prediction property.

Now, we have to determine the last message block in each of the messages. We choose the last message block of the first message, m_4, following the pattern construction property, *i.e.* $m_1 + h_0 = m_4 + h_3$:

$$\begin{aligned}
m_4 &= m_1 + h_0 + h_3 \\
&= m_1 + h_0 + \underbrace{f_3 + h_2 + \theta m_3}_{h_3} \\
&= m_1 + h_0 + f_3 + \underbrace{f_2 + h_1 + \theta m_2}_{h_2} + \theta m_3 \\
&= m_1 + h_0 + f_3 + f_2 + \underbrace{f_1 + h_0 + \theta m_1}_{h_1} + \theta m_2 + \theta m_3 \\
&= m_1 + f_3 + f_2 + f_1 + \theta(m_3 + m_2 + m_1)
\end{aligned}$$

The last block of the second message, m_4^*, is selected in such a way that the difference in the last message block $m_4' = (1+\theta)^2 a + x$. This choice ensures that the input of f in the last iteration (*iteration 4*) equals the input in the first iteration (*iteration 1*). Consequently, the output of f will be the same as in the first iteration, hence it will have the same difference as in the first iteration, namely $f_1' = f_4' = a + \theta x$ (*cf.* Figure 5.1). Working out the equations, we see

that the difference in h_4 becomes:

$$\begin{aligned} h'_4 &= f'_4 + h'_3 + m'_4 \\ &= f'_4 + h'_3 + \theta x + (1+\theta)^2\theta a \\ &= a + \theta x + (1+\theta)^2 a + \theta x + (1+\theta)^2\theta a \\ &= a + (1+\theta^2)a + (1+\theta^2)\theta a \\ &= \theta a + \theta^2 a + \theta^3 a \\ &= a(1 + (1+\theta)^3) \end{aligned} \tag{5.5}$$

Since we assumed that we can choose a θ such that $(1+\theta)$ has order 3, the difference in (5.5), $h'_4 = a(1+(1+\theta)^3)$, equals zero and we have produced a collision for our simple SMASH instance SMASH-ORD3. Due to the collision after the last iteration ($h'_4 = 0$), the final hash computation (5.3) has no impact on the result.

The attack on SMASH-ORD3 can be generalized to break the simplified SMASH instances, referred to as SMASH-ORDy. The instances SMASH-ORDy are defined by choosing a θ such that the order of the element $(1+\theta)$ equals y, where y is an arbitrary value with $y \geq 3$. As we have observed for the instance SMASH-ORD3, we have a collision in iteration 4. Along the same lines it is easy to show that for SMASH-ORDy we can construct collisions for messages consisting of at least $y+1$ blocks without counting in the last message block that results from MD strengthening.

Collisions for SMASH-256 and SMASH-512

For the attacks on SMASH-ORD3 and SMASH-ORDy, we assumed that we can choose a certain finite field element θ. This is not possible for the specific instances of SMASH. For SMASH-256 the finite field element θ is defined as a root of the irreducible polynomial $q(\theta) = \theta^{256} + \theta^{16} + \theta^3 + \theta + 1$, *i.e.* $q(\theta) = 0$. Also for SMASH-512, θ is defined as a root of the irreducible polynomial. In order to show whether or not the previously described attacks on SMASH-ORD3 and SMASH-ORDy can be applied to SMASH-256 and SMASH-512, we have to compute the order of the element $(1+\theta)$ for the specified θ. For SMASH-256, the order of $(1+\theta)$ equals $((2^{256}-1)/5)$ and for SMASH-512 the order of $(1+\theta)$ is $(2^{512}-1)$. For both SMASH-256 and SMASH-512, we computed the order of the element $(1+\theta)$ using Maple. Based on the order of the element $(1+\theta)$ the attack requires at least $(((2^{256}-1)/5)+1)$ message blocks for SMASH-256 and 2^{512} message blocks for SMASH-512, respectively. As specified in [87], SMASH-256 can be used to hash messages of bit length less than 2^{128}. This corresponds to $(2^{120}-1)$ message blocks of 256 bits each. SMASH-512 is specified for $(2^{247}-1)$ message blocks of 512 bits each. Hence, the order of the element $(1+\theta)$ is for both hash functions larger than the maximum number of message blocks. This means, that we can still produce colliding messages but these messages are no longer valid inputs according to the SMASH-256 and SMASH-512 specification.

Previously, we exploited the pattern construction property by considering message pairs that introduce a non-zero input difference x into f twice: once at the beginning and once at the end of the message. As already mentioned in Section 5.1.2, we can extend this property further by considering message pairs that introduce the difference x three or more times. In other words, we exploit the pattern construction property in several iterations. Every time the input difference to f is non-zero, we make sure that the values of the two message blocks equal the values in the first message blocks. Consequently, the output difference of f will be every time the same, namely $(a + \theta x)$, *cf.* Figure 5.1.

In this way, we can produce almost any desired difference in the chaining variable h_t. In order to find a collision, we want to construct a difference that is a multiple of the irreducible polynomial, *i.e.* a difference of the form $h'_t = a \cdot q(\theta) = a \cdot 0 = 0 \pmod{q(\theta)}$.

Based on Property 5.1 and Property 5.2, it can be shown that we can construct differences in the chaining values of the form

$$h'_t = a \sum_{i=1}^{t} (1+\theta)^{t-i} \delta_i \ , \tag{5.6}$$

where the $\delta_i \in \{0, 1\}$ are arbitrary (*cf.* Theorem 5.1). Thus, our problem boils down to find values δ_i that fulfill

$$a \cdot q(\theta) = a \sum_{i=1}^{t} (1+\theta)^{t-i} \delta_i \ .$$

How these δ_i's can be determined in general will be shown in Section 5.1.4. For the collision attack on the concrete instances SMASH-256 and SMASH-512, we will simply rewrite the irreducible polynomials $q(\theta)$ in terms of $(1+\theta)^i$ to achieve our goal.

For given $\delta_1, \ldots, \delta_i$, we can then compute the difference m'_i as follows:

$$m'_i = \begin{cases} \sum_{j=1}^{i-1} (1+\theta)^{i-j-1} a\delta_j & \text{if } \delta_i = 0 \\ \\ x + \sum_{j=1}^{i-1} (1+\theta)^{i-j-1} a\delta_j & \text{if } \delta_i = 1 \ , \end{cases} \tag{5.7}$$

where x is the arbitrarily chosen difference (*cf.* Figure 5.1). The values of m_i can be determined as follows. The first block, m_1, can always be selected arbitrarily. If $\delta_i = 0$, then m_i can be selected arbitrarily. If $\delta_i = 1$ and $i > 1$, then m_i has to be equal to $h_{i-1} + m_1 + h_0$ (*cf.* Property 5.2).

Solutions for SMASH-256. In order to show that we can construct a difference that is a multiple of the irreducible polynomial, we have to rewrite the polynomial such that it contains powers of $(1 + \theta)$. For the irreducible polynomial $\theta^{256} + \theta^{16} + \theta^3 + \theta + 1$ we need the powers 256, 16, and 3. We know that $(1 + \theta)^{256} = 1 + \theta^{256}$. The same holds for $(1 + \theta)^{16} = 1 + \theta^{16}$. To get the power

3 we have to use $(1+\theta)^3$ resulting in $(1+\theta)^3 = 1+\theta+\theta^2+\theta^3$. Summing up these three terms results in

$$1+\theta^{256}+1+\theta^{16}+1+\theta+\theta^2+\theta^3 = \theta^{256}+\theta^{16}+\theta^3+\theta^2+\theta+1\ .$$

To eliminate θ^2 we need to introduce $(1+\theta)^2 = 1+\theta^2$. Finally, we add 1 and the irreducible polynomial $q(\theta)$ can be rewritten as:

$$\begin{aligned} \theta^{256}+\theta^{16}+\theta^3+\theta+1 = \\ 1+(1+\theta)^2+(1+\theta)^3+(1+\theta)^{16}+(1+\theta)^{256} \end{aligned} \tag{5.8}$$

Since we want to construct a difference that is a multiple of the irreducible polynomial, we can solve (5.6) for the polynomial (5.8). It follows that $t = 257$ and the solutions are $\delta_i = 1$ for $i = 1, 241, 254, 255, 257$ and $\delta_i = 0$ for all other $i \leq 257$. Given the δ_i's, the differences m'_i can be computed as shown in (5.7). This gives:

$$\begin{aligned} m'_1 &= x \\ m'_i &= (1+\theta)^{i-2}a, && 1 < i \leq 240 \\ m'_{241} &= x+(1+\theta)^{239}a \\ m'_i &= (1+\theta)^{i-2}a+(1+\theta)^{i-242}a, && 241 < i < 254 \\ m'_{254} &= x+(1+\theta)^{252}a+(1+\theta)^{12}a \\ m'_{255} &= x+(1+\theta)^{253}a+(1+\theta)^{13}a+a \\ m'_{256} &= (1+\theta)^{254}a+(1+\theta)^{14}a+(1+\theta)a+a \\ m'_{257} &= x+(1+\theta)^{255}a+(1+\theta)^{15}a+(1+\theta)^2a+(1+\theta)a \end{aligned}$$

Here, x is an arbitrary 256-bit difference. All other differences are defined by the attack method. As explained above, 253 of the message blocks m_i can be chosen arbitrarily, while the remaining 4 are determined by the attack.

The above-defined differences produce the following difference in the chaining variable h_{257}:

$$h'_{257} = (1+\theta)^{256}a+(1+\theta)^{16}a+(1+\theta)^3a+(1+\theta)^2a+a$$

It is clear that $h'_{257} = a \cdot q(\theta) = a \cdot 0 = 0$, and hence we have a collision after iteration 257.

SMASH-256: Example of colliding messages. We give an example of two messages, m and $m^* = m + m'$, that collide after 257 iterations. Note that for this example we omitted padding.

Figure 5.2 shows the two colliding ASCII coded strings. Each message consists of 258 message blocks. Even if we have already a collision after iteration 257 (see also Table 5.1), we added an additional message block, $m_{258} = m^*_{258}$, containing the character '>' (`3e`). We have chosen the two messages in this way, because the message blocks $m_2, \ldots, m_{258}$ and $m^*_2, \ldots, m^*_{258}$ are inside the end

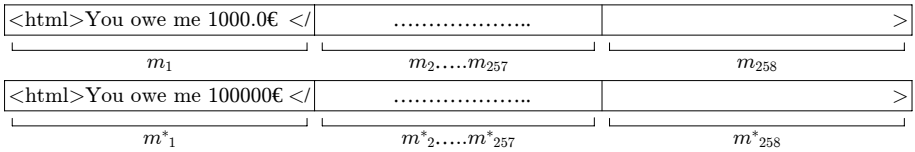

Figure 5.2: Examples of colliding messages for SMASH-256. ASCII coded strings m and m^*.

tag ($< / m_2, \ldots, m_{258} >$) and hence are not displayed in a standard *HTML* viewer or web browser (*e.g.* Mozilla Firefox 1.0.3). Therefore, at first sight, only the two message blocks m_1 and m_1^* are visible.

Using hexadecimal notation, Table 5.1 shows the input message blocks m_i and m_i^* for $i = 1, 241, 254, 255, 257, 258$, the initial chaining variables $h_0 = h_0^* = f(\text{iv}) + \text{iv}$, the chaining variables h_{257}, h_{257}^*, h_{258}, and h_{258}^*, and the colliding outputs h_{259} and h_{259}^*. The message blocks $m_2, \ldots, m_{240}, m_{242}, \ldots, m_{253}$, and m_{256} can be chosen arbitrarily. In this simple example each of these message blocks contains only space characters (20).

Table 5.1: Colliding messages m and m^* for SMASH-256. Table entries are given in hexadecimal representation.

h_0=	55214e9e	237290c2	3ff782f7	c2073a8c	2105c5f1	6ccb0855	9c71b7c1	e7ecceac
h_0^*=	55214e9e	237290c2	3ff782f7	c2073a8c	2105c5f1	6ccb0855	9c71b7c1	e7ecceac
m_1=	3c68746d	6c3e596f	75206f77	65206d65	20313030	302e3030	80202020	20203c2f
m_1^*=	3c68746d	6c3e596f	75206f77	65206d65	20313030	30303030	80202020	20203c2f
				.....				
m_{241}=	346a6100	4e3cbc5b	f472d355	b41311b2	4b7df46d	e6b4028f	6aaf9c4d	97a6f169
m_{241}^*=	4afe2771	dd8507d9	a25082bc	dac25578	f34abb1c	5501e05e	d9874798	6aa679d3
				.....				
m_{254}=	60358467	cfde2276	534a4038	d3555d7e	576415d4	5c151dbb	7664ed09	f97bb393
m_{254}^*=	de2cdecd	6e323f7e	de8e653b	7d887168	f94cebb5	7370fc51	0e2e0226	8f1e25ba
m_{255}=	40088790	06da5567	eb2a1d6e	2869d96f	02fb791a	dc8799ca	0df2d9de	9dec9799
m_{255}^*=	f81b73fc	db0f8afe	28947e42	699822e0	6de2b3b5	0b5536fe	c3d1ebb0	7744d316
				.....				
m_{257}=	cc4a7c9c	9b4a99e1	8d275de9	3a44a2e7	4640484b	3cb2abb4	f1af679f	4e6e142f
m_{257}^*=	1a5d75eb	71ea319e	be76a60e	abc9278b	329ff04f	5e932f4d	cc04996a	9e6c4183
h_{257}=	ddc0b465	b42b5072	c34ad69d	b47c8e2c	30a36a7b	218a7bbe	99ffd185	831e8ddf
h_{257}^*=	ddc0b465	b42b5072	c34ad69d	b47c8e2c	30a36a7b	218a7bbe	99ffd185	831e8ddf
m_{258}=	20202020	20202020	20202020	20202020	20202020	20202020	20202020	2020203e
m_{258}^*=	20202020	20202020	20202020	20202020	20202020	20202020	20202020	2020203e
h_{258}=	da7c8fa1	4389e3c5	7299afdd	ad027de9	4c595315	c981c2f8	95390053	37c2fa00
h_{258}^*=	da7c8fa1	4389e3c5	7299afdd	ad027de9	4c595315	c981c2f8	95390053	37c2fa00
h_{259}=	2ffeac86	08bc1142	a3ddf493	6455bcd8	673dea34	c6365ec3	92b1bc79	15c1487e
h_{259}^*=	2ffeac86	08bc1142	a3ddf493	6455bcd8	673dea34	c6365ec3	92b1bc79	15c1487e

SMASH-512: Equations and solutions. To determine the differences m_i' that produce a collision for SMASH-512, we use the same definitions and equations as we have used for SMASH-256.

SMASH-512 is specified by setting $n = 512$ and by defining the finite field $\mathrm{GF}(2^{512})$ via the irreducible polynomial $q(\theta)$

$$q(\theta) = \theta^{512} + \theta^8 + \theta^5 + \theta^2 + 1 \ .$$

Rewriting the polynomial $q(\theta)$ such that it consists of powers of $(1+\theta)$ works along the same lines as for SMASH-256. It can be rewritten as follows:

$$q(\theta) = 1 + (1+\theta) + (1+\theta)^2 + (1+\theta)^4 + (1+\theta)^5 + (1+\theta)^8 + (1+\theta)^{512}$$

The solution of (5.6) is given by $\delta_i = 1$ for $i = 1, 505, 508, 509, 511, 512, 513$ and $\delta_i = 0$ for all other $i \leq 513$. Given the δ_i's, the differences m'_i can be computed as shown in (5.7). This gives:

$$\begin{aligned}
m'_1 &= x \\
m'_i &= (1+\theta)^{i-2}a, && 1 < i \leq 504 \\
m'_{505} &= x + (1+\theta)^{503}a \\
m'_i &= (1+\theta)^{i-2}a + (1+\theta)^{i-506}a, && 505 < i < 508 \\
m'_{508} &= x + (1+\theta)^{506}a + (1+\theta)^2 a \\
m'_{509} &= x + (1+\theta)^{507}a + (1+\theta)^3 a + a \\
m'_{510} &= (1+\theta)^{508}a + (1+\theta)^4 a + (1+\theta)a + a \\
m'_{511} &= x + (1+\theta)^{509}a + (1+\theta)^5 a + (1+\theta)^2 a + (1+\theta)a \\
m'_{512} &= x + (1+\theta)^{510}a + (1+\theta)^6 a + (1+\theta)^3 a + (1+\theta)^2 a + a \\
m'_{513} &= x + (1+\theta)^{511}a + (1+\theta)^7 a + (1+\theta)^4 a + (1+\theta)^3 a + (1+\theta)a + a
\end{aligned}$$

Here x is an arbitrary 512-bit difference. All other differences are defined by the attack method. For SMASH-512, 507 of the message blocks m_i can be chosen arbitrarily, while the remaining 6 are determined by the attack.

The above-defined differences produce the following difference in the chaining variable h_{513}:

$$\begin{aligned}
h'_{513} &= (1+\theta)^{512}a + (1+\theta)^8 a + (1+\theta)^5 a \\
&+ (1+\theta)^4 a + (1+\theta)^2 a + (1+\theta)a + a
\end{aligned}$$

It is clear that $h'_{513} = a \cdot q(\theta) = a \cdot 0 = 0$, and hence we have a collision for SMASH-512 after iteration 513.

5.1.4 Second Preimage Attack

In the following we describe the remarkable fact that the collision attack on the SMASH design strategy can be generalized resulting in a second preimage attack. The attack also exploits the fact that we can control the differences throughout the application of the compression function.

In the following, we define the messages that we need in order to construct arbitrary differences in a chaining variable of SMASH: the *base message* and the

offset message. The aim is to construct a base message in such a way that by constructing an according offset message we can produce an arbitrary difference of our choice in the chaining variable. By doing so, we can construct preimages or second preimages as we will show in the following sections. Firstly, we consider the SMASH design strategy in general and then we will look at the specific instances SMASH-256 and SMASH-512.

Base Message

We define a *base message* consisting of n message blocks. The first block m_1 can be chosen arbitrarily. The next blocks are defined as follows:

$$m_i = m_1 + h_0 + h_{i-1} \quad \text{for } 1 < i \leq n \ , \tag{5.9}$$

where h_i denotes the value of the chaining variable after processing message block m_i. The value h_0 is defined by (5.1) and h_i is defined by (5.2). The purpose of this base message is to ensure that the inputs to the function f are the same in all iterations. In other words, we can use the pattern construction property (Property 5.2) in each iteration.

Offset Message

We denote an n-block *offset message* by $m^*(\delta)$, where δ is an n-bit vector. The single elements of this vector are denoted by $\delta_1, \ldots, \delta_n$, where $\delta_i \in \{0, 1\}$. The message blocks of $m^*(\delta)$ are denoted by m_i^*. Furthermore, we denote the value of the chaining variable after processing the message block m_i^* by h_i^* according to (5.1) and (5.2). Let x be an arbitrary n-bit block, different from zero, and let a denote the value

$$\begin{aligned} a &= f(m_1 + h_0) + f(m_1 + x + h_0) + \theta x \\ &= f(m_i + h_{i-1}) + f(m_i + x + h_{i-1}) + \theta x \ , \end{aligned} \tag{5.10}$$

where the second equality follows from the definition of the base message in (5.9). Then, we can define the offset message as follows:

$$\begin{aligned} m_1^* &= m_1 + \delta_1 x \\ m_i^* &= m_i + \delta_i x + a \sum_{j=1}^{i-1} \delta_j (1+\theta)^{i-j-1} \quad \text{for } 1 < i \leq n \end{aligned} \tag{5.11}$$

Constructing the Desired Offset Message

Using Property 5.1 and Property 5.2, we show the following theorem which is also given without proof in Section 5.1.3, *cf.* (5.6). The proof is based on the fact that (5.9) and (5.11) have the following effect on the input to the f function. Let $y_i = m_i + h_{i-1}$ be the inputs to f when processing the base message m.

Similarly, let y_i^* be the inputs to the compression function when processing the offset message m^*. In case of the base message m, we have

$$\forall i, \quad 1 \le i < n : y_i = m_1 + h_0 \ .$$

In case of the offset message m^*, the inputs to f depend on δ but still can have only two values:

$$\forall i, \quad 1 \le i < n : y_i = m_1 + h_0 + \delta_i x$$

Theorem 5.1. *For any nonzero value x, defining m, δ, a, and $m^*(\delta)$ as before, it holds that*

$$h_n + h_n^* = a \sum_{j=1}^{n} \delta_j (1+\theta)^{n-j} \ .$$

Proof. We prove the theorem by induction, showing that

$$h_t + h_t^* = a \sum_{j=1}^{t} \delta_j (1+\theta)^{t-j} \quad \text{for } 1 \le t \le n \ . \tag{5.12}$$

In the first step, we set $t = 1$. If $\delta_1 = 0$, then we have from (5.11) that $m_1 = m_1^*$ and hence $h_1 + h_1^* = 0$. If $\delta_1 = 1$, then $m_1^* = m_1 + x$ and

$$\begin{aligned} h_1 + h_1^* &= (f(h_0 + m_1) + h_0 + \theta m_1) + (f(h_0 + m_1 + x) + h_0 + \theta(m_1 + x)) \\ &= a \ . \end{aligned}$$

Assume now that (5.12) holds for index $t \ge 1$. Applying (5.11) and the induction hypothesis gives

$$m_{t+1} + m_{t+1}^* + \delta_{t+1} x = a \sum_{j=1}^{t} \delta_j (1+\theta)^{t-j} = h_t + h_t^* \ . \tag{5.13}$$

We also know that:

$$h_{t+1} + h_{t+1}^* = f(h_t + m_{t+1}) + f(h_t^* + m_{t+1}^*) + h_t + h_t^* + \theta(m_{t+1} + m_{t+1}^*) \tag{5.14}$$

Hence, if $\delta_{t+1} = 0$ we get from (5.13) that $m_{t+1} + m_{t+1}^* = h_t + h_t^*$ and by using (5.12), (5.14) becomes:

$$\begin{aligned} h_{t+1} + h_{t+1}^* &= h_t + h_t^* + \theta(m_{t+1} + m_{t+1}^*) \\ &= h_t + h_t^* + \theta(h_t + h_t^*) \\ &= (1+\theta)(h_t + h_t^*) \\ &= (1+\theta) a \sum_{j=1}^{t} \delta_j (1+\theta)^{t-j} \\ &= a \sum_{j=1}^{t+1} \delta_j (1+\theta)^{t+1-j} \end{aligned}$$

If $\delta_{t+1} = 1$, then we know from (5.13) that $m_{t+1} + m^*_{t+1} + x = h_t + h^*_t$ and hence by using (5.10) and (5.12), (5.14) becomes:

$$\begin{aligned} h_{t+1} + h^*_{t+1} &= (a + \theta x) + h_t + h^*_t + \theta(m_{t+1} + m^*_{t+1}) \\ &= (a + \theta x) + h_t + h^*_t + \theta(h_t + h^*_t + x) \\ &= (a + \theta x) + a \sum_{j=1}^{t} \delta_j (1+\theta)^{t-j} + \theta \left(a \sum_{j=1}^{t} \delta_j (1+\theta)^{t-j} + x \right) \\ &= a \left(1 + (1+\theta) \sum_{j=1}^{t} \delta_j (1+\theta)^{t-j} \right) \\ &= a \sum_{j=1}^{t+1} \delta_j (1+\theta)^{t+1-j} \end{aligned}$$

□

Theorem 5.1 can be used to compute the value $h_n + h^*_n$ corresponding to a given base message m, difference x, and n-bit vector δ. Conversely, when given a value for m, x, and $h_n + h^*_n$ the corresponding value for δ can be computed by solving a set of n linear equations over $GF(2)$ in the unknowns $\delta_1, \ldots, \delta_n$. If the polynomials $(1+\theta)^i \bmod 2$, where $0 \leq i < n$, are linearly independent, then there is always exactly one solution. This follows from the fact that we have an inhomogeneous system of equations with full rank. Once δ has been computed, the corresponding offset message m^* can be constructed using (5.11). The polynomials $(1+\theta)^i \bmod 2$ are independent if the element $(1+\theta)$ is not in a proper subfield of $GF(2^n)$. If n is a power of 2, then the number of such elements is $2^n - 2^{n/2}$. Hence a randomly selected θ will give linear independent polynomials with overwhelming probability. This is the only condition that is required for the attack to work deterministically on a hash function design according to the SMASH design strategy.

Controlling the Output Difference of SMASH-256 and SMASH-512

In the case of SMASH-256 and SMASH-512 the element θ is defined as a root of the irreducible polynomials $q(\theta)$, and from the computed orders we know that the element $(1+\theta)$ is not in a proper subfield. For these cases the n-bit vector δ can be computed with Algorithm 5.1.

For the collision attack presented in Section 5.1.3, the collisions are constructed by choosing an arbitrary base message and selecting a proper offset message. Note that for n-block messages, $h_n + h^*_n = 0$ always leads to an unacceptable solution $\delta = 0$. This follows from the fact that a homogeneous system of equations with full rank only has the trivial solution. Therefore, $(n+1)$-block messages (without padding) have to be used to construct collisions.

Proposition 5.1. *Algorithm 5.1 outputs the right δ and terminates.*

Algorithm 5.1 Compute δ for SMASH-256 and SMASH-512

Input: First preimage $h_n + h_n^*$
Output: $\delta_i \in \{0, 1\}$
Compute base message m as described in (5.9)
Compute a as described in (5.10)
Let $p(\theta)$ be the polynomial representation of $(h_n + h_n^*)a^{-1}$
$i \Leftarrow n$
Initialize δ with 0
repeat
Perform the polynomial division $p_i(\theta) = p_{i-1}(\theta)(1 + \theta) + r_i$ to determine the quotient $p_{i-1}(\theta)$ and the remainder r_i.
$\delta_i \Leftarrow r_i$
$i \Leftarrow i - 1$
until $i = 0$ or $p_i(\theta) = 0$

Proof. We can represent $(h_n + h_n^*)a^{-1}$ as polynomial $p_n(\theta) = c_0 + c_1\theta + \cdots + c_{n-1}\theta^{n-1}$. It follows from Theorem 5.1 that

$$(h_n + h_n^*)a^{-1} = \sum_{i=1}^{n} \delta_i (1 + \theta)^{n-i} \ . \tag{5.15}$$

Division with remainder of the left hand side in (5.15) leads to

$$p_n(\theta) = p_{n-1}(\theta)(1 + \theta) + r_n \ .$$

The special form of the right hand side in (5.15) implies that $\delta_n = r_n \in \{0, 1\}$ since $deg(r_n) < 1$. The same process is repeated for $p_{n-1}(\theta) = p_{n-2}(\theta)(1 + \theta) + r_{n-1}$ and so forth. In every step we have $deg(p_{i-1}(\theta)) < deg(p_i(\theta))$. Thus, the algorithm terminates after a maximum of n steps ($deg(p_n(\theta) < n)$) and produces in every step the desired coefficients δ_i for $i = 1, \ldots, n$. Every step needs $\mathcal{O}(n)$ additions modulo 2 and hence the running time of Algorithm 5.1 is $\mathcal{O}(n^2)$. □

Constructing Second Preimages for SMASH

To show how to construct second preimage for SMASH we consider first a reduced variant of SMASH. This variant is defined by omitting the final output transformation (5.3) and by assuming that the input message consists of a multiple of n blocks (*i.e.* without considering padding). As shown in the previous sections, we can control the output difference of this SMASH variant. From this property, we can derive the following theorem.

Theorem 5.2. *For the reduced variant of SMASH, we can construct preimages for any hash value h.*

Proof. We construct an n-block base message m as shown in (5.9). Then we compute the hash value h^{m} for the base message. Now derive the difference

$h' = h + h^{\mathrm{m}}$. For h' and the base message m, we can construct the n-block offset message m^* according to Theorem 5.1. Therefore, we know that the difference between the hash values of the base message and the offset message is $h' = h^{\mathrm{m}} + h^{\mathrm{m}^*}$. It follows that the offset message m^* is an n-block preimage of h since $h' = h + h^{\mathrm{m}} = h^{\mathrm{m}} + h^{\mathrm{m}^*}$ and thus $h = h^{\mathrm{m}^*}$. □

Note that this is a nice example for a preimage attack that exploits differential cryptanalysis. It follows from Theorem 5.2, that in order to construct preimages, we have to control the message blocks $m_2^*, \ldots, m_n^*$. This is illustrated in Figure 5.3.

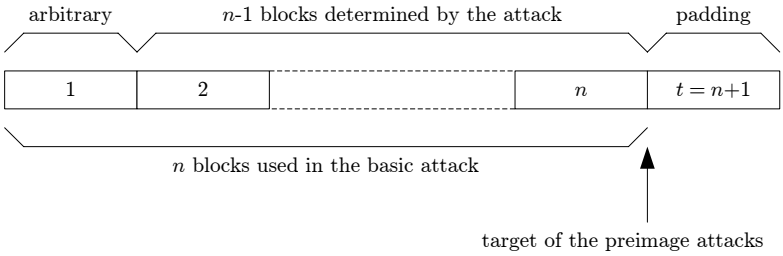

Figure 5.3: Structure of the message used in the (second) preimage attacks.

Due to the padding and the final processing in SMASH, *cf.* (5.3), we cannot control the message input for the last two applications of the compression function. Therefore, we see currently no way of constructing preimages for the SMASH design strategy. However, we can use the fact that we can control the difference in the chaining variable to construct second preimages. For the sake of readability, assume we are given a message M after padding consisting of $t = n + 1$ blocks. We refer to this message as the first preimage. Later on, we will show how to extend the approach to longer and also shorter messages. We can easily compute h_{t-1}, the chaining variable before processing the message block M_t. Now, we construct an n-block preimage for h_{t-1}. Subsequently, we concatenate M_t to produce a second preimage for SMASH. This attack produces second preimages of the same length as the given message (first preimage).

Second Preimages for Longer Messages. For messages consisting of $t \geq n + 2$ blocks, there are two possibilities to construct second preimages using the above-described attack strategy.

1. The target for the preimage is again h_{t-1}. Choose arbitrary values in the first $t - n$ blocks instead of the first block only and then do the attack by controlling the last $n - 1$ message blocks. The resulting message structure is illustrated at the top of Figure 5.4.

2. Instead of constructing a preimage for h_{t-1}, construct a preimage for h_n. Only the first message block can be chosen arbitrarily, the last $t - n$ blocks

are the same for the first message and the second preimage. The resulting message structure is illustrated at the bottom of Figure 5.4.

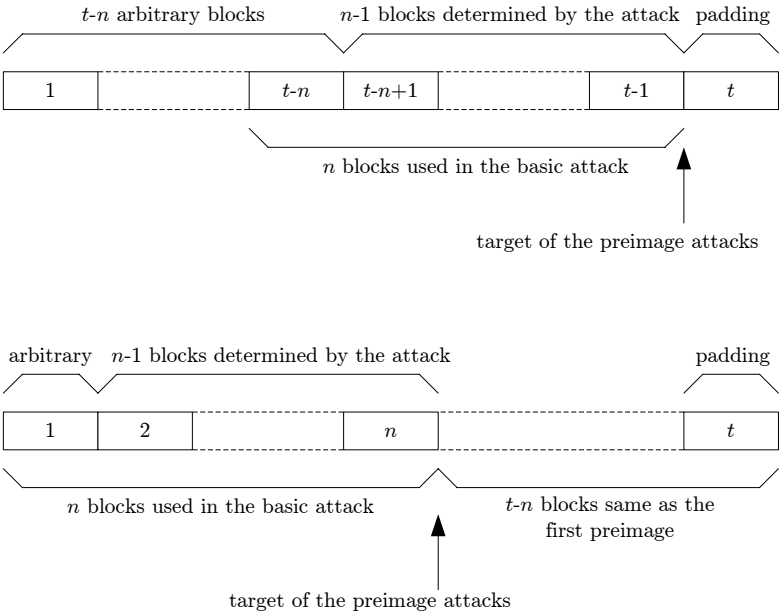

Figure 5.4: Two different structures of the messages used in the (second) preimage attacks for longer messages.

Second Preimages for Shorter Messages. The attack can also be extended to find second preimages for a first preimage M consisting of $t \leq n$ blocks. It follows from Theorem 5.1 that then we get n equations in $t-1$ unknowns:

$$h_{t-1} + h^*_{t-1} = a \sum_{j=1}^{t-1} \delta_j (1+\theta)^{t-j-1}$$

With probability 2^{t-1-n}, there exists a solution δ. If no solution exists, then the attack has to be repeated with another value for m or x. On average, the attack will succeed after 2^{n+1-t} iterations. Hence, the presented attack is faster than the general meet-in-the-middle attack [87] for messages longer than $n/2+1$ blocks.

5.2 A Second Preimage Attack on a DBLH Proposal

In this section, we present a second preimage attack on a double-block-length hash proposal presented at FSE 2006. The attack works if the hash function

proposal is instantiated with a block cipher following the FX construction. This second preimage attack is, as the attack on SMASH in the previous section, a purely structural attack, and allows to construct second preimages deterministically if the given message consist of at least 2 (3) message blocks.

5.2.1 The Proposal

Shoichi Hirose proposed a double-block-length hash function at FSE 2006 [68], which we will refer to as DBLH for the remainder of this chapter. It is an iterated, block-cipher-based hash function and uses MD strengthening. The compression function is defined as follows:

$$\begin{aligned} g_i &= F_{h_{i-1}\|m_i}(g_{i-1}) \oplus g_{i-1} \\ h_i &= F_{h_{i-1}\|m_i}(g_{i-1} \oplus c) \oplus g_{i-1} \oplus c \ , \end{aligned} \tag{5.16}$$

where c is an arbitrary constant with the only restriction that $c \neq 0$, F_k ($k = h_{i-1}\|m_i$) is an arbitrary block cipher, and $h_i\|g_i$ is the chaining value with $h_0\|g_0$ a fixed initial value (*cf.* Figure 5.5). After t message blocks have been processed,

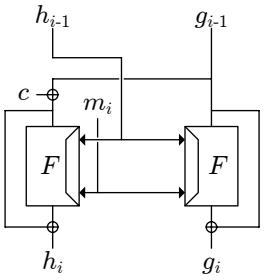

Figure 5.5: The double-block-length hash function proposal of Hirose with F as underlying block cipher.

the final hash value is the concatenation $h_t\|g_t$. As it can be seen in (5.16), the key length of the underlying block cipher F has to be greater than the block length. This is because $|k| = |h_{i-1}| + |m_i|$, where $|h_{i-1}|$ is the block length of the cipher.

In [68], Hirose proved the security with respect to collision resistance of DBLH in the ideal cipher model. Another interesting property of this proposal is that the construction does not require two full encryptions for one iteration since only one key schedule has to be computed. This is in general not the case for double-block-length hash functions as for instance MDC-2 or MDC-4. However, this performance advantage compared to other DBL hash functions is not reflected in the rate of this block-cipher-based hash function (*cf.* Section 2.3.5). If the construction is instantiated for instance with AES-192 a rate $r = 1/4$ and in the case of AES-256 a rate $r = 1/2$ is achieved.

5.2.2 DESX and the General Construction FX

The block cipher DESX [82] was proposed by Rivest to protect DES against exhaustive key search attacks. Kilian and Rogaway proved the security of the DESX construction in [81, 82] against a key-search adversary. The best know attack on DESX known to date has been presented by Biryukov and Wagner in [21].

The general form of this construction is referred to as FX [81, 82], where F can be any block cipher with block length n and key length $|k|$. The FX construction is defined as follows:

$$\mathrm{FX}_{k\|k_1\|k_2}(x) = F_k(x \oplus k_1) \oplus k_2 \ ,$$

where $|k_1| = |k_2| = n$ and the key length is $|k| + 2n$.

Kilian and Rogaway showed that FX with a key length of $|k| + 2n$ bits has an effective key length of at least $k + n - 1 - \log m$ bits, where m bounds the number of $(m, \mathrm{FX}_{k\|k_1\|k_2}(x))$ pairs a key-search adversary can obtain. The effective key length, which is obviously smaller than $|k|+2n$ bits, determines the security margin against a brute force key-search attack. It is obvious that the FX construction is not an ideal cipher. For instance, flipping a bit in k_2 can be directly observed in the ciphertext which allows to distinguish it trivially from an ideal cipher (see Section 2.2).

5.2.3 DBLH with FX

For DBLH with underlying block cipher $\mathrm{FX}_{k\|k_1\|k_2}(x)$, we can construct the following three configurations (see Figure 5.6), where $m_i = l_i\|r_i$.

Configuration I:
$$k\|k_1\|k_2 \quad = \quad l_i\|h_{i-1}\|r_i, \text{ where } \quad |l_i| = |k|, |h_{i-1}| = |r_i| = n$$

Configuration II:
$$k\|k_1\|k_2 \quad = \quad h_{i-1}\|l_i\|r_i, \text{ where } \quad |h_{i-1}| = |k|, |l_i| = |r_i| = n \qquad (5.17)$$

Configuration III:
$$k\|k_1\|k_2 \quad = \quad l_i\|r_i\|h_{i-1}, \text{ where } \quad |l_i| = |k|, |r_i| = |h_{i-1}| = n$$

For each configuration, we can interchange l_i and r_i. However, without loss of generality, we take the configurations defined in (5.17) for the further analysis. Note that if F is a block cipher with $|k| < n$ then, for Configuration II, the chaining variable h_{i-1} needs to be truncated to match the key length $|k|$. Which bits are truncated does not have any impact on the analysis. For the remainder of this section, we assume that F is a block cipher with $|k| = n$.

For the sake of simplicity, we will write DX to denote the instantiation of DBLH with FX as underlying block cipher. If we speak of a specific configuration, we append the number of the configuration. For instance, for DBLH with FX in Configuration II, we write DX-II.

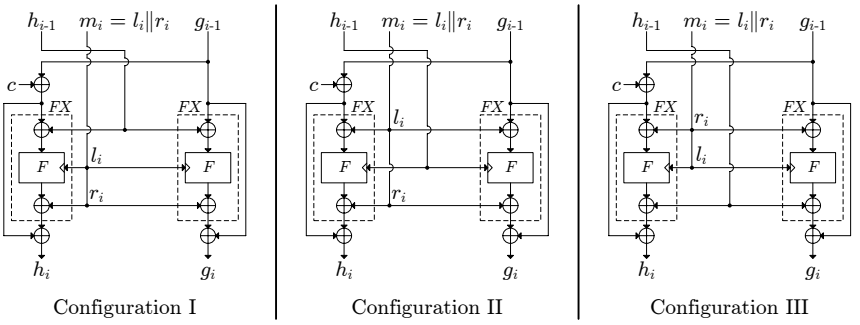

Figure 5.6: Three possible configurations of DBLH with FX as underlying block cipher. The hatch denotes the key input of the block cipher F.

5.2.4 The Second Preimage Attack

For DX, we can find second preimages for all the configurations described in Section 5.2.3. The following three theorems describe how the second preimages can be constructed. The proofs of the theorems are based on techniques from differential cryptanalysis.

Theorem 5.3. *For the iterated hash function DX-I, we can construct second preimages, since for every two-block message $m = m_1 \| m_2$ the following message m^* results in the same hash value:*

$$m^* = (m_1 \oplus (0 \| u')) \| (m_2 \oplus (0 \| u')) \ , \tag{5.18}$$

where $m_i = l_i \| r_i$, $|l_i| = |k|$, $|r_i| = n$, u' any value with $|u'| = n$, and 0 is the $|k|$-bit all-zero binary string.

Proof. Assume, we have the following 2-block messages m, m^*, where:

$$\begin{aligned} m &= m_1 \| m_2 = (l_1 \| r_1) \| (l_2 \| r_2) \\ m^* &= m_1^* \| m_2^* = (m_1 \oplus (0 \| u')) \| (m_2 \oplus (0 \| u')) = (l_1^* \| r_1^*) \| (l_2^* \| r_2^*) \\ l_1^* &= l_1 \oplus 0 = l_1, \quad r_1^* = r_1 \oplus u' \\ l_2^* &= l_2 \oplus 0 = l_2, \quad r_2^* = r_2 \oplus u' \end{aligned}$$

After one iteration, we have

$$\begin{aligned} g_1 &= g_0 \oplus F_{l_1}(g_0 \oplus h_0) \oplus r_1 \\ g_1^* &= g_0 \oplus F_{l_1}(g_0 \oplus h_0) \oplus r_1 \oplus u' = g_1 \oplus u' \ , \text{ and} \\ h_1 &= g_0 \oplus c \oplus F_{l_1}(g_0 \oplus c \oplus h_0) \oplus r_1 \\ h_1^* &= g_0 \oplus c \oplus F_{l_1}(g_0 \oplus c \oplus h_0) \oplus r_1 \oplus u' = h_1 \oplus u' \ . \end{aligned}$$

The outputs after two iterations are

$$\begin{aligned} g_2 &= g_1 \oplus F_{l_2}(g_1 \oplus h_1) \oplus r_2 \\ g_2^* &= g_1 \oplus u' \oplus F_{l_2}(g_1 \oplus u' \oplus h_1 \oplus u') \oplus r_2 \oplus u' \\ &= g_1 \oplus F_{l_2}(g_1 \oplus h_1) \oplus r_2 = g_2 \text{ , and} \\ h_2 &= g_1 \oplus c \oplus F_{l_2}(g_1 \oplus c \oplus h_1) \oplus r_2 \\ h_2^* &= g_1 \oplus u' \oplus c \oplus F_{l_2}(g_1 \oplus u' \oplus c \oplus h_1 \oplus u') \oplus r_2 \oplus u' \\ &= g_1 \oplus c \oplus F_{l_2}(g_1 \oplus c \oplus h_1) \oplus r_2 = h_2 \text{ .} \end{aligned}$$

Hence, $g_2' = g_2 \oplus g_2^* = 0$ and $h_2' = h_2 \oplus h_2^* = 0$. Therefore, the 2-block message m^* defined in (5.18) is a second preimage for the 2-block first preimage (given message) m. □

Theorem 5.4. *For the iterated hash function DX-II, we can construct second preimages, since for every 3-block message $m = m_1 \| m_2 \| m_3$ the following message m^* results in the same hash value:*

$$m^* = (m_1 \oplus (0 \| u')) \| (m_2 \oplus (v' \| w')) \| (m_3 \oplus (z' \| z')) \text{ ,} \tag{5.19}$$

where $m_i = l_i \| r_i$, $|l_i| = |r_i| = n$, u', v' any value with $|u'| = |v'| = n$, and 0 is the n-bit all-zero binary string. Let t' be the output difference of the left F instance in iteration 2:

$$t' = [F_{h_1}(g_1 \oplus c \oplus l_2)] \oplus [F_{h_1 \oplus u'}(g_1 \oplus u' \oplus c \oplus l_2 \oplus v')] \tag{5.20}$$

Then, $w' = u' \oplus t'$ and the difference z' in (5.19) *is defined as*

$$\begin{aligned} z' = [F_{h_1}(g_1 \oplus l_2) \oplus r_2 \oplus g_1] \oplus \\ [F_{h_1 \oplus u'}(g_1 \oplus u' \oplus l_2 \oplus v') \oplus r_2 \oplus w' \oplus g_1 \oplus u'] \text{ .} \end{aligned} \tag{5.21}$$

Proof. We show that for the 3-block messages m and m^*, where

$$\begin{aligned} m &= m_1 \| m_2 \| m_3 = (l_1 \| r_1) \| (l_2 \| r_2) \| (l_3 \| r_3) \\ m^* &= (m_1 \oplus (0 \| u')) \| (m_2 \oplus (v' \| w')) \| (m_3 \oplus (z' \| z')) = (l_1^* \| r_1^*) \| (l_2^* \| r_2^*) \| (l_3^* \| r_3^*) \\ l_1^* &= l_1 \oplus 0, \quad r_1^* = r_1 \oplus u' \\ l_2^* &= l_2 \oplus v', \quad r_2^* = r_2 \oplus w' \\ l_3^* &= l_3 \oplus z', \quad r_3^* = r_3 \oplus z' \end{aligned}$$

the output difference equals zero after three iterations. After one iteration, we have

$$\begin{aligned} g_1 &= g_0 \oplus F_{h_0}(g_0 \oplus l_1) \oplus r_1 \\ g_1^* &= g_0 \oplus F_{h_0}(g_0 \oplus l_1) \oplus r_1 \oplus u' = g_1 \oplus u' \\ h_1 &= g_0 \oplus c \oplus F_{h_0}(g_0 \oplus c \oplus l_1) \oplus r_1 \\ h_1^* &= g_0 \oplus c \oplus F_{h_0}(g_0 \oplus c \oplus l_1) \oplus r_1 \oplus u' = h_1 \oplus u' \text{ .} \end{aligned}$$

After two iterations, chaining variable h_2 is computed as follows

$$\begin{aligned} h_2 &= g_1 \oplus c \oplus F_{h_1}(g_1 \oplus c \oplus l_2) \oplus r_2 \\ h_2^* &= g_1 \oplus u' \oplus c \oplus F_{h_1 \oplus u'}(g_1 \oplus u' \oplus c \oplus l_2 \oplus v') \oplus r_2 \oplus w' \ . \end{aligned}$$

With $w' = u' \oplus t'$ and t' as defined in (5.20), we get

$$\begin{aligned} h_2^* &= g_1 \oplus u' \oplus c \oplus F_{h_1 \oplus u'}(g_1 \oplus u' \oplus c \oplus l_2 \oplus v') \oplus r_2 \oplus u' \\ &\oplus \underbrace{F_{h_1}(g_1 \oplus c \oplus l_2) \oplus F_{h_1 \oplus u'}(g_1 \oplus u' \oplus c \oplus l_2 \oplus v')}_{t'} \\ &= g_1 \oplus u' \oplus c \oplus r_2 \oplus u' \oplus F_{h_1}(g_1 \oplus c \oplus l_2) \\ &= h_2 \ . \end{aligned}$$

The difference in chaining variable g_2 after two iterations is

$$g_2^* = g_2 \oplus z' \ ,$$

where z' is defined in (5.21). After three iterations, we get

$$\begin{aligned} g_3 &= g_2 \oplus F_{h_2}(g_2 \oplus l_3) \oplus r_3 \\ g_3^* &= g_2 \oplus z' \oplus F_{h_2}(g_2 \oplus z' \oplus l_3 \oplus z') \oplus r_3 \oplus z' \\ &= g_2 \oplus F_{h_2}(g_2 \oplus l_3) \oplus r_3 \\ &= g_3 \\ h_3 &= g_2 \oplus c \oplus F_{h_2}(g_2 \oplus c \oplus l_3) \oplus r_3 \\ h_3^* &= g_2 \oplus z' \oplus c \oplus F_{h_2}(g_2 \oplus z' \oplus c \oplus l_3 \oplus z') \oplus r_3 \oplus z' \\ &= g_2 \oplus c \oplus F_{h_2}(g_2 \oplus c \oplus l_3) \oplus r_3 \\ &= h_3 \ . \end{aligned}$$

Therefore, after three iterations the differences in the chaining variables are $g'_3 = g_3 \oplus g_3^* = 0$ and $h'_3 = h_3 \oplus h_3^* = 0$. Therefore, the 3-block message m^* defined in (5.19) is a second preimage for the 3-block first preimage m. □

Theorem 5.5. *For the iterated hash function DX-III, we can construct second preimages, since for every 3-block message $m = m_1 \| m_2 \| m_3$ the following message m^* results in the same hash value:*

$$m^* = (m_1 \oplus (u' \| v')) \| (m_2 \oplus (0 \| z')) \| (m_3 \oplus (0 \| (w' \oplus z'))) \ , \tag{5.22}$$

where $m_i = l_i \| r_i$, $|l_i| = |k|$, $|r_i| = n$, u', v' any value with $|u'| = |k|$ and $|v'| = n$, and 0 is the $|k|$-bit all-zero binary string. Once the values u', v' have been chosen for the given input message block m_1, the differences w' and z' can be computed:

$$\begin{aligned} w' &= [g_0 \oplus c \oplus F_{l_1}(g_0 \oplus c \oplus r_1) \oplus h_0] \\ &\oplus [g_0 \oplus c \oplus F_{l_1 \oplus v'}(g_0 \oplus c \oplus r_1 \oplus u') \oplus h_0] \ , \\ z' &= [g_0 \oplus F_{l_1}(g_0 \oplus r_1) \oplus h_0] \\ &\oplus [g_0 \oplus F_{l_1 \oplus v'}(g_0 \oplus r_1 \oplus u') \oplus h_0] \end{aligned}$$

Proof. As for the proof of Theorem 5.3 and Theorem 5.4, we show that for the 3-block messages m and m^*, where $m = m_1\|m_2\|m_3 = (l_1\|r_1)\|(l_2\|r_2)\|(l_3\|r_3)$ and

$$\begin{aligned}
m^* &= (m_1 \oplus (u'\|v'))\|(m_2 \oplus (0\|z'))\|(m_3 \oplus (0\|(w' \oplus z'))) = (l_1^*\|r_1^*)\|(l_2^*\|r_2^*)\|(l_3^*\|r_3^*)\\
l_1^* &= l_1 \oplus u', \quad r_1^* = r_1 \oplus v'\\
l_2^* &= l_2 \oplus 0, \quad r_2^* = r_2 \oplus z'\\
l_3^* &= l_3 \oplus 0, \quad r_3^* = r_3 \oplus (w' \oplus z')
\end{aligned}$$

the output difference equals zero after three iterations, *i.e.* $g_3' = h_3' = 0$. After the first iteration, we have

$$\begin{aligned}
g_1 &= g_0 \oplus F_{l_1}(g_0 \oplus r_1) \oplus h_0\\
g_1^* &= g_1 \oplus z', \text{where}\\
z' &= [g_0 \oplus F_{l_1}(g_0 \oplus r_1) \oplus h_0]\\
&\oplus [g_0 \oplus F_{l_1 \oplus v'}(g_0 \oplus r_1 \oplus u') \oplus h_0] \text{ , and}\\
h_1 &= g_0 \oplus c \oplus F_{l_1}(g_0 \oplus c \oplus r_1) \oplus h_0\\
h_1^* &= h_1 \oplus w', \text{where}\\
w' &= [g_0 \oplus c \oplus F_{l_1}(g_0 \oplus c \oplus r_1) \oplus h_0]\\
&\oplus [g_0 \oplus c \oplus F_{l_1 \oplus v'}(g_0 \oplus c \oplus r_1 \oplus u') \oplus h_0] \text{ .}
\end{aligned}$$

The difference of the chaining variables after two iterations is

$$\begin{aligned}
g_2 &= g_1 \oplus F_{l_2}(g_1 \oplus r_2) \oplus h_1\\
g_2^* &= g_1 \oplus z' \oplus F_{l_2}(g_1 \oplus z' \oplus r_2 \oplus z') \oplus h_1 \oplus w'\\
&= g_2 \oplus (w' \oplus z') \text{ , and}\\
h_2 &= g_1 \oplus c \oplus F_{l_2}(g_1 \oplus c \oplus r_2) \oplus h_1\\
h_2^* &= g_1 \oplus z' \oplus c \oplus F_{l_2}(g_1 \oplus z' \oplus c \oplus r_2 \oplus z') \oplus h_1 \oplus w'\\
&= h_2 \oplus (w' \oplus z') \text{ .}
\end{aligned}$$

The output difference after three iterations is computed as follows. For the sake of clearness, we write $y' = w' \oplus z'$:

$$\begin{aligned}
g_3 &= g_2 \oplus F_{l_3}(g_2 \oplus r_3) \oplus h_2\\
g_3^* &= g_2 \oplus y' \oplus F_{l_3}(g_3 \oplus y' \oplus r_3 \oplus y') \oplus h_2 \oplus y'\\
&= g_3 \text{ , and}\\
h_3 &= g_2 \oplus c \oplus F_{l_3}(g_2 \oplus c \oplus r_3) \oplus h_2\\
h_3^* &= g_2 \oplus y' \oplus c \oplus F_{l_3}(g_2 \oplus y' \oplus c \oplus r_3 \oplus y') \oplus h_2 \oplus y'\\
&= h_3
\end{aligned}$$

Hence, $g_3' = g_3 \oplus g_3^* = 0$ and $h_3' = h_3 \oplus h_3^* = 0$. Therefore, the 3-block message m^* defined in (5.22) is a second preimage for the 3-block first preimage m. □

5.3 Summary

In this chapter, we have presented second preimage attacks for two recent hash function proposals: the single-block-length hash function design strategy SMASH and the double-block-length hash function proposal DBLH. Both attacks are pure structural attacks in the sense that they are independent of the underlying nonlinear function: for SMASH the n-bit function f and for DBLH the underlying block cipher. It is important to stress, that the second preimage attack on DBLH when instantiated with a block cipher following the FX construction does not disprove the security proof of Hirose. Therefore the hash function proposal can not be considered to be broken. We will exploit the results of this chapter for the discussion on the implications of second preimage attacks on hash-based message authentication codes in Chapter 6.

6

Implications of Collision and Second Preimage Attacks on Hash-Based MACs

In this chapter, we will discuss the implications of recent collision and second preimage attacks on hash-based message authentication codes (MACs) such as NMAC and HMAC. Firstly, we will review the implications on MACs due to recent collision attacks. Secondly, building upon the results of Chapter 5, we also analyze the implications on NMAC and HMAC if one can construct second preimages for the underlying hash function. This chapter includes work presented in [136].

6.1 Message Authentication Codes and Attacks

Message authentication codes (see Section 2.4 for an introduction) such as the CBC-MAC or NMAC/HMAC are subject to the following attacks: forgery attacks and key-recovery attacks. Regarding forgery, we require from a MAC that for an adversary it should be infeasible to find for an arbitrary message (existential forgery) or a message of his choice (selective forgery) the corresponding tag if he has no knowledge of the secret key k. The most devastating attack on a MAC is key recovery. For an *ideal* MAC, key recovery should be as expensive as exhaustive key search, *i.e.* for a $|k|$-bit key it should require about $2^{|k|}$ MAC queries. The number of required message-tag pairs for verifying the correct key is $\lceil |k|/n \rceil$, where n is the length of the tag in bits. It is clear that once an adversary knows the correct key k then he can also perform a (selective) forgery. We speak of a known-message attack if the adversary eavesdrops q message-tag pairs communicated between two legitimate parties: $(m^1, a^1), \ldots, (m^q, a^q)$, where a^i is the corresponding tag. An attack is called a chosen-message attack if the adversary can choose the messages $m^1, \ldots, m^q$ and gets the corresponding tags. Note that a chosen-message attack is considered to be less realistic than a known-message attack. For instance, depending on the practical application, an attacker may not be able to have access to the system and hence cannot ask for the tag of a message of his choice.

The most popular hash-based message authentication codes are NMAC and

HMAC, where the construction used most often in practice is HMAC. This is also reflected by the standardization of HMAC in for instance IETF RFC 2104 [92], ANSI X9.71 [72], and the generalization of these standards in NIST FIPS [129]. For the remainder of this chapter, we will only focus on NMAC and HMAC. They are defined as follows (see also Figure 6.1):

$$\mathrm{NMAC}_{k_1,k_2}(m) = H(k_1, pad(H(k_2, m))) \quad (6.1)$$

$$\mathrm{HMAC}_k(m) = H(iv, (k \oplus opad) \| H(iv, (k \oplus ipad) \| m)) \ , \quad (6.2)$$

where $H(iv, m)$ denotes the application of the iterated hash function H, *e.g.* SHA-1, with initial value iv and a t-block message $m = m_1 \| \ldots \| m_t$ after padding. If a hash function with a block size bigger than the chaining value is used, then an appropriate padding is required for NMAC such that the hash value $H(k_2, m)$ fits the block size of the hash function. This is denoted by $pad(\cdot)$ in (6.1) and in Figure 6.1. For NMAC the initial value iv is replaced by the secret key k_1, respectively k_2, resulting in a keyed hash function. For HMAC the same initial value is used as specified for the underlying hash function and the message input is keyed. Two appropriate padding methods are defined (the one specified for the underlying hash function) to have a key k that is of the same length as the block size of the hash function. Furthermore, two constants *ipad* and *opad* for the secret key k are defined, which is also referred to as the *key derivation function* of HMAC. For NMAC, we call the key k_2 the *inner key* and the according H_{in} the *inner hash*. The key k_1 is called the *outer key* and H_{out} the *outer hash*, respectively. We use the same naming for HMAC with the only difference that $(k \oplus ipad)$ and $(k \oplus opad)$ is called the inner key, respectively the outer key. For the remaining discussion, we assume that we have an n-bit chaining value and an n-bit tag, *i.e.* no output transformation such as truncation is applied. Furthermore, we assume that we have messages that are a multiple of the block length.

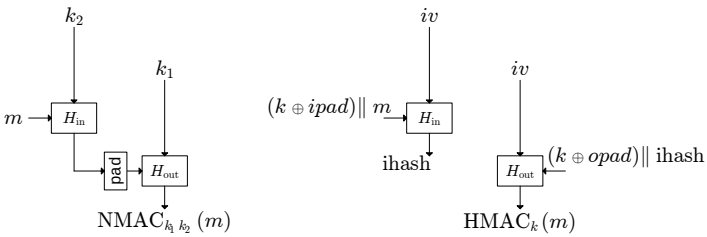

Figure 6.1: The NMAC (left) and HMAC (right) construction based on an iterated hash function H.

Since NMAC and HMAC are iterated MACs the results of Preneel and van Oorschot in [146] apply. They presented generic attacks which are based on the birthday paradox. As a result, a forgery attack is always possible in about $2^{n/2}$ operations. Therefore, in order to speak of a (theoretical) attack on NMAC and HMAC (or any other iterated MAC construction), we require that the complexity for forgery attacks is below $2^{n/2}$ and for key-recovery attacks it is below $2^{|k|}$.

Bellare *et al.* proved the security of NMAC and HMAC in [6]. They show that NMAC is a pseudo-random function PRF (for the definition of a PRF we refer to Goldreich *et al.* [63]). In their security model, the notion of a PRF is stronger than the notion of a secure MAC. Therefore, any PRF is a secure MAC as it has been shown by Bellare *et al.* in [8]. We do not further investigate their security model but we will look at the required properties of the underlying hash function. To prove that NMAC is a PRF, the underlying hash function needs to fulfill the following two properties:

- *P1)* the keyed compression function is a pseudo-random function
- *P2)* the keyed hash function is *weakly collision resistant*

Regarding the definition of the second property P2, we refer to [6], where this notion has been introduced. Roughly speaking, the second property is weaker than the assumption that the keyed hash function has to be collision resistant. Based on the security proof of NMAC they also proved the security of HMAC by introducing an additional assumption:

- *P3)* the key derivation function of HMAC is a pseudo-random function

Due to the recent results in the cryptanalysis of MD5 and SHA-1, the assumption *P2* is no longer valid for these hash functions since collisions can be constructed for any initial value [172, 173]. Note that these recent attacks do not contradict the proofs of Bellare *et al.* They just show that the most commonly used hash functions in practice do not fulfill the requirements anymore. Therefore, Bellare presented new proofs for NMAC and HMAC in [5] where the assumption *P2* is not required anymore. This leads to proofs that have less strong requirements on the underlying hash function than the original proofs. The biggest advantage of these new proofs is that the two most widely used hash functions used in practice for NMAC and HMAC, namely MD5 and SHA-1, still provide the required strength. More precisely, based on the cryptanalytic results to date the keyed compression function for both hash functions is a PRF and hence assumption *P1* still holds.

6.2 Implications of Collision Attacks

Recently, some researchers investigated the impact of collision attacks on hash-based message authentication codes instantiated with MD4, MD5, HAVAL, SHA-0, and SHA-1 as underlying hash function: Kim *et al.* [83], Contini and Yin [32, 33], Rechberger and Rijmen [149], and Fouque *et al.* in [59]. The results of their analyses is that the collision attacks do impact the security of NMAC and HMAC. However, the attack complexities are far beyond being practical.

For the following discussion, we assume that the attacker is able to perform a chosen-message attack. The adversary is given an NMAC/HMAC oracle he can query with messages of his choice and receives the according tags, which is the standard model for analyzing the security of MACs. For the remainder of this chapter, we consider the following attack scenarios:

- **distinguishing attack:** distinguish the MAC from a pseudo-random function
- **forgery attack:** for a message m, find a second message $m^* \neq m$ that results in the same tag
- **key-recovery attack:**
  - NMAC: find the keys k_1 and k_2
  - HMAC: find the key k. Note that an equivalent information for the key k are the values $H(iv, (k \oplus ipad))$ and $H(iv, (k \oplus opad))$ since with these values an adversary can compute the tag for any message

Note that distinguishing attacks on the hash-based MACs are interesting for two reasons. Firstly, the security proof of NMAC and HMAC shows that NMAC/HMAC is a pseudo-random function. This property is useful when HMAC is used as a PRF for key-derivation in for instance TLS [48] and IPsec [65]. Therefore, if we can distinguish the MAC from a pseudo-random function, then we can conclude that the underlying hash function does not fulfill the required properties and one should replace the hash function with a stronger one. Secondly, the kind of distinguishing attacks discussed in this chapter and forgery attacks are closely related to one another. We emphasize that we are only considering a special type of forgery, namely we try to find a second message that results in the same tag as a given one. In general, for a forgery the attacker can also generate a new message resulting in a new valid tag.

6.2.1 Distinguishing and Forgery Attacks

To present the basic idea of a distinguishing attack on NMAC and HMAC, we assume that we are given a collision-producing characteristic with a probability of 2^{-w}, where $w \ll n/2$. For instance, we can use the characteristic for step-reduced SHA-1 presented in [149, Table 6]. This collision-producing characteristic for 34 out of 80 steps has a probability of 2^{-31}. One of the main reasons why such a characteristic can be exploited for a distinguishing attack is that for both NMAC and HMAC, a collision in the inner hash is preserved by the application of the outer hash. Based on this property and the given characteristic, we can distinguish NMAC and HMAC as follows (see also [32, 149]):

- **on-line phase:** collect the tags for pairs of messages with the input difference that defines the characteristic
- **off-line phase:** search for a pair of messages that have the same tag

The number of required trials, *i.e.* queries to the NMAC/HMAC oracle, is related to the probability of the characteristic. We know that after randomly choosing about 2^w message pairs with the given input difference defining the characteristic, we will find with high probability one message pair that has the same tag. Therefore, after about 2^{w+1} queries to the NMAC/HMAC oracle, we

found one message pair with colliding tags. Since we assumed that $w \ll n/2$, we have successfully performed a distinguishing attack on both NMAC and HMAC.

A forgery attack in a chosen-message scenario proceeds as follows: the adversary chooses a message m and queries the oracle to obtain the tag. Now, if the attacker is able to find a second message $m^* \neq m$ that results in the same tag and the complexity for finding this message m^* is smaller than $2^{n/2}$, then he can continue as follows. Since the two messages collide in the inner hash the attacker can just append an arbitrary message e to have successfully performed a forgery: due to the collision of m and m^* both $m\|e$ and $m^*\|e$ result in the same tag. Therefore, we can conclude that a collision attack on the underlying hash function leads to a distinguishing attack and this in turn leads to an immediate forgery attack with the same complexity as the distinguishing attack. The only difference is that the adversary needs one additional chosen query to get the tag for the message $m\|e$.

So far, we assumed that we are given a collision-producing characteristic. But as has been shown for instance in [32] also a related-key attack can be used. More precisely, the related-key assumption gives the adversary additional information about the key: he still does not know the key but a specific relation between certain key bits. Under the related-key assumption, we can use for instance a characteristic resulting in a pseudo-collision in the inner hash. As we have discussed in Chapter 4, such a characteristic can have a significantly better probability than a collision-producing characteristic. Hence, the related-key assumption can lead to lower attack complexities as it has been shown in [32, 149]. Nevertheless, the attack becomes less realistic since it is reduced to the related-key setting.

6.2.2 Key-Recovery Attacks

The distinguishing attack and the forgery attack are also the basis for the key-recovery attack. This is due to the fact that the defined characteristic and the observation of identical tags resulting from a collision in the inner hash, provide the adversary with information about internal state values. We will explain this in more detail on a demonstrative example for NMAC-SHA-1 that only consists of the first six steps, *i.e.* the state update transformation uses f_{IF}. For a more detailed analysis and description of the methods for key recovery, we refer to [32, 59, 149].

For our example, we are using a single local collision as defined in Chapter 4 with the difference that we do not inject the corrections in steps 2, 3, and 4:

$$\begin{aligned} W'_0 &= +2^j \\ W'_1 &= -2^{j+5} \\ W'_5 &= -2^{j-2} \\ W'_2 &= W'_3 = W'_4 = 0 \ , \end{aligned} \tag{6.3}$$

where $0 \leq j \leq 31$. If we observe a collision after six rounds then we know with overwhelming probability (the probability that we find a random collision

is negligible compared to the probability of the 6-step collision) that the input differences to f_{IF} have been blocked in steps 2, 3, and 4 (note that a single-bit difference is blocked with probability of 1/2). This in turn provides information about the internal state as follows. We know from Section 2.5.2 that f_{IF} has the following differential properties for a single-bit difference in bit position j:

$$\begin{aligned}
&\text{step 2 : difference } B'_2 = 2^j \text{ is blocked iff } C_{2,j} \oplus D_{2,j} = 0\\
&\text{step 3 : difference } C'_3 = 2^{j-2} \text{ is blocked iff } B_{3,j-2} = 0\\
&\text{step 4 : difference } D'_4 = 2^{j-2} \text{ is blocked iff } B_{4,j-2} = 1
\end{aligned}$$

Therefore, if we observe a collision in the tags, *i.e.* we found a message pair that follows the characteristic, we gain the following information about the state values at bit position j:

$$\begin{aligned}
C_{2,j} \oplus D_{2,j} = 0 &\Leftrightarrow A_{0,j-2} \oplus B_{0,j-2} = 0\\
B_{3,j-2} = 0 &\Leftrightarrow A_{2,j-2} = 0\\
B_{4,j-2} = 1 &\Leftrightarrow A_{3,j-2} = 1\ ,
\end{aligned}$$

and hence information about the inner key k_2. Note that for the further discussion, we will exploit only the information we gain about $A_{0,j-2}$ and $B_{0,j-2}$.

If we do not observe a collision after the expected number of trials, we can conclude that the difference in step 2 has not been blocked and hence we know that

$$C_{2,j} \oplus D_{2,j} = 1 \Leftrightarrow A_{0,j-2} \oplus B_{0,j-2} = 1\ .$$

Note that we would also not observe a collision if the difference has not been blocked in steps 3 or 4. However, we can assume that the difference has not been blocked in step 2 and not in steps 3, 4, since in these steps we have a probability of 1/2 that the difference is blocked. The probability for steps 3 and 4 follows from the fact that we use randomly chosen messages as input and hence the input to f_{IF} in steps 3, 4 behaves randomly. In step 2 the differential property of f_{IF} depends on the chaining variables $C_{2,j}$ and $D_{2,j}$, namely the values $A_{0,j-2}$ and $B_{0,j-2}$ (parts of the inner key k_2).

It is easy to see that for the characteristic in (6.3), we expect a collision after six steps with a probability of 2^{-4} (without considering carry effects) resulting from a probability of 1/2 in step 0 and steps 2, 3, 4. As opposed to the probability of a local collision (*cf.* Section 4.4), the probability of the characteristic in (6.3) is independent of the bit position j. The reason for this is that, firstly, we assume that the easy conditions are fulfilled. Secondly, we only require the difference in step 2 to be blocked. Hence the sign of the difference is not of interest (zero difference in the message word W_2). Therefore, we have a probability of 1/2 in step 2.

With this information, we can now proceed as follows. For each bit position j, with $0 \leq j \leq 31$, we try about 2^4 message pairs, *i.e.* 2^5 queries. If we observe a collision, then we know with high probability that $A_{0,j-2} \oplus B_{0,j-2} = 0$ and if we do not observe a collision in the tags then we can conclude that

$A_{0,j-2} \oplus B_{0,j-2} = 1$. Therefore, with about $32 \cdot 2^5 = 2^{10}$ queries the key space has been reduced from 2^{160} to $2^{160-32} = 2^{128}$. Thus, we can guess the inner key k_2 of NMAC-SHA-1 reduced to six steps in about 2^{128} trials and we have performed successfully an inner-key-recovery attack for this demonstrative example.

Once the inner key k_2 has been recovered, we can also recover the outer key k_1, which then constitutes a full key recovery on NMAC. Recovering the outer key was for the first time presented in [149]. The idea is as follows. With the knowledge of the inner key k_2, we can generate arbitrary many messages for which we can easily compute the inner hash. Then, we use for instance a pseudo-near-collision characteristic with a zero difference in the message words. More precisely, we use the related-key assumption for recovering the outer key k_1 and we have a characteristic that has a zero input-difference but results in a specific difference in the outer hash. Note that such a difference can be observed directly in the tag of NMAC. An example of such a characteristic for step-reduced SHA-1 is given in [100]. With this characteristic also the outer key can be recovered as has been shown in [149].

As discussed in the previous section, the distinguishing and forgery attack on NMAC can be extended directly to HMAC. Regarding key-recovery attacks the attack scenario is slightly different. More precisely, we cannot gain key information as with NMAC, but we can get equivalent information, *i.e.* by getting information about the initial value for the inner hash, we get information about $H(iv, (k \oplus ipad))$. For recovering the outer key-equivalent information $H(iv, (k \oplus opad))$ one can proceed in a similar way as done for NMAC. We refer to [32, 59, 149] for a more detailed discussion on the key-recovery attacks.

6.2.3 Summary of Forgery and Key-Recovery Attacks on NMAC and HMAC with Step-Reduced SHA-1

In the following, we summarize the results on NMAC and HMAC with SHA-1 as underlying hash function that have been published to date. Kim *et al.* [83] were the first that investigated the impact of collision attacks on both NMAC and HMAC. They presented new distinguishing and forgery attacks if NMAC or HMAC are instantiated with HAVAL, MD4, MD5, SHA, and step-reduced SHA-1. Contini and Yin [32] extended upon these results for MD4, MD5, and step-reduced SHA-1 by considering several published characteristics. They discussed forgery attacks and partial key-recovery attacks on NMAC for recovering the inner key k_2. Rechberger and Rijmen [149] improved further on the work of Contini and Yin by considering easy relations between message words and also by looking at side effects such as the impact of carries on the probability of the characteristics. They presented new characteristics and new attack methods that lead to the best cryptanalytic results on NMAC and HMAC with MD5 and SHA-1 published to date. Additionally, their analysis also considers the truncation of the tag, which is a common output transformation for MACs (see Section 2.4). Fouque *et al.* [59] presented full-key-recovery attacks on NMAC

and HMAC for MD4 without related-key assumption. Their key-recovery attack on NMAC-MD5 is in the related-key setting and leads to the same result as presented in [149]. A summary of the results and attack complexities is given in Table 6.1 and Table 6.2, whereby we only consider the results for NMAC and HMAC with step-reduced SHA-1.

Table 6.1: Summary of forgery attacks on HMAC-SHA-1.

MAC	steps out of 80	data complexity	truncation #bits out of 160	work
HMAC-SHA-1	$0, \ldots, 33$	2^{52}	64	[83]
HMAC-SHA-1	$0, \ldots, 33$	2^{34}	64	[32]
HMAC-SHA-1	$0, \ldots, 33$	2^{32}	64	[149]
HMAC-SHA-1	$20, \ldots, 56$	2^{65}	96	[149]

Table 6.2: Summary of key-recovery attacks on NMAC-SHA-1 and HMAC-SHA-1 presented in [149]. Note that the column *offline complexity* comes from a new key-recovery method presented in [149].

MAC	result	steps out of 80	data compl.	offline compl.	truncation #bits out of 160
NMAC-SHA-1	FRK	$0, \ldots, 33$	2^{153}	2^{156}	64
HMAC-SHA-1	IK	$0, \ldots, 33$	2^{32}	2^{32}	160
NMAC-SHA-1	IRK	$19, \ldots, 79$	2^{100}	2^{100}	128
HMAC-SHA-1	IK	$20, \ldots, 72$	$2^{99.5}$	$2^{99.5}$	128
FRK: Full-related key recovery IRK: Inner related-key recovery IK: Inner key recovery					

6.3 Implications of Second Preimage Attacks

In this section, we will discuss the implications of second preimage attacks on NMAC and HMAC. In particular, we will focus on the attacks on SMASH and the double-block-length hash function proposal DBLH instantiated with a block cipher following the FX construction as presented in Chapter 5.

6.3.1 The Notion of b-Block Bypass

In the following, we introduce a new property of iterated hash functions and show which implications it has. For the remainder of this chapter, we assume

without loss of generality that we have message lengths that are a multiple of the block length. Furthermore, we assume that the blocks required for MD strengthening have been removed.

Definition 6.1. *(b-block bypass) Let H be an iterated hash function. We say that we can construct a* b-block bypass *for H, if for any initial value h_0 and for any b-block message $m = m_1 \| \dots \| m_b$, we can find a b-block message $m^* = m_1^* \| \dots \| m_b^* \neq m$ such that the following holds:*

$$\begin{aligned} &H(h_0, (m_1 \| \dots \| m_i)) \neq H(h_0, (m_1^* \| \dots \| m_i^*)) \quad \text{for } i = 1, \dots, b-1 \\ &H(h_0, (m_1 \| \dots \| m_b)) = H(h_0, (m_1^* \| \dots \| m_b^*)) \end{aligned} \tag{6.4}$$

Remark 6.1. It follows directly from Definition 6.1 that the notions of b-block bypass and second preimage are closely related. To be more precise, if we can construct a b-block bypass for an iterated hash function then it is possible to construct a second preimage m^* for any given message $m = m_1 \| \dots \| m_t \neq m^*$ with $t \geq b$. Furthermore, both the second preimage m^* and the message m are of equal length. Hence, a b-block bypass provides additional details such as the dependency on the chaining value.

Remark 6.2. Controlling the output difference of a hash function after b iterations as has been shown in Section 5.1.4, is similar to Definition 6.1. The differences are that for controlling the output difference, we cannot choose the message m arbitrarily but it has to be the base message such that with the corresponding offset message we can control the difference after iteration b. Also, we do not require that the chaining values of the base message and the offset message are different throughout the iterations $1, \dots, b-1$. While for a b-block bypass we can construct second preimages, controlling the output difference after b iterations allows to construct b-block preimages as shown in Theorem 5.2.

It follows from Definition 6.1 that if we can construct a b-block bypass for a hash function then we can do this for any initial value. Thus we can give the following lemma:

Lemma 6.1. *Let H be an iterated hash function for which we can construct a b-block bypass. Then, for every message $m = m_1 \| \dots \| m_t$ with $t \geq b \geq 1$, we can construct at least*

$$\sum_{j=1}^{\lfloor t/b \rfloor} \binom{t - j(b-1)}{j}$$

distinct second preimages.

Proof. Based on Remark 6.1, we know that for every message with block length greater or equal than b we can construct a second preimage. From Definition 6.1 it follows immediately that it does not matter which b consecutive blocks of the message m are taken to construct a second preimage m^*. This implies that we have at least $t - b + 1$ second preimages for the message m (see Figure 6.2).
On the other hand, if $\lfloor t/b \rfloor \geq 2$ we can apply the fact from Definition 6.1 not

only for one b-block sub-message of m but for j sub-messages, where j can be in the range of $1, \ldots, \lfloor t/b \rfloor$. An illustration of this fact is also shown in Figure 6.2. The problem of counting all these possible second preimages of m boils down to counting the number of possibilities of putting n indistinguishable balls into ℓ distinguishable urns. This number is known to be

$$\binom{n-\ell+1}{\ell-1},$$

cf. [55, page 38, Eq. (5.2)]. In our scenario, we fix a given number of b-block bypass sub-messages. Then the 'urns' correspond to the possible positions where the remaining message blocks can be inserted. That is, between two b-block bypass blocks and on the very left or right end. The remaining message blocks correspond to the 'balls'.
Thus, in the formula above we have $n = t - jb$ and $\ell = j + 1$ so for every $j \in \{1, \ldots, \lfloor t/b \rfloor\}$ we end up with

$$\binom{t-jb+(j+1)-1}{j+1-1} = \binom{t-j(b-1)}{j}$$

possibilities of choosing j sub-messages of length b in the original message m. Thus, the overall number of second preimages resulting from the described construction is

$$\sum_{j=1}^{\lfloor t/b \rfloor} \binom{t-j(b-1)}{j}.$$

□

Remark 6.3. The result of Lemma 6.1 seems intuitive. However, Lemma 6.1 does not necessarily apply to the notion of second preimage but it always holds for the notion of b-block bypass. Therefore, the notion of b-block bypass enables a better insight on the possibilities for constructing a second preimage.

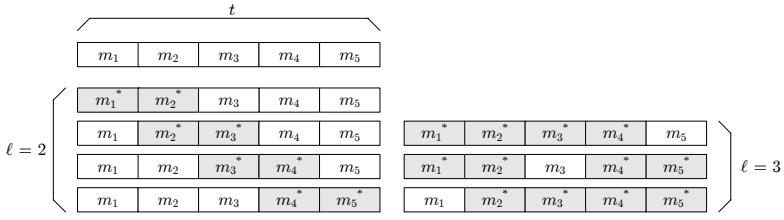

Figure 6.2: For a 2-block bypass we can construct for any 5-block message $m = m_1 \| \ldots \| m_5$ seven distinct second preimages. The shadowed rectangles show which blocks of the original message m have been modified to construct the second preimage.

In the following sections, we will show that the second preimage attacks on SMASH in Section 5.1 and DX in Section 5.2 fit the notion of b-block bypass.

6.3.2 b-Block Bypass for SMASH

As discussed in detail in Section 5.1.4, we can construct efficiently second preimages for SMASH-n. For the following discussion let us first consider SMASH-n omitting padding and omitting the final output transformation in (5.3). For this simplified variant of SMASH-n, we can derive an equation of the following form:

$$h_t = x + y \sum_{j=1}^{t} \delta_j (1+\theta)^{t-j} \ , \tag{6.5}$$

where $1 \leq t \leq n$, x and y are values depending on the used initial value and the compression function f, and the $\delta_i \in \{0,1\}$ are unknowns on which the respective blocks of the preimage $m^* = m_1^* \| \ldots \| m_n^*$ will depend. As mentioned in Section 5.1.4, (6.5) can be interpreted as an inhomogeneous system of n linear equations in t variables over $GF(2)$.
For the solvability of this system, we refer to Section 5.1.4. There we have shown that if $t = n$ in (6.5), we are guaranteed a unique solution $\delta_1, \ldots, \delta_n$ from which an n-block preimage m^* can be constructed.

Because of (5.3), this preimage attack cannot be augmented to the full variant of SMASH-n. However, we can use this result to construct an n-block bypass for an arbitrary message $m = m_1 \| m_2 \| \ldots \| m_n$. Let h_n denote the chaining value computed after n applications of (5.2) starting from our initial message m. Then, the technique described in Section 5.1.4 leads to a message m^* such that $h_n^* = h_n$ and therefore $h_{n+1}^* = h_{n+1}$. Furthermore, the method guarantees that the constructed second preimage m^* differs from m at least in the first message block. Since this can be carried out independently of the choice of the initial value h_0, we arrive at the following theorem:

Theorem 6.1. *For almost all instantiations of SMASH-n, we can construct an n-block bypass. Especially, we can construct a* 256*-block, respectively* 512*-block bypass for the hash functions SMASH-256, respectively SMASH-512.*

It follows from Lemma 6.1 that for an arbitrary message $m = m_1 \| \ldots \| m_t$ with $t \geq n$, we can find at least

$$\sum_{j=1}^{\lfloor t/n \rfloor} \binom{t-j(n-1)}{j}$$

second preimages based on the n-block bypass for SMASH-n.

6.3.3 b-Block Bypass for DBLH with FX

For the DBLH construction instantiated with FX as underlying block cipher we can also construct a b-block bypass. Depending on the configuration, we can construct a 2-block or a 3-block bypass, respectively. This is given by the following theorems.

Theorem 6.2. *For DX-I, we can construct a 2-block bypass. Furthermore, for an arbitrary message $m = m_1 \| \ldots \| m_t$ with $t \geq 2$, we can find at least*

$$\sum_{j=1}^{\lfloor t/2 \rfloor} \binom{t-j}{j}$$

second preimages based on this 2-block bypass.

Proof. The construction of a 2-block bypass for DX-I follows directly from Theorem 5.3 given in Section 5.2.4. After two iterations the differences in the chaining variables are $g'_2 = g_2 \oplus g^*_2 = 0$ and $h'_2 = h_2 \oplus h^*_2 = 0$. Since the differences in chaining variables $g'_0 = h'_0 = 0$, we can construct a 2-block bypass following Definition 6.1. The final statement of the theorem is an immediate consequence of Lemma 6.1 with $b = 2$. □

Note that for DX-I, we have an even stronger property than a b-block bypass given in Definition 6.1. From Theorem 5.3 it follows that we can construct for any 2-block message m a 2-block message m^* such that for any initial value h_0 (6.4) holds. This means that we are independent of the used initial value h_0 for constructing a second preimage consisting of 2 blocks. In other words, the 2-block second preimage m^* as shown in Theorem 5.3, remains always the same, no matter which h_0 is used.

Theorem 6.3. *For both DX-II and DX-III, we can construct a 3-block bypass. Furthermore, for an arbitrary message $m = m_1 \| \ldots \| m_t$ with $t \geq 3$, we can find at least*

$$\sum_{j=1}^{\lfloor t/3 \rfloor} \binom{t-2j}{j}$$

second preimages based on this 3-block bypass.

Proof. The construction of a 3-block bypass for both DX-II and DX-III follows directly from Theorem 5.4 and Theorem 5.5 given in Section 5.2.4. For both configurations the differences in the chaining variables after three iterations are $g'_3 = g_3 \oplus g^*_3 = 0$ and $h'_3 = h_3 \oplus h^*_3 = 0$. Since the differences in chaining variables $g'_0 = h'_0 = 0$, we can construct a 3-block bypass following Definition 6.1. The final statement of the theorem is an immediate consequence of Lemma 6.1 with $b = 3$. □

6.3.4 Implications on NMAC and HMAC

Motivated by the results on the implications of collision attacks on the security of NMAC and HMAC, we now consider the implications of the second preimage attacks. In particular, we will look at NMAC and HMAC using SMASH or one of the DX constructions as underlying hash function. Note that in practice one would not decide for NMAC and HMAC using the DX hash construction but would rather use the underlying block cipher for deriving a block-cipher-based

MAC (see for instance Section 2.4). However, we will use NMAC and HMAC with DX as underlying hash function to highlight the impact of second preimage attacks on hash-based MACs.

If we look at the second preimage attacks, then we see that these attacks are differential attacks. However, as opposed to the collision attacks, the second preimage attacks presented in this chapter are purely structural. Structural in the sense that for constructing the second preimage, we do not consider the nonlinear components of the hash functions. For instance, for the attack on SMASH the nonlinear bijective mapping has no impact—any other mapping can be used and the second preimage attack still succeeds. The same holds for DX in all three configurations. In the case of DX it is even more clear since we instantiate DX already with the general FX construction, meaning that we can employ any block cipher.

DX-I

For the second preimage attack based on the 2-block bypass for the DX-I construction, we can mount directly a forgery attack on both NMAC and HMAC. This follows immediately from Theorem 5.3, which states that the second preimage can be constructed without knowing the initial value. Therefore, for any t-block message $m = m_1 \| \dots \| m_t$ with $t \geq 2 = b$, we can construct a forgery as follows:

$$\begin{aligned} m_1^* &= m_1 \oplus (0 \| u') \\ m_2^* &= m_2 \oplus (0 \| u') \\ m_i^* &= m_i \quad \text{for } i = 3, \dots, t \ , \end{aligned}$$

where u' is an arbitrary value with $|u'| = n$, *i.e.* the value u' has to be of the same size as the block size of the underlying block cipher. The all-zero binary string 0 is of the same size as the key of the employed block cipher (see Theorem 5.3). Note that based on Lemma 6.1 the attacker has several possibilities for constructing the second preimage depending on the number of blocks for the given message m. As opposed to the forgery attacks discussed in Section 6.2, the forgery attack presented here can be mounted in a known-message attack scenario making the attack realistic. Furthermore, for messages with a block length ≥ 2 the attack is deterministic, meaning that if we eavesdrop only a single message-tag, we can construct a forgery.

Let us now look at the implications of this second preimage attack with respect to key recovery. From the notion of b-block bypass in Definition 6.1, we know that the second preimage can be constructed for any initial value h_0. This implies that we cannot gain any information about the used iv or equivalent state information. Therefore, a key-recovery attack based on the second preimage attack is not possible for both MAC constructions.

DX-II, DX-III, and SMASH

For DX-II and DX-III, we can construct second preimages based on a 3-block bypass. It follows immediately from Theorem 5.4 and Theorem 5.5 that for

constructing the second preimages, we require the knowledge of the initial value since we need to compute the output differences of the employed block cipher in iteration 2 and iteration 3. The same is true for SMASH-n. To construct the second preimage based on the n-block bypass, we have to solve the inhomogeneous system of equations given in (6.5). In order to solve this system we need to know the values x, y in (6.5) and these values depend on the initial value. Therefore, the second preimage attacks on these hash functions do not lead to a forgery for both NMAC and HMAC.

Regarding key-recovery attacks, we are in a similar situation as for DX-I. The second preimage attacks are not independent of the initial value (we need the iv to construct the second preimage) but the attack works for any known iv resulting in no conditions on the initial value or any other state values. Therefore, also for DX-II, DX-III, and SMASH-n a key-recovery attack by exploiting the second preimage attack is not possible.

Table 6.3 summarizes the implications of the second preimage attacks on NMAC and HMAC with the hash functions DX-I, DX-II, DX-III, and SMASH-n. As can be seen in Table 6.3, the powerful second preimage attacks on the hash functions SMASH-n and DX in all three configurations have very limited implications on NMAC and HMAC.

Table 6.3: Summary of the implications of second preimage attacks on NMAC and HMAC with DX-I, DX-II, DX-III, and SMASH-n as underlying hash function. 'no' means that an attack based on the second preimage attack on the hash function is not possible and 'yes' means that the attack can be done efficiently.

NMAC/HMAC	forgery attack	key-recovery attack
DX-I	yes	no
DX-II	no	no
DX-III	no	no
SMASH-n	no	no

6.4 Summary

In this chapter, we have discussed the implications of recent collision and second preimage attacks on the security of NMAC and HMAC. In the first part, we reviewed recent research results in this direction. The main result is that the collision attacks on the underlying hash function do impact the security of both NMAC and HMAC. Nevertheless, the complexities for these attacks are far from being practical. Furthermore, all attacks are performed in a chosen-message attack model which makes them less realistic. Also the related-key assumption reduces again the practicability. Regarding SHA-1, one has to note that so far it is only possible to find high probability characteristics for step-reduced variants

of SHA-1 such that the attack complexity is below the generic bounds. If we look at the full key-recovery attack on NMAC then it is also important to mention that for this attack the outer key can only be recovered under the related-key assumption. Nevertheless, even if the discussed attacks are only of theoretical importance, we stress that further research in this direction is necessary to get a better view on the security margins of NMAC and HMAC with for instance MD5 or SHA-1.

In the second part of this chapter, we discussed very rare second preimage attacks on hash functions and their implications on NMAC and HMAC. As a result, we have shown that only in one case (DX-I) we can perform a forgery attack. This forgery attack is practical since it is a known-message attack and a forgery can be performed very efficiently. Nevertheless, for the other hash constructions DX-II, DX-III, and SMASH-n neither forgery nor key-recovery attacks are possible.

Comparing the implications of the collision attacks and the second preimage attacks is difficult since both attacks are entirely different in their nature. However, one common fact is that the powerful attacks on the underlying hash function do have less serious consequences for the MAC constructions. Therefore, a weak hash function does not necessarily mean a weak construction such as NMAC and HMAC derived from this hash function.

7

Design of Cryptographic Hash Functions

In this chapter, we describe and discuss recent hash function proposals. We only treat dedicated and block-cipher-based hash functions and omit hash functions that are based on for instance number theoretic constructions. We start with looking at proposed modifications for SHA-1. The idea of these modifications is firstly to strengthen the security of SHA-1 with respect to existing collision attacks, and secondly, to leave SHA-1 unchanged as much as possible. Furthermore, we treat a new design paradigm for hash functions, called *sponge functions*. This design principle combines techniques from block cipher and stream cipher design. Two new designs based on this design strategy, namely RADIOGATÚN and Grindahl are discussed. We also discuss the block-cipher-based hash function Whirlpool since it is the only block-cipher-based proposal we are aware of, which does not use a double-block-length construction but rather defines a new block cipher. Finally, we propose a new double-block-length hash function AESH-256 and give a security analysis. Based on the design principles of these hash functions, we discuss possible directions in the design of new hash functions.

We emphasize that many other hash constructions have been proposed that we will not discuss in this chapter. For instance, Gauravaram *et al.* proposed hash function constructions called 3C and 3C+ in [60] as an enhancement of the MD design principle. A security analysis presenting shortcomings of these constructions has been presented by Joscák and Tuma in [73]. A dedicated hash function called FORK-256 has been proposed by Hong *et al.* in [70]. Shortly after the publication of FORK-256, weaknesses for simplified variants have been published by Matusiewicz *et al.* in [105] and by Mendel *et al.* in [109]. A collision attack on the FORK-256 hash function has been presented by Matusiewicz *et al.* in [106]. Also a new version of FORK-256 by Hong *et al.* in [71], has been broken by Saarinen in [156]. Recently, a new compression function proposal called MAME, suitable for restricted hardware requirements, has been presented by Yoshida *et al.* in [176]. As a final example of new proposals, we mention the new design strategy called 'zipper hash' that has been presented by Liskov in [99]. We refer to [3] for a more comprehensive list of hash functions and corresponding security analyses.

7.1 Recent Proposals

7.1.1 Modifications of SHA-1

Due to the widespread use of SHA-1 in applications, it might be necessary to find possible short-term modifications such that existing collision attacks are precluded. We are only aware of two proposed modifications of SHA-1, which we will describe and discuss in the following.

A New Message Expansion

After the publication of the collision attacks of Wang *et al.*, Jutla and Patthak [78] proposed an alternative to SHA-1, called SHA1-IME. The hash function SHA1-IME specifies a modified message expansion. With this new message expansion the minimum Hamming weight of the disturbance vector of SHA1-IME is significantly higher than the Hamming weight of the disturbance vector of SHA-1. The message expansion is given by

$$W_i = \begin{cases} M_i, & \text{for } \; 0 \le i \le 15 \\ W_{tmp} \oplus ((W_{i-1} \oplus W_{i-2} \oplus W_{i-15}) \lll 13) & \text{for } 16 \le i \le 35 \\ W_{tmp} \oplus ((W_{i-1} \oplus W_{i-2} \oplus W_{i-15} \oplus W_{i-20}) \lll 13) & \text{for } 36 \le i \le 79 \ , \end{cases}$$

$$\text{with } W_{tmp} = W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16} \ .$$

The state update transformation, the iv, and the step constants of SHA1-IME are the same as defined for SHA-1. As stated in [78] the performance of SHA1-IME is very close to that of SHA-1. However, existing implementations have to be changed accordingly, which makes the new proposal less attractive.

In Section 4.3.1, we described how to construct a generator matrix for the pseudo-near-collision covering the last 60 steps. We did the same for SHA1-IME and the lowest Hamming weight we find is 75 for the last 60 steps considering only state variable A, see Table 7.1.

Jutla and Patthak give a computer aided proof for this Hamming weight. For SHA-1 the minimum Hamming weight is 25 (see Section 4.3.1). Since the minimum Hamming weight of SHA1-IME is three times higher than the minimum weight of SHA-1 the complexity of the attack increases significantly. For the overall attack complexity Jutla and Patthak conjecture the complexity to be $2^{75 \cdot 2.5}$. Assuming on average a probability of $2^{-2.5}$ for a local collision is quite a conservative heuristic. From our detailed analysis of local collisions in Section 4.4 it follows that the best probability for a local collision is at most 2^{-2}. Based on the found low-weight vector, approximately 2/3 of the single disturbances are in bit positions with probabilities around 2^{-4}. Therefore, the attack complexity can be expected to be even higher than $2^{75 \cdot 2.5}$ for the last 60 steps of SHA1-IME.

Based on our short analysis of SHA1-IME, we state that SHA1-IME is more secure against the attacks on SHA-1 presented by Wang *et al.* [172] and De Cannière and Rechberger [28]. However, SHA1-IME has not been studied

Table 7.1: Minimum Hamming weight disturbance vector for last 60 steps of the linearized variant of SHA1-IME. The values are listed in hexadecimal notation. For the sake of readability, zero values are denoted by a dash.

A'_{20} = --1-4---	A'_{40} = --------	A'_{60} = ----4--2
A'_{21} = --1-----	A'_{41} = --1----2	A'_{61} = --1----2
A'_{22} = ----4---	A'_{42} = --1-4---	A'_{62} = --1-----
A'_{23} = -------2	A'_{43} = ----4---	A'_{63} = --------
A'_{24} = --1-----	A'_{44} = --1----2	A'_{64} = -------2
A'_{25} = -------2	A'_{45} = --------	A'_{65} = ----4--2
A'_{26} = ----4---	A'_{46} = ----4---	A'_{66} = ----4---
A'_{27} = --1-4---	A'_{47} = --1-----	A'_{67} = -------2
A'_{28} = --1-4---	A'_{48} = --1-4--2	A'_{68} = --1-----
A'_{29} = --1----2	A'_{49} = --1-----	A'_{69} = --1-4---
A'_{30} = --1-4---	A'_{50} = -------2	A'_{70} = ----4---
A'_{31} = ----4---	A'_{51} = --1----2	A'_{71} = --------
A'_{32} = --1-----	A'_{52} = -------2	A'_{72} = --1-----
A'_{33} = --------	A'_{53} = --------	A'_{73} = --------
A'_{34} = ----4--2	A'_{54} = --1-----	A'_{74} = -------2
A'_{35} = --------	A'_{55} = ----4---	A'_{75} = --1-----
A'_{36} = ----4--2	A'_{56} = -------2	A'_{76} = -------2
A'_{37} = -------2	A'_{57} = --1-4---	A'_{77} = -8-----2
A'_{38} = -------2	A'_{58} = -------2	A'_{78} = -----1-2
A'_{39} = ----4--2	A'_{59} = --1-----	A'_{79} = --2--1-2

carefully enough to date. It is conceivable that a slightly modified attack strategy could bypass the higher Hamming weight of the disturbance vector. Note that the increased Hamming weight of the L-characteristic can imply a more significant decrease in the complexity of the attack following from the carry effect that we discussed in detail in Section 4.4. Therefore, we conclude that a more in-depth security analysis of SHA1-IME is essential in order to get a good understanding of the security margins.

Message Preprocessing

An alternative modification of SHA-1 has been proposed by Szydlo and Yin in [161]. Instead of changing parts of SHA-1 directly, they propose message preprocessing methods. The basic idea is that by message preprocessing the attacker has less freedom and flexibility in finding high probability L-characteristics. So, basically the aim is the same as the method proposed by Jutla and Patthak with the difference that existing SHA-1 implementations can be used further on, which is obviously a nice feature of their proposal.

Szydlo and Yin propose two preprocessing methods: *message whitening* and *message interleaving* that also preserve the streaming property of SHA-1, *i.e.* incoming message blocks can be processed without requiring additional temporary storage. To explain their basic idea, we consider a t-block message $m = m_1 \| \ldots \| m_t$ as the concatenation of 512-bit blocks, where each message

block m_i is further split into 16 32-bit words x_i, *i.e.* $m = x_1 \| \ldots \| x_{16t}$. Without preprocessing, 16 words are used as the input for the compression function of SHA-1 in each iteration. Now, if message whitening is applied, a 512-bit block is constructed by concatenating less than 16 words x_i, say $16-w$, and by inserting zero-words to get a 512-bit block. For instance, if we take the first $16-w$ words, then we simply append w zero-words. For the first whitened message block we get: $m_1^* = x_1 \| \ldots \| x_{16-w} \| 0 \| \ldots \| 0$. Based on this whitening approach, the compression function processes only $16-w$ words of each original message block m_i in one iteration. It is clear that this leads to an according performance slowdown. For instance, if $w = 8$ then whitening can be seen as constructing a new message m^* that is twice as long as the original message m, and hence, the performance is reduced by a factor of two.

The second preprocessing method, message interleaving, does not whiten parts of an input message block but reuses some x_i's. For instance, one possibility is to reuse each 32-bit word x_i twice for building a 512-bit message block. Then the interleaved first input message block can for instance be constructed as follows: $m_1^* = x_1 \| x_1 \| x_2 \| x_2 \| \ldots \| x_8 \| x_8$. It is clear that also this preprocessing method leads to an according performance slowdown compared to SHA-1—for the given example the performance is reduced by a factor of two.

Szydlo and Yin give a quantitative security analysis of the proposed methods. More precisely, their analysis is based on coding theory and investigates how the preprocessing methods do affect the search for good L-characteristics as we have discussed in detail in Chapter 4.3.1. They construct different linear codes based on the linearized variant L-SHA-1 and consider the restriction caused by the preprocessing method. They have performed several experiments for finding low-weight L-characteristics. As shown in Section 4.3.1 and [78] the minimum Hamming weight of the L-characteristic for the last 60 steps of the state update transformation of L-SHA-1 is 25. Based on the performed experiments, they state that the minimum Hamming weight for the last 60 steps can be higher than 80 depending on the used preprocessing method. It is clear that this makes a collision attack on SHA-1 more complex. Nevertheless, one has to be very careful when choosing the preprocessing method, and each variant of it has to be analyzed thoroughly. For instance, if we consider message whitening then it is clear that the performance slowdown can be reduced if fewer blocks are whitened. On one side, this is clearly a goal for applications. On the other side, this can reduce the additional strength one would expect from message preprocessing. Leurent presented in [97] a modified collision search of Wang *et al.* and showed how it can be exploited for collision attacks on MD4 and MD5 with message whitening. He presented a collision attack on MD4 if 11 words of 32 bits are whitened in the 512-bit input-message block. For MD5, he showed a collision attack if only a single 32-bit zero word is used for whitening. Even if there are no results for SHA-1 and even if the quantitative security analysis suggests that high probability L-characteristics are more difficult to find or are less likely to exist, we stress that the message preprocessing methods proposed in [161] need a very careful in-depth analysis before being applied.

7.1.2 Sponge Functions and RADIOGATÚN

RADIOGATÚN is a recent proposal presented by Bertoni *et al.* in [13, 14] as an alternative to the design strategy of the MD family of hash functions. In the following, we present the motivation for a new design paradigm called *sponge functions* and give a description of the specific instance RADIOGATÚN. We stress that this new design paradigm for hash functions is still work in progress and hence, we will only give a short overview and refer to [13, 14, 15] for further details and a security analysis of the new design strategy.

Motivation

Most hash functions in use today are designed following the Merkle-Damgård design principle [41, 118] (see also Section 2.3.1). As discussed in Chapter 3, these constructions are vulnerable to different kind of attacks such as multicollision attacks and second preimage attacks for long messages. Also, it has been shown that certificational weaknesses such as near-collisions or pseudo-near-collisions can be exploited for collision attacks. Therefore, new design principles different than the MD constructions are of high interest and important for new directions in the design of cryptographic hash functions. In [13, 14] a new design principle for hash functions has been proposed and later on generalized in [15]. The basic idea is to provide a new reference model, called *sponge functions*, for the design of new hash functions in order to replace the random oracle model [9]. The reason for this is that it is infeasible to design a hash function that behaves as a random oracle with truncated output. For instance, based on the definition of a random oracle, we know that no internal collisions exist but for existing hash functions they do exist. Therefore, the aim of this new reference model is to provide a design paradigm that is more realistic and considers unideal properties such as internal collisions. The main point is to find a method to show that finding inner collisions is harder than a generic birthday attack on the hash value.

In Figure 7.1 the basic design principle of a sponge function is depicted. The basic components are the internal state, the injection layer, the round function, and the output transformation. The working principle is as follows. A simple round function is iteratively applied to the large internal state which is updated with the injected message blocks. When the entire message has been processed, some blank iterations are performed, *i.e.* the round transformation is applied to the internal state without injecting any message blocks. Then, parts of the state are iteratively output (denoted by the output transformation) resulting in a hash value of variable length.

The question is which properties are required from the single components? For the general model, the round function is defined as a family of functions. For a concrete design one has to fix this function. In this case, the round function defines a permutation but has no cryptographic strength by itself. Furthermore, it is invertible. In [13], the authors introduce the *sponge capacity*, which depends on the state size. Roughly speaking, the higher the state size the higher the

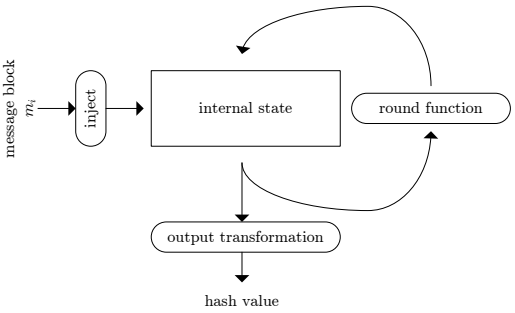

Figure 7.1: The basic components of sponge functions.

capacity and the more difficult it is to find internal collisions. Note that there is no formal way to determine the capacity. For a concrete hash function the designers can only claim it. We refer to [13, 14] for a more detailed description on this new design strategy.

The Round Function of RADIOGATÚN

The design RADIOGATÚN has been presented as a concrete instance of the new design principle relying on sponge functions. RADIOGATÚN borrows techniques from the cipher Panama proposed by Daemen and Clapp in [36] which can act as both stream cipher and hash function. For the Panama hash function collision attacks have been presented by Rijmen *et al.* in [152] and by Daemen in [35]. When designing RADIOGATÚN, these attacks have been considered.

RADIOGATÚN consists of a large state and uses a simple round function to update the state in an iterative way. In each iteration some message words are injected. After the complete message has been processed the round function is applied for several iterations without injecting any message words. Finally, parts of the state are output resulting in the final hash value. In the following, we describe the different parts of the round function in more detail. We follow the notation presented in [14]. RADIOGATÚN is designed generically, *i.e.* the word length can be between 1 and 64 bits. For our description, we focus on the 64-bit variant, which is the default word length.

We start with describing the different parts of the round function and summarize then how the hash value is computed. The state of RADIOGATÚN is composed of two parts (see also Figure 7.2): *Mill* consisting of 19 words $a[i]$ with $i = 0, \ldots, 18$, and *Belt* consisting of 13 stages $b[i]$ of 3 words each $b[i, j]$, $i = 0, \ldots, 12$ and $j = 0, \ldots, 2$. Therefore, the state of the 64-bit variant consists of $(19 + 3 \cdot 13) \cdot 64 = 3712$ bits. An input block consists of 3 words $mb[i]$, and the output block consists of 2 words from the state. Two functions are defined: the *mill function* and the *belt function*. The mill function is borrowed from the cipher Panama [36] and consists of simple Boolean operations such as XOR and AND, and bitwise rotations of the words (see [14, Algorithm 4] for further details). The only difference to the function used in Panama is that it is defined for

19 words instead of 17 words. The mill function is the only nonlinear function in RADIOGATÚN. The belt function is a simple rotation of the state $b[i, j]$, where each row is rotated by one word to the left.

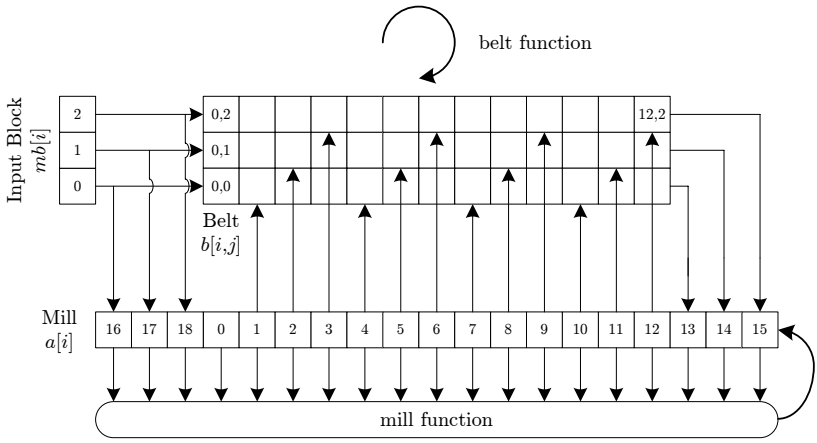

Figure 7.2: The round function of RADIOGATÚN.

Computing the hash value can be summarized as follows (see also Figure 7.2):

- **Step 1** (Initialize): initialize the state (Belt and Mill) to zero.

$$a[i] \leftarrow 0 \text{ for } 0 \leq i \leq 18$$
$$b[i, j] \leftarrow 0 \text{ for } 0 \leq i \leq 12 \text{ and } 0 \leq j \leq 2$$

- **Step 2** (Process one input block):

  1. XOR the 3 input words to Belt and Mill

$$b[0, j] \leftarrow b[0, j] \oplus mb[j] \text{ for } 0 \leq j \leq 2$$
$$a[i + 16] \leftarrow a[i + 16] \oplus mb[i] \text{ for } 0 \leq i \leq 2$$

  2. apply the belt function to Belt

$$b[i, j] \leftarrow b[i - 1 \bmod 13, j] \text{ for } 0 \leq i \leq 12 \text{ and } 0 \leq j \leq 2$$

  3. mix parts of Mill and Belt

$$b[12, j] \leftarrow b[12, j] \oplus a[13 + j] \text{ for } 0 \leq j \leq 2$$

  4. apply the mill function to Mill

$$a \leftarrow mill(a)$$

5. mix parts of Belt and Mill

$$a[i+1] \leftarrow a[i+1] \oplus b[i+1, i \bmod 3] \text{ for } 0 \leq i \leq 11$$

- **Step 3** (Process input message): repeat Step 2 until the entire input message has been processed
- **Step 4** (Blank rounds): compute 16 blank rounds, *i.e.* repeatedly perform Step 2 with a zero input block
- **Step 5** (Output parts of final hash value): apply Step 2 once with zero input block and output the register values $a[0], a[1]$
- **Step 6** (Final hash value): repeat Step 5 until the needed hash size is met. It follows immediately that the hash value is a multiple of the word length, *e.g.* 64 bits.

Looking at the description of RADIOGATÚN, we can see immediately the analogy to a sponge: first the entire message is processed in the internal state (absorbing the message) and only after that, parts of the state are output (squeezing the sponge) resulting in the final hash value.

Applying the blank iterations can be seen as the application of a block cipher with a fixed key (zero message block) and hence acts as a permutation on the internal state. Note that during the blank iterations the attacker has no control since there are no input message blocks. Therefore, this permutation destroys certificational weaknesses such as near-collisions. Clearly, the number of blank iterations has to be chosen carefully. The hash value generation (Step 5 and Step 6) has similarities to the working principle of a stream cipher (see for instance [116, Chapter 6]). The round function can be considered as the feedback for the state. In each iteration, two words are output and the round function is applied to the state again. Then the next two words are output and so forth.

For RADIOGATÚN with a word size of 64 bits, the designers claim a sponge capacity of $l_c = 19 \cdot 64 = 1216$. This is exactly the number of bits in the Mill state. Following the definition of the capacity by Daemen and Rijmen in [39] this results in a complexity of $2^{l_c/2} = 2^{608}$ for finding inner collisions. Note that the birthday paradox still applies. For instance, if a 256-bit hash value is generated by outputting $(2 \cdot 2 \cdot 64)$-bit words, then the complexity for finding a collision for RADIOGATÚN is 2^{128}. Therefore, as long as the size of the hash value is smaller than the capacity, we can say that it is more complex to find inner collisions than a collision based on the generic birthday attack. For this word size of RADIOGATÚN, we can produce hash values of a length up to 1216 bits (as a multiple of 64 bits). If the hash size gets bigger than l_c, then finding inner collisions is less complex than finding collisions by applying the birthday attack.

7.1.3 Grindahl

At FSE 2007, Knudsen *et al.* presented a new proposal for a family of hash functions called Grindahl [90]. Grindahl follows the basic design principle of

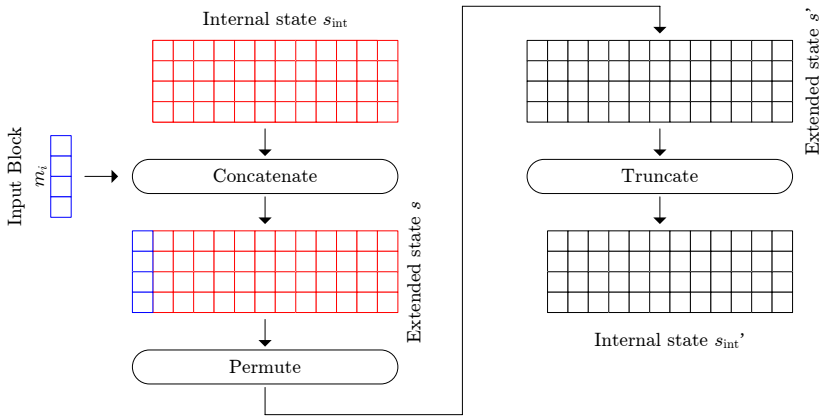

Figure 7.3: Single round of the Concatenate-Permute-Truncate principle of Grindahl.

RADIOGATÚN. The designers call the design principle *Concatenate-Permute-Truncate*, where the permutation borrows the main transformations of AES. In [90], two specific instances of the hash family Grindahl have been specified: Grindahl-256 and Grindahl-512 producing a 256-bit, respectively a 512-bit hash value. For the remainder of this section, we will only focus on Grindahl-256 and for the sake of readability, we will simply write Grindahl to refer to the 256-bit variant. Furthermore, we assume that the reader is familiar with AES and with the defined transformations.

Short Description of Grindahl-256

Due to the chosen padding method (see [90] for the specified padding method) Grindahl can process messages of at most $2^{64} - 1$ blocks, where each block consists of 4 bytes. The input message m is split (after padding) into 32-bit blocks m_i. In each iteration, a single message block is processed. The internal state s_{int} of Grindahl consists of a 4×12 array of bytes which is initialized by zero values. A single round is computed as follows. The input message block is first concatenated to the internal state s_{int} resulting in the so-called *extended state*, which we denote by s. Then the permutation $P : \{0,1\}^{416} \rightarrow \{0,1\}^{416}$ is applied to the extended state. Afterwards, the extended state is truncated by omitting the leftmost column. A graphical illustration of the *Concatenate-Permute-Truncate* round function is depicted in Figure 7.3. When the last input message block is processed, the truncation is omitted. Then 16 blank iterations are performed, *i.e.* the permutation P is applied 16 times to the extended state s. After the blank iterations have been performed, the eight rightmost columns (256 bits) are output as the final hash value.

The permutation P is defined as follows:

$$P(s) = \texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes} \circ \texttt{AddConstant}(s) \ ,$$

where the transformations are applied to the extended state from right to left. The transformations `MixColumns` and `SubBytes` are the same as defined in AES. `ShiftRows` uses the rotation constants $(1, 2, 4, 10)$, meaning that row zero is rotated by 1, row one by 2, row two by 4, and row three by 10 byte positions to the right. As stated in [90] the rotation constants have been chosen in such a way that each injected message byte affects the entire extended state after four rounds. `AddConstant` simply adds a single bit $s'_{3,12} = s_{3,12} \oplus \texttt{01}$. Note that adding this single bit is necessary in order to avoid that an extended state of equal columns can still have equal columns after applying one round if for instance a message block $m_i = 0$ is injected. Flipping this single bit destroys this property (see [90]). Figure 7.4 illustrates how the single transformations operate on the extended state s.

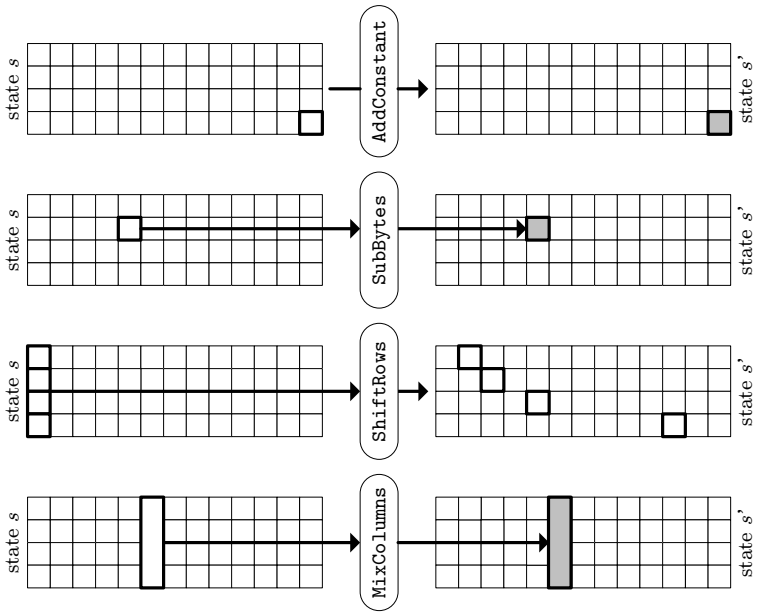

Figure 7.4: Round transformations defined for Grindahl operating on the extended state. Shadowed squares denote altered bytes.

Security Analysis of Grindahl-256

In the design document the authors investigated the security of Grindahl. They claim that for finding collisions, second preimages, and preimages, an attacker needs an effort of about 2^{128} iterations of Grindahl-256. From Section 2.3.2, we know that for an ideal hash function with a 256-bit hash value, we expect that finding (second) preimages should require about 2^n evaluations of the compression function. However, the authors of [90] only claim a resistance of $2^{n/2}$ with the remark that they expect that the complexity of (second) preimage attacks

is far beyond that bound. This is the only hash function proposal we are aware of, where the security claims for collision and (second) preimage resistance are the same. It is clear that there are other hash functions such as MDC-2 that do not achieve the ideal bound with respect to (second) preimage resistance but the lower complexity is argued by an existing generic attack which is not the case for Grindahl. Note that there exist (second) preimage attacks with less than 2^{256} for Grindahl. This has been pointed out in the design document. Also a meet-in-the-middle attack for preimages can be mounted with a complexity of about $2^{13\cdot(4\cdot8)/2} = 2^{208}$ evaluations of the round function since the round function is invertible.

Shortly after the publication of Grindahl a collision attack on Grindahl has been published by Peyrin in [133]. The attack finds collisions based on an inner collision with a complexity of about 2^{112} evaluations of the round function. Before describing the basic attack strategy, we need to introduce some terminology which is common in the analysis of block ciphers such as AES. For the analysis we do not need to look at actual differences but rather at the fact whether or not a difference exists. If there is a nonzero difference in one of the bytes of the state, we call it an *active byte*. We know that the only nonlinear transformation in Grindahl is `SubBytes`. The remaining transformations are linear over $GF(256)$. `MixColumns` takes as input a 4-byte value and returns a 4-byte output. The *branch* number of `MixColumns` is 5 (see Daemen and Rijmen [37] for a definition). This means that in total (input and output) at least 5 bytes are active. This is all we need to know for giving a short overview of the collision attack presented by Peyrin.

An obvious way to find an internal collision is to find a path through several iterations with the requirement that it ends in a state where only the leftmost column has active bytes. In this case, we would have a collision since the leftmost column is truncated at the end of the round function. Clearly, this needs to happen before the blank rounds are applied to the extended state. In order to succeed with this approach, we require that the number of active bytes in the path is as low as possible. However, this seems quite unlikely since in the design document of Grindahl the authors state that finding such a path requires at least 5 iterations. Additionally, in at least one iteration, the path has to go through a state where at least half of the state bytes are active, *cf.* [90, Property 1]. In fact, in [133] it has been shown that the number of iterations is at least 6 to find an inner-collision path. Based on this, Peyrin *et al.* proceed in a different way. They observed that by injecting arbitrary input differences over few rounds a state where all bytes are active can be constructed effectively. Then, starting from the state full of active bytes they construct a path over 8 iterations that leads to a collision (note that this is called macro-differences in [133]). The probability of the exploited path is about 2^{-440}. At first sight this seems too low for a collision attack. However, by cleverly exploiting the freedom due to the input message bytes, they are able to find an inner collision with a complexity of about 2^{112} hash computations.

An interesting approach of their attack is the fact that as long as we consider

only active bytes, we actually do not have to look at the differential properties of `SubBytes`. An input difference of `SubBytes` will always lead to an output difference. Hence `SubBytes` preserves the active byte. The only transformation that is important is `MixColumns`. Due to the branch number of `MixColumns`, we know that at least 5 bytes are active. For instance, if we have 3 active input bytes then we know that this leads to at least 2 active output bytes. However, also 3 and 4 bytes can be active. Since `MixColumns` is a linear transformation it is easy to derive linear conditions such that the number and the position of active bytes can be guaranteed. We follow the notation presented by Daemen and Rijmen in [40] for the analysis of two round differentials in AES. For any *activity pattern*, *i.e.* a vector that denotes the active input/output bytes by a 1, we can derive the so-called u^b and u^d vector. For instance, for the activity pattern $(1111;0001)$ we have $u^b = [\mathtt{9}, \mathtt{D}, \mathtt{B}, \mathtt{E}]$ and $u^d = [\mathtt{0}, \mathtt{0}, \mathtt{0}, \mathtt{1}]$. This means, that any nonzero input difference δ of the form $[\delta\mathtt{9}, \delta\mathtt{D}, \delta\mathtt{B}, \delta\mathtt{E}]$ leads to the following output difference after `MixColumns`: $[0, 0, 0, \delta]$. It is easy to see that only 255 input differences fulfill these conditions, namely all $\delta \in \{1, 2, \ldots, 255\}$. This results in a probability of $(2^8 - 1)/(2^{32} - 1) \approx 2^{-24}$. If we look at activity patterns with more than 5 active bytes then the probability increases. For instance, if we consider 7 active bytes, *i.e.* activity pattern $(1111;1110)$, then the probability can be approximated by $(2^{24} - 1)/(2^{32} - 1) \approx 2^{-8}$. In fact, the probability is slightly lower than this approximation. This can be explained by looking at so-called *bundles* introduced in [40]. Daemen and Rijmen define a bundle as the non-zero differences that follow the u^b and the u^d pattern. Therefore, if we have 7 active bytes, then we do not have 2^{24} possible output differences but in fact slightly less: 64015 bundles representing $64015 \cdot 255 < 2^{24}$ non-zero differences. Therefore, the probability for the activity pattern is $(64015 \cdot 255)/2^{32} - 1 \approx 2^{-8.04}$ instead of 2^{-8}. Nevertheless, for the attack on Grindahl this slight decrease in the probability has not been considered and can in fact be neglected.

It is immediately clear that activity patterns for `MixColumns` with a higher number of active bytes have a better probability. This can also be seen as a motivation why the attack on Grindahl starts from a state full of active bytes. If we look at the path given in [133, Figure 1], we see that most of the activity patterns for `MixColumns` have 7 and 8 active bytes. This increases the probability significantly. Another fact why the attack works are the degrees of freedom an attacker gets due to the injected message bytes. More precisely, the path in [133] has been chosen in such a way that the injected bytes can be exploited to increase the probability of `MixColumns`. For instance, assume we have an activity pattern $(1111;1110)$ which has a probability of 2^{-8}. If an injected byte in the same iteration directly influences this column, then the activity pattern has a probability of approximately 1. Therefore, by cleverly choosing the path such that the number of injected bytes can be exploited to increase the probability of `MixColumns` and by choosing activity patterns for `MixColumns` with high probability, the attack has a complexity below the birthday bound. More precisely, the path with a probability of 2^{-440} can be influenced by the degrees of freedom due to the injected message bytes resulting in a collision

attack with a complexity of about 2^{112} hash computations.

In [133], a second preimage attack has been presented as well. For the second preimage attack the collision attack has been extended accordingly. The attack has a complexity of about 2^{224} hash computations. Nevertheless, due to the security claim of $2^{n/2} = 2^{128}$ for (second) preimages by the designers of Grindahl, this cannot be considered as an attack. To summarize, Grindahl is not collision resistant. In the attack paper, modifications of Grindahl have been proposed. By adding a single column, and by choosing the rotation constants of `ShiftRows` to be $1, 2, 5, 7$ such that the diffusion properties are maximized for the larger state, the collision attack does not work anymore with a complexity below the birthday bound. However, to have a better security margin, the authors of [133] suggest to add some more columns.

7.2 The Hash Function Whirlpool

The iterated hash function Whirlpool proposed by Barreto and Rijmen in [4] is standardized in the ISO/IEC 10118-3:2003 standard for dedicated hash functions [58]. It also has been evaluated and approved by NESSIE [124]. The other hash functions in this standard are the SHA-2 family (SHA-224, SHA-256, SHA-384, and SHA-512) [128], RIPEMD-160 [52], and Tiger [1]. In the following, we give a short description of the compression function and review the security analysis of the designers of Whirlpool.

7.2.1 The Compression Function

Whirlpool [4] is an iterated hash function and processes input message blocks of 512 bits and produces a 512-bit hash value. For the first message block an initial vector iv is defined; for Whirlpool: $iv = 0$. Due to the chosen padding method (see [4] for further details) Whirlpool can hash messages of less than 2^{256} bits. The main component of Whirlpool is the underlying block cipher, referred to as W. The block cipher operates in the Miyaguchi-Preneel mode of operation as shown in Figure 7.5 (see also Section 2.3.5).

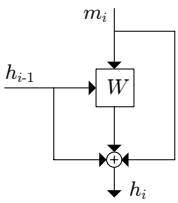

Figure 7.5: The Whirlpool hash function.

The block cipher W is strongly based on the structure of AES [122]. W is a 512-bit block cipher and uses a 512-bit key. The input (plaintext) is the i-th message block m_i to be hashed and the cipher key is the intermediate hash

value from the previous iteration h_{i-1}. The block cipher W can basically be divided into two parts: the datapath and the key schedule (see also Figure 7.6). The datapath processes the input message m_i by iteratively applying the round

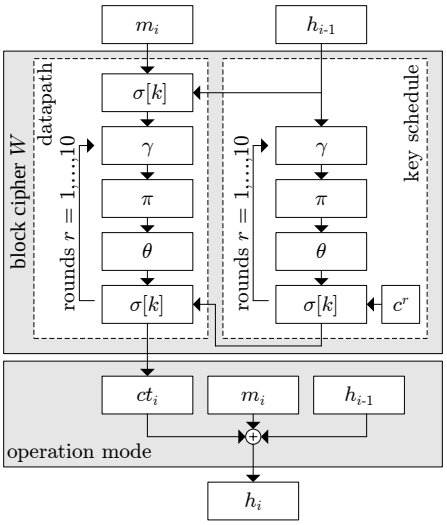

Figure 7.6: The round function of Whirlpool.

transformations for 10 rounds. Each round requires a round key that is derived by the key schedule from the cipher key. The block cipher W uses a 512-bit internal state that is organized as an 8×8 array of bytes. The state stores the input message, the intermediate results for each round, and the ciphertext after 10 rounds. As can be seen in Figure 7.6, both the datapath and the key schedule use the same round transformations. The round transformations are:

- the nonlinear layer γ, where a nonlinear S-Box is applied to each byte of the state independently
- the cyclical permutation π, where the bytes of column j are rotated downwards by j positions
- the linear diffusion layer θ, where the state is multiplied by a constant matrix
- the key addition $\sigma[k]$, where also round constants c^r are introduced

One round $\rho[k]$ of W is performed as follows:

$$\rho[k] \equiv \sigma[k] \circ \theta \circ \pi \circ \gamma \ ,$$

where the transformations are applied to the state from the right to the left.

A single input-message block is processed as follows (see also Figure 7.6). The cipher key (either h_{i-1} or $h_0 = iv$) is added to the message block m_i and

stored in the state. Then, the round transformation $\rho[k]$ is applied to the state for 10 rounds with the round key for each round provided by the key schedule. After 10 rounds, the state containing the ciphertext ct_i, the cipher key h_{i-1}, and the input-message block m_i are added (Miyaguchi-Preneel mode of operation) resulting in the cipher key h_i for the next message block, or the final hash value, if the input message has been completely processed.

When having a closer look at Figure 7.6, it can be seen that the block cipher W is actually composed of two block ciphers: the datapath with round keys provided by the key schedule and the key schedule with the round constants c^r as round keys. Therefore, also a 512-bit internal state is required for the key schedule. Both, the internal state of the datapath and the state for the key schedule are organized as an 8×8 array of bytes.

All arithmetic operations of Whirlpool required for the round transformations (addition and multiplication) are defined in the finite field $GF(256)$, *i.e.* the operations are performed at byte level. This makes Whirlpool very suitable for hardware implementations since operations such as multiplication and addition can efficiently be implemented as has been shown for numerous implementations of AES [54, 69, 102, 137]. Efficient implementation of Whirlpool have been presented for instance in [84, 140].

7.2.2 Security Claims

The designers of Whirlpool give the following security claims:

- collision resistance: $2^{n/2} = 2^{256}$
- (second) preimage resistance: $2^n = 2^{512}$

For the analysis they consider the compression function and the block cipher W. The compression function operates in the Miyaguchi-Preneel mode of operation. This is one of only few modes that have not been broken to date, see [143, 144]. Black *et al.* [24] have proven the security of, amongst others, this mode in the ideal cipher model. Based on these results the Miyaguchi-Preneel mode of operation is a favorable choice.

Barreto and Rijmen carefully analyzed the security of the block cipher W. They have given design rationales for the round transformations based on existing attacks on block ciphers. The most obvious attack is of differential nature. The transformation θ has been designed in such a way that the branch number equals 9. Based on the wide-trail design strategy of AES (see Daemen and Rijmen [37]) the number of active S-Boxes over four consecutive rounds is the square of the branch number, *i.e.* $9^2 = 81$. For the choice of the transformation γ the differential probability is 2^{-5}. Therefore, a characteristic over four rounds has a probability of at most $2^{-5^{81}} = 2^{-405}$. This clearly makes a differential attack very unlikely.

No attacks showing certificational weaknesses of Whirlpool have been published to date. Knudsen analyzed a variant of Whirlpool in [85] reduced to 6 out of 10 rounds. He showed that this variant exhibits non-random properties.

However, these properties are not a weakness for the original Whirlpool. Also the fact that Whirlpool has been standardized, and evaluated and approved by NESSIE, gives confidence in the security of the hash function. Unfortunately, no methods for truncating the output have been specified in the design document which can lead to incompatibility issues in applications where smaller hash values are required. At first sight, truncating the output of Whirlpool seems to pose no security problems except that the security margins are decreased according to the size of the hash value. However, an in-depth analysis for possible truncation methods is necessary. A new variant of Whirlpool, called MAELSTROM-0, has been presented by Filho *et al.* in [56]. As stated in [56], the hash function MAELSTROM-0 shows better performance figures than Whirlpool and can produce hash values with a variable output length up to 512 bits.

7.3 A New Block-Cipher-Based Hash Proposal: AESH-256

In [110], Mendel and Pramstaller proposed a new hash function AESH-256. The proposal is a modification of the double-block-length hash function proposed by Hirose in [68] (see also Section 5.2), which we refer to as DBLH for the remainder of this chapter. A drawback of DBLH is the achievable rate r (see Definition 2.3 in Section 2.3.5), due to the size of the message blocks: if instantiated with AES-192 then only 64-bit message blocks can be processed within one iteration and if AES-256 is used, then 128-bit message blocks can be processed. AESH-256 can process message blocks of 256 bits per iteration resulting in better performance figures compared to DBLH. The hash value has a size of 256 bits. In the following, we define the compression function, the padding method, and fix the iv and constants. After the specification of AESH-256, we present a security analysis with respect to collision and (second) preimage resistance.

7.3.1 The Compression Function of AESH-256

The compression function of AESH-256 is defined as follows, where $\text{AES-256}_k(p)$ denotes the encryption of the plaintext p under the key k with AES-256 (see also Figure 7.7):

$$\begin{aligned} h_i &= g_{i-1} \oplus c \oplus \text{AES-256}_{m_i}(g_{i-1} \oplus c) \text{ , with } c \neq 0 \\ g_i &= \text{AES-256}_{m_i}(g_{i-1}) \oplus g_{i-1} \oplus h_{i-1} \text{ ,} \end{aligned} \tag{7.1}$$

where $|h_i| = |g_i| = 128$ and $|m_i| = 256$. The concatenation of the two blocks h_0 and g_0 is the initial value. After t message blocks have been processed, the final hash value is the concatenation $h_t \| g_t$. The only requirement we have for the constant c is that $c \neq 0$. For AESH-256, we specify the following 128-bit constant consisting of the four 32-bit words $c = c_0 \| c_1 \| c_2 \| c_3$, where

$$c_0 = \texttt{0x00000000} \quad c_1 = \texttt{0x00000000} \quad c_2 = \texttt{0x00000000} \quad c_3 = \texttt{0x0AE256DC} \text{ .}$$

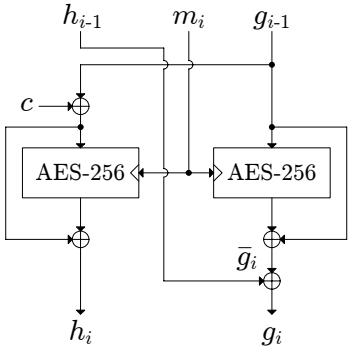

Figure 7.7: The compression function of AESH-256. The hatch marks the key input of AES-256. The value $\bar{g}_i$ will be used for the security analysis in Section 7.3.2.

For the sake of readability, we write $h_i \| g_i = \text{AESH-256}(h_{i-1} \| g_{i-1}, m_i)$ instead of (7.1) for the remainder of this section. As can be seen in (7.1), we need two encryptions per input block. Since we are using AES-256, *i.e.* $|k| = 2n$, we achieve a rate of $r = 1$ following Definition 2.3. If we compare the rate for AESH-256 with the rate of DBLH with AES-256 (see Section 5.2.1), we gain a factor of 2. In other words, the throughput will be twice as high. This is shown also in the performance comparison in Section 7.3.3.

Initial Value

For AESH-256, we define the same iv as for SHA-256, *cf.* [128], *i.e.* we take the first 32 bits of the fractional parts of the square roots of the first eight prime numbers. The 256-bit initial value $iv = h_0 \| g_0$ is specified as follows, where $h_0 = h_{0,0} \| h_{0,1} \| h_{0,2} \| h_{0,3}$ and $g_0 = g_{0,0} \| g_{0,1} \| g_{0,2} \| g_{0,3}$ with

$$\begin{aligned} h_{0,0} &= \texttt{0x6A09E667} & g_{0,0} &= \texttt{0x510E527F} \\ h_{0,1} &= \texttt{0xBB67AE85} & g_{0,1} &= \texttt{0x9B05688C} \\ h_{0,2} &= \texttt{0x3C6EF372} & g_{0,2} &= \texttt{0x1F83D9AB} \\ h_{0,3} &= \texttt{0xA54FF53A} & g_{0,3} &= \texttt{0x5BE0CD19}\ . \end{aligned}$$

MD Strengthening

For MD strengthening, we have to fix the iv and we have to define an unambiguous padding method. We already defined the iv and the constant c for AESH-256. The padding is defined as follows. Take the Y-bit input message m ($Y < 2^{128}$) and append a '1' followed by z '0' bits such that z is the smallest positive integer satisfying

$$Y + 1 + z \equiv 128 \bmod 256$$

and, finally, append the binary representation of the length of the original message m. Due to the chosen padding method, AESH-256 can process messages with a length of at most $(2^{128} - 1)$ bits.

7.3.2 Security Analysis

In this section, we analyze the security of AESH-256 with respect to collision resistance and (second) preimage resistance. Based on this analysis, we expect the following security margins:

1. Collision resistance: $2^{n/2} = 2^{128}$
2. (Second) Preimage resistance: $2^{n/2+1} = 2^{129}$

Note that the attacks presented by Knudsen *et al.* in [88] do not apply to AESH-256. This follows from the fact that the attacks are applied to a general class of rate 1 hash functions, where in each iteration two message blocks are processed. This is clearly not the case for AESH-256—only one message block is processed in each iteration.

If we look for instance at the MDC-2 construction [116], second preimage and preimage resistance is bounded by $2^{3n/4}$. For AESH-256, we can only claim $2^{n/2+1}$. This is due to a structural property of the proposed double-block-length construction. The same property has been exploited by Mendel and Rijmen [115] for a (second) preimage attack on the single-block-length dedicated hash construction HAS-V [131]. We will explain this structural limitation of AESH-256 and the according generic (second) preimage attack in the following sections.

Collision Resistance

Consider the first two iterations of AESH-256. Then, based on the structure of AESH-256, we observe the following property:

Property 7.1. *A zero difference in the chaining variable g_1 leads automatically to a zero difference in h_2 if there is a zero difference in m_2. This is independent of h_1, i.e. if $g_1 = g_1^*$ and $m_2 = m_2^*$ then $h_2 = h_2^*$ for any value of h_1.*

One could now use the following strategy for constructing a collision over two iterations of AESH-256 (see also Figure 7.8). Find a message pair $m_1 \neq m_1^*$ in the first iteration that leads to $g_1' = g_1 \oplus g_1^* = 0$. Then search for a message block m_2 that leads to $g_2' = 0$ in the second iteration. Together with Property 7.1 this would lead to a collision after the second iteration of the compression function. However, it is easy to verify that this is not possible. The reason why this attack based on Property 7.1 does not work is as follows. Assume $g_1' = 0$. Then we know that $h_2' = 0$ if $m_2' = 0$ (Property 7.1). Since we found $m_1 \neq m_1^*$ such that $g_1' = 0$ we know that with overwhelming probability $h_1 \neq h_1^*$ (note that if $h_1' = 0$, we would have already a collision). With $g_1' = 0$ and $m_2' = 0$ the difference after the second iteration is $h_2' = 0$ but since $h_1 \neq h_1^*$ we have $g_2' \neq 0$. To have $g_2' = 0$ we need that $m_2 \neq m_2^*$ but then Property 7.1 does not hold.

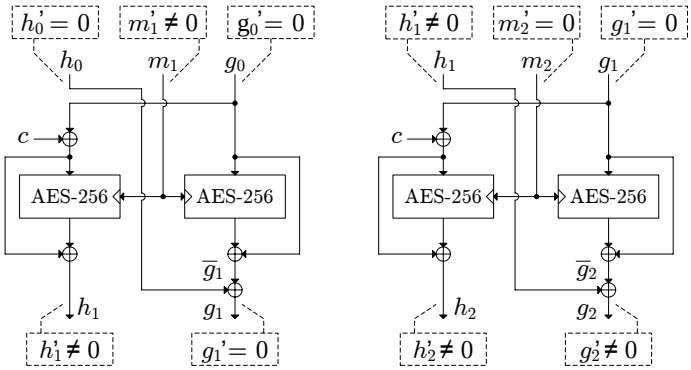

Figure 7.8: Graphical illustration of Property 7.1 for AESH-256. Dashed rectangles denote differences.

We did not find any other attack strategies that exploit Property 7.1. In general, we do not see any other collision shortcut attack that has lower complexity than a birthday attack. However, the structure of AESH-256 allows to construct pseudo-collisions with a complexity of about 2^{64} encryptions. The pseudo-collision attack is described in Algorithm 7.1.

Algorithm 7.1 Constructing a pseudo-collision for AESH-256.

Input: Intermediate chaining value $h_{i-1} \| g_{i-1}$

Output: Two messages m_i and m_i^*, and the difference h'_{i-1} such that AESH-256$(m_i, h_{i-1} \| g_{i-1}) =$ AESH-256$(m_i^*, h_{i-1} \oplus h'_{i-1} \| g_{i-1})$.

- Apply a birthday attack to find two messages m_i and m_i^* such that $h'_i = 0$, *i.e.* $h_i = h_i^*$. This requires about 2^{64} encryptions
- For the messages m and m^* compute the values $\bar{g}_i$ and $\bar{g}_i^*$ for the given g_{i-1}. This determines the difference $h'_{i-1} = \bar{g}_i \oplus \bar{g}_i^*$

(Second) Preimage Resistance

For an ideal hash function with an n-bit hash value one would expect a security margin of 2^n for finding a (second) preimage for the hash function. However, in the case of AESH-256, we can find a (second) preimage with complexity close to $2^{n/2+1}$ hash computations. Firstly, we will describe an attack with complexity $2^{3n/4}$, which is similar to the attack that can be applied to MDC-2, see [93]. Secondly, we present a structural attack following [115] which results in a (second) preimage attack in about $2^{n/2+1}$ evaluations of the compression function.

Fact 7.1. *The compression function is invertible with respect to the chaining variables.*

Algorithm 7.2 A way to invert the compression function.

Input: The intermediate hash value $h_i||g_i$ and an arbitrary message block m_i
Output: The intermediate hash value $h_{i-1}||g_{i-1}$ such that AESH-256$(m_i, h_{i-1}||g_{i-1}) = h_i||g_i$ holds.

- Guess g_{i-1} and check if the following equation holds:

$$h_i = g_{i-1} \oplus c \oplus \text{AES-256}_{m_i}(g_{i-1} \oplus c)$$

  Hence, a correct choice for g_{i-1} can be found in about 2^{128} encryptions

- Calculate $h_{i-1} = g_i \oplus \text{AES-256}_{m_i}(g_{i-1}) \oplus g_{i-1}$

Based on Algorithm 7.2, the hash function is invertible with respect to the chaining variables. In detail, for a given h_i, g_i and an arbitrary message, we can find h_{i-1} and g_{i-1} in about 2^{128} applications of the compression function. This can be used to find a preimage in the hash function with a complexity of about 2^{192} based on Fact 7.2.

Fact 7.2. *An unbalanced meet-in-the-middle attack can be used to find a (second) preimage similar as it has been shown by Lai and Massey in [93].*

Based on Algorithm 7.3, we can find a preimage in the hash function with complexity about $2 \cdot 2^{192}$ applications of the compression functions. Note that a similar attack can also be applied to MDC-2.

Algorithm 7.3 Unbalanced birthday attack to find a preimage of the hash function

Input: The final hash value $h_i||g_i$
Output: The message $m = m_{i-1}||m_i$ such that AESH-256$(m, h_{i-2}||g_{i-2}) = h_i||g_i$.

- Calculate and store 2^{64} candidates for $h_{i-1}||g_{i-1}$ in the list L resulting from a backward computation of the compression function using Algorithm 7.2 ($2^{64} \cdot 2^{128}$ applications of the compression function).
- Calculate $h_{i-1}||g_{i-1}$ resulting from a forward computation of the compression function and check for a match in L.
- After calculating at most 2^{192} candidates for $h_{i-1}||g_{i-1}$ one can find a matching entry in L (a collision) and hence a preimage. Note that a collision is likely to exist due to the birthday paradox.

So far, we achieved the same security margin with respect to (second) preimage resistance as MDC-2. Nevertheless, we explain now a generic attack that is based on [115] resulting in a (second) preimage resistance of 2^{129} for AESH-256. The attack is based on the following fact.

Fact 7.3. *The structure of AESH-256 allows to construct a fixed point in the right stream for an arbitrary value g_i and m_i. That means, we can easily find h_{i-1} and h_i such that*

$$\textit{AESH-256}(h_{i-1}\|g_{i-1}, m_i) = h_i\|g_{i-1} \ .$$

To construct a fixed-point following Fact 7.3, we proceed as follows. Choose an arbitrary value g_{i-1} and an arbitrary m_i. Then apply the compression function to compute the values h_i and $\bar{g}_i$ (see Figure 7.7). Finally, compute $h_{i-1} = g_i \oplus \bar{g}_i$ resulting in the fixed point. Thus, constructing a fixed point in the right stream of AESH-256 requires only one application of the compression function.

We describe now how one can find a preimage for AESH-256, *i.e.* we are given the final hash value $(h_t\|g_t)$ and want to find the according message m. For the discussion of the attack, we assume that we have messages with a length that is a multiple of the block length (*i.e.* a multiple of 256 bits) and that no padding is applied (padding will be treated separately). A graphical illustration is given in Figure 7.9.

1. **Step 1:** For the given hash value $(h_t\|g_t)$ we take the initial value $g_{t-1} = g_0$ and choose random messages m_t until we find the according h_t. Since h_t is a 128-bit value, we expect to find the right h_t after at most 2^{128} randomly chosen messages. Once we found h_t, we choose h_{t-1} such that we match the value g_t, *i.e.* $h_{t-1} = g_t \oplus \bar{g}_t$. Now, since $g_{t-1} = g_0$, we can exploit the fact that we can construct fixed points in the right stream of AESH-256 to preserve the value g_0.

2. **Step 2:** In the second phase of the attack, we compute $2 \cdot 2^{128}$ fixed-points as described before and store the triplets (h_{i-1}, h_i, m_i) in the list L. Now, for each entry we have a fixed point such that AESH-256$(h_{i-1}\|g_0, m_i) = (h_i\|g_0)$. Based on the 2^{129} entries in the list, we expect to have two entries in L with the same h_{i-1} and two entries with the same h_i. This fact can be exploited for the preimage attack. Step 2 has a complexity of 2^{129} evaluations of the compression function.

3. **Step 3:** Based on the list L constructed in Step 2, we build two trees (see Figure 7.9), referred to as the upper and lower tree. We start with the upper tree using the value h_0 as the first node. Then, for each node we get two new nodes since we expect two entries that are identical in the list L (see Step 2). We continue like this until we have a tree of depth k, knowing that at this depth we have in total 2^k nodes with different h_i's. We do the same for the lower tree by starting from h_{t-1}. Therefore, we have two trees of depth k. The choice of k depends on the bit size of the chaining value h_i. For AESH-256, we have 128 bits. Setting $k = 64$, we know that we have 2^{64} h_i's in the upper and lower tree at depth k. Due to the birthday paradox it is very likely that we find two matching h_i's between both trees. Hence, finding a path from $(h_{t-1}\|g_0)$ to $(h_0\|g_0)$ requires about 2^{129} evaluations of the compression function.

Based on this description, we can give the complexity of this preimage attack on AESH-256: constructing a preimage requires about $2^{128} + 2 \cdot 2^{128} \approx 2^{129}$ evaluations of the compression function. Furthermore, the preimage for $(h_t \| g_t)$ has a length of $64 + 64 + 1 = 129$ message blocks. Note that for estimating the complexity, we did not consider the cost of building the trees. The memory requirements for storing the trees is about $2 \cdot \sum_{i=1}^{64} 2^i \approx 2^{66}$ message blocks. Note that there exist alternative ways to construct the trees requiring less storage but in this cases the complexity for constructing the trees increases.

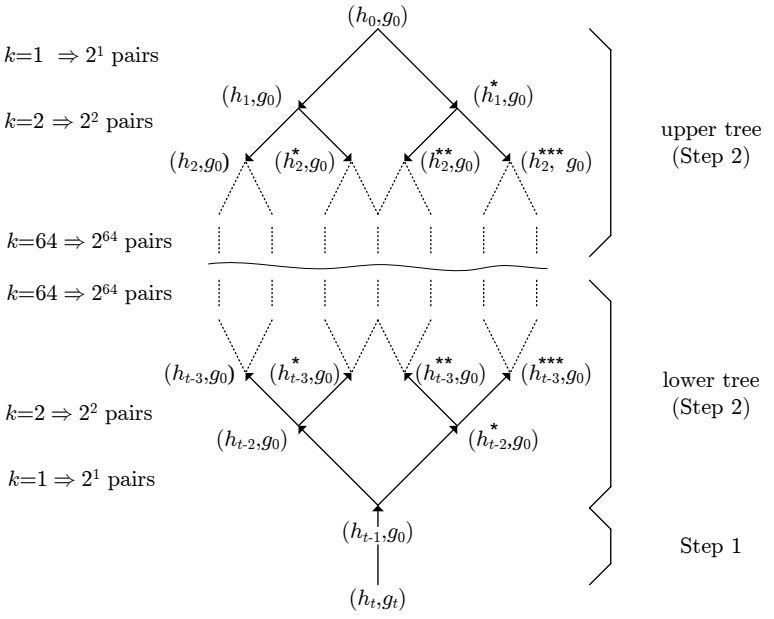

Figure 7.9: Graphical illustration of the preimage attack on AESH-256.

So far, we omitted the padding in the discussion. Including the padding leads to a slightly modified preimage attack. However, the basic attack strategy is the same, since we know that the preimage consists of 129 blocks. Therefore, we start with constructing the final message block resulting from padding. Computing once the inverse of the compression function leads to the chaining values $(h_{t-1} \| g_{t-1})$, which are then the target values (the first preimage) for the preimage attack. Computing the inverse for the padding block can be done along the same lines as described in Step 2, with the only difference that the message block is fixed. Therefore, this step additionally requires about 2^{128} evaluations of the compression function.

Note that the preimage attack implies a second preimage attack. In this case we are not given only the hash value $h_t \| g_t$ but also a message m that results in this hash value. We can construct for any given message a second preimage in the same way as we construct preimages. The difference is, that the second preimage will always consist of 129 message blocks. We can also construct

preimages consisting of less blocks. In this case the preimage attack has to be modified. This results in a higher attack complexity. This can be explained by looking at Step 3. As described, we add to each node two new nodes resulting in 2^k nodes at a depth of k in the lower and upper tree. Actually, we are not restricted by adding two nodes, but in fact, we can also add for instance 4 nodes. This leads to a smaller depth of the trees. For instance, if we add four new nodes at each node then we have 2^{2k} nodes at a depth of k. Therefore, at the depth $k = 32$, we have 2^{64} different h_i's. This results in a (second) preimage consisting of $32 + 32 + 1 = 65$ message blocks instead of 129. It is clear that adding 4 new nodes leads to accordingly more values in the list L. This in turn results in a higher complexity for Step 2 of the attack. Nevertheless, the attack still works. Based on this generic attack, we claim a (second) preimage resistance for AESH-256 of $2^{n/2+1} = 2^{129}$.

7.3.3 Performance Evaluation and Comparison

In this section, we give a relative performance comparison between MDC-2, DBLH, and AESH-256. Note that the MDC-2 construction requires a block cipher with a key length less or equal than the block length, *cf.* [116]. For instance, in [166] the MDC-2 construction instantiated with AES-128 as underlying block cipher is presented.

One big advantage of DBLH and AESH-256 is that the key schedule only has to be computed once, whereas for MDC-2 the key schedule has to be computed twice in each iteration. For the implementation, we used the IAIK Crypto Toolkit [162] and the C-implementation of AES of Gladman [62]. The performance comparison is given in Table 7.2.

Table 7.2: Throughput comparison of MDC-2, DBLH, and AESH-256.

	MDC-2 AES-128	DBLH AES-192	DBLH AES-256	AESH-256
PC32C [1]	23.6 MB/sec	11.9 MB/sec	19.5 MB/sec	40.8 MB/sec
PC32J [2]	3.8 MB/sec	2.2 MB/sec	3.8 MB/sec	7.5 MB/sec
PC64J [3]	7.9 MB/sec	3.7 MB/sec	6.7 MB/sec	13.7 MB/sec

[1] Intel Pentium M, 1.7 GHz 32-bit architecture, Microsoft VC++ compiler
[2] Intel Pentium 4, 2 GHz 32-bit architecture, Java VM: SUN JDK 1.5.0_06
[3] AMD Opteron, 2 GHz 64-bit architecture, Java VM: SUN JDK 1.5.0_04

As can be seen in Table 7.2, AESH-256 compared to MDC-2 achieves an improvement of about 70-90%, depending on the AES implementation and on the architecture. Compared to DBLH, AESH-256 performs approximately four times (DBLH with AES-192) or twice as fast (DBLH with AES-256). This also correlates to the rate of the hash functions, *cf.* Section 7.3.1.

Compared to DBLH instantiated with either AES-192 or AES-256, we achieve better performance figures with the disadvantage that we do not have a security proof for the collision resistance of AESH-256. In [68], Hirose does not comment on (second) preimage resistance. However, the attacks on AESH-256 presented in this chapter do not apply. We expect that the security margins for DBLH with respect to (second) preimage resistance are way beyond $2^{n/2}$. If we compare our proposal with the MDC-2 construction instantiated with AES-128, then we can claim the same security margins for collision resistance but not for (second) preimage resistance. In addition to the higher security margin for (second) preimage attacks, very recently, a proof for the collision resistance of MDC-2 has been presented by Steinberger in [157].

In the following, we list some advantages of AESH-256:

- AESH-256 has a very simple structure and therefore can be implemented easily in existing systems that support AES-256. Furthermore, the simple structure makes the security analysis easier
- AESH-256 shows better performance than MDC-2 and DBLH
- AESH-256 uses the popular AES as underlying block cipher. For AES, no weaknesses have been found to date

7.4 Summary and Discussion

In this chapter, we reviewed some recent proposals for modifications of SHA-1. The two discussed proposals, a new message expansion and two message preprocessing methods, make the modified SHA-1 more secure against existing attacks. However, as discussed in this chapter, an in-depth security analysis is required to get a good insight in the strengthened security, since in some special cases collision attacks are still possible.

A very interesting approach in designing new hash functions are the discussed sponge functions. Sponge functions are intended to be a new reference model for modeling an ideal hash function. As opposed to the random oracle model, sponge functions consider more realistic properties such as internal collisions. It is believed that the required properties of sponge functions can be realized with iterated designs. An example for a hash functions that uses sponge functions as a reference model is the hash function RADIOGATÚN. This type of hash function borrows techniques from block cipher and stream cipher design. As we have shown, the basic idea is to work on a bigger internal state and only simple transformations operating on this state are defined. An interesting property of this proposal is that first the entire input message is processed before a certain number of blank rounds are applied. The blank rounds are introduced in order to withstand attacks that are based on certificational weaknesses such as near-collisions. This approach was actually also used for the design strategy SMASH where a single blank round is applied after processing the entire input message. RADIOGATÚN is still work in progress and the designers motivate its

security analysis. Grindahl is a new proposal that follows the design principles of RADIOGATÚN and borrows the basic transformations from the block cipher AES. The hash function can be efficiently implemented in both hardware and software and the reuse of transformations of AES makes it a very *elegant* design. Also, analyzing the security is facilitated by the fact that one can operate at byte level. Nevertheless, a collision attack has been presented shortly after the hash function Grindahl has been proposed. In order to withstand the attack a possible approach is to increase the size of the internal state.

A potential drawback of both RADIOGATÚN and Grindahl is the huge internal state. For instance, the 64-bit variant of RADIOGATÚN has an internal state size of 3712 bits. Clearly, for implementations in restricted environments this can be a crucial drawback. Compared to RADIOGATÚN, Grindahl has a significant smaller internal state, namely $13 \cdot 4 \cdot 8 = 416$ bits. Nevertheless, in order to thwart the collision attack the state size still needs to be increased.

From the discussion of these recent hash function proposals, we can derive some remarks and suggestions for approaching the design of new cryptographic hash functions. What we have learned from the past years, is that dedicated designs such as SHA-1 and a lot of other proposals designed from scratch, have shown weaknesses, once researchers invest enough effort to analyze these primitives. Therefore, designing a new hash function along the same lines as the MD family is questionable. It is likely that the collision attacks on the MD family presented by Wang *et al.* [169, 172, 173] can be extended to other hash functions that are based on similar design principles. In particular, the recent results on SHA-1 by De Cannière and Rechberger in [28] and De Cannière *et al.* in [27] make an extension more likely. Also the fact that hash functions following the Merkle-Damgård design principle can for instance not achieve a security margin of 2^n for second preimages (see the generic second preimage attack for long messages discussed in Section 3.2.2), emphasizes that it may be advisable to move away from MD constructions.

An alternative to dedicated hash functions are block-cipher-based hash functions. One of the biggest advantages of block-cipher-based constructions is that a popular block cipher such as AES has gone through a public security analysis over several years. This clearly gives confidence in using this primitive. We have discussed two block-cipher-based hash proposals. The first is Whirlpool that has been designed in such a way that the size of the hash value is large enough to fulfill nowadays security requirements. By doing so, no double-block-length construction is required and one of the provable modes of operations can be used. However, the underlying block cipher has been designed from scratch. Even if similar design rationales as for AES are given, the new block cipher W has definitely not been analyzed as thoroughly as for instance AES. What should be required from a hash function such as Whirlpool is the specification of truncation modes due to compatibility reasons. Clearly, truncation methods should be part of the security analysis. The new variant of Whirlpool, namely MAELSTROM-0, considers various sizes of the hash value. The new block-cipher-based hash function AESH-256 uses AES as underlying block cipher. In order to achieve the

required security margins a double-block-length construction is used. What we have seen with this hash function is that structural weaknesses can be exploited to lower the security bounds. For instance, a (second) preimage attack can be performed with a complexity of about $2^{n/2+1} = 2^{129}$ instead of the expected 2^n. Note that the recent results of Knudsen and Rijmen in [91] may have implications on the security of block-cipher-based hash functions including Whirlpool and AESH-256—this needs to be further investigated.

With block-cipher-based hash constructions a fixed-size hash value can be computed with the advantage of using a well known primitive. Other hash sizes can only be achieved by truncating the output. This in turn requires an independent security analysis. If we look at RADIOGATÚN, then this design approach allows to produce a variable hash size with the only restriction that it will be a multiple of the internal word size. Depending on the size of the internal state and the claimed capacity, this hash function can produce hash values of various sizes. An advantage of RADIOGATÚN is that the security margins can be estimated accurately for the different output sizes. Based on the existing security analyses of this new design principle for hash functions, it is clearly too early to recommend the use of this kind of hash function. Further analysis is required to avoid a similar situation we encountered with the hash functions of the MD family, in particular MD5 and SHA-1.

Other points to consider when designing hash functions are the security margins for collision and (second) preimage resistance. For the hash functions in this chapter, we have seen different approaches. First of all, most of the cryptanalytic results focus on collision resistance. For (second) preimage resistance, we are assuming in general the bound we expect from an ideal hash function, namely 2^n for a hash function with an n-bit hash value. In the example of Grindahl, we have seen that even if there is no known (second) preimage attack, the claimed margins are in the same order of magnitude as for collision resistance. This is the first proposal, where this was done. Clearly, there are other proposals where the security margins are below the ones expected from an ideal hash function (*e.g.* MDC-2) but in these cases there exist generic attacks that force the claim of lower security margins. The same holds for AESH-256. However, it remains an open discussion whether or not it is advisable to claim the same security margins for collision and for (second) preimage resistance, even if there do not exist any attacks that force lower claims.

8

Conclusion and Further Research Directions

In this thesis, we have focused on the analysis and design of cryptographic hash functions. We have reviewed general cryptanalytic techniques for the analysis of iterated hash functions. The collision attack on SHA-1 has been reviewed in detail. Furthermore, we have shown how coding theory can be exploited to find high-probability linear characteristics for the state update transformation of SHA-1, which plays a crucial role in the collision attack. We have refined the estimate of the probability of linear characteristics by considering side-effects such as the impact of carries. This resulted in a slight improvement of the attack complexity compared to the original collision attack by Wang *et al.*

We have presented an efficient collision attack on the new hash function design strategy SMASH. A generalization of this attack resulted in a powerful second preimage attack. Furthermore, we have shown for a new block-cipher-based double-block-length hash construction DBLH that the underlying block cipher is crucial for the security. If a block cipher following the FX construction is employed in DBLH, second preimages can be constructed efficiently. Motivated by these results, we have described and discussed how recent results in the cryptanalysis of hash functions affect the hash-based message authentication codes NMAC and HMAC. For the analysis of the impact on NMAC and HMAC, we have considered the existing collision attacks on hash functions such as SHA-1 and also the second preimage attacks on the hash functions SMASH and DBLH. It turned out that the cryptanalytic results on hash functions do affect the security of NMAC and HMAC.

Regarding the design of hash functions, we firstly have discussed some modifications of SHA-1 that have been proposed in order to increase the complexity of existing collision attacks. Secondly, we have described and discussed selected hash function designs such as Whirlpool and the two new proposals RADIOGATÚN and Grindahl. We also have proposed a new block-cipher-based double-block-length hash construction AESH-256 and have given a security analysis with respect to collision and (second) preimage resistance.

8.1 Cryptanalysis of Hash Functions

As discussed in this thesis, differential cryptanalysis is a powerful method for the analysis of hash functions. The majority of dedicated hash functions based on the Merkle-Damgård design principle has been (theoretically) broken by exploiting and extending methods from differential cryptanalysis. Exceptions are for instance the SHA-2 family of hash functions (including amongst others SHA-256 and SHA-512) standardized in FIPS-180-2 [128], and RIPEMD-160 designed by Dobbertin *et al.* in [52]. Only few cryptanalytic analyses of these hash functions have been published to date, *e.g.* see [113] for an analysis of RIPEMD-160, and [61, 66, 107, 112, 175] for preliminary results on SHA-256 and SHA-512. Compared to SHA-1, these hash functions have a bigger internal state and more complex internal operations are employed. Nevertheless, the main concern regarding the security of these hash functions is that the constructions follow similar design principles as SHA-1. Therefore, an important open question is whether or not the known attack methods that have been used to break SHA-1 can be refined and extended in such a way that they can also be applied to break these hash functions.

In the past years, hash functions have been analyzed mostly with respect to collision resistance. Only few results regarding (second) preimage attacks have been published to date. It follows, that an interesting open research problem is to investigate whether collision attacks can be extended in order to construct second preimages. For the hash function MD4, first results in this direction have been published by Yu *et al.* in [177], but no similar results for the hash functions MD5 and SHA-1 are known to date.

8.2 Design of Hash Functions

The design of new hash functions will be the most challenging task in the future. Designers are faced with the difficulty of having no clear view on which security requirements a hash function should fulfill. So far, the reference model for an ideal hash function was, and still is, the random oracle model. However, the results of the past years have shown that we are far away from this model with respect to what can be achieved by iterated hash function designs. Therefore, an important open research problem is to find alternative reference models that consider more realistic scenarios. One step in this direction are sponge functions. Even if this new reference model is a promising alternative to the random oracle model, further in-depth analysis is required.

Similar to the development process for the Advanced Encryption Standard, NIST is initiating a public effort to develop one or more hash functions. The tentative schedule foresees 5 years including several workshops during the evaluation phase of submitted hash function proposals. In 2012, NIST plans to revise FIPS-180-2 [128] depending on the results of the public effort (see [127] for the tentative timeline). As a first step, NIST has drafted the minimum requirements for hash function proposals. The following items are used for judging new

proposals with respect to security [126]:

1. security with respect to collision and (second) preimage resistance compared to other submissions
2. the extent to which the algorithm output is indistinguishable from a random oracle
3. soundness of the mathematical basis for the security of the hash function
4. all other security factors that may arise during the public evaluation phase of the submitted hash function proposals

These draft requirements do not really include any new requirements on hash functions, except the fact that NIST prefers proposals that support an elegant security analysis. Therefore, the first step is to clearly specify what we expect from a hash function by refining these minimum requirements. It is advisable to not only specify the minimum security requirements for the hash function itself but also for hash-based constructions and applications including cryptographic protocols. In other words, we need to specify general minimum requirements for the security of hash functions as a primitive, and additionally, we need to specify application-dependent security requirements. Such a clear specification of the security requirements is the first step to design new hash functions that satisfy our needs for the next decades.

It is an open question whether or not the public effort in designing new hash functions, will be as successful as the selection process of the Advanced Encryption Standard. Independent of this, it will definitely lead to new insights and new directions in both the analysis and design of cryptographic hash functions and hash-based constructions.

Bibliography

[1] Ross J. Anderson and Eli Biham. TIGER: A Fast New Hash Function. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *LNCS*, pages 89–97. Springer, 1996.

[2] Steve Babbage, Carlos Cid, Norbert Pramstaller, and Håvard Raddum. An Analysis of the Hermes8 Stream Ciphers. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *Information Security and Privacy, 12th Australasian Conference, ACISP 2007, Townsville, Australia, July 2-4, 2007, Proceedings*, volume 4586 of *LNCS*, pages 1–10. Springer, 2007.

[3] Paulo S.L.M. Barreto. The Hash Function Lounge. `http://paginas.terra.com.br/informatica/paulobarreto/hflounge.html`.

[4] Paulo S.L.M. Barreto and Vincent Rijmen. The Whirlpool Hashing Function, 2000, revised in May 2003. `http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html`.

[5] Mihir Bellare. New Proofs for NMAC and HMAC: Security Without Collision Resistance. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *LNCS*, pages 602–619. Springer, 2006.

[6] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *LNCS*, pages 1–15. Springer, 1996.

[7] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The Security of Cipher Block Chaining. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *LNCS*, pages 341–358. Springer, 1994.

[8] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The Security of the Cipher Block Chaining Message Authentication Code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.

[9] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[10] Mihir Bellare and Phillip Rogaway. CSE 207 Course Notes: Introduction to Modern Cryptography, 2005. `http://www-cse.ucsd.edu/~mihir/cse207/`.

[11] Elwyn R. Berlekamp, Robert J. McEliece, and Henk C. A. van Tilborg. On the Inherent Intractability of Certain Coding Problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[12] Daniel J. Bernstein. The Poly1305-AES Message-Authentication Code. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *LNCS*, pages 32–49. Springer, 2005.

[13] Guido Bertoni, Joan Daemen, and Gilles Van Assche. RADIOGATÚN, a Belt-and-Mill Hash Function, 2006. `http://radiogatun.noekeon.org/`.

[14] Guido Bertoni, Joan Daemen, and Gilles Van Assche. RADIOGATÚN, a Belt-and-Mill Hash Function. Presented at the Second Cryptographic Hash Workshop hosted by NIST, 2006.

[15] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge Functions. Presented at the Dagstuhl Seminar 07021: Symmetric Cryptography, 2007. `http://kathrin.dagstuhl.de/files/Materials/07/07021/07021.DaemenJoan1.Slides.pdf`.

[16] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A New Block Cipher Proposal. In Serge Vaudenay, editor, *Fast Software Encryption, 5th International Workshop, FSE 1998, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *LNCS*, pages 222–238. Springer, 1998.

[17] Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *LNCS*, pages 290–305. Springer, 2004.

[18] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and Reduced SHA-1. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 36–57. Springer, 2005.

[19] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[20] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard.* Springer, 1993. ISBN 0-387-97930-1.

[21] Alex Biryukov and David Wagner. Advanced Slide Attacks. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques,*

Bruges, Belgium, May 14-18, 2000, Proceeding, volume 1807 of *LNCS*, pages 589–606. Springer, 2000.

[22] John Black. The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *LNCS*, pages 328–340, 2006.

[23] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. UMAC: Fast and Secure Message Authentication. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *LNCS*, pages 216–233. Springer, 1999.

[24] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *LNCS*, pages 320–335. Springer, 2002.

[25] Karl Brincat and Chris J. Mitchell. New CBC-MAC Forgery Attacks. In Vijay Varadharajan and Yi Mu, editors, *Information Security and Privacy, 6th Australasian Conference, ACISP 2001, Sydney, Australia, July 11-13, 2001, Proceedings*, volume 2119 of *LNCS*, pages 3–14. Springer, 2001.

[26] Johannes Buchmann. *Einführung in die Kryptographie.* Springer, 1999. ISBN 3-540-66059-3.

[27] Christophe De Cannière, Florian Mendel, and Christian Rechberger. Collisions for 70-step SHA-1: On the Full Cost of Collision Search. In *Selected Areas in Cryptography, 14th Annual International Workshop, SAC 2007, Ottowa, Ontario, Canada, August 16-17, 2007. Revised Papers*, to appear in LNCS. Springer, 2007.

[28] Christophe De Cannière and Christian Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006.

[29] Anne Canteaut and Florent Chabaud. A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998.

[30] Florent Chabaud. On the Security of Some Cryptosystems Based on Error-Correcting Codes. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *LNCS*, pages 131–139. Springer, 1995.

[31] Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *LNCS*, pages 56–71. Springer, 1998.

[32] Scott Contini and Yiqun Lisa Yin. Forgery and Partial Key-Recovery Attacks on HMAC and NMAC Using Hash Collisions. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *LNCS*, pages 37–53. Springer, 2006.

[33] Scott Contini and Yiqun Lisa Yin. Forgery and Partial Key-Recovery Attacks on HMAC and NMAC Using Hash Collisions. Cryptology ePrint Archive, Report 2006/319, 2006. Available online at `http://eprint.iacr.org/`.

[34] Don Coppersmith. The Data Encryption Standard (DES) and its Strength Against Attacks. *IBM Journal of Research and Developement*, 38(4):243–250, 1994.

[35] Joan Daemen and Gilles Van Assche. Producing Collisions for Panama, Instantaneously. In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg City, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *LNCS*, pages 1–18, 2007.

[36] Joan Daemen and Craig S. K. Clapp. Fast Hashing and Stream Encryption with PANAMA. In Serge Vaudenay, editor, *Fast Software Encryption, 5th International Workshop, FSE'98, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *LNCS*, pages 60–74, 1998.

[37] Joan Daemen and Vincent Rijmen. *The Design of Rijndael.* Information Security and Cryptography. Springer, 2002. ISBN 3-540-42580-2.

[38] Joan Daemen and Vincent Rijmen. A New MAC Construction ALRED and a Specific Instance ALPHA-MAC. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *LNCS*, pages 1–17. Springer, 2005.

[39] Joan Daemen and Vincent Rijmen. A New MAC Construction ALRED and a Specific Instance ALPHA-MAC. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *LNCS*, pages 1–17. Springer, 2005.

[40] Joan Daemen and Vincent Rijmen. Understanding Two-Round Differentials in AES. In Robert De Prisco and Moti Yung, editors, *Security and Cryptography for Networks, 5th International Conference, SCN 2006, Maiori, Italy, September 6-8, 2006, Proceedings*, volume 4116 of *LNCS*, pages 78–94. Springer, 2006.

[41] Ivan Damgård. A Design Principle for Hash Functions. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *LNCS*, pages 416–427. Springer, 1989.

[42] Magnus Daum. *Cryptanalysis of Hash Functions of the MD4-Family.* PhD thesis, Ruhr Universität Bochum, 2005. Available online at `http://www.cits.rub.de/imperia/md/content/magnus/dissmd4.pdf`.

[43] D.W. Davies and D.O. Clayden. The Message Authenticator Algorithm (MAA) and its Implementation. Report DITC 109/88, National Physical Laboratory, U.K., February 1988.

[44] Richard Drews Dean. *Formal Aspects of Mobile Code Security.* PhD thesis, Princeton University, 1999.

[45] Bert den Boer and Antoon Bosselaers. An Attack on the Last Two Rounds of MD4. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *LNCS*, pages 194–203. Springer, 1991.

[46] Bert den Boer and Antoon Bosselaers. Collisions for the Compressin Function of MD5. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *LNCS*, pages 293–304. Springer, 1993.

[47] Alex W. Dent and Chris J. Mitchell. *User's Guide To Cryptography And Standards.* Artech House Publishers, 2004. ISBN 1-580-53530-5.

[48] Tim Dierks and Christopher Allen. The TLS Protocol Version 1.0, Request for Comments (RFC) 2246, The Internet Engineering Task Force, 1999.

[49] Hans Dobbertin. Cryptanalysis of MD4. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *LNCS*, pages 53–69. Springer, 1996.

[50] Hans Dobbertin. The Status of MD5 After a Recent Attack. *RSA Laboratories' CryptoBytes*, 2(2):1–6, 1996.

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

[52] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A Strengthened Version of RIPEMD. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *LNCS*, pages 71–82. Springer, 1996.

[53] Orr Dunkelman and Bart Preneel. Generalizing the Herding Attack to Concatenated Hashing Schemes. Presented at the ECRYPT Hash Workshop, May 2007.

[54] Martin Feldhofer, Johannes Wolkerstorfer, and Vincent Rijmen. AES Implementation on a Grain of Sand. *IEE Proceedings on Information Security*, 152(1):13–20, 2005.

[55] William Feller. *An Introduction to Probability Theory and its Applications. Vol. I.* Third edition. John Wiley & Sons Inc., New York, 1968.

[56] Décio L. G. Filho, Paulo S. L. M. Barreto, and Vincent Rijmen. The MAELSTROM-0 Hash Function. In Symposium Proceedings of The 6th Brazilian Symposium on Information and Computer System Security, 2006. Available online at `http://www.cryptolounge.org/w/images/f/f5/Maelstrom-0.pdf`.

[57] International Organization for Standardization. ISO/IEC 9797-1:1999 Information Technology – Security Techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms Using a Block Cipher., 1999.

[58] International Organization for Standardization. ISO/IEC 10118-3:2004 Information Technology – Security Techniques – Hash-Functions – Part 3: Dedicated Hash-Functions., 2004.

[59] Pierre-Alain Fouque, Gaëtan Leurent, and Phong Q. Nguyen. Full Key-Recovery Attacks on HMAC/NMAC-MD4 and NMAC-MD5. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *LNCS*, pages 13–30. Springer, 2007.

[60] Praveen Gauravaram, William Millan, Ed Dawson, and Kapali Viswanathan. Constructing Secure Hash Functions by Enhancing Merkle-Damgård Construction. In Lynn Margaret Batten and Reihaneh Safavi-Naini, editors, *Information Security and Privacy, 11th Australasian Conference, ACISP 2006, Melbourne, Australia, July 3-5, 2006, Proceedings*, volume 4058 of *LNCS*, pages 407–420. Springer, 2006.

[61] Henri Gilbert and Helena Handschuh. Security Analysis of SHA-256 and Sisters. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 14-15, 2003, Revised Papers*, volume 3006 of *LNCS*, pages 175–193. Springer, 2003.

[62] Brian Gladman. AES and Combined Encryption/Authentication Modes. `http://fp.gladman.plus.com/AES/index.htm`.

[63] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to Construct Random Functions. *Journal of the Association for Computing Machinery (ACM)*, 33(4):792–807, 1986.

[64] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography.* Springer, 2004. ISBN 0-387-95273-X.

[65] Dan Harkins and Dave Carrel. The Internet Key Exchange (IKE), Request for Comments (RFC) 2409, The Internet Engineering Task Force, 1998.

[66] Philip Hawkes, Michael Paddon, and Gregory G. Rose. On Corrective Patterns for the SHA-2 Family. Cryptology ePrint Archive, Report 2004/207, 2004. Available online at `http://eprint.iacr.org/`.

[67] Martin E. Hellman, Ralph C. Merkle, Richard Schroeppel, Lawrence C. Washington, Whitfield Diffie, Stephen C. Pohlig, and P. Schweitzer. Results of an Initial Attempt to Cryptanalyze the NBS Data Encryption Standard. Technical Report, Stanford University, U.S.A, 1976.

[68] Shoichi Hirose. Some Plausible Constructions of Double-Block-Length Hash Functions. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *LNCS*, pages 210–225, 2006.

[69] Alireza Hodjat and Ingrid Verbauwhede. Area-Throughput Trade-Offs for Fully Pipelined 30 to 70 Gbits/s AES Processors. *IEEE Transactions on Computers*, 55(4):366–372, 2006.

[70] Deukjo Hong, Donghoon Chang, Jaechul Sung, Sangjin Lee, Seokhie Hong, Jaesang Lee, Dukjae Moon, and Sungtaek Chee. A New Dedicated 256-Bit Hash Function: FORK-256. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *LNCS*, pages 195–209. Springer, 2006.

[71] Deukjo Hong, Donghoon Chang, Jaechul Sung, Sangjin Lee, Seokhie Hong, Jesang Lee, Dukjae Moon, and Sungtaek Chee. New FORK-256. Cryptology ePrint Archive, Report 2007/185, 2007. Available online at `http://eprint.iacr.org/`.

[72] American National Standards Institution. ANSI X9.71, Keyed Hash Message Authentication Code, 2000.

[73] Daniel Joscák and Jirí Tuma. Multi-Block Collisions in Hash Functions Based on 3C and 3C+ Enhancements of the Merkle-Damgård Construction. In Min Surp Rhee and Byoungcheon Lee, editors, *Information Security and Cryptology - ICISC 2006, 9th International Conference, Busan, Korea, November 30 - December 1, 2006, Proceedings*, volume 4296 of *LNCS*, pages 257–266. Springer, 2006.

[74] Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *LNCS*, pages 306–316. Springer, 2004.

[75] Antoine Joux and Thomas Peyrin. Hash Functions and the (Amplified) Boomerang Attack. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *LNCS*, pages 244–263. Springer, 2007.

[76] Charanjit S. Jutla and Anindya C. Patthak. A Matching Lower Bound on the Minimum Weight of SHA-1 Expansion Code. Cryptology ePrint Archive, Report 2005/266, 2005. Available online at `http://eprint.iacr.org/`.

[77] Charanjit S. Jutla and Anindya C. Patthak. A Simple and Provably Good Code for SHA Message Expansion. Cryptology ePrint Archive, Report 2005/247, 2005. Available online at `http://eprint.iacr.org/`.

[78] Charanjit S. Jutla and Anindya C. Patthak. Provably Good Codes for Hash Function Design. In Eli Biham and Amr M. Youssef, editors, *Selected Areas in Cryptography, 13th International Workshop, SAC 2006, Montreal, Canada, August 17-18, 2006 Revised Selected Papers*, volume 4356 of *LNCS*, pages 376–393. Springer, 2006.

[79] John Kelsey and Tadayoshi Kohno. Herding Hash Functions and the Nostradamus Attack. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *LNCS*, pages 183–200. Springer, 2006.

[80] John Kelsey and Bruce Schneier. Second Preimages on n-bit Hash Functions for Much less than 2^n Work. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus,*

Denmark, May 22-26, 2005. Proceedings, volume 3494 of *LNCS*, pages 474–490. Springer, 2005.

[81] Joe Kilian and Phillip Rogaway. How to Protect DES Against Exhaustive Key Search. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109, pages 252–267. Springer, 1996.

[82] Joe Kilian and Phillip Rogaway. How to Protect DES Against Exhaustive Key Search (an Analysis of DESX). *Journal of Cryptology*, 14(1):17–35, 2001.

[83] Jongsung Kim, Alex Biryukov, Bart Preneel, and Seokhie Hong. On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1 (Extended Abstract). In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks, 5th International Conference, SCN 2006, Maiori, Italy, September 6-8, 2006, Proceedings*, volume 4116 of *LNCS*, pages 242–256. Springer, 2006.

[84] Paris Kitsos and Odysseas G. Koufopavlou. Efficient Architecture and Hardware Implementation of the Whirlpool Hash Function. *IEEE Transactions on Consumer Electronics*, 50(1):208–213, 2004.

[85] Lars R. Knudsen. Non-Random Properties of Reduced-Round Whirlpool. `https://www.cosic.esat.kuleuven.be/nessie/reports/phase2/uibwp5-016-2.pdf`.

[86] Lars R. Knudsen. Chosen-Text Attack on CBC-MAC. *Electronic Letters*, 33(1):48–49, 1997.

[87] Lars R. Knudsen. SMASH - A Cryptographic Hash Function. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *LNCS*, pages 228–242. Springer, 2005.

[88] Lars R. Knudsen, Xuejia Lai, and Bart Preneel. Attacks on Fast Double Block Length Hash Functions. *Journal of Cryptology*, 11(1):59–72, 1998.

[89] Lars R. Knudsen and Bart Preneel. MacDES: MAC Algorithm Based on DES. *Electronic Letters*, 34(9):871–873, 1998.

[90] Lars R. Knudsen, Christian Rechberger, and Søren S. Thomsen. The Grindahl Hash Functions. In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg City, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *LNCS*, pages 39–57, 2007.

[91] Lars R. Knudsen and Vincent Rijmen. Known-Key Distinguishers for Some Block Ciphers. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *LNCS*, pages 315–324. Springer, 2007.

[92] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication, Request for Comments (RFC) 2104, The Internet Engineering Task Force, 1997.

[93] Xuejia Lai and James L. Massey. Hash Functions Based on Block Ciphers. In Rainer A. Rueppel, editor, *Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings*, volume 658 of *LNCS*, pages 55–70. Springer, 1992.

[94] Xuejia Lai, Rainer A. Rueppel, and Jack Woollven. A Fast Cryptographic Checksum Algorithm Based on Stream Ciphers. In Jennifer Seberry and Yuliang Zheng, editors, *Advances in Cryptology - AUSCRYPT '92, Workshop on the Theory and Application of Cryptographic Techniques, Gold Coast, Queensland, Australia, December 13-16, 1992, Proceedings*, volume 718 of *LNCS*, pages 339–348. Springer, 1993.

[95] Mario Lamberger, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Second Preimages for SMASH. In Masayuki Abe, editor, *Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, San Francisco, CA, USA, February 5-9, 2007, Proceedings*, volume 4377 of *LNCS*, pages 101–111. Springer, 2007.

[96] Jeffrey S. Leon. A Probabilistic Algorithm for Computing Minimum Weights of Large Error-Correcting Codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988.

[97] Gaëtan Leurent. Message Freedom in MD4 and MD5 Collisions: Application to APOP. In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg City, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *LNCS*, pages 309–328, 2007.

[98] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and Their Applications.* Cambridge University Press; Second edition (August 26, 1994), 1994. ISBN 0-521-46094-8.

[99] Moses Liskov. Constructing an Ideal Hash Function from Weak Ideal Compression Functions. In Eli Biham and Amr M. Youssef, editors, *Selected Areas in Cryptography, 13th International Workshop, SAC 2006, Montreal, Canada, August 17-18, 2006 Revised Selected Papers*, volume 4356 of *LNCS*, pages 358–375. Springer, 2006.

[100] Jiqiang Lu, Jongsung Kim, Nathan Keller, and Orr Dunkelman. Differential and Rectangle Attacks on Reduced-Round SHACAL-1. In Rana Barua and Tanja Lange, editors, *Progress in Cryptology - INDOCRYPT 2006, 7th International Conference on Cryptology in India, Kolkata, India, December 11-13, 2006, Proceedings*, volume 4329 of *LNCS*, pages 17–31. Springer, 2006.

[101] Florence J. MacWilliams and Neil J. A. Sloane. *The Theory of Error-Correcting Codes.* North Holland, 1983. ISBN 0-444-85193-3.

[102] Stefan Mangard, Manfred J. Aigner, and Sandra Dominikus. A Highly Regular and Scalable AES Hardware Architecture. *IEEE Transactions on Computers*, 52(4):483–491, 2003.

[103] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully Attacking Masked AES Hardware Implementations. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *LNCS*, pages 157–171. Springer, 2005.

[104] Mitsuru Matsui. Linear Cryptoanalysis Method for DES Cipher. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *LNCS*, pages 386–397. Springer, 1993.

[105] Krystian Matusiewicz, Scott Contini, and Josef Pieprzyk. Weaknesses of the FORK-256 Compression Function. Cryptology ePrint Archive, Report 2006/317, 2006. Available online at `http://eprint.iacr.org/`.

[106] Krystian Matusiewicz, Thomas Peyrin, Olivier Billet, Scott Contini, and Josef Pieprzyk. Cryptanalysis of FORK-256. In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *LNCS*, pages 19–38. Springer, 2007.

[107] Krystian Matusiewicz, Josef Pieprzyk, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Analysis of Simplified Variants of SHA-256. In Christopher Wolf, Stefan Lucks, and Po-Wah Yau, editors, *WEWoRC 2005 - Western European Workshop on Research in Cryptology, July 5-7, 2005, Leuven, Belgium*, volume 74 of *LNI*, pages 123–134. GI, 2005.

[108] R. J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. The Deep Space Network Progress Report, DSN PR 42-44, January and February 1978, pages 114-116.

[109] Florian Mendel, Joseph Lano, and Bart Preneel. Cryptanalysis of Reduced Variants of the FORK-256 Hash Function. In Masayuki Abe, editor, *Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, San Francisco, CA, USA, February 5-9, 2007, Proceedings*, volume 4377 of *LNCS*, pages 85–100. Springer, 2007.

[110] Florian Mendel and Norbert Pramstaller. A New Block-Cipher-Based Hash Proposal: AESH-256. IAIK Krypto Group Technical Report 2006/09/18 (updated on 2007/10/05). `http://www.iaik.tugraz.at/research/krypto`.

[111] Florian Mendel, Norbert Pramstaller, and Christian Rechberger. Improved Collision-Attack on the Hash Function Proposed at PKC'98. In Min Surp Rhee and Byoungcheon Lee, editors, *Information Security and Cryptology - ICISC 2006, 9th International Conference, Busan, Korea, November 30 - December 1, 2006, Proceedings*, volume 4296 of *LNCS*, pages 8–21. Springer, 2006.

[112] Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Analysis of Step-Reduced SHA-256. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *LNCS*, pages 126–143. Springer, 2006.

[113] Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. On the Collision Resistance of RIPEMD-160. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings*, volume 4176 of *LNCS*, pages 101–116. Springer, 2006.

[114] Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. The Impact of Carries on the Complexity of Collision Attacks on SHA-1. In Matt Robshow, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, 2006.

[115] Florian Mendel and Vincent Rijmen. Weaknesses in the HAS-V Compression Function. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *Information Security and Cryptology - ICISC 2007, 10th International Conference, Seoul, Korea, November 29-30, 2007, Proceedings*, volume 4817 of *LNCS*, pages 335–345. Springer, 2007.

[116] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boston, 1997.

[117] Ralph C. Merkle. *Secrecy, Authentication, and Public Key Systems.* PhD thesis, Stanford University, 1979. Available online at `http://www.merkle.com/papers/Thesis1979.pdf`.

[118] Ralph C. Merkle. One Way Hash Functions and DES. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *LNCS*, pages 428–446. Springer, 1989.

[119] Judy H. Moore and Gustavus J. Simmons. Cycle Structure of the DES for Keys Having Palindromic (or Antipalindromic) Sequences of Round Keys. *IEEE Transactions on Software Engineering*, 13(2):262–273, 1987.

[120] Sean Murphy. The Cryptanalysis of FEAL-4 with 20 Chosen Plaintexts. *Journal of Cryptology*, 2(3):145–154, 1990.

[121] National Institute of Standards and Technology (NIST). FIPS-46-3: Data Encryption Standard, October 1999. Available online at `http://www.itl.nist.gov/fipspubs/`.

[122] National Institute of Standards and Technology (NIST). FIPS-197: Advanced Encryption Standard, November 2001. Available online at `http://www.itl.nist.gov/fipspubs/`.

[123] National Institute of Standards and Technology (NIST). Recommendation for Block Cipher Modes of Operation – Methods and Techniques, December 2001. Available online at `http://csrc.nist.gov/publications/nistpubs/`.

[124] NESSIE. New European Schemes for Signatures, Integrity, and Encryption. IST-1999-12324. `http://cryptonessie.org/`.

[125] K. Nishimura and M. Sibuya. Probability To Meet in the Middle. *Journal of Cryptology*, 2(1):13–22, 1990.

[126] NIST. Announcing the Development of New Hash Algorithm(s) for the Revision of Federal Information Processing Standard (FIPS) 1802, Secure Hash Standard, 2007. `http://www.csrc.nist.gov/pki/HashWorkshop/FederalRegister/Federal%20Register%20Notice%20for%20Requirements%20&%20Criteria%20-%20E7-927.pdf`.

[127] NIST. Cryptographic Hash Competition – Tentative Timeline, 2007. `http://csrc.nist.gov/groups/ST/hash/documents/HashWshop_2005_Report.pdf`.

[128] National Institute of Standards and Technology (NIST). FIPS-180-2: Secure Hash Standard, August 2002. Available online at `http://www.itl.nist.gov/fipspubs/`.

[129] National Institute of Standards and Technology (NIST). FIPS PUB 198, The Keyed-Hash Message Authentication Code (HMAC), March 2002. Available online at `http://www.itl.nist.gov/fipspubs/`.

[130] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A Side-Channel Analysis Resistant Description of the AES S-Box. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *LNCS*, pages 413–423. Springer, 2005.

[131] Nan Kyoung Park, Joon Ho Hwang, and Pil Joong Lee. HAS-V: A New Hash Function with Variable Output Length. In Douglas R. Stinson and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 14-15, 2000, Proceedings*, volume 2012 of *LNCS*, pages 202–216. Springer, 2000.

[132] Erez Petrank and Charles Rackoff. CBC MAC for Real-Time Data Sources. *Journal of Cryptology*, 13(3):315–338, 2000.

[133] Thomas Peyrin. Cryptanalysis of Grindahl. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *LNCS*, pages 551–567. Springer, 2007.

[134] Vera Pless. *Introduction to the Theory of Error-Correcting Codes.* Wiley-Interscience; 3 edition, 1998. ISBN 0-471-19047-0.

[135] Norbert Pramstaller, Frank K. Gürkaynak, Simon Häne, Hubert Kaeslin, Norbert Felber, and Wolfgang Fichtner. Towards an AES Crypto-Chip Resistant to Differential Power Analysis. In M. Steyaert and C.L. Claeys, editors, *30th European Solid-State Circuits Conference, ESSCIRC 2004, Leuven, Belgium September 21-23, 2004, Proceedings*, pages 307–310. IEEE, 2004.

[136] Norbert Pramstaller, Mario Lamberger, and Vincent Rijmen. Second Preimages for Iterated Hash Functions and their Implications on MACs. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *Information Security and Privacy - ACISP 2007, Townsville, Queensland, Australia, July 2-4, 2007, Proceedings*, volume 4586 of *LNCS*, pages 68–81. Springer, 2007.

[137] Norbert Pramstaller, Stefan Mangard, Sandra Dominikus, and Johannes Wolkerstorfer. Efficient AES Implementations on ASICs and FPGAs. In Hans Dobbertin, Vincent Rijmen, and Aleksandra Sowa, editors, *Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers*, volume 3373 of *LNCS*, pages 98–112, 2004.

[138] Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Exploiting Coding Theory for Collision Attacks on SHA-1. In Nigel P.

Smart, editor, *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, volume 3796 of *LNCS*, pages 78–95. Springer, 2005.

[139] Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Impact of Rotations in SHA-1 and Related Hash Functions. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, volume 3897 of *LNCS*, pages 261–275. Springer, 2005.

[140] Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. A Compact FPGA Implementation of the Hash Function Whirlpool. In Steven J. E. Wilton and André DeHon, editors, *Proceedings of the ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays, FPGA 2006, Monterey, California, USA, February 22-24, 2006*, pages 159–166. ACM, 2006.

[141] Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Breaking a New Hash Function Design Strategy Called SMASH. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, volume 3897 of *LNCS*, pages 233–244. Springer, 2006.

[142] Norbert Pramstaller and Johannes Wolkerstorfer. A Universal and Efficient AES Co-Processor for Field Programmable Logic Arrays. In Jürgen Becker, Marco Platzner, and Serge Vernalde, editors, *Field Programmable Logic and Application, 14th International Conference, FPL 2004, Leuven, Belgium, August 30-September 1, 2004, Proceedings*, volume 3203 of *LNCS*, pages 565–574. Springer, 2004.

[143] Bart Preneel. *Analysis and Design of Cryptographic Hash Functions.* PhD thesis, Katholieke Universiteit Leuven, 1993.

[144] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *LNCS*, pages 368–378. Springer, 1993.

[145] Bart Preneel and Paul C. van Oorschot. MDx-MAC and Building Fast MACs from Hash Functions. In Don Coppersmith, editor, *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, volume 963 of *LNCS*, pages 1–14. Springer, 1995.

[146] Bart Preneel and Paul C. van Oorschot. On the Security of Iterated Message Authentication Codes. *IEEE Transactions on Information Theory*, 45(1):188–199, 1999.

[147] Oliver Pretzel. *Error-Correcting Codes and Finite Fields.* Oxford University Press, USA; Student edition, 1996. ISBN 0-192-69067-1.

[148] Jean-Jacques Quisquater and Jean-Paul Delescaille. How Easy is Collision Search. New Results and Applications to DES. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *LNCS*, pages 408–413. Springer, 1989.

[149] Christian Rechberger and Vincent Rijmen. Note on Distinguishing, Forgery, and Second Preimage Attacks on HMAC-SHA-1 and a Method to Reduce the Key Entropy of NMAC. In *Financial Crypto 2007*, to appear in LNCS. Springer, 2007.

[150] Vincent Rijmen and Elisabeth Oswald. Update on SHA-1. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *LNCS*, pages 58–71. Springer, 2005.

[151] Vincent Rijmen and Elisabeth Oswald. Update on SHA-1. Cryptology ePrint Archive, Report 2005/010, 2005. Available online at `http://eprint.iacr.org/`.

[152] Vincent Rijmen, Bart Van Rompay, Bart Preneel, and Joos Vandewalle. Producing Collisions for PANAMA. In Mitsuru Matsui, editor, *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, volume 2355 of *LNCS*, pages 37–51, 2001.

[153] Ronald L. Rivest. The MD4 Message-Digest Algorithm, Request for Comments (RFC) 1320, The Internet Engineering Task Force, 1992.

[154] Ronald L. Rivest. The MD5 Message-Digest Algorithm, Request for Comments (RFC) 1321, The Internet Engineering Task Force, 1992.

[155] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *LNCS*, pages 371–388. Springer, 2004.

[156] Markku-Juhani O. Saarinen. A Meet-in-the-Middle Collision Attack Against the New FORK-256. In *Progress in Cryptology - INDOCRYPT*

2007, 8th International Conference on Cryptology in India, Chennai, India, December 9-13, 2007, Proceedings, to appear in LNCS. Springer, 2007.

[157] John P. Steinberger. The Collision Intractability of MDC-2 in the Ideal-Cipher Model. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007: 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007. Proceedings*, volume 4515 of *LNCS*, pages 34–51. Springer, 2007.

[158] Jacques Stern. A Method for Finding Codewords of Small Weight. In G. Cohen and J. Wolfmann, editors, *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November, 1988, Proceedings*, volume 388 of *LNCS*, pages 106–113. Springer, 1989.

[159] Marc Stevens, Arjen K. Lenstra, and Benne de Weger. Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *LNCS*, pages 1–22. Springer, 2007.

[160] Makoto Sugita, Mitsuru Kawazoe, Ludovic Perret, and Hideki Imai. Algebraic Cryptanalysis of 58-Round SHA-1. In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *LNCS*, pages 349–365. Springer, 2007.

[161] Michael Szydlo and Yiqun Lisa Yin. Collision-Resistant Usage of MD5 and SHA-1 Via Message Preprocessing. In David Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, volume 3860 of *LNCS*, pages 99–114. Springer, 2006.

[162] IAIK Crypto Toolkit. `http://jce.iaik.tugraz.at/products/core_crypto_toolkits/jca_jce`.

[163] Jacobus H. van Lint. *Introduction to Coding Theory (Graduate Texts in Mathematics)*. Springer; Third rev. and exp. ed. edition, 1998. ISBN 3-540-64133-5.

[164] Paul C. van Oorschot and Michael J. Wiener. Parallel Collision Search with Application to Hash Functions and Discrete Logarithms. In *ACM Conference on Computer and Communications Security*, pages 210–218, 1994.

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

[166] John Viega. The AHASH Mode of Operation, 2004. `http://www.cryptobarn.com/papers/ahash.pdf`.

[167] David Wagner. A Generalized Birthday Problem. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *LNCS*, pages 288–303. Springer, 2002.

[168] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Xiuyuan Yu. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199, 2004. Available online at `http://eprint.iacr.org/`.

[169] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 1–18. Springer, 2005.

[170] Xiaoyun Wang, Andrew Yao, and Frances Yao. Cryptanalysis of SHA-1. Presented at the Cryptographic Hash Workshop hosted by NIST, October 2005.

[171] Xiaoyun Wang, Andrew Yao, and Frances Yao. New Collision Search for SHA-1, August 2005. Presented at rump session of CRYPTO 2005.

[172] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.

[173] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.

[174] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *LNCS*, pages 1–16. Springer, 2005.

[175] Hirotaka Yoshida and Alex Biryukov. Analysis of a SHA-256 Variant. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, volume 3897 of *LNCS*, pages 245–260. Springer, 2005.

[176] Hirotaka Yoshida, Dai Watanabe, Katsuyuki Okeya, Jun Kitahara, Hongjun Wu, Özgül Küçük, and Bart Preneel. MAME: A Compression Function with Reduced Hardware Requirements. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *LNCS*, pages 148–165. Springer, 2007.

[177] Hongbo Yu, Gaoli Wang, Guoyan Zhang, and Xiaoyun Wang. The Second-Preimage Attack on MD4. In Yvo Desmedt, Huaxiong Wang, Yi Mu, and Yongqing Li, editors, *Cryptology and Network Security, 4th International Conference, CANS 2005, Xiamen, China, December 14-16, 2005, Proceedings*, volume 3810 of *LNCS*, pages 1–12. Springer, 2005.

[178] Gideon Yuval. How to Swindle Rabin. *Cryptologia*, 3(3):187–189, 1979.

List of Publications

In Refereed Conference Proceedings

1. Norbert Pramstaller, Mario Lamberger, and Vincent Rijmen, Second Preimages for Iterated Hash Functions and their Implications on MACs. *In Proceedings of Information Security and Privacy - ACISP 2007, Townsville, Queensland, Australia, July 2-4, 2007, pp. 68-81, LNCS 4586*

2. Steve Babbage, Carlos Cid, Norbert Pramstaller, and Håvard Raddum, An Analysis of the Hermes8 Stream Ciphers. *In Proceedings of Information Security and Privacy - ACISP 2007, Townsville, Queensland, Australia, July 2-4, 2007, pp. 1-10, LNCS 4586*

3. Mario Lamberger, Norbert Pramstaller, Christian Rechberger and Vincent Rijmen, Second Preimages for SMASH. *In Proceedings of Topics in Cryptology - CT-RSA 2007, San Francisco, USA, February 5-9, 2007, pp. 101-111, LNCS 4377*

4. Florian Mendel, Norbert Pramstaller, and Christian Rechberger, Improved Collision-Attack on the Hash Function Proposed at PKC'98. *In Proceedings of Information Security and Cryptology - ICISC 2006, Busan, Korea, November 30 - December 1, 2006, pp. 8-21, LNCS 4296*

5. Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, On the Collision Resistance of RIPEMD-160. *In Proceedings of Information Security - ISC 2006, Samos, Greece, August 30 - September 2, 2006, pp. 101-116, LNCS 4176*

6. Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, The Impact of Carries on the Complexity of Collision Attacks on SHA-1. *In Proceedings of Fast Software Encryption - FSE 2006, Graz, Austria, March 15-17, 2006, pp. 278-292, LNCS 4047*

7. Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, Analysis of Step-Reduced SHA-256. *In Proceedings of Fast Software Encryption - FSE 2006, Graz, Austria, March 15-17, 2006, pp 126-143, LNCS 4047*

8. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, A Compact FPGA Implementation of the Hash Function Whirlpool. *In Proceedings of 14th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA 2006, Hyatt Regency Monterey, Monterey, California, February 22-24, 2006, pp. 159-166, ACM Press ISBN 1-59593-292-5*

9. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, Breaking a New Hash Function Design Strategy Called SMASH. *In Proceedings of 12th Annual Workshop on Selected Areas in Cryptography - SAC2005,*

Queen's University, Kingston, Ontario, Canada, August 11-12, 2005, pp. 233-244, LNCS 3897

10. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, Impact of Rotations in SHA-1 and Related Hash Functions. *In Proceedings of 12th Annual Workshop on Selected Areas in Cryptography - SAC2005, Queen's University, Kingston, Ontario, Canada, August 11-12, 2005, pp. 261-275, LNCS 3897*

11. Krystian Matusiewicz, Josef Pieprzyk, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, Analysis of simplified variants of SHA-256. *In Western European Workshop on Research in Cryptology - WEWoRC 2005, LNI P-74, Gesellschaft für Informatik, 2005*

12. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, Exploiting Coding Theory for Collision Attacks on SHA-1. *In Proceedings of 10th IMA International Conference on Cryptography and Coding, Royal Agricultural College, Cirencester, UK , December 19-21, 2005, pp 78-95, LNCS 3796*

13. Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald, Successfully Attacking Masked AES Hardware Implementations. *In Proceedings of Workshop on Cryptographic Hardware and Embedded Systems - CHES 2005, Edinburgh, Scotland, August 29 - September 1, 2005, pp. 157-171, LNCS 3659*

14. Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen, A Side-Channel Resistant Description of the AES S-box. *In Proceedings of Fast Software Encryption - FSE 2005, Paris, France, February 21-23, 2005, pp. 413-423, LNCS 3557*

15. Norbert Pramstaller, Elisabeth Oswald, Stefan Mangard, Frank K. Gürkaynak, and Simon Häne, A Masked AES ASIC Implementation. *In Proceedings of Austrochip 2004, pp.77-81, ISBN 3-200-00211-5, Villach, Austria, October 8th 2004*

16. Norbert Pramstaller and Manfred Aigner, A Universal and Efficient SHA-256 Implementation for FPGAs. *In Proceedings of Austrochip 2004, pp.89-93, ISBN 3-200-00211-5, Villach, Austria, October 8th 2004*

17. Norbert Pramstaller, Frank K. Gürkaynak, Simon Häne, Hubert Kaeslin, Norbert Felber, and Wolfgang Fichtner, Towards an AES Crypto-chip Resistant to Differential Power Analysis. *In Proceedings of European Solid-State Circuits Conference - ESSCIRC 2004, Leuven, September 21-23, 2004, pp. 307-310, ISBN 0-7803-8480-6*

18. Norbert Pramstaller, Stefan Mangard, Sandra Dominikus, and Johannes Wolkerstorfer, Efficient AES Implementations on ASICs and FPGAs. *In Proceedings of Fourth Conference on the Advanced Encryption Standard - AES 2004, Bonn, May, 2004, pp.98-112, LNCS 3373*

19. Norbert Pramstaller and Johannes Wolkerstorfer, A Universal and Efficient AES Implementation for Filed Programmable Logic Arrays. *In Proceedings of Field-Programmable Logic and Applications - FPL 2004, Antwerp, August 30 - September 2, 2004, pp. 565-574, LNCS 3203*

20. Martin Feldhofer, Michael Gross, Johann Großschädl, Thomas Popp, Norbert Pramstaller, Christian Pühringer, Karl Scheibelhofer, Alexander Szekely, Stefan Tillich, and Karl C. Posch, Rapid Prototyping of a SPARC-V8-based Firewall-on-Chip. *In Proceedings of Austrochip 2003, pp. 4145, ISBN 3-200-00021-X, Linz, Austria, October 3rd, 2003*

21. Norbert Pramstaller, Johannes Wolkerstorfer, An Efficient AES Implementation for Re-configurable Devices. *In Proceedings of Austrochip 2003, pp. 58, ISBN 3-200-00021-X, Linz, Austria, October 3rd, 2003*

Contributions to Workshops Without Proceedings

1. Steve Babbage, Carlos Cid, Norbert Pramstaller, and Håvard Raddum, Cryptanalysis of Hermes8F, *SASC The State of The Art of Stream Ciphers, January 31 - February 1, Bochum, Germany, 2007*

2. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, Breaking a new Hash Function Design Strategy called SMASH. *Workshop Hash Functions, Przegorzaly (Krakow), June 23-24, Poland, 2005*

3. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, On Variable Bit-Rotations in SHA-1-like Hash Functions. *Workshop Hash Functions, Przegorzaly (Krakow), June 23-24, Poland, 2005*

4. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, Preliminary Analysis of the SHA-256 Message Expansion . *Western European Workshop on Research in Cryptography - WEWoRC, July 05-07, Leuven, Belgium, 2005*

5. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, An Efficient FPGA Implementation of Whirlpool. *Western European Workshop on Research in Cryptography - WEWoRC, July 05-07, Leuven, Belgium, 2005*

Preprints

1. Steve Babbage, Carlos Cid, Norbert Pramstaller, and Håvard Raddum, Cryptanalysis of Hermes8F, *eSTREAM, ECRYPT Stream Cipher Project, Report 2006/048 and Cryptology ePrint Archive, Report 2006/269*

2. Mario Lamberger, Norbert Pramstaller, and Vincent Rijmen, Second Preimages for Iterated Hash Functions Based on a b-Block Bypass, *Cryptology ePrint Archive, Report 2006/116*

3. IAIK Krypto Group, Preliminary Analysis of DHA-256, *Cryptology ePrint Archive, Report 2005/398*

Invited Book Chapters

1. Vincent Rijmen and Norbert Pramstaller, Chapter 6, Cryptographic Algorithms in Constrained Environments, *in Handbook of Wireless Security: From Specifications to Implementations, edited by N. Sklavos and X. Zhang, CRC Press, ISBN: 9780849387715, 2007*

Curriculum Vitae

Norbert Pramstaller was born in Brunico (Italy), on June 29, 1978. From 1998 to 2004 he studied Telematics at Graz University of Technology, where he obtained the Master degree with distinction. His master thesis was about DPA resistant AES ASIC implementations and was performed at ETH Zurich during the winter term 2003/2004. From June 2004 to December 2004, he was working as researcher in the VLSI and security group at IAIK TU Graz. In January 2005, he started to work towards his PhD degree with Prof. Vincent Rijmen at the IAIK Krypto Group. Since then, he is scientific assistant at Graz University of Technology.