Maria Eichlseder

Differential Cryptanalysis of Symmetric Primitives

PhD Thesis

Supervised by Florian Mendel and Christian Rechberger

SCIENCE · PASSION · TECHNOLOGY

TU Graz

Maria Eichlseder

Differential Cryptanalysis of Symmetric Primitives

Doctoral Thesis

to achieve the university degree of
Doktorin der technischen Wissenschaften

submitted to
Graz University of Technology

Advisor: Florian Mendel

Assessors: Christian Rechberger
Graz University of Technology

Joan Daemen
Radboud University Nijmegen

Institute of Applied Information Processing and Communications
Graz University of Technology

Graz, March 2018

Abstract

We cryptanalyze several symmetric encryption and hashing algorithms. A central factor in the security of symmetric cryptographic algorithms is the resistance of their core building block, the primitive, against cryptanalytic attacks such as differential, linear, and algebraic cryptanalysis. The fundamental idea of differential cryptanalysis is to extract secret information or forge malicious messages by investigating the behavior of the primitive for two related, slightly different inputs, and has proven both very powerful and highly versatile since its inception in the 1990s. Resistance against such attacks is thus one of the cornerstones in the design of block ciphers.

More recently, alternative symmetric primitives have risen to general attention: Permutations and tweakable block ciphers in particular have shown the potential to rival block ciphers in their role as the ideal primitive for efficient and elegant schemes. However, the available cryptanalytic tools and theory on the design and analysis of these alternative primitives are arguably less mature than for block ciphers.

We investigate the security of these primitives against differential cryptanalysis. Compared to classic block ciphers, adversaries who target permutations or tweakable block ciphers can take advantage of known, chosen, or related round-key material. We find that in some cases, the designers' block-cipher-based design strategies do not sufficiently protect against variations of the classical differential strategy. In particular, we break the full security claims of the tweakable block cipher MANTIS-5 and the permutation Simpira v1. We provide a key recovery attack for the round-reduced block cipher LowMC and analyze the authenticated cipher Prøst in a related-key setting. We also develop techniques to improve the computer-aided differential analysis of unkeyed primitives, leading to the best practical collision attacks on the round-reduced hash standard SHA-2.

Acknowledgements

It is my pleasure to thank everyone who helped make this thesis possible.

First and foremost, I would like to thank my advisors Florian Mendel and Christian Rechberger for their support throughout my PhD studies. Florian, thank you for your guidance when I began this quest, and for providing me with the opportunity to collaborate on many interesting questions in the past years. Your cryptanalytic intuition is inspiring. Thanks for your patience, encouragement, and the occasional board gaming evening. Christian, thank you for your support, for introducing me to interesting new design challenges, and giving me the freedom to pursue my research interests. Also thanks to Stefan Mangard for the enjoyable time in the SESYS group.

I am grateful to Joan Daemen for agreeing to assess my thesis and coming to Graz to join the defense committee, as well as valuable comments and interesting discussions at past conferences.

Thanks to all my coauthors for giving me the opportunity to collaborate on many interesting ideas. In particular, I owe thanks to Florian Mendel, Christoph Dobraunig, Martin Schläffer, Daniel Kales, Ange Albertini, Jean-Philippe Aumasson, Tomislav Nad, and Vincent Rijmen, with whom I collaborated on papers discussed in this thesis. Thank you for many fruitful discussions and not losing your nerves during the usual five-minutes-to-midnight submissions. I also want to thank Christian Rechberger, Lorenzo Grassi, Stefan Mangard, Thomas Korak, Robert Primas, Thomas Unterluggauer, Andrey Bogdanov, Virginie Lallemand, Martin M. Lauridsen, Gregor Leander, Eik List, Victor Lomné, Elmar Tischhauser, and Andreas Hülsing & the Sphincs+ team. Thanks for widening my horizon.

To Florian, Christian, Christoph, and Daniel, thanks for taking the time to provide feedback on drafts of this thesis, and for your valuable comments.

My time at IAIK would not have been as enjoyable without my former and current colleagues. Thanks for an amazing time. Christoph, I really enjoyed working with you – thank you for both thoughtful discussions and entertaining coffee breaks. Thanks to Martin Schläffer, Mario Lamberger, and Elisabeth Oswald for introducing me to cryptography by supervising my master's thesis, bachelor's thesis, and student internships, respectively.

I would like to (again!) express my thanks to the Ascon team, Christoph, Florian, and Martin, for great ideas, inspiring discussions, and enjoyable cryptanalysis sessions both before and after the CAESAR deadline. Special thanks to Hannes Groß for a veritable zoo of efficient implementations.

Finally, I want to thank my fiancé Daniel Gruß, my family, and my friends for all the good times shared throughout the years. Danke Mama, Papa, und Michi für eure Unterstützung. Thank you, Daniel – for everything.

Maria Eichlseder
Graz, March 2018

Contents

List of Figures

List of Tables

1 Introduction

Differential cryptanalysis is one of the central techniques for analyzing the security of symmetric cryptographic algorithms such as block ciphers. The fundamental idea to extract secret information or forge malicious messages by investigating the behavior of the algorithm for two related, slightly different inputs has proven both very powerful and highly versatile. Resistance against such differential attacks has been one of the cornerstones in the design of block ciphers since the first published attack in 1990.

The art of designing and using block ciphers has long been the primary focus of symmetric cryptography. However, more recently, alternative symmetric primitives have risen to general attention: Permutations and tweakable block ciphers in particular have shown the potential to rival block ciphers in their role as the ideal primitive for efficient and elegant schemes. However, the available theory on the design and analysis of these alternative primitives is arguably less mature than for block ciphers.

In this thesis, we analyze the security of several recently designed primitives against variants of differential cryptanalysis. We show that several of them can be broken in spite of the designers' claim of resistance against differential cryptanalysis. In addition, we develop techniques to improve the computer-aided differential analysis of unkeyed primitives.

This work lead to contributions in 18 peer-reviewed publications and several informal publications, of which 8 are partially or completely covered in this thesis.

In this chapter, we provide an introduction to the main goals and results of this thesis. Section 1.1 introduces the general context to motivate the goals and contributions. We discuss the individual contributions in more detail and provide an outline of the thesis in Section 1.2. Finally, we provide a list of all co-authored publications.

1.1. Motivation

Symmetric cryptography is concerned with solutions for the classic application scenario of information security that we encounter ubiquitously in communication networks and information storage systems: A small “team of insiders” (typically the two notorious secret-mongers Alice and Bob) wants to protect their communication from the wide world of potentially malicious outsiders. For this purpose, they use a symmetric cryptographic scheme to process their plaintext communication data before transmission. The insiders share similar preconditions; that is, they may share access to the same secret keys, and they cooperate towards a mutual goal.

Two types of symmetric schemes cover the lion’s share of applications: hash functions and authenticated ciphers. Hash functions map a message of arbitrary length to a short, fixed-size fingerprint, which can serve as placeholder, identifier, or commitment of this message. Authenticated ciphers map a message to an authenticated ciphertext for transmission based on a shared secret key, which protects the message’s confidentiality (no outsider can derive information about the message) and authenticity (no outsider can manipulate the communicated data without detection).

Under the hood, these schemes operate a central primitive, classically a block cipher such as AES. The scheme’s security is rooted in the security of this primitive, which is operated by both communication partners in similar ways and with the same secret key material.

This is in contrast to asymmetric cryptography, where the two communicating parties have fundamentally different roles, which manifests in different key material and interfaces. This allows the definition of complex and subtle interfaces, roles, and security goals, and there seems to be an inexhaustible stream of newly emerging types of cryptographic primitives and schemes in this field. For symmetric cryptography, on the other hand, the functional requirements for schemes like (authenticated) encryption schemes and hash functions, as well as the underlying primitives and basic operations, have remained comparably stable for years and decades. Thus, it is valid to ask why there is still demand for new designs, why symmetric cryptography is not “solved and done”. In the following, we discuss some factors and challenges that drive the ongoing evolution of symmetric cryptography.

1.1.1. Challenges

Security. The security of symmetric schemes and primitives is evaluated through cryptanalysis, and is as such never finally asserted. Every year, new attack angles, new techniques and methods, new scenarios are proposed. Even for designs like AES, which have been extensively studied for 20 years, new insights and observations are still being made [GRR17].

In addition to algorithmic advances, it may become necessary to consider stronger attackers, and adapt both security parameters and security notions accordingly. For example, networks are now easily fast enough to breach the birthday bound for 64-bit block ciphers like 3DES in everyday applications [BL16]. Attack scenarios previously presumed moot, such as related-key and known-key attacks, can become more relevant with the rise of new constructions, such as permutations and tweakable block ciphers constructed from block ciphers [JNP14a]. Further requirements and questions arise when considering very-long-term security, for example, related to post-quantum security.

Efficiency. While the attackers' computational capabilities continuously increase, the same cannot necessarily be said for users' available resources. Cryptographic implementations are increasingly deployed in previously unprotected applications, where individual resource parameters can be strictly constrained (energy and power, code and state area, latency and throughput, bandwidth and format, randomness and statefulness, . . .). Additionally, novel application scenarios and implementation security requirements may introduce new efficiency metrics that determine, for example, the computational overhead for side-channel resistance of the implementation, or the cost in very specialized circuit frameworks [ARS+15].

Usability. The practical suitability of a cryptographic scheme is often not only (or not even primarily) a question of efficiency, but of usability and developer-friendliness. Usability aspects include, but are not limited to, availability of secure implementations with simple, suitable, high-level interfaces, clear documentation and user guidance, and robustness in the face of sub-optimal implementations or misuse.

For example, the most prominent standards in symmetric cryptography have previously focused on relatively low-level primitives, in particular, block ciphers. The higher-level constructions, such as modes of operation and their generic compositions, were treated more as second thoughts, and thus sometimes designed and used less carefully than the primitives. More recently, the situation has however improved considerably. Protocols like TLS v1.3 and standardization efforts are beginning to consider higher-level schemes such as authenticated ciphers, rather than primitives, as the lowest level of modularity. Moreover, higher-level schemes are increasingly being designed with a focus on clear, useful interfaces in terms of both functionality (such as intermediate tags) and robustness (such as clear guidance on nonce use and decryption, or discussion of side-channels).

Transparency. Besides its technical challenges, cryptography is also often a balancing act between political, commercial, and public interests. This raises the question who can be trusted to design such schemes, and who can be trusted to evaluate how reliable the security expectations are. The question has become increasingly more pressing with the rise of internet services for every imaginable purpose. For most of its history, and up to the late 1960s, cryptography was the almost exclusive domain of military and governmental research. The early 1970s saw the first demands for cryptography for public, commercial applications, such as securing banking or pay-TV connections. From internal documents that surfaced only much later [Joh09], it can be seen that the NSA was rather opposed to this idea, and feared that "a competent industry standard could spread into undesirable areas, like Third World government". Nevertheless, they finally acceded to the standardization of IBM's DES algorithm, though not without modifications, which caused some heated reactions. Even today, many commercial applications concoct their own, secret crypto schemes, rather than relying on publicly verifiable schemes.

Cryptographic competitions aim to increase the trust in symmetric schemes by focusing the academic community's efforts on the transparent design and analysis of particularly central schemes. The most prominent examples of this approach are the AES and SHA-3 competitions organized by NIST, but also some European efforts, like NESSIE and eStream. The most recent incarnation is the CAESAR competition, with the goal of identifying efficient and interesting new authenticated encryption algorithms.

Curiosity. Last but far from least, many developments are simply results of curiosity. Criteria in this context include elegance, provability, minimalism, or feasibility.

1.1.2. Directions

Novel Primitives. Block ciphers have long been the workhorses of information security, and their design and proper use the central research topic of symmetric cryptography. They seem to be the perfect primitive to securely process data in reasonably-sized portions (the block size of typically 128 bits) with a reasonably-sized security level (the key size of typically 128 bits).

However, when we consider a mode of operation to turn a block cipher into an (authenticated) encryption scheme or a hash function, it soon becomes apparent that there is one piece of information that does not quite seem to fit: the context. For encryption, this context defines the meaning of a particular chunk of plaintext by positioning it in space and time, for example by its memory address or message number. For authentication purposes, the context additionally corresponds to the accumulated information so far. An adversary must not be able to subvert this context, for example by repositioning or dropping plaintext blocks, or by observing equal plaintext chunks in different contexts. Modes of operation have to jump through hoops in order to squeeze this extra bit of information into the limited block-cipher interface.

Two alternative approaches have surfaced in the past years. The first is to explicitly account for the context with an additional input parameter to the block cipher which "tweaks" the resulting mapping. In other words, the tweak and the key together select the permutation that maps the plaintext to the ciphertext and vice-versa. The resulting primitive is termed a tweakable block cipher, and can be constructed either as a dedicated primitive or in a generic way from a block cipher or other building blocks. This approach is particularly intuitive for pure encryption schemes, but has also been shown to be useful for authentication [Rog04a].

The second approach is to drop all barriers between key, context, and plaintext information, and use a public permutation with a sufficiently large block size to accumulate this information implicitly. The resulting modes share aspects not only with block cipher modes like CBC, but also with classical stream cipher and hashing modes. Such permutation-based designs work most naturally for hashing and authentication purposes, as illustrated by the keyed and unkeyed sponge constructions, but have also proven popular for (authenticated) encryption [BDPV07; BDPV11b].

The adoption of alternative primitives is connected with several of the previously discussed challenges: It partially answers some of the questions, but raises different new ones. In terms of efficiency, different improvements are possible: both permutation-based and tweak-based constructions may have fewer calls to the primitive and may require a lower effective state size than classic block-cipher constructions. Tweak-based constructions appear particularly suitable for parallelized implementations and random-access encryption, whereas permutation-based constructions seem naturally suited for high-throughput serialized processing of data; they also scale well for different security levels and state size constraints.

In terms of usability, several schemes based on alternative primitives introduce flexible, clean and stream-compatible interfaces. The permutation-based schemes of the extended sponge family in particular have shown remarkable flexibility with cleanly integrated features such as sessions with intermediate tags, alternating authenticate-only and authenticated-encryption inputs, and more. Finally, many classical block cipher modes suffer from restrictive query limits due to indexing limits or due to the quadratic security loss with an increasing number and length of queries with respect to the message block size (the "birthday bound"). This can be circumvented in tweak-based and permutation-based schemes by expanding relevant parts of the internal state with the tweak or context.

For the security analysis, on the other hand, alternative primitives come with new challenges. On a higher level, this includes the definition of proper security models for the primitives that are both realistic and useful for proofs, particularly for the keyless permutations. The positive side is that these modes are usually comparably clean and elegant and thus lend themselves well to both intuitive and rigorous security arguments. On a lower level, the design of primitives is a delicate task that has so far been mostly studied with respect to block ciphers. Permutations are often characterized as block ciphers without a key (schedule). However, this is not necessarily a constructive approach for designing permutations, for several reasons: Permutation-based modes require permutations with significantly larger input sizes than block ciphers and may thus favor different word sizes and operations; the known-key security of block ciphers is not particularly well-studied, since many classical attacks target key recovery and assume a "randomizing" effect in every round due to the key; and distinguishing attacks may use inside-out computations or similar techniques. Similarly, the analysis of dedicated tweakable block ciphers can be compared to the related-key analysis of block ciphers.

Tools for Analysis. Designing symmetric primitives and schemes is a complex and often laborious task and relies on extensive cryptanalysis efforts both before and after publishing the result. Cryptographic competitions have emerged as a popular means to foster such efforts in a focused way, to identify the most promising candidates, and to increase the general confidence in their security. An organizing body such as the United States National Institute of Standards and Technology (NIST) or ECRYPT, a Network of Excellence funded by the European Union, announces such a public competition for a particular type of cryptographic schemes, and the international cryptographic research community can both propose candidate ciphers and collectively evaluate them over a few years. The most prominent examples of this approach are the AES (1997–2000) and SHA-3 (2007–2012) competitions organized by NIST, but also European efforts like NESSIE (2000–2003) and eSTREAM (2004–2008). The most recent incarnation is the CAESAR competition (2014–ongoing), with the goal of identifying efficient and interesting authenticated encryption algorithms. The result is then either a single winner that is subsequently standardized in the NIST competitions, or a portfolio of recommendations.

The success of these competitions depends crucially on a thorough analysis of all competitors. With the increasing number of candidates in the more recent competitions (such as the 57 first-round submissions to CAESAR), this has become more and more of a challenge. Hardly any of the candidates will receive as much manual attention as the purportedly more than a dozen person-years invested in the design of DES, though this is hard to measure for community-driven analysis. The complex definition of some of the candidates, coupled with design approaches that are less than optimally suited for manual cryptanalysis, have made it necessary to leave a significant amount of effort to (semi-)automated cryptanalysis tools. They can serve as both heuristic indicators (e.g., finding easily verifiable differential characteristics, but without any guarantee that these are the strongest possible solutions), and perform a sort of exhaustive proofs for certain categories of cryptanalysis – anything that is computationally expensive, but can be easily defined in terms of a search or optimization problem, and can then be solved either with existing tools like satisfiability (SAT) and mixed-integer linear programming (MILP) solvers or with dedicated tools using techniques from related fields. Popular applications include differential cryptanalysis, linear cryptanalysis, the division property, and more. Alternative primitives are a particularly interesting target where manual or straightforward approaches often fail for reasons such as large state sizes, missing or related round keys, and weakly aligned operations.

1.2. Contributions

Outline

The contributions of this thesis can be grouped into two main parts.

In Part I (Chapters 3, 4, 5, 6), we cryptanalyze several recently published designs of primitives and schemes: the tweakable block cipher MANTIS, the permutation Simpira, the block cipher LowMC, and the permutation-based authenticated cipher Prøst. We show that the security margin of the first three primitives against variants of differential cryptanalysis is smaller than expected, including full breaks of the designers' security claims for Simpira-4 v1 and MANTIS-5. Our analysis highlights the challenges of designing unkeyed, tweakable, or otherwise atypical novel primitives. Among others, we exploit properties caused by dependencies in the unkeyed rounds of a permutation, the tweak schedule, or unkeyed inner rounds.

In Part II (Chapter 7), we target primitives that are very hard to analyze by hand by improving and applying tools for automated cryptanalysis. We extend a dedicated search tool for differential characteristics with several techniques to improve the efficiency of the search for larger primitives. We apply the tool to find practical semi-free-start collisions for the round-reduced hash standard SHA-512 and its truncated variants. Additionally, we show how the tool can be used to insert collision backdoors in malicious variants of SHA-1 with a few modified round constants.

We introduce the relevant background, notation, and other preliminaries for these two parts in Chapter 2, and conclude in Chapter 8.

In the following, we give a more detailed outline of the main part of this thesis. We summarize the individual contributions of each part and the associated publications.

In addition to the work presented in this document, we also contributed to other lines of research in the area of cryptanalysis and implementation security for symmetric cryptography. In particular, we had the opportunity to join the Ascon team in designing the authenticated cipher Ascon currently competing as a finalist in the CAESAR competition, which connects with several topics of this thesis.

We provide a list of all 18 co-authored peer-reviewed publications and some informal reports at the end of this chapter.

Part I: Differential Cryptanalysis of Novel Designs

In Chapter 3, we analyze the tweakable block cipher MANTIS, published at CRYPTO 2016 [BJK+16]. We develop an approach for clustering a large number of differential characteristics in a related-tweak setting. The approach is inspired by truncated characteristics, but by applying a more fine-grained analysis that takes the linear tweak schedule into consideration, we obtain much higher estimated differential probabilities than either standard or truncated differential cryptanalysis. With this approach, we can show that MANTIS' lightweight round function does not interact well with its α-reflective construction and tweakey schedule. We also identify some additional differential properties of the S-box that impact the analysis. We derive an attack on MANTIS-5 using about 2^{30} chosen plaintexts to recover the full 128-bit key in less than 2^{38} computational complexity, or about an hour in practice, which is less than 2^{96} as claimed by the designers. The attack was published at FSE 2017 (Tokyo) and selected among the best three papers; as part of subsequent follow-up work, we further refined and generalized the analysis:

- C. Dobraunig, M. Eichlseder, D. Kales, and F. Mendel. Practical Key-Recovery Attack on MANTIS5. In: IACR Transactions on Symmetric Cryptology 2016.2 (2017), pp. 248–260. DOI: `10.13154/tosc.v2016.i2.248-260`. IACR: `2016/754`.
- M. Eichlseder and D. Kales. Clustering Related-Tweak Characteristics: Application to MANTIS-6. IACR Cryptology ePrint Archive, Report 2017/1136. 2017. IACR: `2017/1136`.

In Chapter 4, we analyze the permutation Simpira v1 [GM16a]. We show that the designers' computer-aided security analysis does not take into consideration several dependencies between intermediate variables of the permutation, which invalidates their conclusions on the maximum probability of differential characteristics based on the minimum number of 75 differentially active AES S-boxes. With about 2^{110} computational complexity, we find differential fixed-points for the full-round 512-bit permutation Simpira-4, corresponding to collisions after a feed-forward. Our result illustrates how unpredictable the effective security margin can be for unkeyed primitives due to dependencies and the attacker's capability of controlling intermediate values. The attacks violate the designers' security claims that there are no structural distinguishers below 2^{128}. In response, the designers published an updated Simpira v2 at ASIACRYPT 2016 [GM16b]. Our attack was published at SAC 2016 (St John's):

🗋 C. Dobraunig, M. Eichlseder, and F. Mendel. Cryptanalysis of Simpira v1. In: Selected Areas in Cryptography – SAC 2016. Ed. by R. Avanzi and H. M. Heys. Vol. 10532. LNCS. Springer, 2016, pp. 284–298. DOI: `10.1007/978-3-319-69453-5_16`. IACR: `2016/244`.

In Chapter 5, we analyze the block cipher LowMC published at EUROCRYPT 2015 [ARS+15]. The goal of LowMC's designers is to have a particularly low multiplicative complexity, measured in the total number and longest chain length of binary multiplications in the encryption circuit. This is achieved with incomplete S-box layers and dense, unstructured linear layers. We exploit these properties in a higher-order differential attack, and show how to extend the basic zero-sum distinguishers such as discussed by the designers by up to 4 rounds (out of a 5-round security margin). To achieve this result, we construct and chain different invariant subspaces of the S-box layer, and optimize the key recovery with FFT summation. We propose round-reduced attacks for different recommended parameter sets, such as 10 out of 11 or 12 rounds of the 128-bit security variant LowMC-$128^{512,86}$ with about 2^{67} complexity. The designers have since proposed an updated version LowMC v2 [ARS+16]. Our attack was published at ICISC 2015 (Seoul):

🗋 C. Dobraunig, M. Eichlseder, and F. Mendel. Higher-Order Cryptanalysis of LowMC. In: Information Security and Cryptology – ICISC 2015. Ed. by S. Kwon and A. Yun. Vol. 9558. LNCS. Springer, 2015, pp. 87–101. DOI: `10.1007/978-3-319-30840-1_6`. IACR: `2015/407`.

In Chapter 6, we propose a related-key forgery attack on the permutation-based CAESAR Round-1 candidate Prøst-OTR [KLL+14]. We showed how for this mode, observing any two messages encrypted under any two related keys with some known xor difference allows an attacker to forge a tag for a third message, with negligible complexity. This property is an effect of the combination of different secure constructions (Even-Mansour cipher in OTR mode of operation), and is completely independent of the Prøst permutation. The observation is unexpected but does not threaten the security of Prøst-OTR in the single-key setting. Our result was published at FSE 2015 (Istanbul):

🗋 C. Dobraunig, M. Eichlseder, and F. Mendel. Related-Key Forgeries for Prøst-OTR. In: Fast Software Encryption – FSE 2015. Ed. by G. Leander. Vol. 9054. LNCS. Springer, 2015, pp. 282–296. DOI: `10.1007/978-3-662-48116-5_14`. IACR: `2015/091`.

Part II: Automating Differential Cryptanalysis

In Chapter 7, we improve and apply a practical collision search tool for the round-reduced hash standard SHA-2, with a special focus on SHA-512.

This work builds on an existing dedicated heuristic search tool developed for SHA-256 by Mendel, Nad, and Schläffer [MNS11b; MNS13b], based on an earlier tool for SHA-1 by De Cannière and Rechberger [DR06]. This tool takes as input the specification of a compression function or other primitive, a specification of initial constraints, and a specification of search heuristics. Based on these bitwise specifications and constraints, the tool successively finds a compatible and consistent assignment of differences or values for all variables required in the respective phase. The search algorithm itself is an instance of the guess-and-determine approach widely used in combinatorial constraint solvers, such as SAT solvers: It repeatedly picks an undetermined variable, assigns a tentative value ("guess"), and applies a form of constraint propagation to derive the implications of this assignment, as well as potential contradictions ("determine"). Mendel, Nad, and Schläffer [MNS13b] successfully applied this tool to find semi-free-start collisions for up to 38 out of 64 rounds ("steps") of SHA-256.

We adapted several aspects of the search tool in order to improve the performance for primitives with larger state sizes, such as those of SHA-512 and SHA-3/Keccak. Two particular problems when applying the previous search algorithm to larger primitives are related to detecting contradictions soon enough: First, the search algorithm has to select and guess the critical bits that reveal the contradiction within a much larger search space; and second, it has to determine the resulting contradiction in a larger circuit. We developed, implemented, and evaluated several techniques to address these two problems. The first is a look-ahead branching heuristic that aims to identify critical bits by peeking at the implications of several candidate bits and picking the most impactful bit. The second is an approach for efficiently propagating partial information through large linear layers. Together, they significantly decrease the time spent in "dead ends" of the search space.

This allows tackling SHA-512 collision challenges with comparable round numbers as SHA-256. The best results include semi-free-start collisions for up to 39 out of 80 steps of SHA-512. We also show how to extend these results to even more steps of the truncated, wide-pipe variants SHA-512/t. Finally, we explore how this tool can be used not only by a cryptanalyst, but by a malicious designer who wishes to include a collision backdoor in a malicious SHA-1 variant with tweaked round constants.

This part includes contributions published as parts of papers at WCC 2013 (Bergen), FSE 2014 (London), SAC 2014 (Montréal), and ASIACRYPT 2015 (Auckland), as well as a technical report for the CRYPTREC project, an evaluation and recommendation effort driven by the National Institute of Information and Communication Technology (NICT) in Japan:

- M. Eichlseder, F. Mendel, T. Nad, V. Rijmen, and M. Schläffer. Linear Propagation in Efficient Guess-and-Determine Attacks. In: International Workshop on Coding and Cryptography – WCC 2013, Preproceedings. Ed. by L. Budaghyan, T. Helleseth, and M. G. Parker. 2013, pp. 193–202. ISBN: 978-82-308-2269-2. URL: `http://www.selmer.uib.no/WCC2013/`.
- M. Eichlseder, F. Mendel, and M. Schläffer. Branching Heuristics in Differential Collision Search with Applications to SHA-512. In: Fast Software Encryption – FSE 2014. Ed. by C. Cid and C. Rechberger. Vol. 8540. LNCS. Springer, 2014, pp. 473–488. DOI: `10.1007/978-3-662-46706-0_24`. IACR: `2014/302`.
- A. Albertini, J.-P. Aumasson, M. Eichlseder, F. Mendel, and M. Schläffer. Malicious Hashing: Eve's Variant of SHA-1. In: Selected Areas in Cryptography – SAC 2014. Ed. by A. Joux and A. M. Youssef. Vol. 8781. LNCS. Springer, 2014, pp. 1–19. DOI: `10.1007/978-3-319-13051-4_1`. IACR: `2014/694`.
- C. Dobraunig, M. Eichlseder, and F. Mendel. Analysis of SHA-512/224 and SHA-512/256. In: Advances in Cryptology – ASIACRYPT 2015. Ed. by T. Iwata and J. H. Cheon. Vol. 9453. LNCS. Springer, 2015, pp. 612–630. DOI: `10.1007/978-3-662-48800-3_25`. IACR: `2016/374`.
- C. Dobraunig, M. Eichlseder, and F. Mendel. Security Evaluation of SHA-224, SHA-512/224, and SHA-512/256. Tech. Report CRYPTREC. 2015. URL: `http://www.cryptrec.go.jp/estimation/techrep_id2401.pdf`.

Publication List

Journal Articles

TOSC'17	C. Dobraunig, M. Eichlseder, S. Mangard, F. Mendel, and T. Unterluggauer. ISAP – Towards Side-Channel Secure Authenticated Encryption. In: IACR Transactions on Symmetric Cryptology 2017.1 (2017), pp. 80–105. DOI: `10.13154/tosc.v2017.i1.80-105`. IACR: `2016/952`.
TOSC'16	C. Dobraunig, M. Eichlseder, D. Kales, and F. Mendel. Practical Key-Recovery Attack on MANTIS5. In: IACR Transactions on Symmetric Cryptology 2016.2 (2017), pp. 248–260. DOI: `10.13154/tosc.v2016.i2.248-260`. IACR: `2016/754`.

Conference Proceedings

COSADE'17	C. Dobraunig, M. Eichlseder, T. Korak, and F. Mendel. Side-Channel Analysis of Keymill. In: Constructive Side-Channel Analysis and Secure Design – COSADE 2017. Ed. by S. Guilley. Vol. 10348. LNCS. Springer, 2017, pp. 138–152. DOI: `10.1007/978-3-319-64647-3_9`. IACR: `2016/793`.
ASIACRYPT'16	C. Dobraunig, M. Eichlseder, T. Korak, V. Lomné, and F. Mendel. Statistical Fault Attacks on Nonce-Based Authenticated Encryption Schemes. In: Advances in Cryptology – ASIACRYPT 2016. Ed. by J. H. Cheon and T. Takagi. Vol. 10031. LNCS. Springer, 2016, pp. 369–395. DOI: `10.1007/978-3-662-53887-6_14`. IACR: `2016/616`.
SAC'16	C. Dobraunig, M. Eichlseder, and F. Mendel. Cryptanalysis of Simpira v1. In: Selected Areas in Cryptography – SAC 2016. Ed. by R. Avanzi and H. M. Heys. Vol. 10532. LNCS. Springer, 2016, pp. 284–298. DOI: `10.1007/978-3-319-69453-5_16`. IACR: `2016/244`.

Conference Proceedings

ACNS'16	C. Dobraunig, M. Eichlseder, and F. Mendel. Square Attack on 7-Round Kiasu-BC. In: Applied Cryptography and Network Security – ACNS 2016. Ed. by M. Manulis, A.-R. Sadeghi, and S. Schneider. Vol. 9696. LNCS. Springer, 2016, pp. 500–517. DOI: 10.1007/978-3-319-39555-5_27. IACR: 2016/326.
FSE'16	C. Dobraunig, M. Eichlseder, and F. Mendel. Analysis of the Kupyna-256 Hash Function. In: Fast Software Encryption – FSE 2016. Ed. by T. Peyrin. Vol. 9783. LNCS. Springer, 2016, pp. 575–590. DOI: 10.1007/978-3-662-52993-5_29. IACR: 2015/956.
ICISC'15	C. Dobraunig, M. Eichlseder, and F. Mendel. Higher-Order Cryptanalysis of LowMC. In: Information Security and Cryptology – ICISC 2015. Ed. by S. Kwon and A. Yun. Vol. 9558. LNCS. Springer, 2015, pp. 87–101. DOI: 10.1007/978-3-319-30840-1_6. IACR: 2015/407.
ASIACRYPT'15	C. Dobraunig, M. Eichlseder, and F. Mendel. Analysis of SHA-512/224 and SHA-512/256. In: Advances in Cryptology – ASIACRYPT 2015. Ed. by T. Iwata and J. H. Cheon. Vol. 9453. LNCS. Springer, 2015, pp. 612–630. DOI: 10.1007/978-3-662-48800-3_25. IACR: 2016/374.
ASIACRYPT'15	C. Dobraunig, M. Eichlseder, and F. Mendel. Heuristic Tool for Linear Cryptanalysis with Applications to CAESAR Candidates. In: Advances in Cryptology – ASIACRYPT 2015. Ed. by T. Iwata and J. H. Cheon. Vol. 9453. LNCS. Springer, 2015, pp. 490–509. DOI: 10.1007/978-3-662-48800-3_20. IACR: 2015/1200.
CT-RSA'15	C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. Cryptanalysis of Ascon. In: Topics in Cryptology – CT-RSA 2015. Ed. by K. Nyberg. Vol. 9048. LNCS. Springer, 2015, pp. 371–387. DOI: 10.1007/978-3-319-16715-2_20. IACR: 2015/030.
FSE'15	C. Dobraunig, M. Eichlseder, and F. Mendel. Related-Key Forgeries for Prøst-OTR. In: Fast Software Encryption – FSE 2015. Ed. by G. Leander. Vol. 9054. LNCS. Springer, 2015, pp. 282–296. DOI: 10.1007/978-3-662-48116-5_14. IACR: 2015/091.

SAC'15	C. Dobraunig, M. Eichlseder, and F. Mendel. Forgery Attacks on Round-Reduced ICEPOLE-128. In: Selected Areas in Cryptography – SAC 2015. Ed. by O. Dunkelman and L. Keliher. Vol. 9566. LNCS. Springer, 2015, pp. 479–492. DOI: `10.1007/978-3-319-31301-6_27`. IACR: `2015/392`.
CARDIS'14	C. Dobraunig, M. Eichlseder, S. Mangard, and F. Mendel. On the Security of Fresh Re-keying to Counteract Side-Channel and Fault Attacks. In: Smart Card Research and Advanced Applications – CARDIS 2014. Ed. by M. Joye and A. Moradi. Vol. 8968. LNCS. Springer, 2014, pp. 233–244. DOI: `10.1007/978-3-319-16763-3_14`. IACR: `2015/033`.
SAC'14	A. Albertini, J.-P. Aumasson, M. Eichlseder, F. Mendel, and M. Schläffer. Malicious Hashing: Eve's Variant of SHA-1. In: Selected Areas in Cryptography – SAC 2014. Ed. by A. Joux and A. M. Youssef. Vol. 8781. LNCS. Springer, 2014, pp. 1–19. DOI: `10.1007/978-3-319-13051-4_1`. IACR: `2014/694`.
LATINCRYPT'14	A. Bogdanov, C. Dobraunig, M. Eichlseder, M. M. Lauridsen, F. Mendel, M. Schläffer, and E. Tischhauser. Key Recovery Attacks on Recent Authenticated Ciphers. In: Progress in Cryptology – LATINCRYPT 2014. Ed. by D. F. Aranha and A. Menezes. Vol. 8895. LNCS. Springer, 2014, pp. 274–287. DOI: `10.1007/978-3-319-16295-9_15`.
FSE'14	M. Eichlseder, F. Mendel, and M. Schläffer. Branching Heuristics in Differential Collision Search with Applications to SHA-512. In: Fast Software Encryption – FSE 2014. Ed. by C. Cid and C. Rechberger. Vol. 8540. LNCS. Springer, 2014, pp. 473–488. DOI: `10.1007/978-3-662-46706-0_24`. IACR: `2014/302`.
WCC'13	M. Eichlseder, F. Mendel, T. Nad, V. Rijmen, and M. Schläffer. Linear Propagation in Efficient Guess-and-Determine Attacks. In: International Workshop on Coding and Cryptography – WCC 2013, Preproceedings. Ed. by L. Budaghyan, T. Helleseth, and M. G. Parker. 2013, pp. 193–202. ISBN: 978-82-308-2269-2. URL: `http://www.selmer.uib.no/WCC2013/`.

Miscellaneous

EPRINT'18	C. Dobraunig, M. Eichlseder, L. Grassi, V. Lallemand, G. Leander, E. List, F. Mendel, and C. Rechberger. Rasta: A cipher with low ANDdepth and few ANDs per bit. IACR Cryptology ePrint Archive, Report 2018/181. 2018. IACR: `2018/181`.
EPRINT'18	C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas. Exploiting Ineffective Fault Inductions on Symmetric Cryptography. IACR Cryptology ePrint Archive, Report 2018/071. 2018. IACR: `2018/071`.
EPRINT'17	D. Kales, M. Eichlseder, and F. Mendel. Note on the Robustness of CAESAR Candidates. IACR Cryptology ePrint Archive, Report 2017/1137. 2017. IACR: `2017/1137`.
EPRINT'17	M. Eichlseder and D. Kales. Clustering Related-Tweak Characteristics: Application to MANTIS-6. IACR Cryptology ePrint Archive, Report 2017/1136. 2017. IACR: `2017/1136`.
PQ'17	D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, and P. Schwabe. SPHINCS+. Submission to NIST's Post-Quantum Crypto Project (Round 1). See `https://sphincs.org/`. 2017. URL: `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/SPHINCS_Plus.zip`.
CAESAR'16	C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. Ascon v1.2. Submission to CAESAR: Competition for Authenticated Encryption. Security, Applicability, and Robustness (Round 3). See `http://ascon.iaik.tugraz.at/`. 2016. URL: `http://competitions.cr.yp.to/round3/asconv12.pdf`.
CRYPTREC'15	C. Dobraunig, M. Eichlseder, and F. Mendel. Security Evaluation of SHA-224, SHA-512/224, and SHA-512/256. Tech. Report CRYPTREC. 2015. URL: `http://www.cryptrec.go.jp/estimation/techrep_id2401.pdf`.

2 Preliminaries

In this chapter, we provide the necessary background in symmetric cryptography and differential cryptanalysis for the main part of this thesis.

In the first half of the chapter, we present a brief anatomy of symmetric cryptographic schemes and their functional goals. To achieve security goals such as confidentiality and authenticity of communicated messages, *cryptographic schemes* transform the plaintext messages into an encrypted ciphertext and/or append an authentication tag. Since messages can be (almost) arbitrarily long, cryptographic schemes usually chop the message down into blocks, which are iteratively processed by *cryptographic primitives* such as block ciphers or compression functions. Internally, primitives repeat a high number of very basic mathematic *operations*. We give a bottom-up overview of these different construction levels.

In the second half of this chapter, we introduce methods to analyze the security of symmetric cryptographic schemes. Since most results in this thesis are based on ideas from and generalizations of *differential cryptanalysis*, this is our main focus. Additionally, we cover some related terminology and results, such as higher-order differentials, which will star in guest appearances later.

Both parts together provide the foundations for the main part of this thesis, and introduce the necessary terminology and concepts. We assume that the reader is somewhat familiar with mathematical concepts from abstract algebra and probability theory, such as finite fields or discrete probability distributions. For a general introduction to the design and analysis of symmetric primitives and an overview of the state-of-the-art in block cipher design, we refer the interested reader to books such as Daemen and Rijmen's [DR02], Knudsen and Robshaw's [KR11a], or Avanzi's [Ava16].

2.1. A Brief Anatomy of Symmetric Cryptology

In this first half of the chapter, we introduce the ingredients necessary to build symmetric cryptographic schemes bottom-up. In Section 2.1.1, we start with the elementary operations and their most basic properties, which as such are not sufficient yet to define any kind of cryptographic security properties. In Section 2.1.2, we define the smallest building blocks that do provide a quantifiable security level, but are not yet directly practically applicable: cryptographic primitives such as block ciphers or compression functions. Finally, in Section 2.1.3, we provide a brief overview of cryptographic schemes for variable-length data, such as hash functions and authenticated ciphers, and their security notions.

2.1.1. Basic Operations

Symmetric and asymmetric cryptography differ significantly not only in their key-distribution setting, but also in how they achieve secure cryptographic primitives. Whereas asymmetric schemes typically rely on the difficulty of solving a concise, high-level mathematical problem, such as the discrete logarithm problem in a very large finite group, symmetric schemes do the opposite. They decompose the inputs into very small units and apply a large number of small operations over small mathematical structures, such as byte-sized finite fields, register-sized finite groups, or even individual bits – usually a mix of several structures. The security is then derived from the large network of small operations, whose interaction prevents the successful application of statistical analysis.

Shannon [Sha49] identified the two main goals of the operations as *confusion* (making the relation between secret key and published ciphertext as complex as possible) and *diffusion* (spreading the effects of each input bit over the whole output). Shannon's definitions do not exactly reflect modern concepts of cryptographic security – for instance, he puts considerable emphasis on the inherent redundancy within the plaintexts – and have thus been re-interpreted in several different ways. They are often mapped to the design of modern ciphers as follows. The first goal, confusion, is achieved primarily with local, nonlinear operations, such as S-boxes or bit-sliced Boolean functions. The latter goal, diffusion, can be achieved with larger, but usually linear operations, such as bit permutations or multiplications with a constant matrix. In the following, we briefly introduce the two most popular families of operations, whose exact boundaries are blurry.

Register-oriented operations, ARX: Software-oriented schemes such as the MD5/SHA hash family [Dan12] split the data blocks into the most natural size for their target platform: the CPU register size b, e.g., 32 bits. These register words serve as operands for various operations:

(A) **Modular Addition** $x \boxplus y$: Adds two registers as integers mod 2^b.

(R) **Rotate** $x \ggg r$**, Shift** $x \gg r$: Permutes the register's bits by a cyclic right/left rotation or shift by r bit positions.

(X) **Xor** $x \oplus y$, **And** $x \wedge y$, etc.: Applies Boolean operation to each bit position in two input registers to compute one output register.

Here, **(A,R)** can contribute to diffusion within register words and **(X)** across words; **(A)** (or nonlinear bitwise operations) contribute to confusion.

The following Boolean functions and the corresponding word-oriented, bitwise Boolean function versions will be used in the following chapters. For a b-bit word $x \in \{0,1\}^b$, we denote by x_i the i-th coordinate starting from the least significant bit (LSB) x_0 to the most significant bit (MSB) x_{b-1}. We also write (x_i) or $(x_i)_i$ for $x = (x_i)_{i=0,\dots,b-1}$, and identify x with the corresponding integer $x = \sum_{i=0}^{b-1} x_i 2^i \in \mathbb{Z}_{2^b}$.

- Unary and associative binary bitwise Boolean functions $z_i = f(x_i, y_i)$: $\neg x_i$ (not), $x_i \wedge y_i$ (and), $x_i \vee y_i$ (or), $x_i \oplus y_i$ (xor).
- Ternary bitwise Boolean functions with $z_i = f(w_i, x_i, y_i)$:

$$z = \mathsf{eq}(w,x,y): \quad z_i = (\neg w_i \oplus x_i) \wedge (\neg w_i \oplus y_i)\,, \tag{eq}$$
$$z = \mathsf{if}(w,x,y): \quad z_i = (w_i \wedge x_i) \oplus (\neg w_i \wedge y_i)\,, \tag{if}$$
$$z = \mathsf{maj}(w,x,y): \quad z_i = (w_i \wedge x_i) \oplus (w_i \wedge y_i) \oplus (x_i \wedge y_i)\,. \tag{maj}$$

- Unary rotation $\lll r, \ggg r$ and shift $\ll r, \gg r$ functions $z = f_r(x)$:

$$x_{\lll r}: \quad z_i = x_{i-r} \pmod b, \qquad x_{\ggg r}: \quad z_i = x_{i+r} \pmod b, \tag{rot}$$
$$x_{\ll r}: \quad z_i = \begin{cases} x_{i-r} & i \geq r, \\ 0 & \text{else}, \end{cases} \qquad x_{\gg r}: \quad z_i = \begin{cases} x_{i+r} & i < b-r, \\ 0 & \text{else}. \end{cases} \tag{sh}$$

- Modular addition $z = x \boxplus y$ in $\mathbb{Z}_{2^b}$ with carry c:

$$c = \mathsf{carry}(x,y): \quad c_i = \begin{cases} \mathsf{maj}(x_{i-1}, y_{i-1}, c_{i-1}) & i \geq 1\,, \\ 0 & i = 0\,, \end{cases} \tag{carry}$$
$$z = x \boxplus y: \quad z_i = x_i \oplus y_i \oplus c_i\,. \tag{sum}$$

Besides a symbolic representation of Boolean functions as above, we also use representations in truth tables and algebraic normal forms (ANF). The ANF is a canonical representation of Boolean functions $f(v_1, \dots, v_n)$ that writes the output bit as a sum (xor) of distinct products (and) of distinct input bits, or in other words, as a multivariate polynomial in $\mathbb{F}_2[v_1, \dots, v_n]/(v_1^2 \oplus v_1, \dots, v_n^2 \oplus v_n)$, with terms 'monomial', 'coefficient', 'degree', etc., defined accordingly [Gég27]. The coefficients of the ANF can be efficiently computed from the value vector of the function's truth table (and vice versa) via the binary Möbius transform.

Modern schemes of this type range from pure ARX designs like ChaCha [Ber08] ($\boxplus, \ggg, \oplus$) and generalized ARX designs like SHA-2 [Dan12] (ARX plus nonlinearity from bitwise Boolean $\wedge$) to more hardware-friendly, 'purely Boolean' designs [AJN16] (without $\boxplus$).

When such designs include vectorial, purely Boolean nonlinear functions with multiple input and output bits (coordinates) and thus define bit-sliced S-boxes, they cross the boundary towards SPN designs.

Cell-oriented operations, SPN: The term substitution-permutation network (SPN) originally referred to a construction that alternates between a bricklayer cell-substitution layer (S-box) for confusion and a bit permutation between cells for diffusion (P-box), but is now variably used in a more general sense to include other linear P-boxes, such as strongly aligned linear layers as in AES.

Compared to ARX designs, SPNs tend to have a clearer separation of duties between the individual building blocks: An S-box ensures that local changes at its input cause "confusing" effects at the output; and a P-box diffuses these effects to the inputs of the S-boxes of the next round. The desired effect – that flipping a single input bit causes (essentially unpredictable) changes in each output bit – has been classically quantified by properties like completeness [KD79] and the (strict) avalanche criterion [Fei73; WT85]. These definitions already carry a foretaste of the ideas of differential cryptanalysis. Kam and Davida [KD79] also propose a general approach for constructing bit permutations that achieve completeness after one or several rounds: Tree-Structured SPNs. Later designs gradually introduced P-boxes using binary linear operations over $\mathbb{F}_{2^b}$ or $\mathbb{Z}_{2^b}$ for b-bit cells [Mas93], and started representing the P-box as a matrix-vector multiplication over some ring. Perfect diffusion is achieved if the matrix is MDS or, more generally, a multipermutation [Vau94; Dae95].

For quantifying the confusion provided by S-boxes or by their component functions, a number of different metrics have been proposed. These include the algebraic degree of the component functions' ANFs, their nonlinearity that measures the number of times that the best affine approximation over $\mathbb{F}_2$ produces a different output, and more.

The most notable and most widely re-used approach is the design of the block cipher Rijndael/AES by Daemen and Rijmen [DR98; DR02; DBN+01], which splits each block S into a 4×4 matrix of byte-sized cells (bundles) $S = (s_{i,j})$, identified with field elements in $\mathbb{F}_{2^8}$. All operations are aligned to these cells:

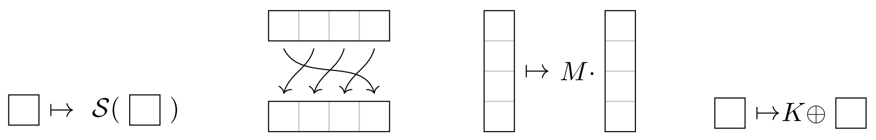
$$S = \begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} .$$

(S) **AddRoundKey** adds the round key $K^{(r)}$ of round r to the state, **SubBytes** substitutes each byte element $s_{i,j}$ with the output of the same nonlinear function, the S-box $\mathcal{S}$:

$$\forall\, i, j : \quad s_{i,j} \mapsto s_{i,j} \oplus K^{(r)}_{i,j}, \qquad \text{(AddRoundKey)}$$
$$\forall\, i, j : \quad s_{i,j} \mapsto \mathcal{S}(s_{i,j}). \qquad \text{(SubBytes)}$$

(P) **ShiftRows** permutes the four elements $s_{i,\cdot}$ of each matrix row i with a cyclic left-shift by i positions to achieve inter-column diffusion, **MixColumns** multiplies the MDS matrix M over $\mathbb{F}_{2^8}^4$ to (each column j of) the state matrix to achieve intra-column diffusion:

$$\forall\, i : \quad (s_{i,j})_j \mapsto (s_{i,j})_j \lll i = (s_{i,j+i \bmod 4})_j, \qquad \text{(ShiftRows)}$$
$$\forall\, j : \quad (s_{i,j})_i \mapsto M \cdot (s_{i,j})_i. \qquad \text{(MixColumns)}$$



(a) SubBytes (b) ShiftRows (c) MixColumns (d) AddRoundKey

Figure 2.1.: AES' cell-oriented operations.

2.1.2. Cryptographic Primitives

Cryptographic primitives are the smallest building blocks of a cryptographic system that already have meaningful cryptographic claims for a specified security level. Like the basic operations, they operate on one or more fixed-size inputs. However, their size is not defined by implementation requirements, but rather by the targeted security level. Typical security claims refer to the hardness of recovering certain inputs to the primitive given its other inputs and outputs, or the hardness of distinguishing the primitive from an idealized primitive, and quantify the minimum workload to solve these problems.

Note that this notion of cryptographic primitives differs somewhat from some textbooks and introductions, which often include some schemes with arbitrary input sizes, like hash functions or MACs, in the list of primitives, but inexplicably exclude others, such as encryption modes. This does not reflect the fact that most cryptographic schemes in use for arbitrary-sized inputs are essentially modes of operation for one of the primitives discussed below, and often reduce their security claims to claims for this primitive.

This thesis focuses on the following four symmetric primitives.

Block ciphers: A b-bit block cipher with k-bit key is a function

$$\begin{aligned} E : \mathbb{F}_2^k \times \mathbb{F}_2^b &\to \mathbb{F}_2^b, \\ K, M &\mapsto C, \end{aligned}$$

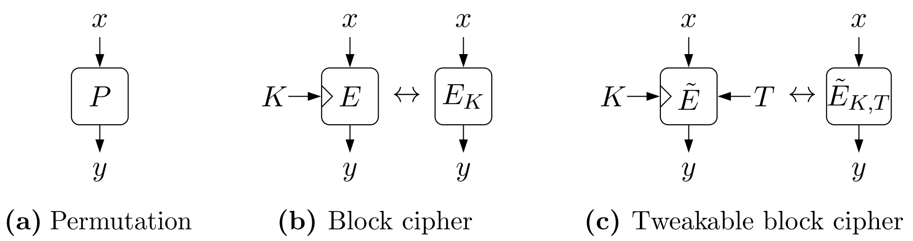
called the encryption function, which maps k-bit key K and b-bit plaintext block M to a b-bit ciphertext block C such that $E_K = E(K, \cdot)$ is a family of efficiently computable permutations (Figure 2.2). Its inverse is the decryption function $E_K^{-1} = D_K$ such that $D_K(E_K(x)) = x$ for all $x \in \mathbb{F}_2^b$:

$$\begin{aligned} D : \mathbb{F}_2^k \times \mathbb{F}_2^b &\to \mathbb{F}_2^b, \\ K, C &\mapsto M. \end{aligned}$$

Practically speaking, the block cipher protects the confidentiality of both the plaintext M (for unknown key K) and of the key K (even if many plaintext-ciphertext pairs $C_i = E_K(M_i)$ are known): An adversary must not be able to learn any function of either the plaintext or the key with success probability higher than random guessing and computational effort significantly lower than 2^s operations for some specified *security level* of s bits. Here, s is at most equal to the key size in bits, k, due to the generic approach of exhaustively testing all 2^k possible keys.

Most cryptanalytic attacks on block ciphers aim to recover (parts of) the fixed secret key K. They differ in the data resources that they grant the adversary: These range from *known ciphertexts* only, over quantities of *known plaintext-ciphertext* pairs, to allowing the adversary *adaptively chosen queries* to encryption and/or decryption *oracles*. In addition, some applications use block ciphers in ways not covered by the standard fixed-key security notion, which motivates alternative security notions. Most prominently, *related-key attacks* give the adversary additional query access to oracles for different keys with a known or chosen relation to the secret target key. Thus, different attacks can be compared based on their computational complexity, data or query complexity (qualitatively and quantitatively), and their objective, such as key recovery.

When block ciphers are employed as primitives in a cryptographic scheme, their assumed security needs to be phrased in a more concise way that serves as assumption for the security proof. The *standard model* for block ciphers is the pseudo-random permutation (PRP) assumption: to assume that for any fixed, but unknown key K that has been randomly selected, the permutation E_K is *indistinguishable* from a permutation truly randomly selected from the set of all $(2^b)!$ permutations. In this notion, the adversary can make queries to E_K (and D_K) for the fixed, unknown key, within the complexity limits defined by the security level. However, in many constructions, this model is not useful since the value K used for E_K is not such a fixed, unknown value. Then, a stronger notion such as the *ideal cipher model* [Sha49] is necessary to argue the security of the scheme: It models E_K for different keys as independent pseudo-random permutations. In other words, $E(\cdot,\cdot)$ is assumed to be picked uniformly at random from among all $(2^b!)^{2^k}$ block ciphers. This sounds intuitive, but many practical ciphers are easily distinguishable in this setting; furthermore, any practical instantiation can essentially void security proofs in this model [Bla06].



(a) Permutation **(b)** Block cipher **(c)** Tweakable block cipher

Figure 2.2.: Symmetric cryptographic primitives as permutation families.

Tweakable block ciphers: A b-bit tweakable block cipher with k-bit key and t-bit tweak is a function

$$\begin{aligned}\tilde{E} : \mathbb{F}_2^k \times \mathbb{F}_2^t \times \mathbb{F}_2^b &\to \mathbb{F}_2^b, \\ K, T, M &\mapsto C,\end{aligned}$$

which maps a k-bit key K, a t-bit public tweak T, and a b-bit plaintext M to a b-bit ciphertext C such that $\tilde{E}_{K,T} = \tilde{E}(K, T, \cdot)$ is a family of efficiently computable permutations (Figure 2.2). Its inverse is the decryption function $\tilde{E}_{K,T}^{-1} = \tilde{D}_{K,T}$.

Tweakable block ciphers generalize the concept of block ciphers by adding an additional public input, the tweak. This tweak plays a role similar to the nonces or initialization values of higher-level modes of operation, and provides additional variation of the individual instances of the cipher family. The concept was formally introduced by Liskov et al. [LRW02; LRW11], who argue that it is more naturally suited as a building block for higher-level modes of operation than block ciphers. Some designs combine the key and tweak into a $k + t$-bit "tweakey" that can flexibly be part-secret, part-public [JNP14b]. The security notions are similar to block ciphers, except for the tweak T, which is generally under full control of the adversary.

Cryptographic permutations: A cryptographic b-bit permutation P is a bijective function

$$\begin{aligned}P : \mathbb{F}_2^b &\to \mathbb{F}_2^b, \\ x &\mapsto y,\end{aligned}$$

such that P (and, if required, its inverse P^{-1}) is easy to evaluate.

Due to the lack of a key, it is not easy to define a clear security notion for unkeyed cryptographic permutations. To be considered secure for cryptographic applications, P must not permit any structural distinguishers. This includes any properties that are not expected for a randomly chosen permutation. In particular, P must resist cryptanalysis approaches as applied to block ciphers, such as linear and differential analysis. A general property of particular relevance for permutations is the hardness of constrained-input constrained-output (CICO) problems defined by Bertoni et al. [BDPV11a]: Given two sets $\mathcal{X} \subseteq \mathbb{F}_2^b$ and $\mathcal{Y} \subseteq \mathbb{F}_2^b$, it should be no easier than for generic permutations to find an input-output pair (x, y) such that $y = P(x)$ and $x \in \mathcal{X}, y \in \mathcal{Y}$.

Compression functions: A t-bit (or $b+t$-to-t-bit) compression function is an efficiently computable function

$$\begin{aligned} F : \mathbb{F}_2^t \times \mathbb{F}_2^b &\to \mathbb{F}_2^t, \\ H_{i-1}, M_i &\mapsto H_i, \end{aligned}$$

which maps a t-bit input chaining value H_{i-1} and b-bit message block M_i to a t-bit output chaining value H_i.

F is generally required to be a *one-way function*: Given a fixed t-bit output chaining value H_i, it should be infeasible to find a *preimage*, that is, an input chaining value H_{i-1} and message M_i such that $F(H_{i-1}, M_i) = H_i$. Additionally, given a solution (H_{i-1}, M_i), it should be infeasible to find a *second preimage* $(H'_{i-1}, M'_i) \neq (H_{i-1}, M_i)$ such that $F(H'_{i-1}, M'_i) = F(H_{i-1}, M_i) = H_i$. The generic complexity for finding a preimage or second preimage for a t-bit compression function is about 2^t.

In addition, F is often required to be *collision-resistant*: It should be infeasible to find a *collision*, that is, a pair of inputs $(H_{i-1}, M_i) \neq (H'_{i-1}, M'_i)$ such that $F(H_{i-1}, M_i) = F(H'_{i-1}, M'_i)$. The generic complexity for finding a collision for a t-bit compression function is about $\sqrt{2^t} = 2^{t/2}$ due to the birthday paradox [Yuv79]. More specifically, the probability that there is a collision among q random queries can be approximated by $1 - \exp(-q^2/2^{t+1})$ [FO89], as illustrated in Figure 2.3.

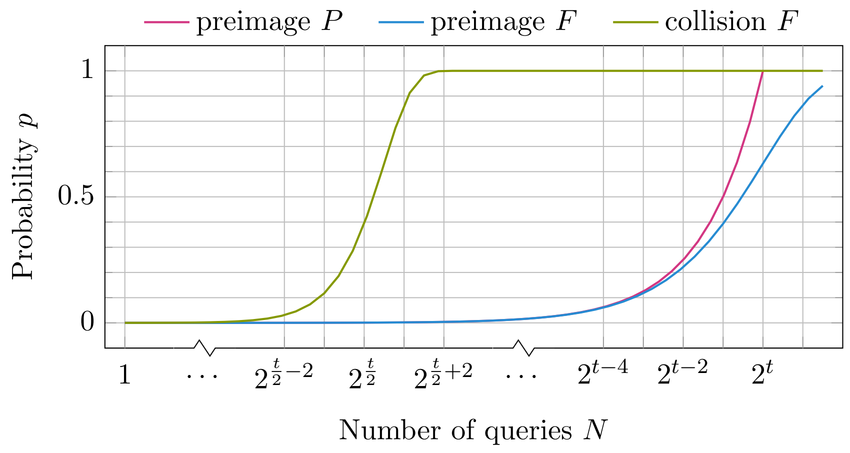


Figure 2.3.: Probability p of a collision or a preimage among N different random queries to an ideal t-bit compression function F or t-bit permutation P.

Constructing primitives. Since cryptographic primitives are the smallest building blocks that are associated with a cryptographic security level, they need to be constructed by assembling basic operations. This assembly needs to be both simple enough to allow efficient descriptions and implementations, but also complicated enough to ensure diffusion and confusion. To achieve both goals, primitives define a reasonably simple transformation, the *round function*, and iteratively repeat this round function reasonably often with only minor variations. The result is what Shannon [Sha49] calls a *product cipher*, combining weak components to build a strong whole. The concept and design of round functions for block-based primitives builds on the work of Feistel [Fei70; Fei73; FNS75]. The primary challenge in symmetric cryptography is to make qualified statements about the security of a function assembled this way.

Each application of the round function $\mathcal{R}_i$ updates an internal *state* of the primitive from its previous value X_{i-1} to a new value X_i. This state is initialized with an initial state X_0, containing (some of) the inputs of the primitive, such as the plaintext, while (part of) the final output X_r after r rounds serves as output of the primitive.

For invertible, keyed primitives like block ciphers, the state is usually further subdivided into a round-key part K_i that is initialized with the key K and updated by a *key schedule* function $\mathcal{R}_i^K(\cdot)$ and a data part X_i that is initialized with the plaintext or permutation input and updated by a keyed round permutation $\mathcal{R}_i^P(\cdot, K_i)$ for $1 \leq i \leq r$:

$$K_0 = K, \qquad K_i = \mathcal{R}_i^K(K_{i-1}), \qquad \text{(Key schedule)}$$

$$X_0 = M, \qquad X_i = \mathcal{R}_i^P(X_{i-1}, K_i). \qquad \text{(Round function)}$$

For tweakable block ciphers, the key schedule K_0 may be initialized with both the key K and the tweak T [JNP14b] or a derived value, or the state X_0 may be initialized with a masked (whitened) plaintext [Rog04a]. For permutations, the key schedule is reduced to a constant schedule.

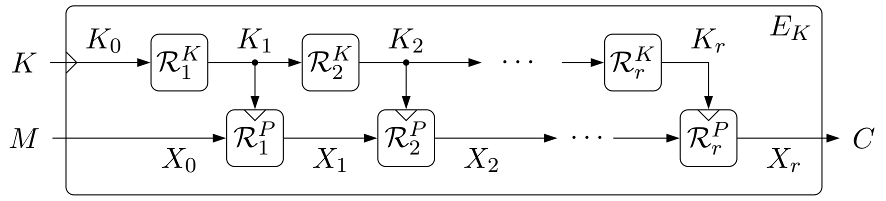
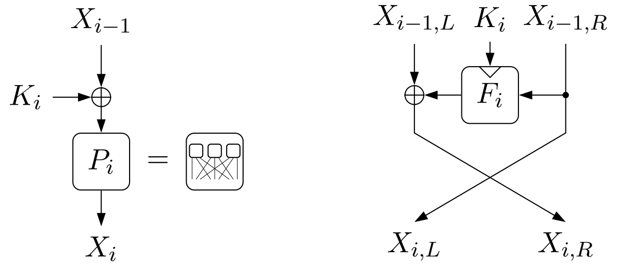


Figure 2.4.: Constructing primitives by iterating a keyed round function.

Two of the most useful constructions for invertible, keyed primitives from round functions are illustrated in Figure 2.5. They represent two different ideas to construct efficiently invertible round functions, which are necessary in many constructions using invertible primitives.

The first is the *key-alternating construction* with a round permutation P_i. It updates the state with a round key addition of K_i and a bijective transformation P_i. If the inverse of P_i needs to be efficiently implementable, P_i is typically an SPN that applies several copies of a small nonlinear substitution transformation in parallel for confusion and a (sparse) linear layer for diffusion. For decryption, each layer needs to be individually inverted, and all layers applied in reverse order.

An alternative construction that avoids inverting layers is the *Feistel network*, analyzed theoretically by Luby and Rackoff [LR88], and popularized by the block cipher DES [US 77]. It applies only a half-sized, key-dependent, not necessarily bijective function F_i to one half of the state and adds the result to the other half, before swapping halves. Since both the addition step (using xor) and the swap are involutions, it is easy to invert the (iterated) round transformation by applying the same operations in reverse order. For other addition operations, such as modular addition, the addition step is no longer involutive, but inversion is similarly simple using subtraction. In case $F_i(X, K) = \tilde{F}_i(X \oplus K)$, this can be seen as a special case of the key-alternating construction. The Feistel construction can be generalized in several ways for mixed addition operations (as in many ARX designs), for more than two branches, including parallel addition steps, expanding and contracting F-functions, and more.



(a) Key-alternating SPN **(b)** Feistel network

Figure 2.5.: Constructing invertible, keyed round functions $\mathcal{R}_i^P$.

Besides building primitives from scratch by iterating weak round functions, there are also a number of generic constructions to build primitives from other primitives. Ideally, this allows a security reduction to assert the security of the new primitive under the assumption that the old primitive is secure. However, such security reductions unfortunately often come with a considerable security loss.

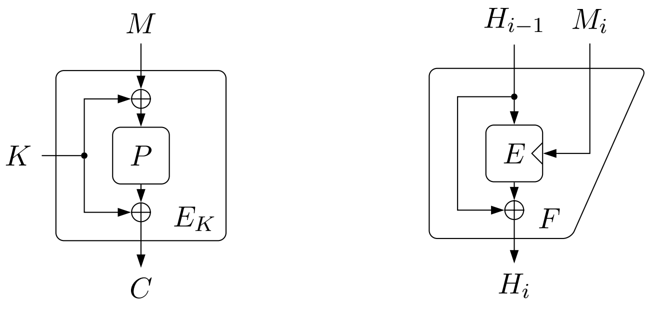
Figure 2.6 illustrates two prime examples which we will revisit later in Chapter 6 and Chapter 7, respectively. The first example is due to Even and Mansour [EM91; EM97] and was refined by Dunkelman et al. [DKS12]. They propose to build a block cipher $E : K, M \mapsto C$ from a public permutation P with the same block size by adding the key K as a *whitening key* before and after applying the permutation to the plaintext:

$$C = E_K(M) := P(M \oplus K) \oplus K.$$

Similar to other constructions that merge a public and a secret input, such as the XE/XEX construction of a tweakable block cipher from a block cipher [Rog04a], this only offers birthday-level security [Dae91]. A closely related idea is the FX construction as proposed in DESX by Rivest in order to increase the key size, with related problems [KR96; KR01].

The second construction, proposed by Davies and Meyer and proven secure by Winternitz [Win84], is one of several notable approaches [PGV93b] to build an n-bit compression function F from an n-bit block cipher E_K (or permutation P, for $b = 0$):

$$H_i = F(H_{i-1}, M_i) := E_{M_i}(H_{i-1}) \oplus H_{i-1}.$$



(a) Even-Mansour: E_K from P **(b)** Davies-Meyer: F from E_K

Figure 2.6.: Constructing primitives from other primitives.

The separation between constructions of primitives from (weak) round functions and other (strong) primitives is not always as clear as outlined above. For example, there is a significant body of literature analyzing the security of Feistel networks (Figure 2.5b) assuming idealized, independent functions F_i. Similarly, idealizing the permutations P_i in key-alternating constructions (Figure 2.5a) leads to an iterated generalization of the Even-Mansour construction (Figure 2.6a), which has in turn been studied intensively.

2.1.3. Cryptographic Schemes

Cryptographic schemes provide cryptographically well-defined security levels not only for small, fixed-size artificial units like blocks, but for practically relevant inputs. They process arbitrary-length inputs such as they naturally occur in applications, and may consider additional implementation constraints, such as inputs whose full value or even length is not yet known when computation starts. To achieve this, schemes build on top of cryptographic primitives and iteratively apply them to process subsequent parts of the input. Ideally, the scheme will provide a reduction proof or convincing arguments that show that the scheme preserves (part of) the security level of the underlying primitives (up to some security degradation, which may depend for example on the number of processed message blocks or the bitsize of each block).

Encryption schemes: A symmetric encryption scheme with k-bit key and n-bit nonce is a function

$$\begin{aligned} E^* : \mathbb{F}_2^k \times \mathbb{F}_2^n \times \mathbb{F}_2^* &\to \mathbb{F}_2^*, \\ K, N, M &\mapsto C, \end{aligned}$$

called the encryption function, which maps a k-bit key K, n-bit nonce N, and plaintext M of arbitrary length m to a ciphertext C of arbitrary length $c = c(m)$ such that ${E^*}_K^N = E^*(K, N, \cdot)$ is a family of efficiently computable injective functions. Its inverse is the decryption function $({E^*}_K^N)^{-1} = {D^*}_K^N$ such that ${D^*}_K^N({E^*}_K^N(x)) = x$ for all $x \in \mathbb{F}_2^*$:

$$\begin{aligned} D^* : \mathbb{F}_2^k \times \mathbb{F}_2^n \times \mathbb{F}_2^* &\to \mathbb{F}_2^*, \\ K, N, C &\mapsto M. \end{aligned}$$

Note that E^* may impose a (generous) limit on the length of $M \in \mathbb{F}_2^*$, and that E^* is not necessarily surjective, so some values in $\mathbb{F}_2^*$ may not be valid ciphertexts and cause an error $\perp$ when decrypting. For simplicity of notation and to avoid confusion with cryptographic verification of integrity, we nevertheless use $\mathbb{F}_2^*$ to denote both the domain and the codomain of E_K^N for any K, N.

Intuitively, an encryption mode extends the security notion of block ciphers from fixed-length messages to arbitrary-length messages. Like block ciphers, encryption schemes protect the confidentiality of the plaintext M and key K. An additional input, the nonce N (or initial value IV), serves to randomize the ciphertext and thus hide plaintext equality. Schemes usually require that the nonce never repeats ("nonce-based"), and may impose additional requirements, such as random, unpredictable nonces ("IV-based", for example in CBC). The above definition of (nonce-based) encryption schemes surfaces this nonce as an input to the encryption function E^*, as suggested by Rogaway [Rog04b], thus making E^* a deterministic, stateless algorithm. A more classic approach, inspired from analogy with asymmetric schemes [GM82; BDJR97], is to let $E^* = E^*_\$$ be a *probabilistic* algorithm that generates N by itself using internal coin tosses and explicitly or implicitly yields it as an output as part of the ciphertext C:

$$\begin{aligned} E^*_\$: \mathbb{F}_2^k \times \mathbb{F}_2^* &\xrightarrow{\$} \mathbb{F}_2^*, \\ K, M &\overset{\$}{\mapsto} C. \end{aligned}$$

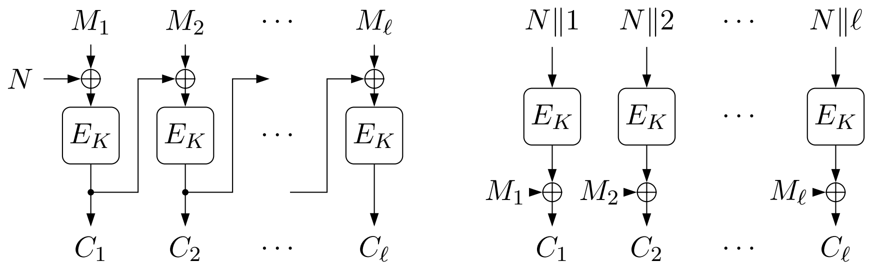
Several security notions have been proposed to formalize and generalize the requirements for a secure encryption scheme, starting with the seminal work of Goldwasser and Micali [GM82; GM84] on the *semantic security* of probabilistic asymmetric encryption schemes. This central concept requires that an adversary cannot efficiently extract any partial information about the message from a ciphertext that she could not obtain without the ciphertext. While semantic security captures the intuitive requirements well, it is typically very hard to prove for a specific construction. Furthermore, the original notion only considered passive adversaries which cannot obtain other plaintext-ciphertext pairs. Instead, it has proven more useful to consider notions based on *indistinguishability* (IND), where the adversary interacts with one of two oracles and has to decide which one it is. The *advantage* of an adversary is the difference between her success probability and the success probability $\frac{1}{2}$ of a random guess, and should be negligible for an adversary with reasonable resources.

The adversary has to distinguish an oracle that encrypts one or more plaintexts of the adversary's choice from one which responds with random strings (IND\$ [RBBK01]), which encrypts random strings (Real-or-Random [BDJR97]), or which encrypts a second provided chosen plaintext (Find-then-Guess [GM84; BDJR97], Left-or-Right [BDJR97]). The oracle may provide different levels of access to the encryption and decryption functions, ranging from no access to fully adaptive access, except queries that would lead to a trivial win. For example, the following levels are usually considered for a Find-then-Guess adversary, who has to decide whether a challenge ciphertext C encrypts plaintext M_0 or M_1: IND-CPA (chosen plaintexts) allows queries to the encryption function $E_K^{*,\cdot}$; IND-CCA1 (chosen ciphertexts) allows the queries of IND-CPA, plus queries to the decryption function $D_K^{*,\cdot}$, although these queries need to be made before receiving C; and IND-CCA2 (adaptive chosen ciphertexts) allows the queries of IND-CCA1, plus queries for any ciphertext except C after receiving C. Several authors have categorized and analyzed the relations between different access levels [KY00a] and indistinguishability games [BDJR97; Rog11]. Cryptanalysis efforts, on the other hand, have focused primarily on key recovery and, to a lesser degree, on plaintext recovery.

Two popular modes to construct an encryption scheme from a block cipher are illustrated in Figure 2.7, and two modes for permutations in Figure 2.8. In the remainder of this thesis, they will primarily make an appearance as building blocks of authenticated encryption modes, such as GCM and CCM in Chapter 6. To process messages M of arbitrary length, all modes chop the message into blocks $M = M_1 \| M_2 \| \cdots \| M_\ell$ of a predefined length, such as the block length of the underlying block cipher. Note that the descriptions below are simplified and omit details of the original schemes, such as padding rules or detailed considerations about block lengths.

- Cipher Block Chaining mode (CBC) was introduced in a patent by Ehrsam et al. [EMST76] and standardized by the US National Bureau of Standards [US 80] as one of the modes recommended for encryption with DES. Messages are encrypted block by block, but each message block M_i is modified by adding the previous ciphertext block C_{i-1} (or the nonce $C_0 = N$ for the first block) before the block cipher call:

$$C_i = E_K(M_i \oplus C_{i-1}), \qquad C_0 = N, \qquad \text{(Encryption)}$$
$$M_i = D_K(C_i) \oplus C_{i-1}. \qquad \text{(Decryption)}$$



(a) Cipher block chaining (CBC) **(b)** Counter (CTR)

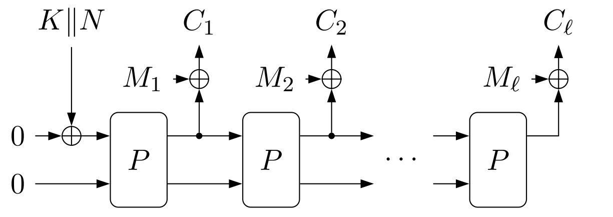
Figure 2.7.: Encryption modes based on a block cipher E_K.

Note that CBC requires a random, unpredictable initial value N (or some algorithmic changes, such as $C_0 = E_K(N)$) in order to achieve IND-CPA security and similar notions when instantiated with an idealized version of E_K [BDJR97], and that it does not achieve IND-CCA2. Notable properties of CBC include its parallelizable decryption (though encryption is strictly sequential), and the necessity of padding the message of original length m to a multiple $m^* = \ell \cdot b \geq m$ of the block cipher's message block length b, which will also be the length $c = m^*$ of the ciphertext C. Padding and the ensuing ciphertext expansion can be avoided by a variety of techniques such as ciphertext stealing (CTS), also standardized by NIST [Dwo10], but these may introduce their own complications regarding domain separation and similar issues [RWZ12]. Other frequently cited properties of CBC, such as its "error propagation" and its self-synchronization, are somewhat less relevant for contemporary protocols, since these aspects are usually handled on a different layer in the protocol stack.

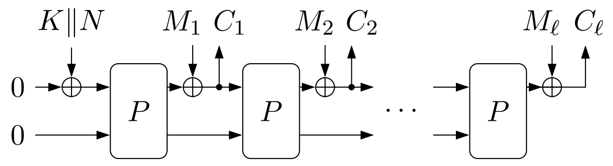
- Counter mode (CTR) was first discussed by Diffie and Hellman [DH79] and later standardized as one of the recommended modes for AES [Dwo01]. Counter mode produces a key-stream by encrypting successive nonce-dependent counter values, and adds this key-stream to the plaintext to produce the ciphertext blocks:

$$C_i = E_K(N\|i) \oplus M_i, \qquad \text{(Encryption)}$$
$$M_i = E_K(N\|i) \oplus C_i. \qquad \text{(Decryption)}$$

In contrast to CBC, both encryption and decryption are fully parallelizable, and the inverse block cipher D_K is not necessary for



(a) Sponge stream cipher



(b) Duplex sponge encryption

Figure 2.8.: Encryption modes based on a permutation P.

implementing decryption. Furthermore, there is no need to pad the message to a multiple of the block length, since the key-stream can be truncated to arbitrary plaintext length m. If N is random and unpredictable, the concatenation $N\|i$ can be replaced by $N \oplus i$ or similar while maintaining IND-CPA security.

- Sponge-based stream ciphers were proposed by Bertoni et al. [BDPV07; BDPV08]. In these constructions, an internal state is initialized with the key K and nonce N, as in classical stream ciphers, and iteratively updated by applying an unkeyed permutation P. The key-stream is generated by extracting part of this state (the outer part), whereas the rest (the inner part) remains internal. Like other stream ciphers, this mode does not require the inverse of the internal primitive, and no message padding is required. Unlike CBC and CTR, N can be implemented as a counter or a similarly predictable unique value without affecting IND-CPA security.

- Duplex sponge encryption schemes were introduced by Bertoni et al. [BDPV11b] as a useful building block for authenticated ciphers. Unlike the stream cipher construction, the state is updated to depend on all previous message blocks, which allows re-using the final state to derive an authentication tag.

Hash functions: A t-bit hash function is a function

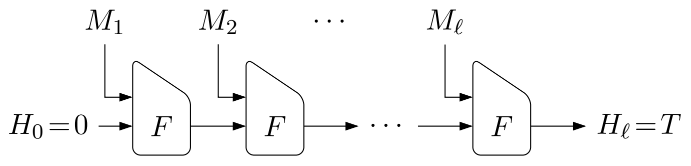
$$\begin{aligned} H : \mathbb{F}_2^* &\to \mathbb{F}_2^t, \\ M &\mapsto T, \end{aligned}$$

which is efficiently computable and maps a message M of arbitrary length m to a t-bit hash digest T. Like encryption schemes E^*, hash functions H may impose a (generous) limit on the length of $M \in \mathbb{F}_2^*$.

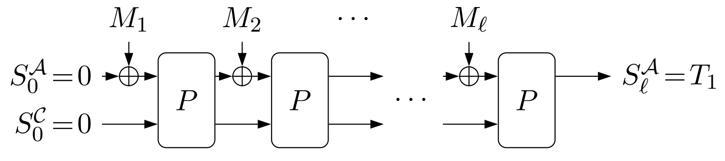
Intuitively, a hash function extends the security notion of compression functions from fixed-length messages to arbitrary-length messages, and protects the integrity of the message M. Necessary conditions for a secure hash function are specified by Rabin [Rab78] and Merkle [Mer79]: *Preimage resistance* requires that, given H and a digest T, it must be infeasible to find a message M such that $H(M) = T$. Here, depending on the author, the challenge T is either sampled uniformly from all possible digests, or by hashing a uniformly sampled message, or can also be any fixed value such as 0 (sometimes called zero-resistance). *Second-preimage resistance* requires that, given H and a message M, it must be infeasible to find a different message $M' \neq M$ such that $H(M') = H(M)$. *Collision resistance* requires that, given H, it must be infeasible to find any two different messages M and $M' \neq M$ such that $H(M) = H(M')$. These three conditions are clearly necessary for the primary applications of hash functions, such as the hash-then-sign paradigm. Other, related requirements have been put forth in the meantime to support a variety of applications, such as the construction of keyed message authentication codes (MACs) from hash functions. For example, the SHA-3 call [Kay07] requires that candidate submissions must be resistant to length-extension attacks, and that the three basic requirements also hold for any k-bit subset of the hash digest (with the definition of "infeasible" adapted accordingly).

Formal security notions, on the other hand, are harder to come by, since notions like "collision resistance" are hard to formalize in absence of a key or function family. Indeed, treating hash functions as families of functions in spite of practical reality is the preferred view both in some theory papers (e.g., for studying the basic security notions) and in some more application-oriented papers (e.g., to prevent the security degradation of preimage resistance in a multi-target setting, similar to salting). Formalized versions of the three intuitive security requirements and implications between these notions have been studied, among others, by Stinson [Sti06] and Rogaway and Shrimpton [RS04]. The idealized version of hash functions is the (truncated) *random oracle*, which consistently returns a random digest for any queried message. While random oracles are useful as a tool for proofs, they can unfortunately not be substituted with any fully specified hash function without risking total loss of security [CGH04]. The *indifferentiability* notion [MRH04] provides a useful framework to prove the soundness of a hash construction by showing its indistinguishability from a random oracle even for an adversary who can additionally query the underlying public primitives, such as compression functions or permutations.

Two constructions for hash functions from either compression functions or permutations are sketched in Figure 2.9, omitting details like padding.



(a) Merkle-Damgård



(b) Sponge

Figure 2.9.: Hashing using a compression function F or permutation P.

- The Merkle-Damgård (MD) construction (Figure 2.9a) was originally proposed by Merkle [Mer79] and later proved secure independently by Damgård [Dam89] and Merkle [Mer89]. Implemented in MD5, SHA-1, and SHA-2 variants, it is currently the most widely used hash construction. To compute the t-bit hash value T, this construction iteratively updates a chaining value H_i using a $(t + b)$-to-t-bit compression function F and the b-bit message blocks M_i. H_0 is initialized with a constant initial value (IV), denoted by 0 in the following. Then, T is derived from the final H_ℓ with a finalization function τ (usually the identity mapping, or truncation if $|T| < |H_\ell|$):

$$\begin{aligned} H_0 &= 0, \\ H_i &= F(H_{i-1}, M_i), \qquad 1 \le i \le \ell, \\ T &= \tau(H_\ell). \end{aligned}$$

The MD construction allows lifting several properties of the compression function F, most notably its collision resistance, to the hash function H. A necessary prerequisite is that M is padded to $M_1 \| \cdots \| M_\ell$ by appending an MD-compliant padding, i.e., two messages of the same length must be padded to the same length; and for two messages M, M' with different length, $M_\ell \neq M'_{\ell'}$ [GB08, p. 145]. The classical padding scheme, also called Merkle-Damgård strengthening, is to append a fixed-size representation of the length of M, preceded by a 1-bit plus the necessary number of zeros to fill to the next multiple of the block size b.

Note that conversely, finding a collision for F, that is, a pair $(H_{i-1}, M_i) \neq (H'_{i-1}, M'_i)$ such that $F(H_{i-1}, M_i) = F(H'_{i-1}, M'_i)$, does not necessarily imply finding a collision for H, that is, a pair $M \neq M'$ such that $H(M) = H(M')$. A collision for F is also called a *free-start collision* for H (if $H_{i-1} \neq H'_{i-1}$), or a *semi-free-start collision* (if $H_{i-1} = H'_{i-1} \neq 0$ or the specified IV) [LM92]. While (semi-)free-start collisions usually do not directly threaten the security of H, they can be a useful indicator to judge how the security margin of H holds up to cryptanalytic advances.

The original, "narrow-pipe" MD construction updates an internal state H_i of the same size t as the final hash value. However, this approach has some shortcomings, and inner collisions in H_i can be exploited in several generic attacks that target both the classical security requirements and additional properties. Examples include multicollisions [Jou04] and related results, such as second preimages for long messages [Dea99; KS05] and chosen-target forced-prefix preimages [KK06].

An obvious countermeasure is to increase the size of the chaining variable and to use, for example, a "wide-pipe" $2t$-bit compression function and to obtain the t-bit hash value by compressing or truncating H_ℓ accordingly [Luc05; CDMP05]. However, this invalidates the reduction proof of the original construction. The approach is illustrated, for example, by the SHA-512/t members of the SHA-2 family, which implement the chop-MD construction [CDMP05]. It is worth noting that while all these attacks are faster than the corresponding generic attacks on a random oracle, they are no faster than collision attacks, and simply show that more applications than previously expected require collision-resistant hash functions.

- The sponge construction (Figure 2.9b) was originally introduced as a theoretical model for hash functions [BDPV07] and proved indifferentiable by Bertoni et al. [BDPV08], but the design and standardization of Keccak/SHA-3 [BDPV11d; Dwo15] have also demonstrated its value for practical designs. Unlike the Merkle-Damgård construction or block-cipher–based encryption schemes, the sponge does not try to lift specific security properties, such as collision-resistance, from the underlying primitive to the scheme. Instead, it builds on the most generic primitives: random permutations P (P-sponge) or random arbitrary transformations (T-sponge). The input and output space of P are each written as the direct product $\mathcal{A} \times \mathcal{C}$ of an outer part $\mathcal{A}$ (r bits, the rate) and an inner part $\mathcal{C}$ (c bits, the capacity). The state $S_i = (S_i^{\mathcal{A}}, S_i^{\mathcal{C}})$ is updated as follows to produce the tag $T = T_1 \| T_2 \| \cdots$, which can be truncated to the desired length t:

$$\begin{aligned} S_0 &= (0,0) && && \text{(Initialization)} \\ S_i &= P(S_{i-1}^{\mathcal{A}} \oplus M_i, S_{i-1}^{\mathcal{C}}), && 1 \le i \le \ell && \text{(Absorbing)} \\ S_{\ell+i} &= P(S_{\ell+i-1}), && 1 \le i && \text{(Squeezing)} \\ T_i &= S_{\ell+i-1}^{\mathcal{A}} \end{aligned}$$

  Distinguishing this construction from a random oracle requires the detection of *inner collisions* in $\mathcal{C}$. This is reflected in the flat sponge claim, which states that "the success probability of any attack should be smaller than or equal to the maximum of that for a random oracle and $1 - \exp(2^{2y-(c+1)})$, with $N = 2^y$ the number of calls to the round function (or its inverse)" [BDPV07]; in other words, the sponge provides the expected security up to about $2^{c/2}$ queries.

Authentication schemes: A t-bit message authentication code (MAC) is an efficiently computable function H which maps k-bit key K and a message M of arbitrary length m to a t-bit tag T:

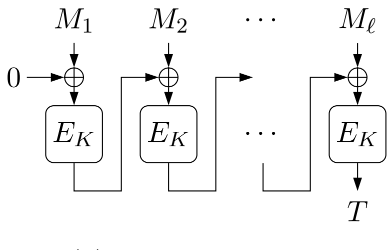
$$\begin{aligned} H : \mathbb{F}_2^k \times \mathbb{F}_2^* &\to \mathbb{F}_2^t, \\ K, M &\mapsto T. \end{aligned}$$

The MAC protects the authenticity of messages: To confirm that a received message M and tag T were produced by an owner of key K, the MAC is recomputed as $T' = H_K(M)$, and the resulting tag checked for equality $T = T'$. We refer to this use of the function H as an authentication scheme, which provides an authentication function $A^* : M \mapsto (M,T)$ and the verification function $V^* : (M,T) \mapsto M$ or $\perp$ as defined above, where $\perp$ signifies the failure event $T \neq T'$. In contrast to encryption, authentication schemes require no nonce N to achieve their security goals, but may support nonces to enable composition with encryption schemes.

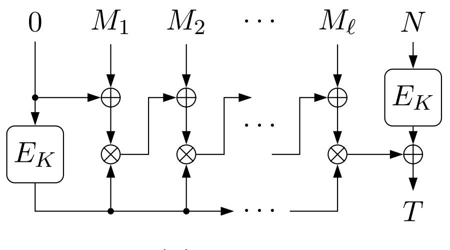
The expected security for an authentication scheme can be described in similar terms as that of a signature scheme [BKR94], and is captured by the *(strong) unforgeability* (SUF-CMA) notion: An adversary who can obtain $A^*(M)$ for adaptively chosen messages M of her choice should be unable to produce a pair (M, T) that was not obtained from A^* as a reply to a query, but is nevertheless accepted by V^* (with non-negligible probability). Note that if H_K is a probabilistic function, then the adversary even succeeds if she can produce a new, different tag $T' \neq T$ for a previous query response (M, T) such that (M, T') is accepted by V^*.

We very briefly illustrate two well-known example constructions in Figure 2.10 due to their role as building blocks for authenticated encryption.

- Plain CBC-MAC (Figure 2.10a) [Nat85] is based on CBC encryption, but returns only the last ciphertext block as tag. Note that the plain version is insecure and needs to be tweaked, e.g., by prepending the message length, or additional processing of the tag [BKR94].
- GMAC (Figure 2.10b) [MV04; Dwo07] is an example of a very different approach. The underlying iterative construction is the *universal hash function* with final nonce-based mask introduced by Carter and Wegman [CW77; WC81]. This function essentially evaluates a polynomial in the secret variable $E_K(0)$ using Horner's scheme, with polynomial coefficients given by the (length-padded) message M, plus a constant coefficient $E_K(N)$ that depends on a nonce N.



(a) Plain CBC-MAC



(b) GMAC

Figure 2.10.: Message authentication codes based on a block cipher E_K and finite-field multiplications $\otimes$.

Authenticated encryption schemes: An authenticated encryption scheme with associated data (AEAD) is a function

$$\begin{aligned}\mathcal{E} : \mathbb{F}_2^k \times \mathbb{F}_2^n \times \mathbb{F}_2^* \times \mathbb{F}_2^* &\to \mathbb{F}_2^* \times \mathbb{F}_2^t, \\ K, N, A, M &\mapsto C, T\end{aligned}$$

which maps a k-bit key K, n-bit nonce N, associated data A of arbitrary length a, and a message M of arbitrary length m to a ciphertext C of length $c = c(m)$ and a t-bit tag T such that $\mathcal{E}_K^{N,A} = \mathcal{E}(K, N, A, \cdot)$ is a family of efficiently computable, injective functions. Its inverse $(\mathcal{E}_K^{N,A})^{-1} = \mathcal{D}_K^{N,A} = \mathcal{D}(K, N, A, \cdot, \cdot)$ for each K, N, A defines the corresponding verified decryption function such that $\mathcal{D}_K^{N,A}(\mathcal{E}_K^{N,A}(M)) = M$ for all plaintexts M, and $\mathcal{D}_K^{N,A}(\cdot, \cdot) = \perp$ for all other inputs:

$$\begin{aligned}\mathcal{D} : \mathbb{F}_2^k \times \mathbb{F}_2^n \times \mathbb{F}_2^* \times \mathbb{F}_2^* \times \mathbb{F}_2^t &\to \mathbb{F}_2^* \cup \{\perp\}, \\ K, N, A, C, T &\mapsto M \text{ or } \perp.\end{aligned}$$

An authenticated encryption scheme meaningfully combines an encryption scheme and an authentication scheme, thus protecting both confidentiality and authenticity of data. In fact, it has been repeatedly suggested to use authenticated encryption instead of pure encryption even when confidentiality is (or seems to be) the only security goal [BU02; DP07; Rog11]. For example, schemes that only provide IND-CPA are highly sensitive to innocent implementation decisions, like error messages [Vau02]. On the other hand, an encryption scheme that is also unforgeable provides security against adaptive chosen ciphertext attacks [KY00b].

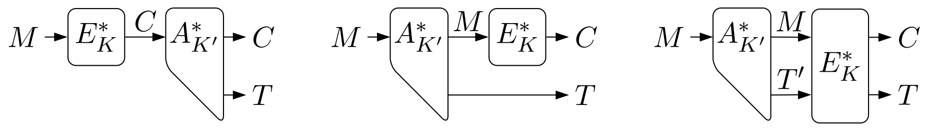
Like pure encryption schemes, authenticated encryption schemes and their security notions were originally formalized in terms of probabilistic algorithms [BN00; KY00b]. The explicit, user-controlled nonce [RBBK01; Rog04b] and associated data [Rog02] are later additions. The CAESAR call proposes an additional optional input, the secret message number, but this has not yet found widespread adoption [CAE13; NRS13].

The confidentiality notions of indistinguishability extend quite naturally from encryption to authenticated encryption. For authenticity, the unforgeability notions need to be adapted such that a forgery consists either of a ciphertext plus the corresponding tag for a new, not previously queried message (integrity of plaintexts, INT-PTXT), or of a new ciphertext with tag that was not previously returned as a result to any query (integrity of ciphertexts, INT-CTXT).

A succinct all-in-one security notion that covers both properties was proposed by Rogaway and Shrimpton [RS06]: The adversary is given access either to a pair of authenticated encryption and verified decryption oracles $(\mathcal{E}_K, \mathcal{D}_K)$, or to a pair of oracles $(\$, \perp)$, where $\$(\cdot)$ returns a random string of the correct length, and $\perp(\cdot)$ returns $\perp$ on every input. The distinguishing advantage is defined as the difference between the adversary's success probabilities (i.e., probabilities of returning 1) in these two cases. It should be negligible for any adversary with reasonable resources as long as the queries to the first oracle are nonce-respecting and no outputs of the first oracle are forwarded to the second oracle. While originally proposed to formalize misuse-resistant authenticated encryption (MRAE), where the adversary is not limited to nonce-respecting queries, the notion works just as well for nonce-based schemes.

From a cryptanalysis perspective, the adversary's goal is to forge valid ciphertexts with tag or recover the key (or, more rarely, learn information about the plaintext). Attacks are quantified in terms of the number, type and size of required queries, computational complexity, and success probability, but also in terms of damage potential, with key recovery the worst case. While any forgery breaks the cipher's authenticity, forgery attacks are sometimes further subclassified into existential (the attacker may not even know the plaintext), selective or chosen-plaintext (the attacker adaptively chooses the plaintext during the attack), almost universal (meaningful plaintexts), and universal or random-plaintext forgeries (the attacker can forge for all plaintexts, a total break) [KY00b].

A general approach for obtaining authenticated ciphers is by functional composition of an encryption scheme E^* and an authentication scheme A^* (or MAC). Figure 2.11 illustrates the three classic generic compositions: Encrypt-then-Authenticate (EtA), Encrypt-and-Authenticate (E&A), and Authenticate-then-Encrypt (AtE). In all three variants, associated data can be included as an additional (properly domain-separated) input to A^*, and both E^* and A^* may be probabilistic algorithms or include a nonce.



(a) Encrypt-then-Auth. **(b)** Encrypt-and-Auth. **(c)** Auth.-then-Encrypt

Figure 2.11.: Generic compositions for authenticated encryption.

Krawczyk [Kra01] and Bellare and Namprempre [BN00; BN08] analyzed the generic security of these constructions for probabilistic E^*_K and found that only EtA is generally secure for secure $A^*_{K'}$ and E^*_K with independent keys K, K'. The security of AtE and E&A depends on details of A^* and E^* beyond the usual security notions. Namprempre et al. [NRS14] revisit the question for nonce-based E^*_K, which is much more relevant for practical schemes, and show more differentiated results. Several of the most popular authenticated ciphers are instances of such generic compositions:

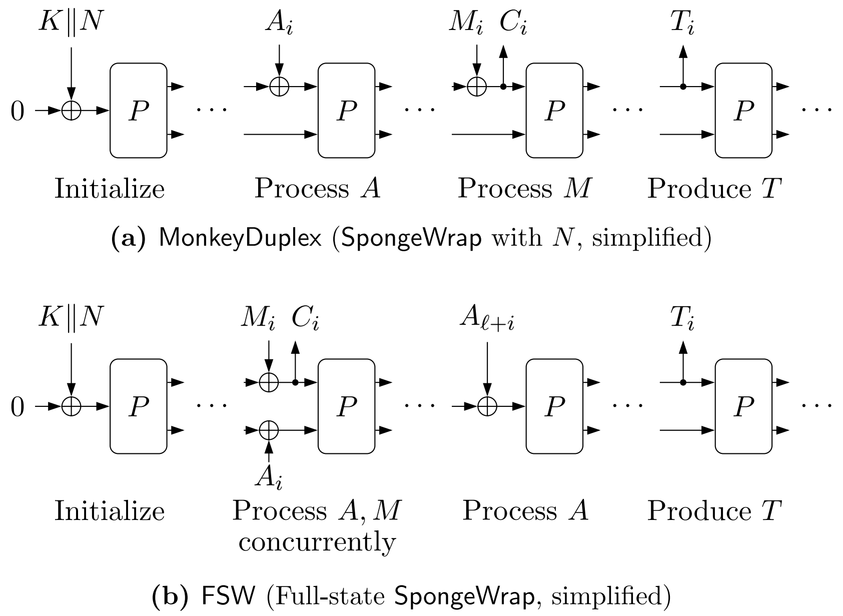
- Counter-with-CBC-MAC mode (CCM) was designed by Whiting et al. [WHF03] and standardized by NIST [Dwo04]. It essentially combines CBC-MAC (Figure 2.10a) of the nonce N and message length m, followed by data A and message M, in an Authenticate-then-Encrypt construction with CTR encryption (Figure 2.7b) of the message M and tag T. Possible disadvantages include the need for two block cipher calls per block and knowing m and N in advance.
- Galois/Counter mode (GCM) by McGrew and Viega [MV04; Dwo07] is an Encrypt-then-Authenticate construction using CTR encryption of M (Figure 2.7b) and GMAC of A and C (Figure 2.10b). It fixes some criticized properties of CCM by processing a, m, N after A, C, and replacing block cipher calls with finite-field multiplications.

More recent designs, including several of the CAESAR round-3 candidates [CAE16], show that dedicated constructions can be very competitive both in terms of efficiency and in terms of simplicity:

- Tweakable block cipher (TBC) variants of standard modes, such as TAE [LRW02; LRW11] and ΘCB3 [KR11b] of OCB, may permit simpler proofs than the original block cipher (BC) variants. Dedicated TBC modes like SCT [PS16] address one of the main issues of BC modes by providing beyond-birthday security, besides other features. On the downside, generic constructions for TBCs from BCs like XE/XEX [Rog04a] either need more than one BC call, or provide only birthday-level security [Men15; WGZ+16].
- Duplex schemes as proposed by Bertoni et al. [BDPV11b] achieve authenticated encryption with almost no overhead compared to encryption (Figure 2.8b) by squeezing the tag from the sponge state after the final ciphertext block (Figure 2.12a). The approach is easily adapted to accommodate intermediate tags, variable-length variants of (K, N, A) (SUV: secret unique value), etc., as illustrated by round-3 CAESAR candidates Keyak [BDP+16b] and Ketje [BDP+16a].

Keyed sponges like the KeyedSponge [BDPV11c] and keyed Duplex [BDPV11b] can process data more efficiently than unkeyed sponges. The first reason is that a dedicated proof not based on sponge indifferentiability reveals that keyed sponges can provide security beyond the birthday bound on the capacity c, as long as the online data complexity remains well below this birthday bound $2^{c/2}$ [BDPV11c; JLM14]. In particular, Andreeva et al. [ADMV15] show that the time complexity is at least $\min\{2^k, 2^c/\mu\}$, where μ is the multiplicity [BDPV10b], which is usually very small for nonce-based schemes.

The second reason is that keyed sponges can securely absorb more data per permutation call by using not just the outer part, but also the inner part to absorb. First proposed for the DonkeySponge MAC by Bertoni et al. [BDPV12], this idea was also generalized to associated data in duplex sponges by Sasaki and Yasuda [SY15]. The most general variant is FSW, the Full-state SpongeWrap variant (Figure 2.12b), specified and proved secure by Mennink et al. [MRV15]. It concurrently absorbs associated data blocks A_i in the inner part and message blocks M_i in the outer part, or, if the number of A_i blocks is larger than of M_i, absorbs A in both the inner and outer part (minus a few frame bits for padding and domain separation).



(a) MonkeyDuplex (SpongeWrap with N, simplified)

(b) FSW (Full-state SpongeWrap, simplified)

Figure 2.12.: Authenticated encryption modes based on permutation P.

2.2. Differential Cryptanalysis

Differential cryptanalysis is one of the most effective and most versatile cryptanalytic techniques for analyzing symmetric primitives. The approach was introduced for the analysis of DES by Biham and Shamir [BS90; BS91; BS93], culminating in the first attack faster than brute-force for full-round DES [BS92] (given sufficient amounts of chosen plaintexts). The latter result is particularly noteworthy in the light of later revelations regarding the design criteria of DES [Cop94], which show that IBM and NSA were already aware of elements of this attack as early as 1974, and strengthened the cipher accordingly. Since then, defending against differential cryptanalysis has been one of the primary security requirements for new designs. However, differential cryptanalysis has proven to be exceptionally versatile, which means that variants and generalizations of the attack may succeed even when the original attack does not.

In the most general sense, differential cryptanalysis considers pairs of inputs with some fixed relation or "input difference", and analyzes the resulting difference relation of the outputs produced by the (round-reduced) cryptographic primitive. This can be done by tracing the effect of each internal operation on the differences in intermediate variables of the computation. The resulting statistical information about the output difference, such as particular differences with very high probability, can then be used to thwart the security aims of the primitive, for example by recovering the secret key or finding collisions.

In Section 2.2.1, we introduce the necessary notation and background to investigate the effect of local operations and their composition, such as a network of S-boxes, with respect to differences. In Section 2.2.2, we show how the differential behavior of several rounds can be exploited to thwart the security aims of different primitives, in particular block ciphers and compression functions. This motivates Section 2.2.3, where we discuss approaches to identify (as an attacker) or prevent (as a designer) exploitable differential behavior. Finally, in Section 2.2.4, we discuss generalizations of and concepts related to differential cryptanalysis, such as truncated and higher-order differentials, which may be applicable even when a primitive appears secure against standard differential attacks.

2.2.1. Basic Operations: A Differential View

In differential cryptanalysis, we consider *pairs* of two variables x and x^*, and evaluate our knowledge of the relation, or *difference* Δx, between these values. Usually, for bitstrings $x, x^* \in \mathbb{F}_2^n$, this difference is defined in terms of the bitwise exclusive-or operation $\oplus$ and is thus itself a bitstring:

$$\Delta x = x^* \oplus x.$$

If x and x^* are inputs to two instances of some cryptographic function f, we are interested in the resulting difference Δy of the two outputs y, y^*:

$$\Delta y = y^* \oplus y = f(x \oplus \Delta x) \oplus f(x).$$

The analysis of DES by Biham and Shamir [BS90] laid the foundations for systematically analyzing Δy and its dependency of Δx, as well as possibilities to exploit information on Δy. Similarly defined differences also appear in the analysis of FEAL [Boe88; Mur90]. This difference notion is also easily generalized to other groups [LMM91]; one example of interest is the modular difference $x^* \boxminus x := x^* \boxplus x^-$ induced by modular addition $\boxplus$ of $x, x^* \in \mathbb{Z}_{2^n}$ with corresponding additive inverse $(\cdot)^-$, as used in the attacks on the MD4/MD5 family [Dob96]. Unless noted otherwise, difference refers to exclusive-or in the remainder of this section.

Derivation of Boolean functions

Let f be some operation, written as a (vectorial) Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$. For a fixed input difference Δx, the output difference Δy depends on the value x. Thus, for any fixed $\alpha = \Delta x \in \mathbb{F}_2^n$, this induces another function on $\mathbb{F}_2^n$, the forward directional derivative by α:

$$\Delta_\alpha f(x) := f(x \oplus \alpha) \oplus f(x).$$

One reason for the success of differential cryptanalysis is that these derivatives $\Delta_\alpha f$ may be more amenable to analysis than the initial function f. This derivation operator shares many properties with the derivations of differential calculus, including its linearity ("sum rule") and a variant of Leibniz's rule ("product rule", with respect to bitwise multiplication) [Lai94]. In particular, if f is a Boolean function and $\deg f$ denotes its algebraic degree, then

$$\deg \Delta_\alpha f \leq \deg f - 1.$$

Differential cardinality, probability, and propagation

Even if the value x is unknown, we can still derive some statistical information about the possible values of $\Delta_\alpha f(x)$. We refer to a pair of candidate values for the input difference $\alpha = \Delta x \in \mathbb{F}_2^n$ and the output difference $\beta = \Delta y \in \mathbb{F}_2^m$ as a *differential* $\delta = (\alpha \mapsto \beta)$. The solution set $S(\alpha, \beta)$ of the differential with *cardinality* $s(\alpha, \beta) := |S(\alpha, \beta)|$ is then

$$S(\alpha, \beta) := \{x \in \mathbb{F}_2^n : \Delta_\alpha f(x) = f(x \oplus \alpha) \oplus f(x) = \beta\} .$$

We call the differential *impossible* if $s(\alpha, \beta) = 0$, and *possible* otherwise. For example, for $\alpha = 0$, only the *trivial* differential $(0 \mapsto 0)$ is possible. Pairs $(x, x \oplus \alpha)$ with $x \in S(\alpha, \beta)$ are called *valid.* Note that some authors count the cardinality of the set of unordered pairs $\{x, x \oplus \alpha\} = \{x \oplus \alpha, x\}$, leading to an alternative cardinality notion $\tilde{s} = s/2$ (if $\alpha \neq 0$) [DR07].

If n, m are not too large, all values $(s(\alpha, \beta))_{\alpha \in \mathbb{F}_2^n \setminus \{0\}, \beta \in \mathbb{F}_2^m}$ can be computed and stored in a table, the *differential distribution table* (DDT) [BS90]. The multiset of values in this table is referred to as the *differential spectrum* of f, and its maximum as the *differential uniformity* du_f of f [Nyb93]:

$$\mathsf{du}_f := \max_{\alpha \neq 0, \beta} s(\alpha, \beta).$$

Cardinalities $s(\alpha, \beta)$ are even, so $\mathsf{du}_f \geq 2$. Furthermore, $\sum_\beta s(\alpha, \beta) = 2^n$, so if $n \leq m$, then for any α, $s(\alpha, \beta) = 0$ for at least half of all values β. Functions with $m = n$ and $\mathsf{du}_f = 2$ are called *almost perfect nonlinear* (APN) and have been proposed for designing ciphers resistant against differential cryptanalysis [NK92; NK95]. Unfortunately, all known APN S-boxes have an odd number of input bits, with the exception of one 6-bit S-box due to Dillon [BDMW10].

We denote the corresponding probability that f maps an input difference $\Delta x = \alpha$ to an output difference $\Delta y = \beta$ for uniformly random x by

$$\mathsf{dp}(\alpha, \beta) = \mathbb{P}_x[\alpha \overset{f}{\mapsto} \beta] := \mathbb{P}_x\left[f(x \oplus \alpha) \oplus f(x) = \beta\right] = \frac{s(\alpha, \beta)}{2^n} \leq \frac{\mathsf{du}_f}{2^n}.$$

The logarithm of a non-zero differential probability is referred to as the *weight*, *cost*, or sometimes as the *number of conditions* of this differential:

$$\mathsf{dw}(\alpha, \beta) := -\log_2(\mathsf{dp}(\alpha, \beta)) \in \{0, \dots, n-1\} .$$

Where not clear from the context, we also write $S_f(\alpha, \beta)$, $\mathsf{dp}_f(\alpha, \beta)$, etc.

In many cases, it is infeasible to explicitly list the full DDT because n is too large. It may nevertheless be easy to find $\mathsf{dp}(\alpha, \beta)$ for given α, β. For example, if f is an $\mathbb{F}_2$-affine function $f(x) = \ell(x) \oplus c$ with linear part $\ell(x)$ and the input difference is $\alpha = \Delta x$, then we know that the only one value β with non-zero probability is $\beta = \ell(\alpha) = \Delta_\alpha f(x)$.

A particularly relevant case is that of addition modulo 2^w, $w \in \{32, 64\}$. The probability of a differential $(\alpha_1, \alpha_2) \mapsto \beta$, with $\alpha_1, \alpha_2, \beta \in \mathbb{F}_2^w$, was shown by Lipmaa and Moriai [LM01] to be efficiently computable as

$$\begin{aligned}\mathsf{dp}(\alpha_1, \alpha_2, \beta) &= \mathbb{P}_{x_1,x_2}\big[((x_1 \oplus \alpha_1) \boxplus (x_2 \oplus \alpha_2)) \oplus (x_1 \boxplus x_2) = \beta\big] \qquad (2.1)\\ &= \begin{cases} 0, \text{ if } \mathsf{eq}(\alpha_{1\lll 1}, \alpha_{2\lll 1}, \beta_{\lll 1}) \wedge (\alpha_1 \oplus \alpha_2 \oplus \beta \oplus \alpha_{1\lll 1}) \neq 0, \\ 2^{-\mathsf{hw}\left((\neg \mathsf{eq}(\alpha_1, \alpha_2, \beta))_{\lll 1}\right)} \quad \text{else,} \end{cases}\end{aligned}$$

where $\mathsf{eq}(w, x, y)$ is defined as in Section 2.1.1 and $\mathsf{hw}(x)$ denotes the Hamming weight of $x \in \mathbb{F}_2^w$.

In SPN constructions, we encounter another type of function f with large inputs $x = (x_1, \dots, x_k) \in (\mathbb{F}_2^s)^k$: A parallel application of several small s-bit S-boxes $\mathcal{S}_i : \mathbb{F}_2^s \to \mathbb{F}_2^s$, followed by a linear function $\ell : \mathbb{F}_2^{sk} \to \mathbb{F}_2^{sk}$ (or vice-versa). Based on the DDTs of the individual S-boxes, it is very easy to derive the differential properties of f for any differential $\delta = (\alpha \mapsto \beta)$. Assume f is a permutation and $\gamma = (\gamma_1, \dots, \gamma_k) = \ell^{-1}(\beta)$, then

$$\begin{aligned} S_f(\alpha, \beta) &= S_{\mathcal{S}_1}(\alpha_1, \gamma_1) \times \dots \times S_{\mathcal{S}_k}(\alpha_k, \gamma_k)\,, \\ \mathsf{dp}_f(\alpha, \beta) &= \prod_{i=1}^{k} \mathsf{dp}_{\mathcal{S}_i}(\alpha_i, \gamma_i)\,. \qquad (2.2)\end{aligned}$$

We refer to the S-boxes $\mathcal{S}_i$ with $\alpha_i \neq 0$ as active S-boxes for this differential, and to the others as inactive. Clearly, only the properties of the active S-boxes contribute to the overall differential properties of f.

Since many transitions $\alpha \overset{f}{\mapsto} \beta$ are impossible, knowing $\alpha = \Delta x$ allows deterministically deriving some information about Δy without knowing x, x^*. We refer to this as *propagation* of information. An explicit characterization of possible α, β such as given above for modular addition is particularly useful for propagation.

Differential characteristics: Keying and composing components

The notions described so far are useful for functions that are both simple and known. However, cryptographic targets are usually neither of those. If we pick a function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ uniformly at random from all such functions, then the differential probability $\mathsf{dp}(\alpha, \beta)$ (and, similarly, $s(\alpha, \beta)$ and $\mathsf{dw}(\alpha, \beta)$) of any non-trivial differential $\delta = (\alpha, \beta)$ becomes a stochastic variable. Daemen and Rijmen [DR07] showed that this variable is distributed according to a (scaled) binomial distribution $2^{1-n} \cdot \mathcal{B}(2^{n-1}, 2^{-m})$:

$$\mathbb{P}_f \left[\mathsf{dp}(\alpha, \beta) = \frac{k}{2^{n-1}}\right] = (2^{-m})^k (1 - 2^{-m})^{2^{n-1}-k} \binom{2^{n-1}}{k},$$
$$\mathbb{E}_f[\mathsf{dp}(\alpha, \beta)] = 2^{-m}.$$

For cryptanalysis, we are interested in $\mathsf{dp}(\alpha, \beta)$ for a fixed function, but this function is selected uniformly at random from a set of functions $\{f_K\}$ by an unknown key K. Since we cannot compute $\mathsf{dp}(\alpha, \beta)$, we can instead consider the *expected differential probability* of the set $\{f_K\}$ [LMM91]:

$$\mathsf{edp}(\alpha, \beta) := \mathbb{E}_K[\mathsf{dp}(\alpha, \beta)] = \mathbb{P}_{K,x}[\alpha \overset{f_K}{\mapsto} \beta] = \mathbb{P}_{K,x}[f_K(x \oplus \alpha) \oplus f_K(x) = \beta]\,,$$

and hope $\mathsf{dp}(\alpha, \beta) \approx \mathsf{edp}(\alpha, \beta)$: the *hypothesis of stochastic equivalence*. We call $\alpha \mapsto \beta$ an impossible differential of the set $\{f_K\}$ if $\mathsf{edp}(\alpha, \beta) = 0$.

But how to estimate $\mathsf{edp}(\alpha, \beta)$ for a (large) family of (large) functions? We are particularly interested in a certain class of function families $\{f_K\}$: the key-alternating construction, which iteratively updates the state x_i using a round function $x_i = \mathcal{R}_i(x_{i-1} \oplus K_{i-1})$ with a round key K_i produced by the key schedule. This round function is iterated r times to compute $y = f_K(x) = f_K(x_0) = x_r \oplus K_r$. For simplicity, we also write $x_i = \mathcal{R}^i(x)$.

The round function $\mathcal{R}_i(\cdot \oplus K_i)$ is much simpler than f_K, so we assume it is very easy to characterize the differential behavior of $\mathcal{R}_i$, but very hard for f_K. Then, instead of *differentials* $\delta = (\alpha, \beta)$ for $y = f_K(x)$, we can consider *differential characteristics* χ for f_K [BS90], in some contexts also referred to as *trails* or *paths*. A characteristic χ is a consistent sequence of differentials δ_i for each round function $\mathcal{R}_i$:

$$\chi = (\alpha = \chi_0, \chi_1, \dots, \chi_r = \beta)\,,$$
$$\delta_i = (\chi_{i-1}, \chi_i), \qquad 1 \le i \le r\,.$$

We can now define analoga of the concepts for differentials δ also for characteristics χ. This includes the fixed-key properties like solution set $S(\chi)$, cardinality $s(\chi)$, differential probability $\mathsf{dp}(\chi)$, and weight $\mathsf{dw}(\chi)$, as well as the family property of expected differential probability $\mathsf{edp}(\chi)$. All properties are defined as before with respect to the solution set

$$S(\chi) := \left\{x \in \mathbb{F}_2^n : \textstyle\bigwedge_{i=1}^r \Delta_\alpha \mathcal{R}^i(x) = \mathcal{R}^i(x \oplus \alpha) \oplus \mathcal{R}^i(x) = \chi_i\right\} .$$

While the fixed-key properties usually remain elusive for the cryptanalyst, we can derive some family properties under reasonable assumptions. In particular, if all round keys are independent and uniformly random, then

$$\mathsf{edp}(\chi) = \prod_{i=1}^{r} \mathsf{edp}(\delta_i)\,. \tag{2.3}$$

More generally, Lai et al. [LMM91] call any family $\{f_K\}$ of iterated functions with rounds $\mathcal{R}_i(\cdot, K_i)$ a *Markov cipher* if its round function satisfies the following property: The probability that a uniformly random round key K_i maps an input difference χ_{i-1} to an output difference χ_i is independent of the input x, i.e., for any x, the family $\{\mathcal{R}_i(\cdot, K)\}_K$ satisfies

$$\mathbb{P}_K[\mathcal{R}_i(x \oplus \chi_{i-1}, K) \oplus \mathcal{R}_i(x, K) = \chi_i] = \mathsf{edp}(\chi_{i-1}, \chi_i).$$

The (expected) probability of a characteristic χ gives a lower bound for the (expected) differential probability of $\delta = (\chi_0, \chi_r)$, as

$$\mathsf{edp}(\chi_0, \chi_r) = \sum_{\chi_1} \cdots \sum_{\chi_{r-1}} \mathsf{edp}(\chi_0, \chi_1, \ldots, \chi_r)\,.$$

Summarizing, we use the following assumptions and abstractions. First, we characterize the differential behavior of the unkeyed round function. The required differential weight $\mathsf{dw}(\chi_{i-1}, \chi_i)$ for any differential can be obtained, for example, from the DDTs of all active S-boxes for SPN (2.2) or by counting bitwise conditions for ARX (2.1). Then, we implicitly assume a key-alternating construction with independent and uniformly random round keys to replace the key/round-constant schedule and thus obtain the associated "long-key cipher" [DR07]. We can compute the expected differential properties of characteristics for this long-key cipher by adding the differential weights of all round functions (2.3), and hope that this gives a good approximation for fixed-key properties of most keys. Finally, if we have found a characteristic χ for the iterated function this way with probability significantly higher than the generically expected differential probability 2^{-m} (or $1/(2^m - 1)$ for permutations) of (χ_0, χ_r), we may have found a useful distinguisher that can be further used in different ways to break the function's security properties, as discussed next.

2.2.2. Cryptographic Primitives: A Differential View

Differential characteristics are a remarkably flexible tool for analyzing the security of various primitives. If we find good characteristics for a primitive, we can recover confidential information or inject forged data.

Block ciphers

In block ciphers $E_K : \mathbb{F}_2^b \to \mathbb{F}_2^b, M \mapsto C$, the attacker's main goal is to recover the fixed, secret key K by observing or querying plaintext-ciphertext combinations for this key. There are two primary key recovery methods, sometimes referred to as 0R and 1R (or 1+R) attacks. Both are chosen-plaintext attacks and target the last few (or first few) round keys for an r-round cipher, but they are based on slightly different assumptions.

0R attacks. For 0R attacks [BS90], we require a full r-round characteristic χ with probability $\mathsf{dp}(\chi) \gg 2^{-b}$. We consider the corresponding differential $\delta = (\chi_0, \chi_r)$. We query the ciphertexts for a large number of chosen plaintext pairs (M, M^*) with $\Delta M = \chi_0$ and keep only the pairs that follow the differential, i.e., those with $\Delta C = \chi_r$. Assuming that the remaining pairs that follow δ also follow χ, and in particular the last round follows (χ_{r-1}, χ_r), we can derive the individual differentials (α_i, β_i) of the S-boxes (or other nonlinear operations) of the last round. Under this assumption, the set of possible values for the intermediate variables at the input and output of each S-box is then limited to the solution set $S(\alpha_i, \beta_i)$. By combining this information with the observed ciphertext values (C, C^*), we learn a set of possible values for the last round key K_r and thus reduce the possible key-space. If possible, this process can be iterated recursively for more than one final round of the characteristic. Note that the approach will usually produce at least two equivalent key candidates per S-box due to the structure of the solution sets.

The underlying assumption is that χ is the dominating characteristic for δ, so if we observe δ, it was most likely produced by χ. To this end, it is usually assumed that E_K behaves like a random function for all inputs except for the *right pairs* that follow χ, so the probability that a *wrong pair* produces δ is 2^{-b}, and the overall differential probability of δ is about $\mathsf{dp}(\chi) + 2^{-b}$. If $\mathsf{dp}(\chi) \gg 2^{-b}$, the necessary data complexity is proportional to $\mathsf{dp}(\chi)^{-1} = 2^{\mathsf{dw}(\chi)}$ chosen-plaintext pairs to find one or a few right pairs.

1+R attacks. For 1R (or 2R, 3R, …) attacks [BS90] on an r-round cipher, we use differentials $\delta = (\alpha, \beta)$ with probability $\mathsf{dp}(\delta) \gg 2^{-b}$ for $r-1$ (or $r-2$, $r-3$, …) rounds, instead of characteristics for r rounds. Such differentials are in practice often found by discovering a single good characteristic χ for differential $(\alpha, \beta) = (\chi_0, \chi_{r-1})$ and using it as a lower bound $\mathsf{dp}(\delta) \geq \mathsf{dp}(\chi)$, but this is no necessity. We again query the ciphertexts for a large number of chosen plaintext pairs (M, M^*) with $\Delta M = \alpha$, hoping to hit several *right pairs* that follow δ. We will test this for every ciphertext pair (C, C^*). Let $\gamma = \Delta C$. If the differential (β, γ) is impossible, this is definitely a wrong pair and can be *filtered*. If (β, γ) is possible, its solution set $S(\beta, \gamma)$ will suggest several candidates for the last round key, similar to the 0R attack. For a truly right pair, the correct key will be among these; for an unfiltered wrong pair, it might or might not be. To efficiently enumerate the candidates, we only guess the relevant parts of the round key, partially decrypt the last round for both C and C^*, and upvote the partial key candidate if the resulting difference matches β. When all data has been processed, we accept the most upvoted key candidate and brute-force the remaining reduced key-space.

The underlying assumption is the *wrong-key randomization hypothesis*: After decrypting the last round of a pair with a wrong key candidate, the observed difference is uniformly random. In particular, the target difference β will not stand out, so the associated counter of each wrong key will be similarly low. The right key, on the other hand, will be suggested by each of several right pairs in addition to this noise, and the resulting higher counter will be distinguishable.

The data complexity depends primarily on the expected necessary number of queries to find one right pair, given by the inverse probability $\mathsf{dp}(\delta)^{-1}$, but several additional parameters must also be taken into account. Unlike the 0R attack, we cannot eliminate all wrong pairs, so we need enough data to find several right pairs to ensure that the correct key indeed ends up as the candidate with the highest counter. A relevant metric in this context is the *signal-to-noise ratio* SNR of the key recovery approach, which gives the ratio between the number of right pairs (a lower bound for the counter of the correct candidate) and the average counter of the incorrect candidates. If the number of pairs is N, the number of tested key candidates is 2^k, the average number of suggested key candidates per pair is A, and the fraction of pairs that survive filtering is B, then [BS90]

$$\mathsf{SNR} = \frac{N \cdot p_{\mathrm{r}}}{N \cdot p_{\mathrm{w}}} = \frac{\mathsf{dp}(\delta)}{A \cdot B \cdot 2^{-k}} \,.$$

Based on experiments, Biham and Shamir [BS90] propose to use enough data for about 20 to 40 right pairs if $1 < \mathsf{SNR} \leq 2$, and about 3 to 4 right pairs if SNR is "much higher".

Selçuk [SB02; Sel08] analyzes the success probability and its dependency on the invested data complexity in more detail. He considers a more general setting where the attack is successful if the correct candidate is among the top 2^{k-a} out of 2^k candidate counters, where a is the *advantage* of the attack. Assuming that the counters are independently and approximately normally distributed with mean and variance $\mu_{\mathrm{r}} = (p_{\mathrm{r}} + p_{\mathrm{w}})N$, $\sigma_{\mathrm{r}}^2 \approx \mu_{\mathrm{r}}$ (for the right key) and $\mu_{\mathrm{w}} = p_{\mathrm{w}}N$, $\sigma_{\mathrm{w}}^2 \approx \mu_{\mathrm{w}}$ (for each wrong key), he shows that the necessary number of pairs to succeed with probability p is

$$N = \frac{\left(\sqrt{\mathsf{SNR}+1}\cdot\Phi^{-1}(p) + \Phi^{-1}(1-2^{-a})\right)^2}{\mathsf{SNR}} \cdot \mathsf{dp}(\delta)^{-1},$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution. However, a comparison with experimental data shows that the theoretical model overestimates the success probability [Sel08].

The overall attack complexity is the sum of this analysis phase on the one hand, and the effort of the brute-force phase for recovering the rest of the key on the other hand. The analysis phase requires data and time proportional to $N = c \cdot \mathsf{dp}(\delta)^{-1}$ for obtaining and filtering the ciphertexts, plus time for testing $B \cdot N \cdot 2^k$ differences (possibly in a structured and more efficient way) and storing the results in a memory of 2^k counters. For the brute-force phase, the time is a fraction of 2^{-a} of the time for exhaustively enumerating the full key candidates, assuming that the candidates for the missing bits can be enumerated efficiently.

Variants. Many variants of this basic key-recovery approach have been proposed to address different performance bottlenecks, or to exploit different statistical distinguishers (see Section 2.2.4). Examples of performance-oriented improvements in Biham and Shamir's DES attacks include an approach to generate pairs from known, rather than chosen, plaintexts [BS93], initial structures to produce more pairs per queried plaintexts [BS91; BS92], and a memoryless variant that tests candidates immediately without storing many counters [BS92]. Albrecht and Cid [AC09] proposed an alternative approach to identify right pairs and recover the key based on observed runtimes of an algebraic solver. Its runtime is hard to predict and usually not better than the standard approach, but it may succeed with lower data complexity [WSMP11].

Tweakable block ciphers

For tweakable block ciphers $\tilde{E}_{K,T} : \mathbb{F}_2^b \to \mathbb{F}_2^b, M \mapsto C$, the attacker's goal is the same as for block ciphers: to recover the fixed key K. However, in addition to querying $\tilde{E}$ with different plaintexts, the attacker also has control over the tweak T. This has several important implications for differential cryptanalysis.

First, it requires an adaptation of the concept of differentials and characteristics compared to Section 2.2.1. Any observed plaintext-ciphertext differential δ is with respect to a fixed key K and with respect to two fixed, possibly different tweaks T and T^*. Differences can be introduced not only in the plaintext (which is processed by the permutation $\tilde{E}_{K,T}$), but also in the tweak (which defines the permutation $\tilde{E}_{K,T}$). Thus, it makes sense to consider the differential probability of the function $\tilde{E}_K(T, P)$ for an input difference $(\Delta T, \Delta P) = (\tau, \alpha)$ and output difference $\Delta C = \beta$:

$$\mathsf{dp}(\tau, \alpha, \beta) = \mathbb{P}_{T,P}[\tau, \alpha \overset{\tilde{E}_K}{\mapsto} \beta] = \mathbb{P}_{T,P}[\tilde{E}_K(T + \tau, P + \alpha) \oplus \tilde{E}_K(T, P) = \beta]\,.$$

We are interested in tweakable block ciphers where each permutation $\tilde{E}_{K,T}$ "looks like" an instance of a closely related block cipher. For example, in ad-hoc constructions like the TWEAKEY framework [JNP14b], the tweak T is absorbed via the key schedule. Every round key addition may introduce (or cancel) additional differences, and this additional freedom may decrease the minimum number of active S-boxes for the best differential characteristics. The effect is similar to differential characteristics in a related-key scenario for block ciphers [Bih93; Bih94a]. For example, 4 rounds of AES-128 have 25 active S-boxes, but in a related-key scenario, characteristics with 13 and truncated characteristics with 9 active S-boxes can be found [BN10]. A derived tweakable block cipher, KIASU-BC [JNP14a], has a minimum of 8 active S-boxes for 4 rounds, or even 0 active S-boxes for 2 rounds.

Second, the tweak input increases the size of the *full codebook* from 2^b plaintexts to 2^{b+t} plaintext-tweak combinations, whereas the generic probability of a differential remains about 2^{-b}. This may render attacks feasible for tweakable block ciphers even though the data requirements $2 \cdot c \cdot \mathsf{dp}(\delta)^{-1} > 2^b$ would be too large to apply to block ciphers, in particular if the block size b is smaller than the key size k.

We revisit the practical implications in Chapter 3 for the attack on MANTIS.

Compression functions

For compression functions $F : \mathbb{F}_2^t \times \mathbb{F}_2^b \to \mathbb{F}_2^b$, both the attacker's goals and methods differ significantly from keyed primitives. Instead of recovering secret information, we want to craft message inputs with certain properties. We focus on the goal of creating collisions, that is, inputs with non-zero input difference $(\Delta H_{i-1}, \Delta M_i) \neq (0, 0)$ and zero output difference $\Delta H_i = 0$. The applicability of differential cryptanalysis for finding collisions was already noted by Biham and Shamir [BS90], including first practical examples. We can consider several subtypes of collisions which are relevant in the context of iterated Merkle-Damgård hash functions [LM92]:

- *Free-start collisions* $(\Delta H_{i-1}, \Delta M_i) \neq (0, 0)$ or *compression function collisions*, where any differences are allowed both for the chaining value and the message. This most general case is the easiest to find and violates the preconditions for the MD proof [Mer89], but usually cannot be extended to an actual attack on the hash function.
- *Semi-free-start collisions* $\Delta M_i \neq 0$, $\Delta H_{i-1} = 0$, where the input chaining value is required to have zero difference.
- *Collisions* $\Delta M_i \neq 0$, $\Delta H_{i-1} = 0$, $H_{i-1} = H_{i-1}^* = 0$ (IV), where the input chaining value matches the hash function's initial value and the compression function collision thus implies a hash function collision.

If the compression function is built in a generic construction from a block cipher, such a collision corresponds to a pair of block cipher inputs with non-zero difference in the plaintext or key and a suitable ciphertext difference. For example, a free-start collision for a Davies-Meyer compression function (Figure 2.6b) corresponds to a solution for a related-key differential with $(\Delta M, \Delta K) \neq (0, 0)$ and $\Delta M = \Delta C$. Similar to tweakable block ciphers, this means we also need to consider a characteristic that covers the key schedule, but unlike the case of tweakable block ciphers, we do not have a random, unknown key to justify probability estimates based on a round-by-round evaluation. The latter can still give useful estimates for the complexity of an attack, but is much less suited for quantifying the function's security margin.

To qualify as a successful attack on the compression function, the complexity of the differential attack must be less than the roughly $2^{b/2}$ function evaluations of the generic attack [Yuv79] or its memoryless variants [Bre80].

Generally, a differential collision attack on a compression function is loosely divided into two phases: First, identify a differential characteristic for the full-round function with reasonably high probability whose differential implies a collision. Second, find a solution for this characteristic. In practice, these two phases are often less clearly separated than for keyed primitives.

While the first phase of finding a characteristic can be more difficult than for keyed differences due to the additional requirements for the differential, the second phase is often significantly easier. Preneel et al. [PGV93a] identified several important differences between keyed and unkeyed primitives for the second phase. All inputs are known to and under control of the attacker, which implies several advantages: All computations can be performed offline and in parallel, without the bottleneck of impractical data requirements. A single right pair is often sufficient for an attack and is easily detected without any statistical evaluation. For the first few rounds, the attacker can actively modify the inputs to deterministically satisfy the conditions of the characteristic (*message modification* [WLF+05; WY05]). For the later rounds, the attacker hopes that the conditions of the characteristic are probabilistically satisfied; an early-abort approach can detect contradictions before the full function is evaluated. Inside-out approaches like the rebound attack [MRST09; LMS+15] similarly find many solutions for part of the characteristic, and some probabilistically satisfy the rest.

In Chapter 7, we improve the automatic search for such collisions.

Cryptographic permutations

For permutations $P : \mathbb{F}_2^b \to \mathbb{F}_2^b$, the attacker's goal and approach depend on the intended (keyed or unkeyed) use. Many attack scenarios require either a good characteristic (keyed) or a single right pair (unkeyed) for any differential where parts of the input and output difference are fixed to zero difference. For example, consider a (keyed or unkeyed) sponge mode with rate r and capacity $c = b - r$. If we find a good characteristic whose input and output difference is zero on the c-bit inner part, we can try to build a two-block forgery or collision by introducing the input difference with the first block and cancelling the output difference with the second block. In the unkeyed case, we can take advantage of similar effects as for (semi-free-start) collisions of compression functions.

In Chapter 4, we illustrate how unpredictable the effective security margin is for unkeyed applications by providing an attack on full-round Simpira.

2.2.3. Searching and Bounding Characteristics

Both the designer and the attacker are interested in good characteristics in order to achieve their goals: The attacker wants to find a differential whose probability is high enough for an attack, usually by finding a single good characteristic. The designer wants to show that no such differential can be practically found (or, ideally, that none exists), usually by showing that even the best characteristic is by far not good enough for an attack. In the following, we briefly discuss different approaches to search for good characteristics (attacker's perspective) or to bound the probability of the best characteristic or differential (designer's perspective).

Designer's perspective

The first proposals of ciphers with "provable security" against differential cryptanalysis emerged quickly after the attack became public [Ada92]. Nyberg and Knudsen [NK92; NK95] proposed CRADIC (aka KN cipher), whose Feistel round function uses a single large APN S-box such that (under some independence assumptions) the probability of any r-round differential is within $2^{1-n} \pm 2^{1-n}$ and thus not sufficiently different from the random case to be exploitable for key recovery. Matsui [Mat96; Mat97] proposed the block cipher MISTY as a practical instantiation of this approach. Both proposals prove that no differential with reasonably high expected probability exists, so standard differential cryptanalysis is not applicable. However, they do not necessarily prevent the application of all of its variants, such as higher-order differential cryptanalysis [JK97]. Another approach in a similar vein is decorrelation theory [Vau98].

A different, more practically inspired approach that may lead to more balanced designs is to bound the expected probability of the best characteristic, rather than the best differential. This is only a necessary, not a sufficient criterion for resistance against differential cryptanalysis, but such ciphers are often considered "practically secure" [Knu93]. This approach is more easily applicable to a wider range of designs, in particular to Substitution-Permutation Networks with higher number of less computationally expensive rounds [HT94; HT96]. According to the previous assumptions (2.2,2.3), the maximum expected probability (or minimum weight) of any characteristic can be bounded based on (a) the minimum number of active S-boxes (bundle weight [DR01]) in any characteristic, and (b) the maximum differential probability (or minimum weight) of any non-trivial S-box differential. Since the latter is easy to compute, the problem is reduced to bounding the number of active S-boxes. This can be achieved either by enumerating candidate characteristics (see below) or, if the design permits, by higher-level arguments.

The *wide-trail strategy* by Daemen and Rijmen [Dae95; DR01] is a general approach for designing primitives with strong bounds by design. Whereas the "provably secure" designs discussed so far focus on maximizing the minimum S-box weight to obtain very strong rounds with large S-boxes, the wide-trail approach focuses on maximizing the minimum number of active S-boxes over multiple rounds. The authors introduce the *branch number* $\mathcal{B}$ [Dae95] as a metric for the diffusion achieved by the round function. They consider key-alternating SP constructions for $m \cdot n = b$-bit blocks where the linear transformation ("permutation") λ is defined as any linear function of the m-bit S-box outputs (n *bundles* or *cells*), usually as n linear combinations over $\mathbb{F}_2^m$. The branch number $\mathcal{B}(\lambda) \leq n+1$ of such a linear transformation $\lambda : (\mathbb{F}_2^m)^n \to (\mathbb{F}_2^m)^n$ (and the resulting round function) is the minimum total number of active bundles in the input and output of the transformation, or equivalently, the minimum number of active S-boxes over 2 rounds. Transformations λ which attain the maximum branch number $\mathcal{B}(\lambda) = n+1$ are referred to as *multipermutations* and can be constructed using the matrices of *MDS* codes, as suggested by Vaudenay [Vau94] and implemented for instance by SHARK [RDP+96]. This also allows bounding the maximum probability of differentials of ≥ 2 rounds by $\mathsf{du}_{\mathcal{S}}^n$ [HLL+00] based on the S-box $\mathcal{S}$, but the resulting bounds are usually not sufficiently tight to be useful. An alternative approach that may lead to a more balanced, efficient design is to subdivide the state into columns of n' bundles, choose locally MDS intra-column transformations with $\mathcal{B} = n' + 1$, and combine with a complementary inter-column dispersion function to get $\mathcal{B}^2$ active S-boxes after 4 rounds [DR01], as in AES [DR98].

Attacker's perspective

If a cipher was not designed according to such a bound-based strategy, or when we want to obtain actual characteristics for an attack, it is usually necessary to ask a computer for help. More specifically, we need a search algorithm that can quickly enumerate many potentially relevant characteristics, and outputs optimal (or reasonably good) characteristics as a result. There are two primary directions to create such an algorithm: Developing dedicated tools, and applying general-purpose tools.

General-purpose tools. An alternative direction is to leverage existing general-purpose search tools and optimization frameworks. This means that the cryptanalytic task must be translated to an instance of the general problem framework that the tool solves, such as Boolean satisfiability. A primary advantage of this approach is that the cryptographer can profit from the intensive efforts invested by the solver's developers, which are sometimes based on decades of research and have produced sophisticated heuristics and finely tuned, carefully optimized implementations. The task is reduced to mere translation. On the downside, the cryptographer is limited by the expressiveness of the solver's problem framework. Cryptographic insights or heuristics that would help the search can be hard to translate. If the cryptographic task can only be artificially translated to a problem instance, the solver's heuristics may be less efficient.

The two most popularly used general-purpose search tools in differential cryptanalysis are SAT solvers and MILP optimizers. While SAT models approach the search problem from the logical, Boolean, combinatoric point of view, MILP models take a less obvious approach originally grounded in mathematical methods for continuous, real-valued optimization problems.

Boolean satisfiability (SAT) solvers find a satisfying assignment (or "interpretation") in $\mathbb{F}_2^n$ for a set of Boolean variables $\{v_i\}_{i=1}^n$ such that a given Boolean function (or "propositional logic formula"), specified in Conjunctive Normal Form (CNF), evaluates to 1:

$$\bigwedge_{s=1}^{S} \bigvee_{t=1}^{T} \ell_{s,t} = 1, \qquad \ell_{s,t} \in \{v_i\}_{i=1}^n \cup \{\neg v_i\}_{i=1}^n \,.$$

The decisional SAT problem is NP-hard [Coo71], but modern SAT solvers can solve practical problem instances of surprising size and complexity thanks to important developments in search heuristics and problem preprocessing. The core search algorithm underlying most modern SAT solvers, such as `Lingeling` and `Glucose`, is the Davis–Putnam–Logemann–Loveland (DPLL) [DLL62] algorithm and its variants, like Conflict-Driven Clause Learning (CDCL) [SS96]. SAT problems are commonly encoded in DIMACS-CNF format.

In the context of differential cryptanalysis, the most common approach is to encode the bitwise differential behavior as Boolean constraints and then either query for the (non-)existence of characteristics subject to additional differential constraints like zero output difference, or query for characteristics with a fixed weight or cost. The latter requires to model an integer counter to accumulate the cost of active S-boxes or active modular additions. In the last few years, there has been a veritable explosion of such papers, most popularly targeting lightweight SPN ciphers with 4-bit S-boxes or ARX ciphers [MP13]. Advantages of a SAT-based approach include the wide range of openly available, efficient and (more or less) accessible SAT solvers and natural suitability of propositional logic for modeling differential behavior. Disadvantages arise from the relatively inefficient models of linear functions, in particular long chains of xors in the linear layers and the integer summation of costs. Furthermore, most SAT solvers are optimized for finding solutions of satisfiable formulas, not for proving unsatisfiability, which is necessary when proving bounds.

Satisfiability problems in other higher-level domains with more human-readable descriptions are often solved by translating (parts of) the problem to an equisatisfiable SAT instance. When cryptanalytic problems are solved with SAT solvers, it is usually via such higher-level domains. Commonly used terms for higher-level domain search problems include *Satisfiability Modulo Theories (SMT)* and *Constraint Programming (CP)*, with input languages like CVC or SMTLibv2 and solvers like `STP`, `Z3`, `Choco`, `IBM ILOG CP`, and many more. Dedicated tools for cryptanalysis include the solver `CryptoMiniSAT` [Soo16] and the higher-level interfaces CryptoSAT [Laf13] and CryptoSMT [Köl14].

Mixed-Integer Linear Programming (MILP) is an optimization problem for a linear objective function and linear constraints over the real numbers, where some variables may be constrained to integers. For decision variables $\{x_i\}_{i=1}^n$, the goal is to find an assignment $x \in \mathbb{Z}^k \times \mathbb{R}^{n-k}$ that minimizes

$$\min \quad cx = \sum_{i=1}^{n} c_i x_i \qquad\qquad \text{s.t.} \quad Ax \leq b$$

for coefficients $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{M \times n}$, $b \in \mathbb{R}^M$. MILP problems are often encoded in `LP` syntax, which is widely supported by solvers such as `IBM ILOG CPLEX` or `Gurobi`.

The application of MILP solvers in differential cryptanalysis was first proposed for easily finding the minimal number of active S-boxes in wide-trail designs [MWGP11; WW11]. In their model, the activity of each S-box is represented by a binary decision variable. Then, for AES and AES-like designs, the constraints for valid activity patterns can be easily described by modeling solely the differential branch number of the MixColumns-step with the help of one binary column activity helper variable per column.

The approach has since been successfully adapted to bitwise differences in lightweight ciphers with 4-bit S-boxes [SHW+14a; SHW+14b] (or even 8-bit S-boxes [AST+17]) and ARX designs [FWG+16]. Compared to SAT models, MILP models are naturally suited for accounting the cost of a characteristic by summing and bounding partial costs; on the other hand, the description of Boolean constraints in the form of tables, or the highly combinatorial nature of cryptographic problems in general, are less well-suited for MILP solvers. Translating a given relation table such as the DDT to a set of linear inequalities is referred to as converting the V-representation (vertex representation) of a convex point set to the H-representation (half-space representation), and can be done with existing tools, but often requires heuristic optimizations to become solvable Sun et al. [SHW+14a; SHW+14b].

Dedicated tools. Before the wide-spread adoption of comfortable off-the-shelf general-purpose solvers for cryptanalysis, characteristics and solutions were usually either found by hand or by dedicated tools. In this context, we refer to software tools as “dedicated” if the most time-consuming parts of the software are not off-the-shelf libraries such as SAT solvers, but code written by the cryptanalyst for this specific application, with the particular requirements and structures of cryptanalytic attacks in mind. Prominent examples include the widely-reused branch-and-bound algorithm of Matsui [Mat94] for finding the best linear approximations and differential characteristics of DES-like ciphers, more recently also extended to related-key characteristics [BN10] and ARX designs [BV14; BVL16]. Even now, dedicated tools often have significant advantages compared to general-purpose tools and may succeed in solving problems infeasible for the latter, at the cost of a higher implementation effort.

A particularly active area is the differential analysis of hash functions and unkeyed primitives. One reason is that the lack of keys allows the cryptanalyst to better steer the process of developing characteristics and solving the resulting equations. Additionally, practical results such as practical collisions for currently-used hash functions are (unfortunately) both within computational grasp [SBK+17] and of global impact due to their reusability, without repeating expensive computations. Examples include several tools developed for the analysis of the primitives underlying the MD5/SHA-1/SHA-2 family [SO06; DR06; SLW07; MNS11b; MNS13b], competitors in the SHA-3 competition [Leu12; Leu13], and SHA-3/Keccak [DV12; MDV17].

2.2.4. Generalizations and Related Concepts

So far, we focused on classical differential cryptanalysis close to the original approach of Biham and Shamir. However, none of the attacks presented in this thesis directly follows this classical approach. One of the main factors contributing to the lasting success and popularity of differential cryptanalysis in cryptographic research is the generality and flexibility of the underlying ideas. Indeed, by generalizing and varying different aspects of the approach, many fundamental ideas in symmetric cryptanalysis can be (naturally or artificially) cast in the light of differential cryptanalysis. In the following, we briefly discuss some related concepts that are relevant for the main part of this thesis.

Difference notions and partial knowledge

In Section 2.2.1, we already saw that the notion of differences can be defined with respect to other group operations [LMM91]. In addition, it has proven fruitful to consider not only one or all individual characteristics for one fixed differential, but more general collections of characteristics.

Truncated differences and multiple differentials. The idea to use several characteristics for an attack is almost as old as differential cryptanalysis itself. In the simplest case, all characteristics contribute to the same differential [BS91], and the differential's probability is estimated based on the sum of the characteristics' probabilities. Blondeau and Gérard [BG11] and Blondeau et al. [BGN12] analyze a more complex setup with characteristics for *multiple* (different) *differentials*, where the attacker essentially efficiently performs several differential attacks in parallel.

Knudsen [Knu94] observed that it can often be much easier and similarly useful to consider differences that are not fully specified, but only specified for selected bits, while the exact difference in the remaining bits is disregarded: *truncated differentials*. This is particularly useful if the selected bits can be evaluated deterministically due to incomplete diffusion, but the approach also applies to probabilistic transitions.

In practice, the most successful application of this concept is to strongly aligned ciphers, where the truncation is done on cell level: Each multi-bit cell is either required by the truncated characteristic to have zero difference or it can have arbitrary difference (which may or may not include zero).

Truncated characteristics are not only useful as a distinguisher based on their estimated probability, but also as an intermediate step when optimizing the number of active S-boxes while searching or bounding individual characteristics (Section 2.2.3). Consequently, there are two relevant probability metrics associated with a truncated characteristic of this kind: First, (bounds on) the probability of the best individual characteristic that is compatible with the truncated constraints, based on the number of active S-boxes and compared with the generic probability of observing a fixed output difference; and second, (estimates for) the collective probability of all compatible characteristics, averaged over all compatible input differences. The latter usually depends on transitions of the linear layer (such as MixColumns in AES) and is compared with the generic probability of the truncated output difference.

Modular differences, signed differences, and generalizations. If a cryptographic primitive features integer additions modulo 2^b, it is natural to consider the *modular difference* $x^* \boxminus x \in \mathbb{Z}_{2^b}$. A prime example is the analysis of hash functions of the MD/SHA family, starting with the cryptanalysis of MD4 by Dobbertin [Dob96; Dob98]. However, modular differences are inconvenient when considering the other bitwise operations of the primitive, since a modular difference can correspond to one of several different bitwise xor differences, and vice versa. For example, for $b = 8$, a modular difference of $\Delta^{\boxminus} = +1 = \texttt{01}$ may correspond to any xor difference $\Delta^{\oplus} \in \{\texttt{01}, \texttt{03}, \texttt{07}, \dots, \texttt{FF}\}$. Conversely, an xor difference of $\Delta^{\oplus} = \texttt{01}$ corresponds to a modular difference $\Delta^{\boxminus} \in \{\pm 1\} = \{\texttt{01}, \texttt{FF}\}$.

Wang et al. [WY05; WLF+05] introduced *signed differences* as a convenient generalization that uniquely determines both modular and xor differences: For each bit position i, the signed difference $\Delta^{\pm} \in \{0, +1, -1\}^b$ specifies if the corresponding bit pair is equal with $(x_i^*, x_i) \in \{(0,0),\ (1,1)\}$, or has a positive $((x_i^*, x_i) = (1,0))$ or negative $((x_i^*, x_i) = (0,1))$ difference. The signed difference $\Delta^{\pm}$ implies an xor difference of $\Delta^{\oplus} = (|\Delta_i^{\pm}|)_i$ and a modular difference of $\Delta^{\boxminus} = \sum_i 2^{\Delta_i^{\pm}} \pmod{2^b}$.

This notion was further generalized for the context of dedicated search tools for characteristics of ARX primitives. There it is useful on the one hand to capture additional constraints across bits (e.g., *two-bit conditions* [MNS11b]); on the other hand, one can consider less constrained descriptions of a family of characteristics as an intermediate result on the way to a fully specified characteristic or pair (e.g., *generalized conditions* [DR06]).

Distinguishing differences

Classical differential cryptanalysis is interested in differentials for almost the full cipher with an unusually high probability, and recommends the key candidate with the highest observed incidence of the output difference.

Impossible differential cryptanalysis, introduced by Knudsen [Knu98] and Biham et al. [BBS99a; BBS05], does the opposite and rejects all key candidates that lead to an output difference known to be impossible. Such impossible differentials can be found, for instance, with the miss-in-the-middle approach [BBS99b] by concatenating two inconsistent probability-1 characteristics (one forward, one backward). Evaluating the complexity of such an attack is however non-trivial and apparently error-prone, and generic complexity estimates have been proposed and discussed recently [BNS14; Der16; Blo17]. As a generalization of both classical and impossible differentials, as well as sets like in truncated or multiple differentials, Albrecht and Leander [AL12] propose to analyze and distinguish the entire *multi-dimensional distribution* of output differences for a set of input differences, which is only feasible for ciphers with a very small block size.

Higher-order differentials consider the difference of more than two paired messages. The idea was first introduced by Lai [Lai94] and follows naturally from the definition of the *derivative* $\Delta_\alpha f(x)$ (Section 2.2.1) by iteratively applying this differential operator. The d-th order derivative of the (vectorial) Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ by $\alpha_1, \ldots, \alpha_d \in \mathbb{F}_2^n$ is

$$\Delta^{(d)}_{\alpha_1,\ldots,\alpha_d} f(x) := \Delta_{\alpha_d} \cdots \Delta_{\alpha_1} f(x) = \bigoplus_{\alpha \in A} f(x \oplus \alpha)\,,$$

where the sum ranges over all 2^d elements of the span

$$A = \langle \alpha_1, \ldots, \alpha_d \rangle = \{\lambda_1\alpha_1 \oplus \ldots \oplus \lambda_d\alpha_d \mid \lambda \in \mathbb{F}_2^d\}\,.$$

As a consequence of the product rule, the algebraic degree of $\Delta^{(d)} f$ is

$$\deg \Delta^{(d)}_{\alpha_1,\ldots,\alpha_d} f(x) \leq \deg f - d\,.$$

In particular, if $d \geq \deg f + 1$, then for any offset x, $\bigoplus_{\alpha \in A} f(x \oplus \alpha) = 0$.

This property has given rise to several different cryptanalytic approaches. Knudsen [Knu94] applies it to break the cipher CRADIC and thus already demonstrates that a cipher with excellent properties with respect to classic differentials can be very susceptible to higher-order differential attacks.

Another direct application is in building *zero-sum (partition) distinguishers* for cryptographic primitives with a relatively low algebraic degree, i.e., sets of inputs $(x \oplus A)$ such that both the input set and the resulting output set sum to zero [AM09]. If the complexity is less than generic approaches [BDPV10a], then these can either serve directly as distinguishers for unkeyed or known-key functions, or as a basis for key recovery.

Cube attacks as proposed by Dinur and Shamir [DS09] and related attacks [Vie07] use a different key recovery approach. Instead of considering the case that (deterministically) $\Delta^{(d)} = 0$, they target the case that $\Delta^{(d)}$ is (probably) a linear function that depends on some key bits. This linear function is first recovered in a precomputation phase and then evaluated in an offline phase to learn one bit per *cube* (A, x). While the first cube attacks focused on classical stream ciphers, more recent improvements like dynamic, conditional, or borderline cube attacks frequently target Keccak and related schemes due to its degree-2 S-box.

Another notable, but very different second-order approach is the *boomerang attack* by Wagner [Wag99], which uses classical differential characteristics to obtain a second-order differential property. Its first victim was another cipher, COCONUT'98, based on decorrelation theory [Vau98], with "provable security" against classic differential cryptanalysis. Differential characteristics can not only be chained with other differential characteristics as in boomerang attacks, but for example with linear characteristics as in the *differential-linear attack* of Langford and Hellman [LH94], later applied to the same cipher [BDK02]. The approach was recently analyzed more closely [BLN14; BLN17] based on the known duality results between differential and linear cryptanalysis [CV94]. More generally, this duality [Mat94; Bih94b; CV94; DGV94a] pervades much of the history of symmetric design and analysis of the past two-and-a-half decades.

Other distantly related attacks that work with (Λ-, δ-)sets of plaintexts and their collective characteristics include the *Square* [DKR97] and *integral attacks* [KW02] as well as some *meet-in-the-middle attacks* [DS08]. Another example related to higher-order differentials is *(impossible) polytopic analysis* [Tie16].

The fact that an observed differential (partially) determines the corresponding solutions for a fixed, known permutation with whitening keys has generally been widely used, from theoretical generic attacks on permutation-based primitives [Dae91] down to practical, non-cryptanalytic "cheating" attacks like differential fault attacks [BS97].

Part I.

Differential Cryptanalysis of Novel Designs

Introduction to Part I

Since the original publication of differential cryptanalysis by Biham and Shamir [BS90; BS91; BS93], intense research has led to a common understanding of how to secure a block cipher against this attack vector. An important result of these efforts is the wide-trail design strategy, applied in the design of AES, by Daemen [Dae95; DR01]. It provides a way to make rigorous statements about a cipher's security against certain attacks based on two statistics: The maximum differential probability of its S-box, and the minimum number of active S-boxes over several rounds.

This classical approach has proven reliable and efficient for general-purpose block ciphers. However, new challenges for symmetric design arise from novel requirements and application scenarios. For example, there is an increasing demand for permutations and tweakable block ciphers instead of block ciphers, since they appear better suited for some higher-level modes of operation. In addition, applications with severe performance constraints call for very lightweight primitives with reduced security margins and non-ideal building blocks. To benefit from the large corpus of existing analysis, novel designs may also re-use existing building blocks in new configurations, and try to borrow the corresponding security arguments.

In this part, we analyze the impact of these constraints and design trends on the ciphers' security. We present four attacks on novel designs that show how challenging the design of modern symmetric primitives can be. The first two attacks on the tweakable block cipher MANTIS (Chapter 3) and the permutation Simpira (Chapter 4) show the difficulties of extending the wide-trail design strategy to these primitives: We demonstrate differential attacks that break the designers' security claims, even though both ciphers come with provable bounds against differential cryptanalysis. Third, we present a higher-order differential attack on the (slightly) round-reduced block cipher LowMC (Chapter 5). The final result targets not the primitive, but the permutation-based mode of operation of the authenticated cipher Prøst-OTR (Chapter 6), which permits simple related-key attacks. While this result does not threaten the cipher's security, it shows an unexpected consequence of composing existing, well-analyzed building blocks, similar to the first two attacks. All our results underline the need for intense individual scrutiny of the security margin of novel designs, even if bounds or existing analysis are available.

3 Key Recovery for MANTIS

In this chapter, we analyze the tweakable block cipher MANTIS, published at CRYPTO 2016 by Beierle et al. [BJK+16]. We show that its lightweight round function does not interact well with the α-reflective construction and tweakey schedule. This allows us to cluster many good differential characteristics and thus achieve much higher differential probabilities. A practical implementation recovers the full 128-bit key of family member MANTIS_5 in about an hour with 2^{30} chosen plaintexts, which violates the designers' security claim.

The results in this chapter are based on joint work with Christoph Dobraunig, Daniel Kales, and Florian Mendel. I am the main author and proposed the central cluster of characteristics, significant parts of the key recovery approach, and the details of the analysis. The following text is a significantly extended version of the paper published and selected among the best three papers at FSE 2017 [DEKM17]. Some of the additional analysis is also part of an online preprint with followup work, which extends the attack to MANTIS_6 [EK17].

3.1. Introduction

Tweakable block ciphers generalize the concept of block ciphers by adding an additional public input, the tweak. This tweak plays a role similar to the nonces or initialization values of higher-level modes of operation, and provides additional variation of the individual instances of the cipher family. The concept was formally introduced by Liskov et al. [LRW02], who defined it as a family of permutations $\tilde{E} : \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n$. $\tilde{E}$ maps a k-bit key K, t-bit tweak T and n-bit plaintext M to an n-bit ciphertext C, such that $\tilde{E}(K,T,\cdot)$ is a permutation. The recent popularity of tweakable block ciphers, for instance in the CAESAR competition, shows that tweakable block ciphers may be more naturally suited as building blocks for higher-level modes of operation than block ciphers.

A particularly relevant application area for tweakable block ciphers is memory and disk encryption, where the address of each data item defines the tweak. However, generic constructions to turn block ciphers $E(K, M)$ into secure tweakable block ciphers $\tilde{E}(K, T, M)$ are often not well-suited for such applications, since they incur a significant latency overhead compared to a plain block cipher call and/or come with considerable security loss due to the birthday bound. This motivates the design of dedicated tweakable block ciphers with low latency.

Compared to generic constructions that use some block cipher as a black box, dedicated constructions try to provide more efficient designs with full security by integrating the tweak in the core primitive design. With the TWEAKEY framework, Jean, Nikolić, and Peyrin [JNP14b] propose to treat the tweak in almost the same way as the key in a key-alternating construction. This approach, and in particular the special case STK with its linear tweak schedule, has been adopted in several CAESAR candidates (Deoxys, Joltik, KIASU), as well as standalone tweakable block cipher designs like SKINNY and MANTIS [BJK+16] or QARMA [Ava17].

Regarding the cryptanalytic implications of this approach, one central aspect is the possibility of related-tweak attacks. The tweak is usually assumed to be under the attacker's control, although in practice, the definition of the mode of operation that uses the cipher may impose some constraints. In particular, this means that the attacker can introduce differences via the key schedule, similar to related-key attacks on classical block ciphers. This increases the number of rounds necessary for security against differential cryptanalysis, as well as certain other attacks [DEM16b], such as integral distinguishers or Meet-in-the-Middle attacks.

For designers, this means that they must analyze bounds for the differential probability in the related-key model. Standard search approaches for finding or lower-bounding the best characteristics, such as mixed-integer linear programming (MILP), satisfiability (SAT) or constraint programming (CP) solvers, can usually be adapted to the related-tweak case. The output of such a search is either an optimal differential characteristic or, more often, a truncated differential characteristic with the minimum number of active S-boxes, referred to as "minimal characteristic" in the following. For standard strongly aligned block ciphers in the fixed-key model, the bounds derived from such a minimal characteristic are usually both reasonably tight and reasonably reliable to estimate the security margin. However, several recent papers have discussed issues which indicate that the bounds obtained from minimal characteristics of STK-based tweakable block ciphers can be less useful. The main reason for this is the deterministic behavior of the linear tweak schedule with respect to the input tweak difference. For example, Cid et al. [CHP+17] show that if this is not considered in the search, the resulting minimal characteristics are often invalid, and that tighter bounds can be obtained by adapting the search model accordingly.

MANTIS is a TWEAKEY/STK-based tweakable block cipher published at CRYPTO 2016 by Beierle, Jean, Kölbl, Leander, Moradi, Peyrin, Sasaki, Sasdrich and Sim [BJK+16]. To optimize the design for low-latency implementations, the designers use the same α-reflective structure as PRINCE [BCG+12; BCKL17], but combine it with the round function of Midori [BBI+15]. According to their analysis [BJK+16], this improves both the latency and the security compared to the original PRINCE, since Midori's variant of ShiftRows is designed to provide a higher bound on the minimum number of active S-boxes.

The recommended version MANTIS_7 has $14 + 2$ rounds (7 forward, 7 backward, 2 reduced inner rounds), but the authors also give a reduced security claim for the $10+2$-round version, MANTIS_5. They claim security against practical attacks, which they define as related-tweak attacks with data complexity 2^d less than 2^{30} chosen plaintexts (or 2^{40} known plaintexts), and computational complexity at most 2^{126-d} block cipher calls based on the underlying FX construction with whitening keys [KR96; KR01], similar to PRINCE [BCG+12].

Our contributions

We propose a key-recovery attack against MANTIS_5 with 2^{28} chosen plaintexts and a computational complexity of about 2^{38} block cipher calls, which violates this security claim. We verified the validity of the attack in a practical implementation. The implementation revealed an additional differential property of the Midori S-box that complicates some steps of the attack due to differentially equivalent keys. An adapted version of the attack recovers the full key in about 1 core hour using 2^{30} chosen plaintexts.

The attack exploits several specific properties of the MANTIS design, but the general approach is relatively generic and may be useful for the analysis of other tweakable block ciphers. Specific properties of MANTIS that we exploit include the lightweight near-MDS mixing layer and certain differential properties of the involutive S-box, both inherited from Midori. These properties make it relatively easy to find a differential characteristic with the claimed optimal probability in the related-tweak setting. Using the same properties, this differential characteristic can then be expanded to a family of characteristics with a corresponding initial structure that makes efficient use of the low data complexity limit of only 2^{30} chosen plaintexts. Furthermore, the choice to keep the original Midori order of linear operations (first permute, then mix) makes the PRINCE-like middle rounds differentially less effective than the ordering used by PRINCE (first mix, then permute). Midori's order preserves a 'Superbox' structure over 4 S-box layers in the middle rounds, instead of 2.

The general approach explores the middle ground between classical differential characteristics and truncated differential characteristics. Our aim is to pick the best of both worlds for the context of tweakable block ciphers with a linear tweak schedule. We consider not only one, but many clustered characteristics to improve the overall probability and generate pairs more efficiently compared to the single best differential characteristic. On the other hand, a straightforward truncated approach cannot take advantage of the high-probability transitions in the S-box, incurs significant costs from the linear constraints imposed by the tweak schedule, and does not provide a fixed output difference that can be used, for instance, for boomerang attacks.

Outline

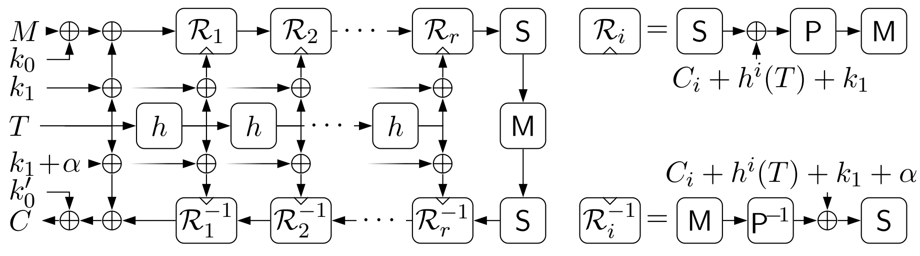
In Section 3.2, we provide a brief description of the tweakable block cipher MANTIS and some of its cryptographic properties. In Section 3.3, we introduce a family of differential characteristics and a corresponding initial structure of messages for MANTIS_5 that lead to a good filter after 9.5 rounds. In Section 3.4, we use this initial structure and filter to mount a multi-staged key recovery attack on MANTIS_5. Finally, we discuss the results of a practical implementation of the attack and its applicability to MANTIS_7 in Section 3.5.

3.2. Description of MANTIS

3.2.1. The Tweakable Block Cipher MANTIS

MANTIS was published at CRYPTO 2016 by Beierle et al. [BJK+16]. The designers propose several variants MANTIS_r that differ only in the number of rounds. All variants operate on a 64-bit message block $M = M_0\|M_1\|\cdots\|M_{15}$ and work with a 64-bit tweak $T = T_0\|T_1\|\cdots\|T_{15}$ and $(64+64)$-bit key $K = (k_0, k_1)$. All 64-bit values are mapped to 4×4 states S of 4-bit cells S_j, where $S_0, \ldots, S_3$ is the first row, etc.

The cipher's structure is similar to PRINCE, with r forward rounds $\mathcal{R}_i$ and r backward rounds $\mathcal{R}_{2r+1-i} = \mathcal{R}_i^{-1}$, separated by an involutive, unkeyed middle layer $\mathsf{S} \circ \mathsf{M} \circ \mathsf{S}$. The 64-bit subkey k_1 is used as round key for the outer forward and backward rounds, while the other 64-bit subkey k_0 and the derived $k_0' = (k_0 \ggg 1) + (k_0 \gg 63)$ serve as whitening keys. The tweak T is added together with k_1 in every round according to the TWEAKEY construction, with a simple cell permutation h as a tweak schedule. The construction is illustrated in Figure 3.1a.

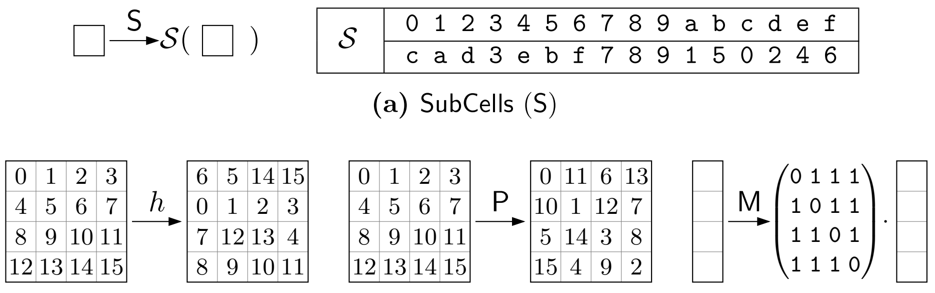


(a) PRINCE-like α-reflective cipher structure **(b)** Midori-like round function

Figure 3.1.: Design of the tweakable block cipher MANTIS_r.

3.2.2. The Round Functions $\mathcal{R}_i$ and $\mathcal{R}_i^{-1}$

The round function $\mathcal{R}_i$ is very closely related to that of Midori [BBI+15]. It updates the 4×4 state of 4-bit cells by means of the sequences of transformations $\mathcal{R}_i$ and $\mathcal{R}_i^{-1}$, as illustrated in Figure 3.1b. Its S-box layer (SubCells) and linear layer (PermuteCells, MixColumns) are directly inherited from Midori [BBI+15]. In the following, we briefly describe the individual operations. For a more detailed description of the MANTIS family, we refer to the design paper [BJK+16].



$\mathcal{S}$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	c	a	d	3	e	b	f	7	8	9	1	5	0	2	4	6

(a) SubCells (S)

(b) Tweak permutation h. **(c)** PermuteCells (P). **(d)** MixColumns (M).

Figure 3.2.: Transformations of the MANTIS round function $\mathcal{R}_i$.

- SubCells (S) applies the involutive 4-bit S-box $\mathcal{S}$ given in Figure 3.2a to each state cell. For our attack, we are primarily interested in the differential behavior of $\mathcal{S}$. The differential distribution table (DDT) in Figure 3.3a shows that $\mathcal{S}$ has 24 differential transitions with a probability of 2^{-2}; for two of the input differences, 2 and a, each of the four possible output differences is observed with probability 2^{-2}. This is due to the algebraic properties of $\mathcal{S}$: only 12 of the 15 component functions have algebraic degree 3.

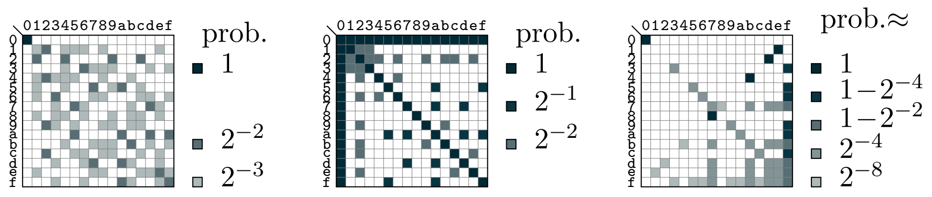
  We also define the set transition function $\sigma : 2^{\mathcal{X}} \to 2^{\mathcal{X}}$ to describe the differential behavior of $\mathcal{S}$ for sets $X \subseteq \mathcal{X} = \{\mathtt{0}, \dots, \mathtt{f}\}$ of input differences: $\sigma(X) := \{y \in \mathcal{X} \mid \exists x \in X : \mathrm{DDT}[x, y] > 0\}$.

  In addition to the first derivative $\mathcal{S}_a(x) := \mathcal{S}(x) + \mathcal{S}(x+a)$ of the S-box, as tabulated in the DDT, we will also refer to some properties of the second derivative $\mathcal{S}_{a,\tau}(x) := \mathcal{S}_a(x) + \mathcal{S}_a(x+\tau)$, in particular the case $\mathcal{S}_{a,\tau} = 0$ as tabulated in the differential invariance table (DIT) in Figure 3.3b.

- AddTweakey$_i$ (A) and AddConstant$_i$ (C) add the round constant C_i, the subkey k_1 (for $\mathcal{R}_i$) or $k_1 + \alpha$ (for $\mathcal{R}_i^{-1}$), and the round tweakey $h^i(T)$ to the state. The tweakey update function h simply permutes the order of cells as specified in Figure 3.2b.

- PermuteCells (P) permutes the state cells as specified in Figure 3.2c.

- MixColumns (M) multiplies columns with involutive near-MDS matrix M over $\mathbb{F}_{2^4}$ given in Figure 3.2d, whose truncated differential behavior per column is illustrated in Figure 3.3c.



(a) SubCells: $\mathrm{DDT}[a,b]$ $\mathbb{P}_x[\mathcal{S}_a(x)=b]$ **(b)** SubCells: $\mathrm{DIT}[a,\tau]$ $\mathbb{P}_x[\mathcal{S}_a(x)=\mathcal{S}_a(x+\tau)]$ **(c)** MixColumns: DDT (Truncated)

Figure 3.3.: Differential properties of MANTIS transformations.

3.3. A Family of Differential Characteristics

We first revisit the designers' security analysis and then propose a family of single-key, related-tweak differential characteristics for MANTIS_5.

3.3.1. Bounds and Security Claim

The designers of MANTIS analyze the security of the cipher against differential cryptanalysis by modeling the differential behavior (truncated to state cells) as a mixed-integer linear program [BJK+16]. They analyzed the minimum number of active S-boxes for different round numbers, both in a fixed-tweak and a related-tweak setting. The design document provides lower bounds for full and round-reduced MANTIS.

For MANTIS_5, the minimum number of active S-boxes in the related-tweak setting is 34 (for the full MANTIS_7: 50), and the maximum differential probability of the S-box is 2^{-2}. The designers conclude that "no related tweak linear or differential distinguisher based on a characteristics is possible for MANTIS_5" [BJK+16]. In particular, they claim that MANTIS_5 is secure against "practical attacks", here defined as related-tweak attacks with data complexity 2^d at most 2^{30} chosen plaintexts (or 2^{40} known plaintexts), and computational complexity at most 2^{126-d}.

Our attack is based on a truncated differential characteristic for the related-tweak setting that meets this lower bound of 34 active S-boxes, and on an optimal differential characteristic whose differential probability meets the corresponding upper bound. However, instead of considering only a single fixed input difference and differential characteristic, we will cluster several related differential characteristics following the same truncated differential characteristic, thus obtaining a much better probability.

3.3.2. Finding an Optimal Differential Characteristic

To find optimal differential characteristics, we first model the truncated differential behavior of MANTIS in a related-tweak setting as a mixed-integer linear program (MILP). Then, we take advantage of the differential properties of SubCells and MixColumns to find differential characteristics that follow this truncated differential characteristic.

MILP models of truncated differential characteristics

The MILP model uses the following decision variables for r-round MANTIS_r, indexed by their round $i \in \mathcal{R} = \{1, \dots, r\}$, forward or backward direction $\pm \in \{+, -\}$, where $i^\pm \in \mathcal{R}^\pm$ is shorthand for $(i, \pm) \in \mathcal{R} \times \{+, -\}$, and cell position $b \in \mathcal{B} = \{0, \dots, 15\}$, or state row $x \in \mathcal{X} = \{0, \dots, 3\}$ and column $y \in \mathcal{Y} = \{0, \dots, 3\}$, such that $b = 4x + y$ (Figure 3.4):

- $\alpha_i^\pm[b] \in \{0, 1\}$ for $i^\pm \in (\mathcal{R} \cup \{0\})^\pm$, $b \in \mathcal{B}$: Truncated difference of cell S_b of the input and output state of SubCells in forward round $\mathcal{R}_i$ or backward round $\mathcal{R}_i^{-1}$.
- $\beta_i^\pm[b] \in \{0, 1\}$ for $i^\pm \in \mathcal{R}^\pm$, $b \in \mathcal{B}$: Truncated difference of cell S_b of the output state of $\mathsf{AddTweakey}_i$ in forward round $\mathcal{R}_i$ (or the input state in backward round $\mathcal{R}_i^{-1}$).
- $\tau[b] \in \{0, 1\}$ for $b \in \mathcal{B}$: Truncated difference of cell S_b of the tweak T. Note that we do not need to model the differences in the plaintext m and in the ciphertext c to optimize the number of active S-boxes.

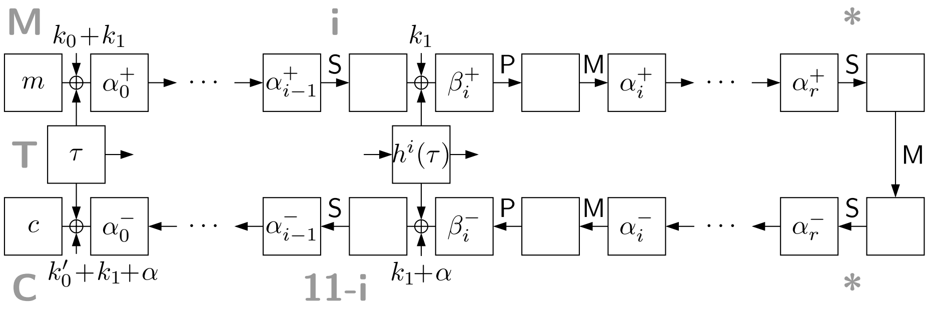
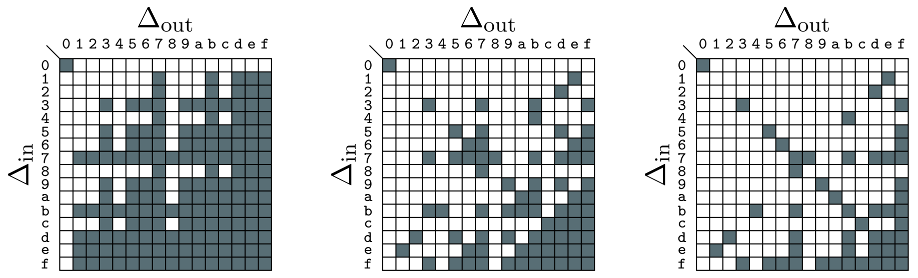


Figure 3.4.: MILP variables for a truncated differential model of forward/backward rounds $\mathcal{R}_i/\mathcal{R}_i^{-1}$, $1 \le i \le r$ and inner rounds of MANTIS_r.



(a) Branch number model **(b)** Xor model **(c)** Exact model

Figure 3.5.: Approximations of the truncated DDT of MixColumns.

In addition, we need helper variables to model the behavior of the linear functions AddTweakey and MixColumns. Note that the MILP model must be linear over $\mathbb{R}$ ($+$ denotes integer addition in the MILP inequalities), whereas the linear layers are linear over $\mathbb{F}_2$ ($+$ or $\oplus$ denotes xor).

The easiest approach is to reduce the behavior of all relevant linear building blocks to their differential branch numbers, in particular the 8-variable MixColumns to its branch number 4 and its bijectivity, and 3-variable AddTweakey to its branch number 2. A corresponding helper variable for each such linear function denotes whether the individual linear functions are differentially active, i.e., if any of the involved cells is active:

- $\chi_i^{\pm}[b] \in \{0,1\}$ for $i^{\pm} \in \mathcal{R}^{\pm}$, $b \in \mathcal{B}$: Activity helper variable for truncated xor additions $\beta_i^{\pm} = \alpha_{i-1}^{\pm} \oplus h^i(\tau)$.
- $\mu_{i,y}^{\pm}, \mu_y^{*} \in \{0,1\}$ for $i^{\pm} \in \mathcal{R}^{\pm}$, $y \in \mathcal{Y}$: Activity helper variables for MixColumns of column y in rounds $\mathcal{R}_i$ and $\mathcal{R}_i^{-1}$ (helpers $\mu_{i,y}^{\pm}$) and in the inner round (helpers μ_y^{*}).

However, as illustrated in Figure 3.5a, this would allow many impossible transitions. In particular, a large proportion of the transitions with exactly 4 active cells are impossible for MixColumns. To restrict the MILP solver to the relevant transitions, we need to add more linear inequalities to our model of MixColumns. In general, translating a given relation table such as the DDT to a set of linear inequalities is also referred to as converting the V-representation (vertex representation) of a convex point set to the H-representation (half-space representation). Sun et al. [SHW+14b; SHW+14a] discuss general approaches to solve this problem or approximate it efficiently. In our simple case of binary MixColumns, efficient solutions can easily be found by hand.

A possible solution is to explicitly write down the 8-variable matrix multiplication as a set of 4 xor additions with 4 variables each, modeled in turn by their branch number. The resulting approximation of the truncated DDT is illustrated in Figure 3.5b. However, this is still not a precise model, since it allows transitions from 2 to 3 active cells. Due to the binary coefficients, these transitions are impossible. To obtain an exact model, observe that the valid MixColumns transitions from truncated difference $a = (a_x)_{x\in\mathcal{X}}$ to $b = (b_x)_{x\in\mathcal{X}}$ are exactly those that satisfy one of the following two constraints $C_1 \vee C_2$ (Figure 3.5c):

$$\sum_{x\in\mathcal{X}} a_x + \sum_{x\in\mathcal{X}} b_x \geq 6 \quad \vee \quad \forall x \in \mathcal{X} : a_x \oplus b_x = \bigoplus_{x'\in\mathcal{X}} a_{x'}.$$

Here, $+$ again denotes addition in $\mathbb{R}$, whereas $\oplus$ is xor addition in $\mathbb{F}_2$ and thus needs to be modeled by separate constraints. Additional helper variables denote which of all these constraints are satisfied:

- $\nu^{\pm}_{i,y,c}, \nu^{*}_{y,c} \in \{0,1\}$ for $i^{\pm} \in \mathcal{R}^{\pm}$, $y \in \mathcal{Y}$, and $0 \leq c \leq 10$: Various condition helper variables for the exact model of MixColumns of column y in rounds $\mathcal{R}_i$ and $\mathcal{R}_i^{-1}$ (helpers $\nu^{\pm}_{i,y,c}$) and in the inner round (helpers $\nu^{*}_{y,c}$). Indices $c \in \{0,\dots,3\}$ indicate if condition C_2 is partially satisfied for $x = c$. Indices $c \in \{4,\dots,7\}$ are activity helper variables for the corresponding xor equations. Index $c = 8$ indicates if condition C_1 is satisfied. Indices $c \in \{9,10\}$ are helper variables for the right-hand side of condition C_2.

Using this exact model of MixColumns, the truncated differential behavior of MANTIS$_r$ in related-tweak settings can be modeled by the constraints and objective function given in Figure 3.6.

For MANTIS$_5$, the final model has 1025 variables and 1056 inequalities (excluding bounds for binary variables). Solving this optimization problem for MANTIS$_5$ yields several truncated differential characteristics matching the lower bound of 34 active S-boxes as claimed by the designers. Two examples, one with 34 and one with 36 active S-boxes, are given in Figure 3.7. A noteworthy property of both solutions is that every active MixColumns exactly matches its branch number of 4 and thus satisfies constraint C_2. Furthermore, in each addition of active tweak cells to the state in AddTweakey, two active cells always cancel. The xors of the truncated model thus all behave like bit xors. We will take advantage of this property in the following. Although the solution of Figure 3.7a has fewer active S-boxes, it is the solution of Figure 3.7b that we will use in the remaining chapter.

$$\min \sum_{i^\pm \in (\mathcal{R} \cup \{0\})^\pm} \sum_{b \in \mathcal{B}} \alpha_i^\pm[b] \qquad \text{(active S-boxes)}$$

$$\text{s.t.} \quad \forall b \in \mathcal{B}, i^\pm \in \mathcal{R}^\pm : \qquad (\mathsf{AddTweakey}_i)$$

$$2\chi_i^\pm[b] \le \alpha_{i-1}^\pm[b] + h^i(\tau)[b] + \beta_i^\pm[b] \le 3\chi_i^\pm[b]$$

$$\forall y \in \mathcal{Y}, i^\pm \in \mathcal{R}^\pm : \qquad (\mathsf{PermuteCells}_i,\ \mathsf{MixColumns}_i)$$

$$4\mu_{i,y}^\pm \le \sum_{x \in \mathcal{X}} \big(\mathsf{P}(\beta_i^\pm)[4x+y] + \alpha_i^\pm[4x+y]\big) \le 8\mu_{i,y}^\pm$$

$$\mu_{i,y}^\pm \le \sum_{x \in \mathcal{X}} \mathsf{P}(\beta_i^\pm)[4x+y], \qquad \mu_{i,y}^\pm \le \sum_{x \in \mathcal{X}} \alpha_i^\pm[4x+y]$$

$$4 \le 4\nu_{i,y,8}^\pm + \sum_{x \in \mathcal{X}} \nu_{i,y,x}^\pm$$

$$6\nu_{i,y,8}^\pm \le \sum_{x \in \mathcal{X}} \big(\mathsf{P}(\beta_i^\pm)[4x+y] + \alpha_i^\pm[4x+y]\big)$$

$$2\nu_{i,y,8}^\pm = \sum_{x \in \mathcal{X}} \mathsf{P}(\beta_i^\pm)[4x+y] - 2\nu_{i,y,9}^\pm - \nu_{i,y,10}^\pm$$

$$\nu_{i,y,10}^\pm = 2\nu_{i,y,x+4}^\pm + 1 - \nu_{i,y,x}^\pm - \mathsf{P}(\beta_i^\pm)[4x+y] - \alpha_i^\pm[4x+y] \qquad \forall x \in \mathcal{X}$$

$$\forall y \in \mathcal{Y} : \qquad \text{(Inner } \mathsf{MixColumns}\text{)}$$

$$4\mu_y^* \le \sum_{x \in \mathcal{X}} \big(\alpha_r^+[4x+y] + \alpha_r^-[4x+y]\big) \le 8\mu_y^*$$

$$\mu_y^* \le \sum_{x \in \mathcal{X}} \alpha_r^+[4x+y], \qquad \mu_y^* \le \sum_{x \in \mathcal{X}} \alpha_r^-[4x+y]$$

$$4 \le 4\nu_{y,8}^* + \sum_{x \in \mathcal{X}} \nu_{y,x}^*$$

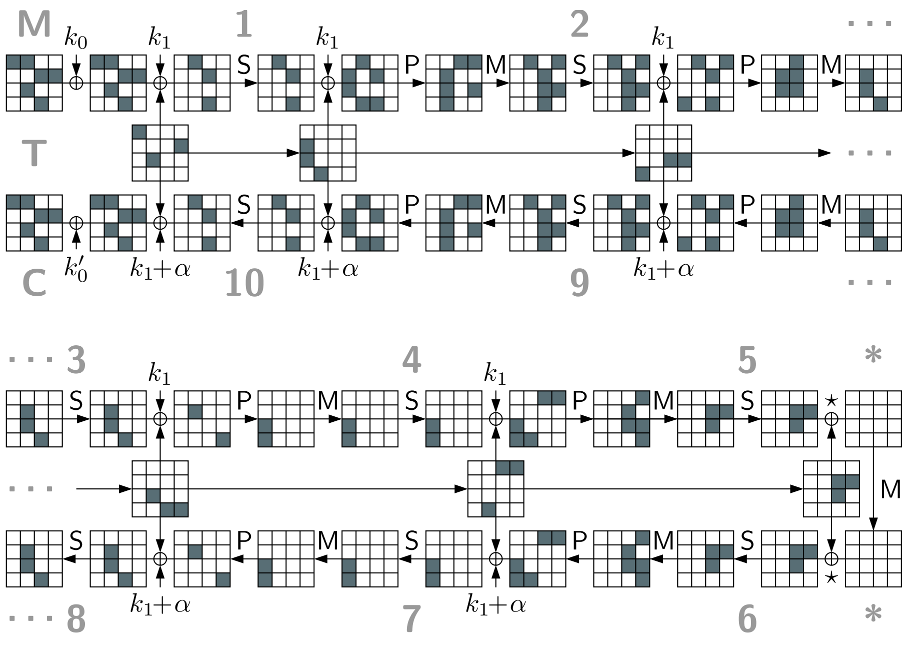
$$6\nu_{y,8}^* \le \sum_{x \in \mathcal{X}} \big(\alpha_r^+[4x+y] + \alpha_r^-[4x+y]\big)$$

$$2\nu_{y,8}^* = \sum_{x \in \mathcal{X}} \alpha_r^+[4x+y] - 2\nu_{y,9}^* - \nu_{y,10}^*$$

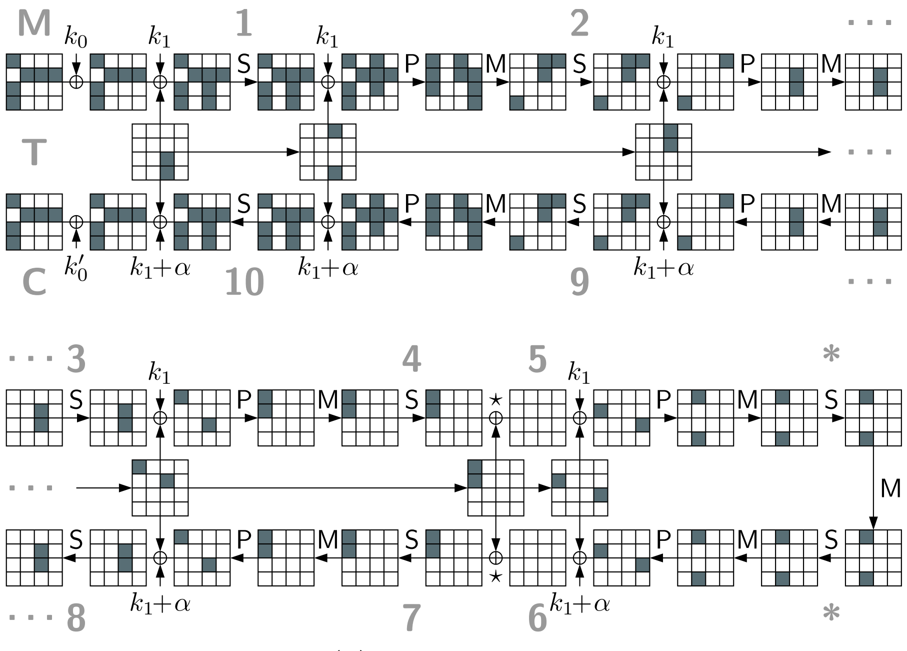
$$\nu_{y,10}^* = 2\nu_{y,x+4}^* + 1 - \nu_{y,x}^* - \alpha_r^+[4x+y] - \alpha_r^-[4x+y] \qquad \forall x \in \mathcal{X}$$

$$1 \le \sum_{b \in \mathcal{B}} \alpha_0^+[b] + \sum_{b \in \mathcal{B}} \tau[b] \qquad \text{(Non-triviality)}$$

Figure 3.6.: Truncated differential MILP model of MANTIS_r.



(a) 34 active S-boxes



(b) 36 active S-boxes

Figure 3.7.: Two truncated differential characteristics for MANTIS_5.

An optimal differential characteristic.

We can now find (bitwise) differential characteristics that follow the previously found truncated differential characteristics. Since MixColumns only has binary coefficients, all transitions that match the branch number of 4 for MixColumns ($1 \to 3$, $2 \to 2$, $3 \to 1$) and 2 for AddTweakey are possible when all active cells have the same fixed difference. For SubCells, this means we need a high-probability differential fixed point. There are 4 candidates with probability $\frac{1}{4}$: $\{\mathtt{a}, \mathtt{b}, \mathtt{e}, \mathtt{f}\}$ (Figure 3.3a). We pick $\mathtt{a}$ and obtain a differential characteristic with the optimal probability $2^{-34 \cdot 2} = 2^{-68}$ from the truncated characteristics with 34 active S-boxes, and $2^{-36 \cdot 2} = 2^{-72}$ with 36 (Figure 3.8).

3.3.3. Clustering Good Differential Characteristics

We will now relax some of these constraints, and also consider characteristics with cell differences other than $\mathtt{a}$ in selected sections of the characteristic. Our goal is to generalize the (near-)optimal differential characteristic into a family of characteristics that is similarly easy to use for key recovery but can take much better advantage of the limited available data.

As an example, consider the inner round $\mathsf{S} \circ \mathsf{M} \circ \mathsf{S}$ between Rounds 5, 6 (Figure 3.8). We want to find other differential characteristics compatible with the given truncated differential characteristic and the fixed tweak difference. AddTweakey in Rounds 5 and 6 implies that any compatible characteristic must have a difference of $(\mathtt{a}, 0, 0, \mathtt{a})^\top$ in column 2 of the input and output of $\mathsf{S} \circ \mathsf{M} \circ \mathsf{S}$. So far, we only considered one characteristic with probability 2^{-8} for $\mathsf{S} \circ \mathsf{M} \circ \mathsf{S}$:

$$(\mathtt{a}, 0, 0, \mathtt{a})^\top \xrightarrow{\mathsf{S}} (\mathtt{a}, 0, 0, \mathtt{a})^\top \xrightarrow{\mathsf{M}} (\mathtt{a}, 0, 0, \mathtt{a})^\top \xrightarrow{\mathsf{S}} (\mathtt{a}, 0, 0, \mathtt{a})^\top .$$

Based on the DDT of SubCells and the definition of MixColumns, any of the following 4 characteristics with $x \in \sigma(\{\mathtt{a}\}) = \{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\}$ can be used equivalently:

$$(\mathtt{a}, 0, 0, \mathtt{a})^\top \xrightarrow{\mathsf{S}} (x, 0, 0, x)^\top \xrightarrow{\mathsf{M}} (x, 0, 0, x)^\top \xrightarrow{\mathsf{S}} (\mathtt{a}, 0, 0, \mathtt{a})^\top .$$

In a similar way, we can describe the family of all possible characteristics consistent with the previous truncated differential characteristic with 36 active S-boxes and with the fixed tweakey difference. Due to the linear

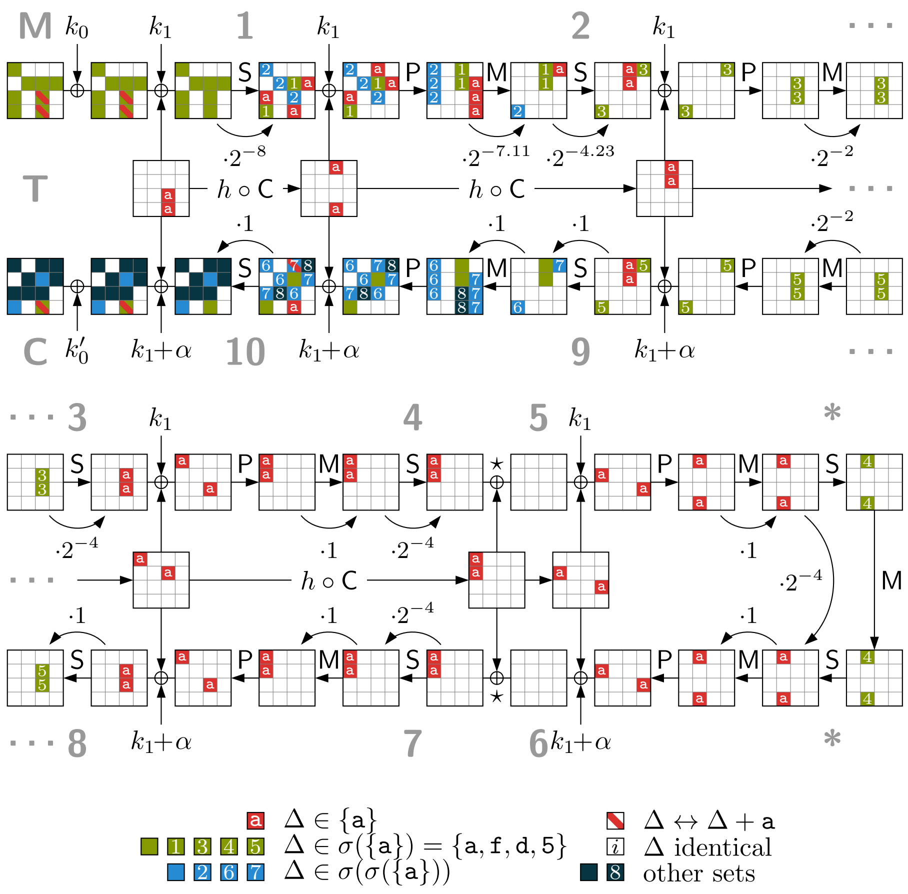


Figure 3.8.: Family of differential characteristics for MANTIS_5.

tweakey schedule, all round-tweakey differences are also fixed, which imposes many constraints on the active cells. Whenever the round tweakey cancels or introduces a difference in the state, it forces an active cell to a (marked 🟥). This is further propagated by SubCells to $\{5, a, d, f\}$ (marked 🟩) and by MixColumns, which requires equality of all differences in transitions that match the branch number (marked with identical identifiers ▢). All characteristics for Rounds 2 to 9 enumerated this way are possible and have optimal probability (see Figure 3.8).

More generally speaking, we specified a semi-truncated characteristic [EK17]. This is a family of characteristics that is characterized by a set $\chi_i \subseteq \mathcal{X}$ of possible cell-wise differences for each active state cell S_i, $0 \leq i \leq 15$, based on a truncated differential characteristic, plus some intra-state and inter-state equality constraints. Then the semi-truncated characteristic is the subset of characteristics in the Cartesian product of all χ_i that satisfy the equality constraints.

First and last rounds.

For a concrete attack, we can configure the sets χ^M of the message input state M and χ^C of the ciphertext output state C, and adapt the states of the neighboring first and last rounds accordingly. In particular, it can be helpful to slightly adapt the truncated characteristic and allow slightly more active S-boxes. The goal of this configuration is to balance the trade-off between the costs of different attack phases, which are influenced by the number of characteristics, their average probability, and the structure of χ^M and χ^C.

For the output C, we define the sets of Round 10 such that all operations starting from SubCells of Round 9 follow the family deterministically.

For the input M, we choose a trade-off with 4 differences per cell in 8 input cells. For Round 1, this selection permits all differences that are also compatible with the previously defined sets for Round 2 and the truncated characteristic.

In summary, our final semi-truncated characteristic is a set of characteristics that contains (a superset of) all differential characteristics that

- have non-zero expected differential probability,
- comply with the starting truncated characteristic with 36 active S-boxes, except for modifications in the message M, Round 10, and the ciphertext C,
- comply with the fixed difference for the tweak T with active cells $(\mathtt{a}, \mathtt{a})$, and
- start with an input difference in the reduced specified set χ^M.

It should be noted the final semi-truncated characteristic no longer includes only possible, optimal characteristics, but also many with lower or zero probability. In the following, we assume that all sets χ_i are reduced to contain only differences that are reachable from both neighboring states [EK17]. Compared to the original characteristic [DEKM17], Figure 3.8 is slightly improved accordingly: Rounds 1 and 2 permit more differences, whereas the sets in Round 10 are refined to exclude a few unreachable differences.

Probability of the family of characteristics.

The probability of a semi-truncated characteristic is defined as the sum of probabilities of all compatible differential characteristics for a fixed input difference, averaged over all compatible input differences. Similarly, the probability of a semi-truncated differential is defined as the probability that any compatible output difference is observed for a fixed input difference, averaged over all compatible input differences. As usual, we will assume that the probability of an individual differential characteristic (for the fixed target key) can be estimated based on the expected differential probability (across all long-keys), which is in turn computed by multiplying the differential probabilities of each round for a Markov cipher.

Instead of enumerating all individual characteristics, we want to estimate the probability directly round by round based on the semi-truncated description. The relevant round operations for evaluating the semi-truncated probability are SubCells (as for individual characteristics) and MixColumns (as for truncated characteristics); the other operations are trivial if the semi-truncated characteristic is reduced [EK17].

In the original paper [DEKM17], we estimated the probability of the family of characteristics cell-by-cell for each round independently. For each round, we consider all differentials and average their probabilities. For example, in Round 2, the individual probability of the SubCells transitions in S_2, S_3, S_6, S_{12} is $2^{-2} \cdot 1 \cdot 2^{-2} \cdot (2^{-1-0} + 2^{-1-1}) \approx 2^{-4.4}$. The individual probability of the MixColumns transition in Round 2 depends on the number of active input cells and set size, and is 2^{-2}. Overall, with the original characteristic, we estimated an overall probability of $2^{-40.51}$. In this straightforward computation we made two Markovian assumptions:

(a) **Uniformity of values:** For each individual characteristic, we make the usual Markov assumption that the input values to SubCells are uniformly distributed; and

(b) **Uniformity of differences:** By using the definition of the probability of a semi-truncated differential and averaging over all input differences, we make a similar uniformity and independence assumption regarding the distribution of the differences in each round among the compatible characteristics.

In the case of MANTIS, the first assumption seems reasonable except in the inner part, which features two successive SubCells layers without a key addition in between. For the specific semi-truncated characteristic used for MANTIS_5, the second assumption is also well-justified in most rounds, for example due to the uniform distribution of the message input or the most frequent transitions with 4 equiprobable differentials. However, in general – and in Round 2 in particular – this assumption does not apply.

To obtain a more accurate estimate, we consider not only the set of differences at each step, but their expected distribution among all compatible, consistent differential characteristics that contribute to the probability. Consider an intermediate state S with semi-truncated characteristic χ. The difference in this state for a random compatible plaintext pair is a random variable $\Delta = (\Delta_0, \dots, \Delta_{15})$. We write $\Delta \in \chi$ for the event $\Delta_i \in \chi_i$ for all i, and $\overline{\Delta} \in \overline{\chi}$ to state that all intermediate differences in the steps up to and including S follow the semi-truncated characteristic for a particular input pair. We are interested in the distribution of Δ in case $\overline{\Delta} \in \overline{\chi}$, and specifically, in the cell-wise conditional distribution defined by the probability mass function φ_i:

$$\varphi_i \ : \qquad \mathcal{X} \to [0,1], \qquad \delta \mapsto \mathbb{P}\big[\Delta_i = \delta \mid \overline{\Delta} \in \overline{\chi}\big].$$

Now consider an operation $f \in \{\mathsf{S}, \mathsf{A}, \mathsf{P}, \mathsf{M}\}$ that is applied to the input state S to produce the output state $S^f := f(S)$. We want to derive the conditional distribution φ_i^f of Δ^f and estimate the probability $\overline{p}^f$ of the semi-truncated characteristic up to this state:

$$p^f = \mathbb{P}\big[\overline{\Delta}^f \in \overline{\chi}^f \mid \overline{\Delta} \in \overline{\chi}\big], \qquad \overline{p}^f = \mathbb{P}\big[\overline{\Delta}^f \in \overline{\chi}^f\big] = p^f \cdot \mathbb{P}\big[\overline{\Delta} \in \overline{\chi}\big].$$

As an intermediate step, we consider the distribution of Δ^f without the constraints χ^f, i.e., $\tilde{\varphi}_i^f$ under the condition $\overline{\Delta} \in \overline{\chi}$ instead of φ_i^f under $\overline{\Delta}^f \in \overline{\chi}^f$ (so $\varphi_i^f(\delta) = 0$ for $\delta \notin \chi_i^f$):

$$\tilde{\varphi}_i^f \ : \qquad \mathcal{X} \to [0,1], \qquad \delta \mapsto \mathbb{P}\big[\Delta_i^f = \delta \mid \overline{\Delta} \in \overline{\chi}\big].$$

For $\mathsf{AddTweakey}$ and $\mathsf{PermuteCells}$, we trivially get $p^f = 1$, and $\tilde{\varphi}_i^f = \varphi_i^f$ is a permuted φ_i. For $\mathsf{SubCells}$, let $\mathbb{P}[\alpha \xrightarrow{\mathcal{S}} \delta]$ denote the differential probability of (α, δ) obtained from the DDT of S-box $\mathcal{S}$. Furthermore, let $\mathbb{1}_{\chi_i}$ denote the indicator function of χ_i: If $\delta \in \chi_i$ then $\mathbb{1}_{\chi_i}(\delta) = 1$, else $\mathbb{1}_{\chi_i}(\delta) = 0$. If we assume that the distributions φ_i are independent, then

$$\tilde{\varphi}_i^{\mathsf{S}}(\delta) = \sum_{\alpha \in \chi_i} \varphi_i(\alpha) \cdot \mathbb{P}\big[\alpha \xrightarrow{\mathcal{S}} \delta\big]\,, \qquad p_i^{\mathsf{S}} = \sum_{\delta \in \chi_i^{\mathsf{S}}} \tilde{\varphi}_i^{\mathsf{S}}(\delta)\,,$$

$$\varphi_i^{\mathsf{S}}(\delta) = \mathbb{1}_{\chi_i^{\mathsf{S}}}(\delta) \cdot \frac{\tilde{\varphi}_i^{\mathsf{S}}(\delta)}{p_i^{\mathsf{S}}}\,, \qquad p^{\mathsf{S}} = \prod_i p_i^{\mathsf{S}}\,.$$

For MixColumns, the distribution needs to be evaluated column by column for each $I \in \mathcal{I}$. Then, assuming the input distributions φ_i are independent, we get the following distribution φ_I^{M} of column differences $\Delta_I = (\Delta_{I_0}, \dots, \Delta_{I_3})$:

$$\tilde{\varphi}_I^{\mathsf{M}}(\delta_I) = \varphi_I(\mathsf{M} \cdot \delta_I) = \prod_j \varphi_{I_j}([\mathsf{M} \cdot \delta_I]_j)\,, \qquad p_I^{\mathsf{M}} = \sum_{\delta_I \in \chi_I^{\mathsf{M}}} \tilde{\varphi}_I^{\mathsf{M}}(\delta_I)\,,$$

$$\varphi_I^{\mathsf{M}}(\delta_I) = \mathbb{1}_{\chi_I^{\mathsf{M}}}(\delta_I) \cdot \frac{\tilde{\varphi}_I^{\mathsf{M}}(\delta_I)}{p_I^{\mathsf{M}}}\,, \qquad p^{\mathsf{M}} = \prod_I p_I^{\mathsf{M}}\,.$$

Based on the column-wise distribution for all differences $\delta_I = (\delta_{I_0}, \dots, \delta_{I_3})$, we obtain the (dependent) cell distributions $\varphi_{I_j}^{\mathsf{M}}$:

$$\varphi_{I_j}^{\mathsf{M}}(\delta) = \sum_{[\delta_I]_j = \delta} \varphi_I^{\mathsf{M}}(\delta_I)\,.$$

For the special case of meeting the branch number bound with $a \in \{1, 2, 3\}$ active input cells and $4 - a$ active output cells, all active cells share the same set χ_*. Then, all active output cells will also share an identical (dependent) distribution φ_*^{M}. For example, in the simplest case that the input cells are also identically (independently) distributed by φ_*:

$$p_I^{\mathsf{M}} = \sum_{\delta \in \chi_*} [\varphi_*(\delta)]^a\,, \qquad \varphi_*^{\mathsf{M}}(\delta) = \mathbb{1}_{\chi_*}(\delta) \cdot \frac{[\varphi_*(\delta)]^a}{p_I^{\mathsf{M}}}\,.$$

Clearly, the independence assumptions required at each step will usually not be satisfied. On the other hand, maintaining a full-state distribution φ for each state is not practicable. As a practical compromise, we consider the dependencies φ_I^{M} introduced by MixColumns in the next SubCells, but assume that the following PermuteCells "clears" the dependencies [EK17]:

$$\tilde{\varphi}_I^{\mathsf{S}}(\delta_I) = \sum_{\alpha_I \in \chi_I} \varphi_I(\alpha_I) \cdot \prod_j \mathbb{P}\big[[\alpha_I]_j \xrightarrow{\mathcal{S}} [\delta_I]_j\big]\,, \qquad p_I^{\mathsf{S}} = \sum_{\delta_I \in \chi_I^{\mathsf{S}}} \tilde{\varphi}_I^{\mathsf{S}}(\delta_I)\,.$$

When applying the approach to the MANTIS_5 characteristic in Figure 3.8, the overall probability is $\overline{p} = 2^{-39.34}$.

The most accurate result would, of course, be obtained with a distribution of full-state differences [CFG+14; Leu15; BDP15]. For example, Canteaut et al. [CFG+14] consider a very structured cluster for PRINCE and derive a closed form for its probability. Leurent [Leu15] gives a general approach using transition probabilities for full-state differences, but this is only feasible with a very low number of active S-boxes per step to keep the number of considered differences per step sufficiently small.

3.3.4. Exploiting Semi-Truncated Characteristics

Data Collection

Once we have fixed a semi-truncated characteristic and determined an estimate for its probability, we need to consider how to efficiently generate message pairs with a compatible input difference, and how to evaluate the resulting output differences. All these considerations depend on the size of the semi-truncated difference set relative to the total number of possible differences. For this purpose, we identify the semi-truncated difference $\chi = (\chi_0, \dots, \chi_{15})$ with the corresponding expanded set of differences $\chi_0 \times \cdots \times \chi_{15} \subseteq \mathcal{X}^{16}$. We then denote the number $|\chi|$ of differences compatible with the semi-truncated difference χ, and their ratio (or filter) $\rho(\chi)$ among all differences, by

$$|\chi| := |\chi_0 \times \cdots \times \chi_{15}| = \prod_i |\chi_i| \quad \in \left[1, |\mathcal{X}|^{16}\right]$$

$$\rho(\chi) := \frac{|\chi|}{|\mathcal{X}^{16}|} = \prod_i \rho(\chi_i) \quad \in \left[2^{-16|\mathcal{X}|}, 1\right].$$

We consider a semi-truncated characteristic with probability p and denote its plaintext-ciphertext differential and tweak difference by (χ^M, χ^C) and χ^T, respectively. Note that the tweak difference is fixed, so $\left|\chi^T\right| = \left|\{\delta^T\}\right| = 1$, whereas $\left|\chi^M\right|, \left|\chi^C\right| \geq 1$.

Plaintext pairs can be generated efficiently with initial structures similar to the case of multiple and truncated differentials: We fix a base plaintext M and base tweak T. Then, we query the ciphertexts for the set $\mathcal{M} \times \mathcal{T}$ of plaintext-tweak combinations, where the message set $\mathcal{M}$ and tweak set $\mathcal{T}$ are defined as follows:

$$\mathcal{T} = T \oplus \left\langle \chi^T \right\rangle = \{T, T \oplus \delta^T\}, \qquad \mathcal{M} = M \oplus \left\langle \chi^M \right\rangle,$$

where $\langle S \rangle$ denotes the linear span generated by a set S, i.e., the set of all linear combinations of elements in S. For each queried message in the first half $T \oplus \mathcal{M}$ of this set, there is a corresponding queried message in the second half $(T \oplus \delta^T) \oplus \mathcal{M}$ for any compatible difference $\delta^M \in \chi^M$. Thus, with $2 \cdot \left|\langle \chi^M \rangle\right|$ chosen-plaintext queries, we obtained $\left|\langle \chi^M \rangle\right| \cdot \left|\chi^M\right|$ compatible plaintext pairs. Among this set of compatible pairs, all message differences compatible with χ^M (and each χ_i^M) appear equally often, consistent with the uniform starting distribution we assumed in Section 3.3.3. We can repeat this procedure several more times with different base inputs M and T to generate pairs at a constant rate of $\left|\chi^M\right|/2$ pairs per query. This is independent of the structure of the sets χ_i^M and the resulting size of $\langle\chi_i^M\rangle$, except for the obtained granularity of the number of pairs.

If we want to generate enough pairs to expect R valid pairs compatible with the full semi-truncated characteristic, the necessary number of queries N_Q is

$$N_\mathrm{Q} = R \cdot \frac{2}{|\chi^M| \cdot p}$$

in case p^{-1} is an integer multiple of $\left|\langle\chi^M\rangle\right| \cdot \left|\chi^M\right|$, or slightly more otherwise. The resulting $N_\mathrm{P} = R/p$ ciphertext pairs can be filtered down to a much smaller number of candidates that still contains about R valid pairs based on the ciphertext difference, which must be in χ^C, resulting in a number of filtered ciphertext pairs N_F of

$$N_\mathrm{F} = R \cdot \frac{\rho(\chi^C)}{p}\,.$$

This filtering can usually be done efficiently without the need to enumerate all R/p ciphertext pairs. For example, we can select the cell positions S_i with the smallest sets χ_i^C, and repeat the following for each base input (T, M): Store the first half of the ciphertexts with tweak T in a hash table indexed by the values of the ciphertext cells C_i. Then, for each ciphertext in the second half with tweak $T \oplus \delta^T$, only check the relevant hash table entries according to χ_i^C for matches on the full output difference χ^C. Ideally, if there are sufficiently many cells with $\left|\chi_i^C\right| = 1$ (depending on the size $\left|\chi^M\right|$), then each filtered ciphertext pair can be identified with minimal amortized cost. In this ideal case, the total complexity is dominated either by the number of queries N_Q or the number of filtered ciphertext pairs N_F, both of which can be significantly smaller than $N_\mathrm{P} = R/p$.

Key Recovery

Different approaches to key recovery are possible depending on the properties of the semi-truncated characteristic, such as $\left|\chi^M\right|, \left|\chi^C\right|, p$, and the cardinalities in the initial and final intermediate rounds. The details also depend heavily on the target cipher and in particular its key schedule. In the remaining paper, we focus on an approach that combines elements of classical 0-round and 1+-round key recovery using standard differential characteristics or differentials. Below, we summarize the basic approach and possible trade-offs, but refer to Section 3.4 for a detailed description of a practical application.

We recover the full key in three phases, where the first phase usually dominates the attack complexity. Note that in this paper, we target a cipher with a key size twice as large as the block size, and also essentially more than twice as large as the key size that the attacker can brute-force, so it is not sufficient to just recover a few key bits and brute-force the rest. We assume we have generated a set of N_{F} filtered ciphertext pairs that contains at least one valid pair compatible with the semi-truncated characteristic, as described above.

In the first phase, we will try to identify this valid pair and recover parts of the initial and final round keys in the process. To this end, we guess parts of the initial and final round key and test for each filtered pair if the resulting intermediate values are compatible with the characteristic. We only keep round key candidates that produce valid intermediate values for at least one characteristic. To estimate how many partial key guesses produce valid intermediate values for a fixed pair, we will assume that the filtered differentials are distributed uniformly among (χ^M, χ^C). Then, we use the same methods as in Section 3.3.3 for estimating probabilities: for the initial rounds, we reuse the probability estimates for the relevant parts of the characteristics; for the final rounds, we compute estimates in essentially the same way, but based on the inverse round function. This phase reduces the space of key candidates for each cell or column, and can be repeated to (almost) uniquely determine the relevant round key values, as well as identify the valid pair.

In the second phase, we repeat a similar approach to test more conditions of the characteristic and recover more key material. Since we only need to test for one or a few valid pairs instead of all N_{F} filtered pairs, we can simultaneously guess larger parts of the key and thus cover more initial and final rounds. Finally, in the third phase, we brute-force the remaining key-space.

As a trade-off to balance the complexities arising from N_{Q} and N_{F}, we can consider minor adjustments of the semi-truncated characteristic. If N_{F} dominates the complexity, we can restrict χ^M in order to exclude the lowest-probability characteristics in the set and thus increase p. As an effect, the product $\left|\chi^M\right| \cdot p$ will slightly decrease (since we excluded several previously valid pairs), leading to a slight increase in the data complexity N_{Q}. Another negative effect is that the first rounds of the cipher will provide a slightly less effective filter for key recovery. On the other hand, N_{F} and the resulting complexity costs for key recovery will be significantly decreased.

3.4. Practical Key Recovery Attack on MANTIS$_5$

We can now use the family of characteristics from Section 3.3 to recover the two 64-bit secret keys k_0 and k_1 of MANTIS$_5$. We first discuss how to collect filtered pairs efficiently in 3.4.1, propose a step-by-step key recovery approach in 3.4.2, and finally verify the attack with a practical implementation in 3.5.1.

3.4.1. Generating and Filtering Enough Pairs

In this section, we will generate 2^{42} plaintext pairs for the differential family of Figure 3.8 and filter the ciphertext pairs down to about $2^{18.55}$ candidates using 2^{27} chosen-plaintext queries and about 2^{30} operations.

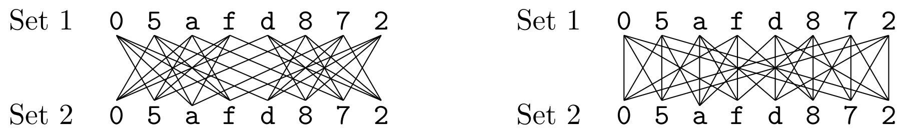
Initial Structure

We now want to generate enough message pairs to expect at least one valid pair, while staying well below the data complexity limit of 2^{30} chosen plaintexts. Obviously, the characteristic's probability is not good enough for a straightforward solution with 2^{29} suitable pairs. However, we can use the set $\{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\}$ of valid differences for each cell to our advantage.

We consider a random base plaintext-tweak pair and query two sets of derived plaintext-tweak pairs: one for the base tweak, and one for the modified tweak with a difference of `a` in two cells, as specified by the truncated differential characteristic in Figure 3.8. The first set for the base tweak contains the following 8^8 modified messages. Each of the 8 active cells (🟩, 🟥) varies over 8 values: the base plaintext plus differences $\{\mathtt{0}, \mathtt{a}, \mathtt{f}, \mathtt{5}, \mathtt{d}, \mathtt{8}, \mathtt{7}, \mathtt{2}\}$. The second set for the modified tweak contains the same 8^8 messages. In total, the number of chosen plaintext-tweak pairs we query is

$$2 \cdot \left|\langle \chi^M \rangle\right| = 2 \cdot 8^8 = 2^{25}.$$

Thus, we could repeat this up to $2^5 = 32$ times and still stay below the data complexity limit.



(a) Differences $\{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\}$ (🟩). **(b)** Differences $\{\mathtt{0}, \mathtt{5}, \mathtt{7}, \mathtt{f}\}$ (▨).

Figure 3.9.: Initial structure with $8 \cdot 4$ pairs from $2 \cdot 8$ queries per cell.

To see how many suitable pairs we can generate from these queries, note that for each value of a cell in the first set, there are exactly 4 (out of 8) values for this cell in the second set that give a valid difference $\{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\}$ (🟩) or $\{\mathtt{0}, \mathtt{5}, \mathtt{7}, \mathtt{f}\}$ (▨), as illustrated in Figure 3.9. Here, we exploited that $\mathtt{a} + \mathtt{5} = \mathtt{f}$, where all these three values are suitable for our family of characteristics. Thus, the number of pairs we get is

$$\left|\langle \chi^M \rangle\right| \cdot \left|\chi^M\right| = 8^8 \cdot 4^8 = 2^{40},$$

and the expected number of valid pairs is at least

$$2^{40} \cdot 2^{-39.34} = 2^{0.66} \approx 1.58\,.$$

We expect that the probability of observing at least one valid pair in this set is roughly 80 % [EK17].

In the following, we will use 4 repetitions $r = 1, \ldots, 4$ of the initial structure. Thus, we need to query $4 \cdot 2^{25} = 2^{27}$ chosen plaintexts with chosen tweaks in order to generate the $4 \cdot 2^{40} = 2^{42}$ plaintext pairs. This is well below the complexity limit of 2^{30} chosen plaintexts for MANTIS$_5$.

Pre-Filtering Ciphertexts for Wrong Pairs

Before we start guessing any keys, we can filter for pairs which definitely do not follow the family of characteristics given in Figure 3.8. The necessary conditions for valid ciphertext pairs are that 5 cells $(S_1, S_4, S_{11}, S_{13}, S_{15})$ have a zero difference (marked □), while the difference in cell S_{14} is in $\{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\}$ after removing the last tweak addition, marked ▨), plus some minor constraints from the other cells. If we assume that plaintext pairs which do not follow our family of characteristics produce a uniformly random difference for corresponding ciphertext pairs, these conditions are fulfilled with a probability of

$$\rho(\chi^C) \approx 2^{-5\times 4} \cdot 2^{-1\times 2} \cdot 2^{-2\times 0.299} \cdot 2^{-5\times 0.093} \cdot 2^{-2\times 0.193} \cdot 2^{-1\times 0} \approx 2^{-23.45}.$$

Hence, we reduce the set of $N_\mathrm{P} = 2^{40}$ pairs per repetition r from the initial structure to a set I_r of about $N_\mathrm{F} = 2^{40-23.45} = 2^{16.55}$ pairs. Each set I_r is still expected to contain $2^{0.66} > 1$ valid pairs that follow the family of characteristics of Figure 3.8.

Complexity and optimizations. A naive implementation of generating and pre-filtering pairs costs $4 \cdot 2^{40}$ state xor operations. However, instead of enumerating all valid pairs and then filtering for matches on 5 cells, it is much more efficient to reverse the process and only generate the relevant pairs as follows. Store each plaintext-tweak-ciphertext of Set 1 in a data structure of 2^{20} partitions, partitioned according to the value of the 5 pre-filter cells $S_1, S_4, S_{11}, S_{13}, S_{15}$. The expected size of each partition is about 2^4. Then, for each plaintext-tweak-ciphertext of Set 2, iterate only over the 2^4 candidates in the correct partition, and check whether the input difference is valid and the difference of output cell S_{14} is valid. The set I_r of remaining filtered pairs is the same, but the computational complexity is reduced to less than 2^{30} state xor operations.

3.4.2. Recovering the Key Step-by-Step

Recovery of 44-bit $k_0' + k_1$

Using these filtered pairs, we can now recover the full key in several steps. The first step of the attack is the partial recovery of 44 bits of the final whitening key $k_0' + k_1$. We want to check our key guesses against the characteristic after AddTweakey in Round 10 (see Figure 3.10):

(C1) Cell S_{14} (marked $\boxed{\texttt{a}}$) must have difference $\texttt{a}$.

(C2) Cells S_0, S_5, S_{10} (marked $\boxed{8}$) must have the same difference.

(C3) So do cells S_2, S_7, S_8 (marked $\boxed{9}$) after compensating for the tweak.

(C4) Cells S_6 and S_{12} (marked $\boxed{6}$, $\boxed{7}$) must have differences $\{\texttt{a}, \texttt{f}, \texttt{5}, \texttt{d}\}$, and additionally, due to the properties of MixColumns, cells S_3 and S_9 (marked $\boxed{\equiv}$) will have the same difference, which is the sum of the differences of S_6, S_{12}.

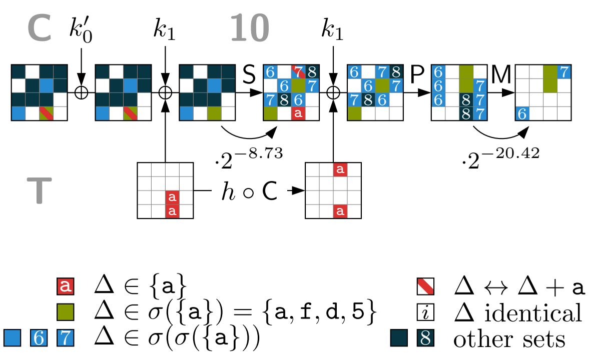


Figure 3.10.: Probability of last-round transitions for key-recovery.

The probability that a random pre-filtered ciphertext pair satisfies these constraints under a 44-bit key guess can be estimated with the same approach as in Section 3.3.3 as $2^{-8.73} \cdot 2^{-20.42} = 2^{-29.15}$.

If we now decrypt one ciphertext pair $i \in I_r$ backward for one SubCells layer under $2^{11 \cdot 4} = 2^{44}$ key guesses, we expect that $2^{44-29.15} = 2^{14.85}$ key guesses will remain which satisfy all these conditions for this ciphertext pair i. We expect the correct key guess to satisfy the conditions for at least one of the ciphertext pairs $i \in I_r$, which follows the family of characteristics in Figure 3.8. Thus, we repeat the procedure for all $2^{16.55}$ pairs and consider the union of all resulting potential key candidates. We expect at most $2^{14.85} \cdot 2^{16.55} = 2^{31.4}$ candidates for the right key guess, which effectively reduces our keyspace by $2^{-12.6}$. So, repeating the attack a total of 4 times with fresh initial structures is sufficient to recover the correct value of 44 bits of $k_0' + k_1$.

Complexity and optimizations. To get the possible key candidates per ciphertext pair, we need $2 \cdot (2^{16} \cdot 4 + 2 \cdot 2^{12} \cdot 3 + 2^4) \approx 2^{19.13}$ S-box lookups, which corresponds roughly to $2^{11.54}$ MANTIS$_5$ encryptions (based on the total number of $16 \cdot 12$ S-boxes in MANTIS$_5$). In total, we have to generate key candidates for $4 \cdot 2^{16.55}$ pairs, corresponding to a total of about $2^{30.09}$ MANTIS$_5$ encryptions.

In a straightforward implementation, we get 4 lists, each containing $2^{31.4}$ key candidates, which dominates our memory requirements. We need to find matches between the 4 lists, which adds a computational complexity of roughly $2^{31.4}$ operations, depending on the implementation.

Note that it is not necessary to guess all 44 bit of the subkey at once per ciphertext pair $i \in I_r$. Instead, we can split up the key guesses condition-wise into a 4-bit subkey for condition (C1) (with a set of valid subkey candidates of expected size $|\mathcal{C}_{14}^{(r,i)}| = 2^2$), a 12-bit subkey for (C2) ($|\mathcal{C}_{0,5,10}^{(r,i)}| = 2^4$), a 12-bit subkey for (C3) ($|\mathcal{C}_{2,7,8}^{(r,i)}| = 2^4$), and a 16-bit subkey for (C4) ($|\mathcal{C}_{3,6,9,12}^{(r,i)}| = 2^4$). The expected set of 2^{14} full key candidates per pair i is then the product set of these sub-candidates. We refer to this structured set of key candidates from repetition r and pair $i \in I_r$ as a bundle $\mathcal{B}^{(r,i)}$, where

$$\mathcal{B}^{(r,i)} = \mathcal{C}_{0,5,10}^{(r,i)} \times \mathcal{C}_{14}^{(r,i)} \times \mathcal{C}_{3,6,9,12}^{(r,i)} \times \mathcal{C}_{2,7,8}^{(r,i)}.$$

Storing all bundles requires only about $4 \cdot 2^{16.55} \cdot 10.25 < 2^{25}$ MANTIS states. To find the correct value of all 44 bits, we now need to compute

$$\bigcap_{r=1}^{4} \bigcup_{\substack{i \in I_r \\ |I_r| \approx 2^{16.55}}} \mathcal{C}_{0,5,10}^{(r,i)} \times \mathcal{C}_{14}^{(r,i)} \times \mathcal{C}_{3,6,9,12}^{(r,i)} \times \mathcal{C}_{2,7,8}^{(r,i)}.$$

The computational complexity of matching the bundles of key candidates is similar to before if the list of bundles per repetition is indexed efficiently per subkey candidate. Then, the bundles can be intersected subkey by subkey, starting with the most restrictive subkey, $\mathcal{C}_{3,6,9,12}^{(r,i)}$.

Recovery of 32-bit $k_0 + k_1$

With the help of the recovered 44 bits of $k_0' + k_1$, we can filter our plaintext pairs $i \in I_r$ so that only the valid plaintext pairs following the family of characteristics in Figure 3.8 remain. The probability that the right key mis-identifies a pair as false positive is $2^{-29.15}$. Therefore, it is likely that only correct pairs (approximately 4) remain after filtering $4 \cdot 2^{16.55}$ pairs. We now use those 4 valid pairs to recover 32 bits of the initial whitening key $k_0 + k_1$ as follows. We guess the key bits for all plaintext cells with differences, $S_0, S_5, S_6, S_7, S_8, S_{10}, S_{12}, S_{14}$. Then we can compute forward through the SubCells layer of Round 1, and check if the resulting difference pattern matches the family of characteristics. A wrong key matches the pattern with a false positive probability of about $2^{-15.11}$, or for all 4 correct pairs with about $2^{-60.44}$. Therefore, we expect that only the correct subkey remains out of the 2^{32} possible candidates.

Complexity. We make a 32-bit key guess for each of 4 pairs, leading to a total of $2 \cdot 4 \cdot 8 \cdot 2^{32} = 2^{38}$ S-box look-ups. This corresponds to about $2^{30.42}$ MANTIS$_5$ encryptions.

Recovery of k_0 and k_1

Up to this point, we have recovered 32 bits of information about $k_0 + k_1$ and 44 bits of information about $k_0' + k_1 = (k_0 \ggg 1) + (k_0 \gg 63) + k_1$. This gives us a system of 76 linearly independent linear equations for k_0 and k_1. To recover the full key, we have to guess 52 remaining bits and identify the right key using trial encryptions. Alternatively, we can also use the SubCells layers of Rounds 2 and 3 (or 9 and 8) to first recover more bits of k_1, based on the previously recovered information. Similar to recovering $k_0 + k_1$, we can apply a guess-and-determine approach to only the 4 valid pairs, for example as summarized in Table 3.1.

Complexity. The guess-and-determine approach recovers 14 of the missing 52 bits of the original 64-bit keys k_0 and k_1 with negligible computational efforts. Completing the key requires 2^{38} trial encryptions, which dominates our attack complexity.

Table 3.1.: Recovering 14 bits of k_0 and k_1.

Recovered key bits		Targeted S-box transitions
$S_0 + S_5 + S_{10}$ of k_1	(1 bit)	S_{12} in R 2 ($\boxed{2} \rightarrow \boxed{3}$)
		and/or R 9 ($\boxed{8} \leftarrow \boxed{5}$)
$S_6 + S_{12}$ of k_1	(1 bit)	S_2, S_6 in R 2 ($\boxed{1} \rightarrow \boxed{a}$)
$S_2 + S_7 + S_8$ of k_1	(4 bits)	S_3 in R 9 ($\boxed{9} \leftarrow \boxed{5}$)
$S_2, S_5, S_6, S_7, S_8, S_{12}$ of k_0, k_1		Guess 1 bit
S_3 of k_1	(4 bits)	S_6, S_{10} in R 3 ($\boxed{3} \rightarrow \boxed{a}$)
		and/or R 8 ($\boxed{5} \rightarrow \boxed{a}$)
S_9 of k_1	(4 bits)	S_2, S_6 in R 9 ($\boxed{6}, \boxed{7} \rightarrow \boxed{a}$)

Summary. In summary, the computational attack complexity is dominated by the final 2^{38} trial encryptions. The query complexity with the parameters used above is 2^{27} chosen-plaintext related-tweak queries (with many queries per tweak value), and results in an estimated success probability of 40 % of at least one valid pair per repetition.

3.5. Discussion

3.5.1. Practical Verification

A practical implementation of the key recovery attack is provided in the original paper [DEKM17] in order to verify the probability estimates and attack complexity. A first straightforward implementation revealed some additional structural properties of MANTIS that negatively affect the success probability of the attack. For this reason, we adapted some aspects of the attack in order to obtain a higher success probability in practice.

The first issue reported in the paper [DEKM17] is that the observed number of valid pairs per repetition is on average higher than expected, but the variance is relatively high: We observed several repetitions with no valid pairs, while other repetitions produced a dozen or more pairs. The first part is partially explained with the improved characteristic and more precise probability evaluation presented in this chapter, which results in numbers closer to (though still slightly below) the observed averages. The variance is a problem for the 44-bit key recovery of Section 3.4.2, which relies on finding at least 1 valid pair per repetition, but the success probability observed in practice was even worse than the estimated 40 %. There are several options to compensate for this. If memory requirements and higher runtime are not an issue, we can simply expand all bundles of key candidates and count the number of occurrences of each candidate, as done in classical key-recovery attacks using ranking statistics [SB02], which will reveal the correct candidate with very high probability. A more practical alternative is to change the initial structures per repetition to contain more structures for different plaintexts, but with fewer queries per structure, in order to decrease the variance. For example, if we use 2^5 different base plaintexts per repetition, but vary only 7 instead of 8 cells, the resulting number of pairs per repetition remains the same at $2^5 \cdot 8^7 \cdot 4^7 = 2^{40}$, but the data complexity increases slightly to $2 \cdot 2^5 \cdot 8^7 = 2^{27}$, or $4 \cdot 2^{27} = 2^{29}$ in total for all repetitions. Additionally or alternatively, we can slightly further increase this number to raise the estimated success probability above 40 %.

The second issue is that during the 32-bit key recovery of Section 3.4.2, we always find at least 2^8 possible key candidates instead of just 1, and 2 key candidates for the 44-bit subkey. Both this and the previous issue are connected to the same structural property of the MANTIS S-box. We filter our keys by checking whether the valid pairs follow the correct differential S-box transitions in Round 1, that is, $\{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\} \mapsto \{\mathtt{a}, \mathtt{f}\}$ for each cell. However, it turns out that whenever a pair of cells (x, x') follows one of these transitions, then so does $(x + \mathtt{a}, x' + \mathtt{a})$. This means that for each cell k of the correct subkey, there is an equivalent value $k + \mathtt{a}$ which also satisfies all the constraints of Round 1, leading to a total of at least 2^8 candidates. This would also increase the complexity of Section 3.4.2 accordingly. Instead of this expensive brute-force approach, we encoded the recovery of the remaining key as a Boolean satisfiability (SAT) problem.

The final adapted attack successfully recovered the full key for several tested random challenges with the original characteristic [DEKM17] in about 1 hour on a single core. The step dominating the computational complexity on paper, performing 2^{38} trial encryptions, was solved by a SAT solver in 1.5 minutes.

3.5.2. Applicability to MANTIS_6 and MANTIS_7

MANTIS_5 is the most aggressive of the proposed MANTIS family members. For their performance comparison with PRINCE, the designers list the variants MANTIS_5 through MANTIS_8, though only MANTIS_5 beats PRINCE in terms of area and latency. Explicit security claims are given for MANTIS_7 (security against related-tweak attacks with data complexity $\leq 2^n$ chosen plaintexts/ciphertexts and computational complexity $\leq 2^{126-n}$ calls) and MANTIS_5 (similar, for $\leq 2^{30}$ chosen or $\leq 2^{40}$ known plaintexts).

The characteristic of Figure 3.8 can be extended to a 6-round characteristic by appending an initial and final round [EK17]. With a careful choice of conditions in the first and last two rounds, it is possible to recover the key for MANTIS_6 with about 2^{55} chosen plaintexts and computational complexity $2^{55.5} \leq 2^{126-55}$. It is worth noting that the attack is based on a full-round characteristic with a probability of only $2^{-67.37}$, which is less than the generic expected probability of the (family of) differentials. To compensate for this, the key recovery process is intertwined with the characteristic over the final 2.5 rounds.

We did not analyze the MANTIS_7 proposal. Many of the observations and methods for MANTIS_5 also apply to MANTIS_7, which certainly casts some doubt on the design's security margin. It is relatively easy to find a very similar optimal differential characteristic with probability 2^{-100} (compared to 2^{-68} for MANTIS_5 and 2^{-88} for MANTIS_6), and to apply the same observations for clustering characteristics and improving the probability. However, a straightforward adaptation of the full key recovery attack is made more difficult by several factors. For example, it is hard to find characteristics for MANTIS_7 which on the one hand have a sufficiently low number of active S-boxes, and, on the other hand have enough active cells at the input and output to be useful for key recovery. Also, due to the small state size, the probability must be relatively high to avoid false positives among the seemingly valid pairs.

3.5.3. Applicability to QARMA

QARMA is a tweakable block cipher published by Avanzi [Ava17] around the same time as MANTIS. Despite its very recent academic publication date, it has already been announced as the standard algorithm for pointer authentication in the new ARMv8-A architecture [Bra16]. Pointer authentication is a new security feature of this architecture intended to prevent exploits based on Return-Orientated-Programming (ROP) or Jump-Orientated-Programming (JOP) [Bra16].

QARMA comes in two block sizes, QARMA-64 and QARMA-128, where QARMA-64 is strikingly similar to MANTIS: It shares the same high-level α-reflective structure (Figure 3.1a), round function structure (Figure 3.1b), and even the same permutations P and h (Figure 3.2). The security claims for 5 and 7 rounds of the construction are also the same as for MANTIS. However, noting some peculiar properties of MANTIS like the Superbox structure of the inner rounds, Avanzi also strengthens some of the other operations. In particular, the inner round of QARMA is strengthened with two additional applications of P and a key addition, and the tweak schedule is extended with an LFSR. The S-box and MixColumns matrix are also more carefully selected. While the building blocks appear stronger than those of MANTIS, they also have their downsides: The MixColumns matrix of QARMA permits related-tweak truncated-differential characteristics with fewer active S-boxes than the matrix of MANTIS, with only 30 (instead of 34 for the MANTIS matrix in either the MANTIS or) active S-boxes for 5 rounds, and 48 (instead of 52) for the full 7 rounds.

The only available third-party analysis so far was presented by Zong and Dong [ZD16]. The authors propose a meet-in-the-middle attack on 4-round QARMA (corresponds to 10 S-box layers) with 2^{53} chosen plaintexts and a computational complexity of about 2^{70} encryptions, which does not threaten the security of QARMA.

The QARMA design fixes several of the issues that we exploited for the attack on MANTIS_5: The strengthened inner round permits no Superbox property, and the new S-box, MixColumns matrix and tweak schedule do not display the same differential fixed points. This means that neither the simple optimal differential characteristic, nor the clustering effects observed in Section 3.3.3 seem applicable. On the other hand, the best truncated differential characteristic has fewer active S-boxes to begin with.

3.5.4. S-box properties of MANTIS and QARMA

In the following, we discuss some properties of the 4-bit MANTIS and QARMA S-boxes relevant for the attack. The four relevant S-boxes are specified in Figure 3.11 and include the involutive MANTIS S-box [BJK+16] borrowed from Midori [BBI+15], as well as the three proposed S-boxes for QARMA [Ava17]: Two involutive S-boxes (the first slightly more "lightweight", the second with slightly better cryptographic properties) and a third non-involutive S-box borrowed from PRINCE [BCG+12]. Some of these properties have also been discussed by the respective designers or in the context of attacks on other primitives, like the invariant subspace attack against Midori64 by Guo et al. [GJN+16].

Figure 3.11 already gives the first indication that QARMA-$\mathcal{S}_1$ and QARMA-$\mathcal{S}_2$ might have cryptographically better properties than QARMA-$\mathcal{S}_0$ and MANTIS-$\mathcal{S}$: The latter has a rather high number of 4 fixed-points $(\mathtt{3}, \mathtt{7}, \mathtt{8}, \mathtt{9})$, QARMA-$\mathcal{S}_0$ has 2 fixed-points $(\mathtt{0}, \mathtt{2})$, the others have none.

Figure 3.12 lists the algebraic normal forms (ANF) for all S-boxes. The attack on MANTIS does not explicitly depend on these ANFs; however, some of the algebraic properties are directly linked to the relevant differential properties we discuss below. Note that all four S-boxes were designed to have the algebraic degree 3, but differ in important details.

This is most obvious when considering the algebraic degree of the individual output bits, or, more generally, of the $2^4 - 1$ component functions. Figure 3.13 lists which output bits depend on which degree-3 monomials $\bar{x}_j = \prod_{\ell \neq j} x_\ell$, and reveals several sub-optimal properties of MANTIS-$\mathcal{S}$:

(a) y_2 only has degree 2,

MANTIS-$\mathcal{S}$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	c	a	d	3	e	b	f	7	8	9	1	5	0	2	4	6

(a) Involutive Sb_0 of MANTIS [BJK+16] and Midori [BBI+15].

QARMA-$\mathcal{S}_0$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	e	2	a	9	f	8	b	6	4	3	7	d	c	1	5

(b) Involutive σ_0 of QARMA [Ava17].

QARMA-$\mathcal{S}_1$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	a	d	e	6	f	7	3	5	9	8	0	c	b	1	2	4

(c) Involutive σ_1 of QARMA [Ava17].

QARMA-$\mathcal{S}_2$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	b	6	8	f	c	0	9	e	3	7	4	5	d	2	1	a

(d) Non-involutive σ_2 of QARMA, affine equivalent to S_6 of PRINCE [BCG+12].

Figure 3.11.: S-boxes of MANTIS and QARMA.

(b) y_2 does not depend on x_2,

(c) $\bar{x}_2 = x_0x_1x_3$ does not appear in any output bit, and

(d) D only has rank 2, i.e., there are only 2 linearly independent degree-3 terms involved: $\bar{x}_1 + \bar{x}_3$ and $\bar{x}_0 + \bar{x}_1 + \bar{x}_3$.

Together, and when combined with the binary linear layer of MANTIS, properties (a) and (c) have a direct influence on the maximum possible degree of multiple rounds: After 1, 2, 3, 4 rounds with a binary MixColumns, degrees in the four output bits of each S-box are limited by $(3, 3, 2, 3)$, $(8, 8, 6, 8)$ (instead of 9), $(22, 22, 16, 22)$ (instead of 27), and $(60, 60, 44, 60)$ (instead of 63), respectively.

Properties (a), (b), (c) also hold in a similar way for $(x_1, \bar{x}_1, y_1)$ of QARMA-$\mathcal{S}_0$. In contrast, both QARMA-$\mathcal{S}_1$ and QARMA-$\mathcal{S}_2$ have full rank and thus involve all degree-3 monomials; each output bit has degree 3 and depends nonlinearly on each input bit. This difference – in particular regarding the optimal or non-optimal rank – is also reflected in the second derivatives of each S-box as discussed below.

$$\begin{aligned}
y_0 &= x_0x_1x_2 + x_0x_2x_3 + x_0x_1 + x_0x_2 + x_2x_3 + 1\\
y_1 &= x_0x_1x_2 + x_0x_2x_3 + x_1x_2x_3 + x_0x_3 + x_0 + x_3 + 1\\
y_2 &= x_0x_1 + x_0x_3 + x_1x_3 + x_1 + x_3\\
y_3 &= x_0x_1x_2 + x_0x_2x_3 + x_1x_2x_3 + x_0x_3 + x_1x_3 + x_2
\end{aligned}$$

(a) MANTIS-$\mathcal{S}$

$$\begin{aligned}
y_0 &= x_0x_1x_2 + x_0x_1x_3 + x_0x_3 + x_1x_3 + x_1 + x_3\\
y_1 &= x_0x_2 + x_0x_3 + x_2x_3 + x_0 + x_3\\
y_2 &= x_0x_1x_2 + x_0x_1x_3 + x_1x_2x_3 + x_0x_1 + x_0x_2 + x_1x_2 + x_2x_3 + x_0 + x_2 + x_3\\
y_3 &= x_0x_1x_3 + x_1x_2x_3 + x_0x_2 + x_1x_2 + x_1
\end{aligned}$$

(b) QARMA-$\mathcal{S}_0$

$$\begin{aligned}
y_0 &= x_0x_1x_2 + x_0x_2 + x_1x_2 + x_1x_3 + x_2x_3 + 1\\
y_1 &= x_0x_1x_3 + x_0x_1 + x_0x_2 + x_0x_3 + x_1x_3 + x_2x_3 + x_1 + x_2 + x_3\\
y_2 &= x_0x_2x_3 + x_0x_1 + x_0x_3 + x_1x_3 + x_2x_3 + x_0 + x_3 + 1\\
y_3 &= x_1x_2x_3 + x_0x_1 + x_0x_2 + x_1x_3 + x_2x_3 + x_0 + x_1 + x_3
\end{aligned}$$

(c) QARMA-$\mathcal{S}_1$

$$\begin{aligned}
y_0 &= x_0x_1x_2 + x_0x_1x_3 + x_0x_2x_3 + x_0x_1 + x_0x_3 + x_2x_3 + x_0 + x_3 + 1\\
y_1 &= x_0x_1x_2 + x_0x_2x_3 + x_0x_2 + x_1x_2 + x_1 + x_3\\
y_2 &= x_0x_1x_3 + x_0x_2x_3 + x_1x_2 + x_2x_3 + x_1 + x_2 + 1\\
y_3 &= x_0x_1x_2 + x_0x_2x_3 + x_1x_2x_3 + x_0x_1 + x_0x_3 + x_1x_3 + x_1 + x_2 + x_3 + 1
\end{aligned}$$

(d) QARMA-$\mathcal{S}_2$

Figure 3.12.: Algebraic normal forms of MANTIS and QARMA S-boxes, with input bits (x_0, x_1, x_2, x_3) and outputs (y_0, y_1, y_2, y_3) (MSB to LSB).

	$\bar{x}_0$	$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$
y_0		✓		✓
y_1	✓	✓		✓
y_2				
y_3	✓	✓		✓

rank $D = 2$

(a) MANTIS-$\mathcal{S}$

	$\bar{x}_0$	$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$
y_0			✓	✓
y_1				
y_2	✓		✓	✓
y_3	✓		✓	

rank $D = 3$

(b) QARMA-$\mathcal{S}_1$

	$\bar{x}_0$	$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$
y_0				✓
y_1			✓	
y_2		✓		
y_3	✓			

rank $D = 4$

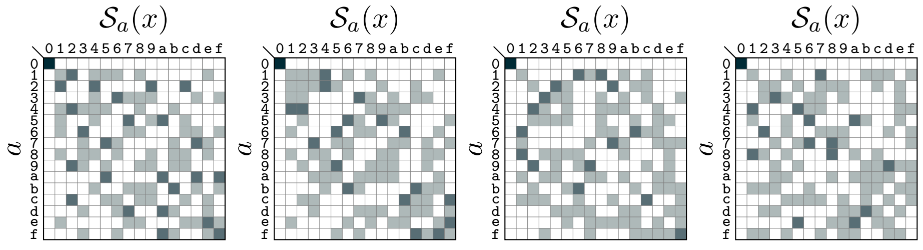
(c) QARMA-$\mathcal{S}_1$

	$\bar{x}_0$	$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$
y_0		✓	✓	✓
y_1		✓		✓
y_2		✓	✓	
y_3	✓	✓		✓

rank $D = 4$

(d) QARMA-$\mathcal{S}_2$

Figure 3.13.: Degree matrix D of MANTIS and QARMA S-boxes, where $D[y_i, \bar{x}_j]$ denotes whether the ANF of y_i contains monomial $\bar{x}_j = \prod_{\ell \neq j} x_\ell$.



(a) MANTIS-$\mathcal{S}$ (b) QARMA-$\mathcal{S}_0$ (c) QARMA-$\mathcal{S}_1$ (d) QARMA-$\mathcal{S}_2$

Figure 3.14.: Differential distribution of MANTIS and QARMA S-boxes, $\mathrm{DDT}[a,b] = \mathbb{P}_x[\mathcal{S}_a(x) := \mathcal{S}(x) + \mathcal{S}(x{+}a) = b]$. Legend: ■ 1 ▨ $\frac{1}{4}$ □ $\frac{1}{8}$.

Figure 3.14 gives the differential distribution table for each S-box. By design, all S-boxes have a maximum differential probability (and maximum absolute linear bias) of $\frac{1}{4}$, though the number of high-probability transitions ranges from 24 (MANTIS-$\mathcal{S}$) to 15 (QARMA-$\mathcal{S}_1$,$\mathcal{S}_2$). All S-boxes except QARMA-$\mathcal{S}_0$ have high-probability differential fixed-points. MANTIS-$\mathcal{S}$ even has two differences with only high-probability transitions ($\mathtt{2}, \mathtt{a}$).

Figure 3.15 highlights two interesting subgraphs of the differential-transition graphs. The first, for MANTIS-$\mathcal{S}$, is the one used in the attack of Chapter 3. It covers the set $\mathcal{A} = \{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\}$ of all possible transitions from and to $\mathtt{a}$, which includes the two high-probability fixed points $\{\mathtt{a}, \mathtt{f}\}$ as well as the sum $\mathtt{a}{+}\mathtt{5} = \mathtt{f}$. The latter two properties are relevant due to their interactions with the binary MixColumns, trivial tweak schedule, and initial structures. Overall, the probability that a starting difference chosen uniformly random from $\mathcal{A}$ is mapped by MANTIS-$\mathcal{S}$ to $\mathcal{A}$ is 50 %, but the output distribution will not be uniform. If we restrict the output to $\mathcal{A}$ and consider the corresponding random walk on the state space $\mathcal{A}$, the distribution will converge to approximately 40.3 % $\mathtt{a}$, 27.2 % $\mathtt{f}$, and 16.2 % each of $\mathtt{d}, \mathtt{5}$. The corresponding probability to map back to $\mathcal{A}$ for the first few steps is about 50 %, 68.8 %, 59.1 %, …, and converges to 62.0 %.

The second subgraph, for QARMA-$\mathcal{S}_1$, covers the set $\mathcal{A}_1 = \{\mathtt{3}, \mathtt{c}, \mathtt{f} = \mathtt{3}{+}\mathtt{c}\}$ and converges to a uniform distribution and constant probability 50 %.

In addition to the first derivative $\mathcal{S}_a(x) := \mathcal{S}(x) + \mathcal{S}(x{+}a)$ of the S-boxes, as tabulated in Figure 3.14, the attack of Chapter 3 also depends on properties of the second derivative,

$$\mathcal{S}_{a,b}(x) := \mathcal{S}(x) + \mathcal{S}(x{+}a) + \mathcal{S}(x{+}b) + \mathcal{S}(x{+}a{+}b).$$

This is most significant for the first S-box layer due to the definition of the initial structure (Figure 3.9), which chooses a set of related input pairs

$$\big\{(x_0+\tau, x_0+\tau+a) \mid a \in \{\mathtt{a},\mathtt{f},\mathtt{d},\mathtt{5}\}, \tau \in \{\mathtt{0},\mathtt{5},\mathtt{a},\mathtt{f},\mathtt{d},\mathtt{8},\mathtt{7},\mathtt{2}\}\big\}.$$

The structure of the output of the first S-box layer also influences the following S-box layers to a certain extent; and finally, the key recovery part is also based on a similar structure, rather than a fixed difference.

Figure 3.16 illustrates the most relevant property of the second derivatives: For any input difference a and offset τ, it gives the probability that adding offset τ to both values of a pair of inputs with difference a does not change the resulting output difference, i.e., the differential behavior of $\mathcal{S}$ is invariant under translation by τ. In terms of the second derivative, this is the probability that, for fixed a and τ, $\mathcal{S}_{a,\tau}(x) = 0$.

The column (or row) sums in this table for any τ depend on the number of high-probability ($\frac{4}{16}$) entries in the DDT for τ, so only MANTIS-$\mathcal{S}$ has two non-trivial values $\tau \in \{\mathtt{2},\mathtt{a}\}$ with high overall probability $\mathbb{P}_{x,a}[\mathcal{S}_{a,\tau}(x)=0]$. The most noteworthy entries are those for $a,\tau \in \mathcal{A}_\mathtt{5} = \{0,\mathtt{5},\mathtt{a},\mathtt{f}=\mathtt{5}+\mathtt{a}\}$ and $a,\tau \in \mathcal{A}_\mathtt{d} = \{0,\mathtt{d},\mathtt{a},\mathtt{7}=\mathtt{d}+\mathtt{a}\}$, which are all $\geq \frac{1}{2}$. In other words, for any $a \in \mathcal{A}_\mathtt{5}$ or $\mathcal{A}_\mathtt{d}$, and any of the at least two differences b with $\mathrm{DDT}[a,b] = \frac{4}{16}$, these 4 solutions are given by $x_0 + \mathcal{A}_\mathtt{5}$ or $x_0 + \mathcal{A}_\mathtt{d}$ for some x_0. Furthermore, for $a \in \{\mathtt{1},\mathtt{3},\mathtt{4},\mathtt{6},\mathtt{9},\mathtt{b},\mathtt{c},\mathtt{e}\}$, and the only b with $\mathrm{DDT}[a,b] = \frac{4}{16}$, these 4 solutions are given by $x_0 + \{0,a,2,a+2\}$.

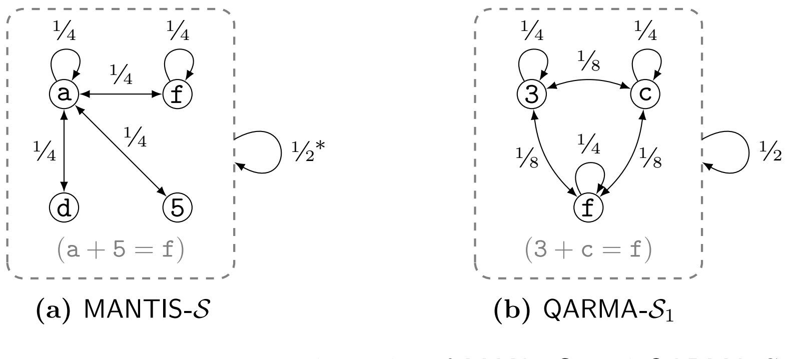
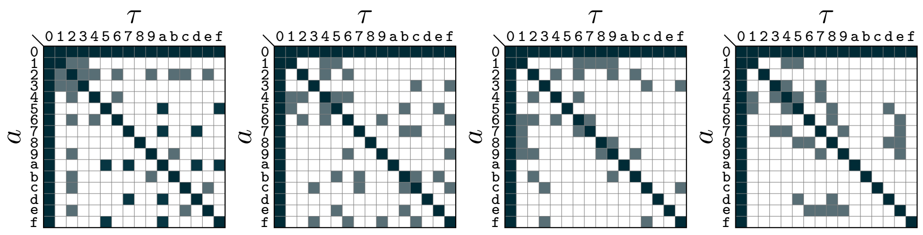


Figure 3.15.: Transition subgraphs of MANTIS and QARMA S-boxes. ½* only if input differences are uniformly selected, else converges to 62 %.



(a) MANTIS-$\mathcal{S}$ **(b)** QARMA-$\mathcal{S}_0$ **(c)** QARMA-$\mathcal{S}_1$ **(d)** QARMA-$\mathcal{S}_2$

Figure 3.16.: Differential invariance of MANTIS and QARMA S-boxes, $\mathrm{DIT}[a,\tau] = \mathbb{P}_x[\mathcal{S}_a(x) = \mathcal{S}_a(x+\tau)] = \mathbb{P}_x[\mathcal{S}_{a,\tau}(x) = 0]$. Legend: ■ 1 ▩ $\frac{1}{2}$ ▨ $\frac{1}{4}$.

3.6. Conclusion

We recover the full 128-bit key for MANTIS$_5$ with a computational complexity of about 2^{38} encryptions, memory requirements of about 2^{25} MANTIS states, and a data complexity of about 2^{27} to 2^{30} chosen plaintexts with chosen tweaks. A practical implementation recovered the correct key in about an hour based on 2^{30} chosen plaintexts. This violates the security claim for MANTIS$_5$: The designers claim resistance against attacks with computational complexity less than $2^{126-30} = 2^{96}$ encryptions based on this data complexity.

Our attack takes advantage of several very lightweight building blocks of MANTIS, most of them inherited from the Midori block cipher. This includes the involutive S-box with its high-probability differential fixed points `a` and `f`, the lightweight near-MDS matrix with its binary coefficients, and the lightweight tweakey schedule. Throughout the analysis, the symmetries of the PRINCE-like design facilitate the repeated exploitation of these properties. Another major issue is the interaction of the Midori-inspired round function with the PRINCE-inspired inner rounds, which leads to a Superbox structure over 4 S-box layers in the inner rounds. Considering all these properties, the security margin of MANTIS may be too optimistic.

4 Collisions for Simpira v1

In this chapter, we analyze the permutation Simpira v1 by Gueron and Mouha [GM16a]. We show that the designers' computer-aided security analysis does not take into consideration several dependencies between intermediate variables of the permutation. This invalidates their conclusions on the maximum probability of differential characteristics, and allows us to mount distinguishing and collision attacks on the full-round permutation. In response to the attack, the designers proposed an updated version of the design, Simpira v2, which was published at ASIACRYPT 2016 [GM16b].

The results in this chapter are based on joint work with Christoph Dobraunig and Florian Mendel. I am the main author of this contribution, pointed out the flaws in the designers' analysis, and developed the following attacks to exploit the resulting dependencies. The following text is an extended and restructured version of the paper published at SAC 2016 [DEM16a].

4.1. Introduction

The Advanced Encryption Standard AES and its underlying wide-trail design strategy are among the most popular building blocks for new symmetric designs. There are several good reasons for this. New AES-like designs profit both from the insights in efficient implementations and from the extensive cryptanalysis and well-understood security bounds of AES. In particular, if new designs not only reuse the general design ideas, but the AES block cipher itself or its round function, then Intel's AES-NI instruction set can provide high software performance on modern CPUs. However, while block ciphers are a versatile building block for other cryptographic primitives, the fixed block size of AES of 128 bits implies a certain limitation. Modern designs often require larger states for efficiency or security. Examples include permutation-based cryptography, wide-block encryption, security beyond 2^{64} inputs without resorting to beyond-birthday-security schemes, and more.

These considerations have motivated the design of numerous cryptographic algorithms based on the AES round function. Notable recent examples of dedicated designs include several authenticated encryption algorithms with excellent software performance, such as the CAESAR round-2 candidates AEGIS [WP14] and Tiaoxin [Nik15], but also more specialized primitives like the Haraka hash function for short inputs [KLMR16b]. Very recently, Jean and Nikolić [JN16] analyzed a more general family of AES-round-based building blocks that generalizes several of the previous dedicated designs. However, except for the last work, these dedicated designs target only specific state sizes, and do not offer scalable, easily reusable building blocks for other cryptographic applications.

Simpira is a recently proposed family of permutations designed by Gueron and Mouha [GM16a] that aims to fill this gap. The design goal is to provide very efficient permutations for arbitrarily large input sizes of $b \times 128$ bits, $b \in \mathbb{N}^+$, while taking advantage of the Intel AES-NI instruction set for optimized software implementations. To achieve these goals, Simpira plugs the AES round function into a generalized Feistel construction. Additionally, the designers provide computer-aided bounds for the minimum number of active S-boxes, and argue that these bounds provide security against a wide range of attack vectors. To showcase the versatility of the Simpira permutations, the designers propose a number of application scenarios, including Even-Mansour block cipher constructions, or a keyless Davies-Meyer variant for hash functions with limited-length inputs. Other applications of Simpira for beyond-birthday-bound secure authenticated encryption were proposed by Forler et al. [FLLW16].

Our contributions

We analyze members of the original Simpira v1 family [GM16a]. We show that the underlying assumptions of independence, and thus the bounds derived from the minimum number of active S-boxes, are incorrect. We focus our analysis on the family member Simpira-4 with its 512-bit state, but similar observations also apply to other family members with larger state sizes. For Simpira-4, we provide differential characteristics with only 40 (instead of 75) independently active S-boxes for the recommended 15 rounds. Based on these characteristics, we propose collision attacks on the proposed Simpira-4 Davies-Meyer hash construction. For 16 rounds of the permutation, we obtain collisions for the full 512-bit hash output with complexity $2^{110.16}$. We also adapt the attack to the originally recommended 15 rounds, providing second-order collisions and truncated collisions. We consider several truncation variants, and obtain, among others, collisions on the truncated 384-bit output with complexity $2^{110.16}$, or collisions on the 256-bit output with complexity $2^{82.62}$ – the details depend on the implemented truncation variant. These attacks violate the designers' security claims that there are no structural distinguishers below 2^{128}.

Related work

Rønjom [Røn16] independently analyzed Simpira v1, and identified invariant subspaces for any even number of rounds of Simpira-4. Both attacks on Simpira v1 exploit properties of the underlying Type-1.x Generalized Feistel Structure by Yanagihara and Iwata [YI14] and the sparse, structured round constants. In response to Rønjom's and our attacks, Gueron and Mouha proposed a new version of the design, Simpira v2 [GM16b], which replaces both the Feistel construction and the round constant schedule. In the remaining document, Simpira always refers to Simpira v1.

Simpira is not the first AES-round-based design with problematic round constants. Other examples include the analysis of the hash function Haraka [KLMR16a] by Jean [Jea16], the analysis of the withdrawn CAESAR round-1 candidate PAES [YWH+14] by Jean et al. [JNSW14; JNSW16], or the analysis of SHAvite-3 [BD09; DB09] by Peyrin [Pey09] and SHAMATA [AKKM08] by Indesteege et al. [IMPS09]. In most cases, the round constants failed to break the symmetry properties of the unkeyed AES round function. Our attack exploits different properties, such as an incomplete diffusion of differences in the structured round constants.

Outline

We first describe the Simpira family of permutations in Section 4.2, and discuss the designers' analysis in Section 4.3.1 and 4.3.2. As a result, we describe an iterative truncated differential characteristic with fewer independently active S-boxes than expected in Section 4.3.3, which serves as the basis for an 8-round differential characteristic with probability $2^{-110.16}$ in Section 4.3.3. Based on this, we propose collision attacks on the 15- and 16-round Simpira-4 hash construction in Section 4.4.1 and Section 4.4.2. Finally, we discuss Rønjom's analysis and the tweaked version Simpira v2 in Section 4.5, and conclude in Section 4.6.

4.2. Description of Simpira v1

Simpira is a family of permutations by Gueron and Mouha [GM16a]. By using the AES round function in a generalized Feistel construction, it can be adapted to any input size of $b \times 128$ bits, $b \in \mathbb{N}^+$. We refer to Simpira family members as Simpira-b.

4.2.1. Permutation and Round Function

The permutation Simpira-b keeps a state of $b \times 128$ bits. The generalized Feistel round function for $b \geq 4$, where $b \neq 6, 8$, is illustrated in Figure 4.1. The final output of Simpira-b for $b \geq 4$, $b \neq 6, 8$, is the state after $6b - 9$ such rounds. Note that if the number of rounds is not a multiple of b, the state words are output in a permuted order to allow for more efficient implementations.

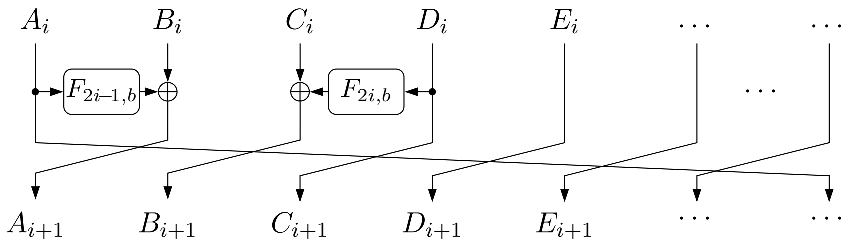


Figure 4.1.: Round function for round i of Simpira-b for $b \geq 4$, $b \neq 6, 8$.

In case of Simpira-4, we denote the 4 state words before round $i \geq 1$ by A_i, B_i, C_i, D_i, so the state update rule corresponds to

$$\begin{aligned} A_{i+1} &= F_{2i-1,4}(A_i) \oplus B_i, \\ B_{i+1} &= F_{2i,4}(D_i) \oplus C_i, \\ C_{i+1} &= D_i, \\ D_{i+1} &= A_i. \end{aligned}$$

The recommended number of rounds for Simpira-4 is 15, with output words $(B_{16}, C_{16}, D_{16}, A_{16})$.

4.2.2. F-Function

The Feistel update function $F = F_{c,b}$ applies two rounds of AES, where the Simpira family member b and the round counter c define the round constants, as illustrated in Figure 4.2.

$$\begin{tikzcd}
S \arrow[r] & \boxed{\mathsf{SB}} \arrow[r] & \boxed{\mathsf{SR}} \arrow[r] & \boxed{\mathsf{MC}} \arrow[r] & \oplus \arrow[r] & \boxed{\mathsf{SB}} \arrow[r] & \boxed{\mathsf{SR}} \arrow[r] & \boxed{\mathsf{MC}} \arrow[r] & F_{c,b}(S) \\
 & & & & C_{c,b} \arrow[u] & & & &
\end{tikzcd}$$

Figure 4.2.: AES-based F-function $F_{c,b}$ of Simpira-b.

Like for AES, the 128-bit intermediate state of F is represented as a 4×4-matrix of bytes, labelled $s_0, \ldots, s_{15}$:

$$S = \begin{array}{|c|c|c|c|}\hline s_0 & s_4 & s_8 & s_{12} \\ \hline s_1 & s_5 & s_9 & s_{13} \\ \hline s_2 & s_6 & s_{10} & s_{14} \\ \hline s_3 & s_7 & s_{11} & s_{15} \\ \hline \end{array} \;.$$

We also refer to the value at byte position s_i in state S as $S[i]$.

The operations SubBytes, ShiftRows, and MixColumns are defined identically to AES, whereas AddConstant adds counters that define an invocation counter and the value b:

- SubBytes (SB): Applies the 8-bit AES S-box $\mathcal{S}$ to each state byte.
- ShiftRows (SR): Rotates row i of the state, $0 \leq i \leq 3$, left by i bytes.
- MixColumns (MC): Multiplies each byte column of the state by the MDS-matrix M over $\mathbb{K} = \mathbb{F}_2[\alpha]/(\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1)$,

$$M = \begin{pmatrix} \alpha & \alpha+1 & 1 & 1 \\ 1 & \alpha & \alpha+1 & 1 \\ 1 & 1 & \alpha & \alpha+1 \\ \alpha+1 & 1 & 1 & \alpha \end{pmatrix} = \begin{pmatrix} \mathsf{02} & \mathsf{03} & \mathsf{01} & \mathsf{01} \\ \mathsf{01} & \mathsf{02} & \mathsf{03} & \mathsf{01} \\ \mathsf{01} & \mathsf{01} & \mathsf{02} & \mathsf{03} \\ \mathsf{03} & \mathsf{01} & \mathsf{01} & \mathsf{02} \end{pmatrix} .$$

- AddConstant (AC): In the cth invocation of F for Simpira-b, xors the following round constant $C_{c,b}$ to the state:

$$C_{c,b} = \begin{array}{|c|c|c|c|}\hline c_0 & b_0 & 0 & 0 \\ \hline c_1 & b_1 & 0 & 0 \\ \hline c_2 & b_2 & 0 & 0 \\ \hline c_3 & b_3 & 0 & 0 \\ \hline \end{array} \;.$$

In the remaining chapter, we focus on Simpira-4, so $b_0 = \mathsf{04}$ and $b_1 = b_2 = b_3 = \mathsf{00}$. Also, since the number of invocations of F is limited to 30 in Simpira-4, $c_1 = c_2 = c_3 = \mathsf{00}$. This constant is only added in the first of the two AES rounds of F, while the second round adds nothing.

To refer to intermediate states of F for an input S, we denote with $S^{\mathsf{SB1}}, S^{\mathsf{SR1}}, S^{\mathsf{MC1}}$ the outputs of first-round $\mathsf{SB}, \mathsf{SR}, \mathsf{MC}$, respectively, S^{AC} the output of first-round AC, and $S^{\mathsf{SB2}}, S^{\mathsf{SR2}}, S^{\mathsf{MC2}}$ the outputs of second-round $\mathsf{SB}, \mathsf{SR}, \mathsf{MC}$.

4.2.3. Permutation-based Hashing

Simpira's designers identify several application areas for the Simpira permutation, such as block ciphers via an Even-Mansour construction. One particular suggested application is permutation-based hashing for short inputs, where "short" means the state size of any Simpira variant. The proposal is to use a single-block, keyless Davies-Meyer-like construction with a feed-forward, and compute the hash $h(x)$ of x as

$$h(x) = \mathsf{Simpira}\text{-}b(x) \oplus x.$$

This approach provides an efficient construction for hashing inputs of limited length, which is required by many applications, such as hash-based signatures [Lam79; BHH+15].

4.3. Revisiting the Differential Bounds

In this section, we revisit the designers' security analysis of Simpira. We show that their bounds on the minimum number of differentially active S-boxes are too optimistic. In particular, we provide an iterative 2-round differential characteristic for Simpira-4 with probability $2^{-27.54}$. For Simpira-4 reduced to 8 out of 15 rounds, this yields a differential characteristic with probability $2^{-110.16}$, compared to the designers' claim of probability $\leq 2^{-25 \cdot 6} = 2^{-150}$ already after 5 rounds. These results are also relevant when round-reduced Simpira-4 is used in a keyed construction, for example as an Even-Mansour block cipher, but they do not threaten the full 15-round variant. While our analysis is focused primarily on Simpira-4, the basic observations also apply to the larger Simpira variants with the same construction approach, that is, Simpira-b with $b \geq 4$, $b \neq 6, 8$.

4.3.1. Observation on Designers' Analysis

The analysis performed by Simpira's designers [GM16a] relies on two basic bounds: full bit diffusion, and minimum number of active S-boxes. The recommended number of rounds for each variant is selected as 3 times the number of rounds necessary to prove full bit diffusion and a minimum number of 25 differentially or linearly active S-boxes. While the proofs for full bit diffusion are based on generic results on the underlying generalized Feistel construction by Yanagihara and Iwata [YI14], the bounds for active S-boxes were obtained with a Mixed-Integer Linear Programming (MILP) model [MWGP11]. For Simpira-4, both full bit diffusion and at least 25 active S-boxes are claimed to be provided by 5 rounds of the round function. For the full number of 15 rounds, this method would imply at least 75 active S-boxes.

The bound is derived under the assumption that all F-function inputs are processed independently. Of course, in an unkeyed primitive like a permutation or a hash function, the S-boxes are not really independent, since there are no random, independent round keys. Nevertheless, it is usually a reasonable assumption that the differential probabilities behave as if the values were independently random. We thus count S-boxes as independently active when it can reasonably be expected that their multiplied differential probabilities give a good estimate for the overall differential probability of the characteristic.

However, for all instances of Simpira-b with $b \geq 4$, $b \neq 6, 8$, this independence is violated by the generalized Feistel construction, and the particular definition of F. Consider the inputs to the left and right F-functions in rounds i and $i + b - 3$, respectively, that is, the inputs A_i to $F_{2i-1,b}$ and D_{i+b-3} to $F_{2(i+b-3),b}$. It is easy to see that these values are in fact identical, since $D_{i+b-3} = E_{i+b-4} = \ldots = A_{i+b-b} = A_i$. Now consider a differential characteristic where some bytes of these inputs are active. Obviously, the characteristic is only consistent if the two input differences are identical. Now recall the definition of $F = F_{c,b}$, in our case $F_{2i-1,b}$ and $F_{2(i+b-3),b}$. The only difference between $F_{2i-1,b}$ and $F_{2(i+b-3),b}$ is the round-constant addition at the end of the first AES round. This means that the inputs and outputs of the S-boxes of the first AES round must be identical as well, i.e., $A_i^{\mathsf{MC1}} = D_{i+b-3}^{\mathsf{MC1}}$. The round constant only differs in state byte s_0 for small i, so this means the S-box transitions in the second AES round will also be identical except in s_0. In fact, the outputs A_i^{MC2} of $F_{2i-1,b}$ an D_{i+b-3}^{MC2} of $F_{2(i+b-3),b}$ will have identical values except for the first column.

4.3.2. Adapted MILP Model of Differential Characteristics

According to the original designers' analysis, Simpira-4 has at least 75 active S-boxes over 15 rounds. The previous observation shows that this number might be double-counting S-boxes that operate on identical inputs. To get a more accurate bound for the number of independently active S-boxes, we adapt the MILP model of the truncated differential behavior of Simpira-4. In the adapted model, identical values will be represented by the same variable, thus avoiding both double-counting and certain inconsistent characteristics. We use an alternative description of the Simpira-4 permutation which stores only words A_i and B_i in each round i, using $C_i = A_{i-2}$ and $D_i = A_{i-1}$, as illustrated in Figure 4.3.

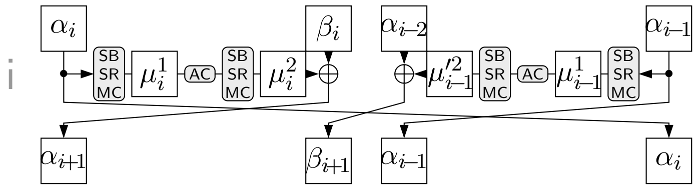


Figure 4.3.: Differential description of Simpira-4 for new MILP model.

The truncated differential model reduces the behavior of all relevant linear building blocks to their differential branch numbers, in particular the 8-variable MixColumns to its branch number 5, and 3-variable Feistel-$\oplus$ to its branch number 2. A corresponding helper variable for each such linear function denotes whether the individual linear functions are differentially active or not. As a cost function, we only count the activity of the left-hand F-functions, and only S-box s_0 for the right-hand F-functions, except in the first round.

The new MILP model uses the following variables for R-round Simpira-4, indexed by their round $i \in \mathcal{R} = \{1, \ldots, R\}$ or $\mathcal{R}_0 = \mathcal{R} \cup \{0\}$, byte position $b \in \mathcal{B} = \{0, \ldots, 15\}$, or state row $x \in \mathcal{X} = \{0, \ldots, 3\}$ and column $y \in \mathcal{Y} = \{0, \ldots, 3\}$, such that $b = 4y + x$:

- $\alpha_i[b] \in \{0, 1\}$ for $i \in \mathcal{R}_0 \cup \{-1, R+1\}$, $b \in \mathcal{B}$: Truncated difference of state byte $A_i[b] = D_{i+1}[b] = C_{i+2}[b]$.
- $\beta_i[b] \in \{0, 1\}$ for $i \in \mathcal{R} \cup \{R+1\}$, $b \in \mathcal{B}$: Truncated difference of state byte $B_i[b]$.

- $\mu_i^1[b] \in \{0,1\}$ for $i \in \mathcal{R}_0$, $b \in \mathcal{B}$: Truncated difference of state byte $A_i^{\mathsf{MC1}}[b] = D_{i+1}^{\mathsf{MC1}}[b]$.
- $\mu_i^2[b] \in \{0,1\}$ for $i \in \mathcal{R}_0$, $b \in \mathcal{B} \cup \{16,\dots,19\}$: Truncated difference of state byte $A_i^{\mathsf{MC2}}[b] = D_{i+1}^{\mathsf{MC2}}[b]$ if $4 \le b \le 15$. The first column differs, so we denote the first column of A_i^{MC2} by $0 \le b \le 3$ (or word μ_i^2), and the first column of D_{i+1}^{MC2} by $16 \le b \le 19$ (or word $\mu_i'^2$).
- $\chi_i^\alpha[b], \chi_i^\beta[b] \in \{0,1\}$ for $i \in \mathcal{R}$, $b \in \mathcal{B}$: Helper variables for truncated xor additions $\alpha_{i+1} = \mu_i^2 \oplus \beta_i$ (helper χ_i^α) and $\beta_{i+1} = \mu_{i-1}'^2 \oplus \alpha_{i-2}$ (helper χ_i^β).
- $\nu_{i,y}^1, \nu_{i,y}^2 \in \{0,1\}$ for $i \in \mathcal{R}_0$, $y \in \mathcal{Y}$ (for $\nu_{i,y}^1$) or $y \in \mathcal{Y} \cup \{4\}$ ($\nu_{i,y}^2$): Helper variables for MixColumns, indexed like μ, where column $y = 4$ corresponds to bytes $b \in \{16,\dots,19\}$.

Using these binary variables, the truncated differential behavior of R-round Simpira-4 can be modeled by the following constraints and objective function:

$$\min \sum_{i\in\mathcal{R}_0}\sum_{b\in\mathcal{B}}\big(\alpha_i[b] + \mu_i^1[b]\big) + \sum_{i\in\mathcal{R}\setminus\{R\}} \mu_i^1[0] \qquad \text{(active S-boxes)}$$

$$\text{s.t. } 5\nu_{i,y}^1 \le \sum_{x\in\mathcal{X}}\big(\mathsf{SR}(\alpha_i)[4y+x] + \mu_i^1[4y+x]\big) \le 8\nu_{i,y}^1 \qquad \forall y \in \mathcal{Y}, i \in \mathcal{R}_0$$

$$5\nu_{i,y}^2 \le \sum_{x\in\mathcal{X}}\big(\mathsf{SR}(\mu_i^1)[4y+x] + \mu_i^2[4y+x]\big) \le 8\nu_{i,y}^2 \qquad \forall y \in \mathcal{Y}, i \in \mathcal{R}_0$$

$$5\nu_{i,4}^2 \le \sum_{x\in\mathcal{X}}\big(\mathsf{SR}(\mu_i^1)[0+x] + \mu_i^2[16+x]\big) \le 8\nu_{i,4}^2 \qquad \forall i \in \mathcal{R} \setminus \{R\}$$

$$\text{(ShiftRows, MixColumns)}$$

$$2\chi_i^\alpha[b] \le \mu_i^2[b] + \beta_i[b] + \alpha_{i+1}[b] \le 3\chi_i^\alpha[b] \qquad \forall b \in \mathcal{B}, i \in \mathcal{R}$$

$$2\chi_i^\beta[b] \le \mu_{i-1}'^2[b] + \alpha_{i-2}[b] + \beta_{i+1}[b] \le 3\chi_i^\beta[b] \qquad \forall b \in \mathcal{B}, i \in \mathcal{R}$$

$$\text{(Feistel-}\oplus\text{)}$$

$$1 \le \sum_{i\in\{-2,-1,0\}}\sum_{b\in\mathcal{B}}\alpha_i[b] + \sum_{b\in\mathcal{B}}\beta_0[b] \qquad \text{(Non-triviality)}$$

Solving this optimization problem for different round numbers reveals that the number of independently active S-boxes is significantly lower than estimated by the designers. For the full 15 rounds of Simpira-4, the new model gives a bound of 40 active S-boxes, rather than the original estimate of at least 75. The minimum number of rounds necessary to achieve 25 active S-boxes according to the new model is 9 instead of 5.

4.3.3. A 2-Round Characteristic with 5 Active S-Boxes

An iterative characteristic that can be expanded to an optimal solution both for the new and the designers' original MILP model is given in Figure 4.4. It is possible to construct similar iterative characteristics with fewer active S-boxes than the designers' bounds for any $b > 4$, $b \neq 6, 8$.

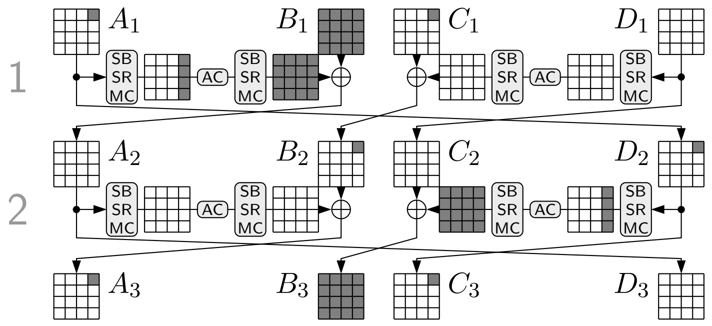


Figure 4.4.: Iterative 2-round characteristic for Simpira-4 with 5 independently active S-boxes. Notation: □ inactive, ■ active

If the F-functions were indeed independent functions, as assumed in the original model, then this 2-round differential characteristic would contain 10 independently active S-boxes. Since the characteristic is iterative, and adds 5 active S-boxes per round, this characteristic demonstrates the tightness of the original 15-round bound under the designers' assumptions.

To evaluate the cost in the new model, consider an internal differential view between the two active F-functions, as illustrated in Figure 4.5. Up to AddConstant, they process exactly the same data. In the second AES round, S-box s_0 is not active, and the differential behavior of MixColumns is independent of the actual values of s_0. This means that the entire output difference of $F_{2i-1,4}$ will be identical to that of $F_{2(i+1),4}$ with probability 1. Consequently, if we fix all full-state, single-byte, and column-wise differences to the same bitwise difference pattern, respectively, the actual cost of the iterative characteristic of Figure 4.4 is equivalent to only 5 active S-boxes per 2 rounds, or 40 S-boxes for the recommended 15 rounds.

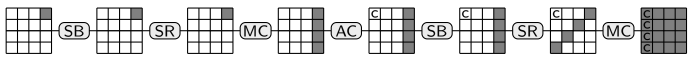


Figure 4.5.: Characteristic for the F-function with 5 active S-boxes. Notation: □ inactive, ■ active, ⊡ internal difference due to round constants

A 2-Round characteristic with probability $2^{27.54}$

We now want to fix the bitwise difference patters of Figure 4.4 suitably to optimize the probability of the characteristic for random input messages. Recall that the AES S-box has maximum differential probability $\frac{4}{256} = 2^{-6}$. For each non-zero input difference, there is exactly one output difference with this probability (and vice versa), while the other probabilities are either $\frac{2}{256} = 2^{-7}$ or 0. We can easily choose difference patterns so that all S-box transitions have this optimal probability, at least for uniformly random round constants. For example, if we fix the one-byte input difference to `75`, the characteristic illustrated in Figure 4.6 satisfies our requirements. The probability of the differential for the F-function is then at least 2^{-30}.

$$\begin{tikzcd}
\begin{bmatrix} \texttt{00} & \texttt{00} & \texttt{00} & \texttt{75} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{00} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{00} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{00} \end{bmatrix} \arrow[r, mapsto, "\mathsf{SB}"] \arrow[r, bend right, "2^{-6}"'] &
\begin{bmatrix} \texttt{00} & \texttt{00} & \texttt{00} & \texttt{fe} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{00} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{00} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{00} \end{bmatrix} \arrow[r, mapsto, "\substack{\mathsf{SR}\\ \mathsf{MC}\\ \mathsf{AC}}"] &
\begin{bmatrix} \texttt{00} & \texttt{00} & \texttt{00} & \texttt{e7} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{fe} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{fe} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{19} \end{bmatrix} \arrow[r, mapsto, "\mathsf{SB}"] \arrow[r, bend right, "2^{-6\times 4}"'] &
\begin{bmatrix} \texttt{00} & \texttt{00} & \texttt{00} & \texttt{f7} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{d8} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{d8} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{b7} \end{bmatrix} \arrow[r, mapsto, "\substack{\mathsf{SR}\\ \mathsf{MC}}"] &
\begin{bmatrix} \texttt{b7} & \texttt{d8} & \texttt{73} & \texttt{f5} \\ \texttt{b7} & \texttt{73} & \texttt{ab} & \texttt{f7} \\ \texttt{c2} & \texttt{ab} & \texttt{d8} & \texttt{f7} \\ \texttt{75} & \texttt{d8} & \texttt{d8} & \texttt{02} \end{bmatrix}
\end{tikzcd}$$

Figure 4.6.: Characteristic for the F-function with probability 2^{-30}.

Note that we are not interested in the probability of the characteristic within the F-function, but rather in the fixed $1 \to 16$ differential. Its probability is higher than that of the characteristic, since several characteristics can contribute to the same differential. In the case of 2-round AES, Keliher and Sui [KS07] proved that for a random round constant, the probability of the differential in Figure 4.6 is $2^{-30} + 74 \times 2^{-35} \approx 2^{-28.272}$.

If we consider additionally that the round constant is not random, but in our case fixed to $(\texttt{00}, \texttt{00}, \texttt{00}, \texttt{00})^\top$ for the relevant bytes, the transition probabilities can increase even further. For example, the differential in Figure 4.7 is satisfied with probability $22 \times 2^{-32} \approx 2^{-27.54}$. Expanded to an 8-round Simpira-4 characteristic, we get a probability of $2^{4\times 27.54} = 2^{-110.16}$.

$$\begin{tikzcd}
\begin{bmatrix} \texttt{00} & \texttt{00} & \texttt{00} & \texttt{40} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{00} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{00} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{00} \end{bmatrix} \arrow[r, mapsto, "\mathsf{SB}"] \arrow[rrr, bend right, "22 \times 2^{-32} \approx 2^{-27.54}"'] &
\begin{bmatrix} \texttt{00} & \texttt{00} & \texttt{00} & \texttt{??} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{00} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{00} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{00} \end{bmatrix} \arrow[r, mapsto, "\substack{\mathsf{SR}\\ \mathsf{MC}\\ \mathsf{AC}}"] &
\begin{bmatrix} \texttt{00} & \texttt{00} & \texttt{00} & \texttt{??} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{??} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{??} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{??} \end{bmatrix} \arrow[r, mapsto, "\mathsf{SB}"] &
\begin{bmatrix} \texttt{00} & \texttt{00} & \texttt{00} & \texttt{2b} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{61} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{61} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{cd} \end{bmatrix} \arrow[r, mapsto, "\substack{\mathsf{SR}\\ \mathsf{MC}}"] &
\begin{bmatrix} \texttt{cd} & \texttt{61} & \texttt{a3} & \texttt{56} \\ \texttt{cd} & \texttt{a3} & \texttt{c2} & \texttt{2b} \\ \texttt{4c} & \texttt{c2} & \texttt{61} & \texttt{2b} \\ \texttt{81} & \texttt{61} & \texttt{61} & \texttt{7d} \end{bmatrix}
\end{tikzcd}$$

Figure 4.7.: Differential for F-function with probability $2^{-27.54}$.

4.4. Collision Attacks on Full-Round Hash

In this section, we show that the number of rounds recommended by the designers is not sufficient to obtain a secure permutation. In particular, we provide collisions for full-round Simpira-4 when used in the permutation-based hash construction suggested by the designers. We first propose collisions for 16 rounds and the full 512-bit hash, then consider the proposed 15-round permutation and a truncated hash output of 256 bits, which is more natural for the 128-bit security claim of the permutation.

Recall that in this short-input Davies-Meyer construction, the $b \times 128$-bit message is used as input to the Simpira permutation, and finally added as a feed-forward to the permutation output to produce the untruncated $b \times 128$-bit hash value:

$$h(x) = \mathsf{Simpira}\text{-}b(x) \oplus x.$$

Our iterative differential characteristic of Figure 4.4 is incidentally very well suited to produce collisions for this feed-forward construction. Observe that if we fix all state differences to the same patterns as discussed in Section 4.3.3, the feed-forward will cancel out the message difference with probability 1 for any even number of rounds. To cover more than 8 rounds while keeping the complexity under the claimed 128-bit security level claimed against permutation distinguishers, we can exploit the absence of any secret keys in this setting.

4.4.1. Collision Attack on 16 Rounds

Since the permutation involves no round keys, we can try to satisfy the conditions for some active F-functions with message modification. We will try to find messages (or rather, initial structures for intermediate Simpira states) such that the conditions for several rounds are satisfied "for free" with probability 1, and append the previous 8-round characteristics of Section 4.3.3 to be satisfied probabilistically. We first propose a simple initial structure covering 6 rounds, and then improve it to satisfy all conditions over 8 rounds, thus extending the previous 8-round characteristic to a 16-round characteristic in the keyless setting with the same probability.

Initial structure for 6 rounds

It is sufficient to set the 4 bytes s_1, s_6, s_{11}, s_{12} of a state A_i to a suitable assignment in order to follow the characteristic for this F-function deterministically. We will refer to these 4 bytes as the diagonal in the following, and to a valid assignment as a valid diagonal. We can reuse one precomputed valid diagonal for all necessary diagonals.

We want to fix the values of the diagonals in A_1, A_3, and A_5 to the valid diagonal. Observe that $A_1 = C_3$, and $A_3 = C_5$. Thus, by fixing the diagonals of A_5 and C_5, we have already satisfied 2 F-characteristics. The remaining $12 + 16 + 12$ bytes of A_5, B_5, C_5 can be filled arbitrarily, which will immediately determine the value of D_3 and thus D_3^{MC2}. If we now set the diagonal of C_3 to the valid diagonal, and fill its remaining 12 bytes with arbitrary values, we completely determine D_5 via B_4 and A_4, and thus complete the state after 4 rounds. By varying the 52 arbitrary byte values, we can obtain the necessary $2^{110.16}$ candidates to satisfy the 8-round characteristic. The approach is illustrated in rounds 1–6 of Figure 4.8, where ▨ and ▧ mark the 52 arbitrary bytes.

Initial structure for 8 rounds by matching diagonals

With some additional effort, we can find initial structures that also satisfy the F-characteristic in round 7. We will again initialize the values of A_5, B_5, C_5, C_3 as in the previous 6-round initial structure. However, we can use the $12+12$ arbitrary bytes of A_5 and C_5 to obtain a valid diagonal in A_7. This will provide us with a 16-round collision attack with the same computational complexity as the 8-round characteristic in Section 4.3.3.

Our goal is to obtain a match between the diagonals of D_5^{MC2} and A_6^{MC2}, as illustrated in Figure 4.8. If these two diagonals sum to zero, the diagonal of A_7 will take the exact same value as that of C_5, which is the valid diagonal. For this purpose, we want to initialize part of the initial structure to generate random values in A_6^{MC2}, and independently a different part of the initial structure, to independently get random values in D_5^{MC2}. Then, any match between the two corresponds to an initial structure that satisfies 4 F-characteristics.

Assume that C_3 and B_5 are already fixed to some arbitrary constants, with the valid diagonal in C_3. We first use the free bytes of A_5 to randomize A_6^{MC2}. Any complete assignment of A_5 will directly determine A_6^{MC2} via

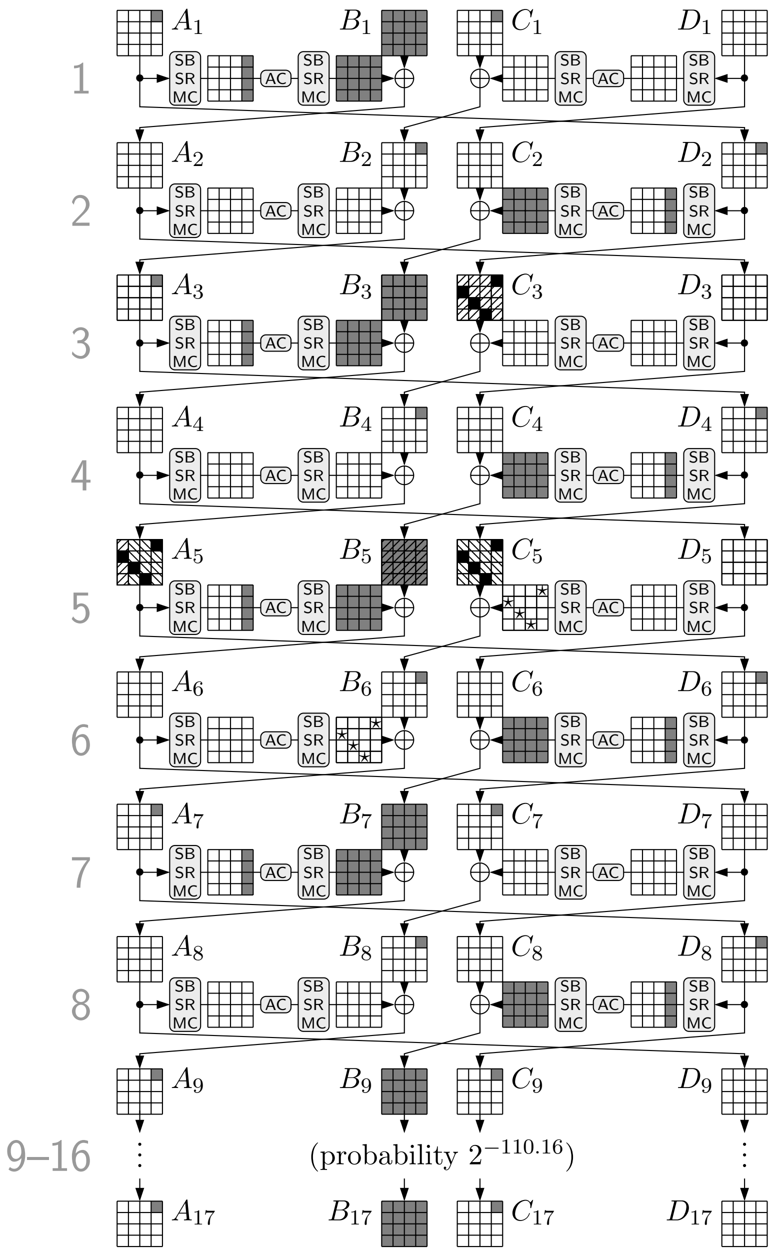


Figure 4.8.: 16-round collision attacks on Simpira-4 hash using 8-round initial structure. Notation: ■ fixed difference, ■ valid diagonal, ▨ arbitrary bytes, ▧ matching inputs, ⋆ match

A_5^{MC2} and A_6. We can assume the values are distributed reasonably close to uniformly random since the values are processed by 4 AES rounds, and only 4 input bytes are fixed.

Independently, we can vary the 12 bytes of C_5 to randomize the diagonal of D_5^{MC2}. To see the independence of the values in A_5, consider the diagonal of A_4^{MC2}. Its values will always be identical to that of D_5^{MC2}, except for the first column, which is influenced by the round constant and will be considered separately in a moment. Since the diagonals of A_5 and C_3 are fixed and predetermined, these values can further be traced back right to D_3^{MC2}. Thus, knowing the diagonal of D_3^{MC2} is equivalent to knowing the target diagonal of D_5^{MC2}, except for 1 byte in s_1. This equivalent diagonal is derived easily from C_5, again by 4 AES rounds via $D_4, D_4^{\mathsf{MC2}}, C_4$.

Evaluating the missing match byte s_1 of D_5^{MC2}. Now we still need to account for the missing byte s_1. Fortunately, with some minor modifications of our guessing strategy, this value can also be computed directly from D_3^{MC2}. Instead of varying all 12 arbitrary bytes of A_5 to produce our matching candidates, we will keep the first column (bytes s_0, s_2, s_3) fixed. In fact, for simplicity, we will set them to the exact same values as the first column of C_3:

$$A_5[0,\dots,3] = C_3[0,\dots,3].$$

This implies that the values of the first column and diagonal (bytes $s_0,\dots,s_3$, s_6, s_{11}, s_{12}) must be identical between D_3^{MC2} and A_4^{MC2}. By partially inverting the last few steps of F, we can also easily verify that this means that

$$D_3^{\mathsf{AC}}[0] = A_4^{\mathsf{AC}}[0].$$

To determine our target value s_1 in D_5^{MC2}, consider a differential view of the intermediate variables in the computations $F(A_4)$ and $F(D_5)$. The input values are identical, but a difference in s_0 is introduced by AddConstant. We are interested in how this difference ΔS^{AC} propagates to the target byte in ΔS^{MC2}. Since we only introduced a single-byte difference before the final MixColumns, we get

$$\begin{aligned}\Delta S^{\mathsf{MC2}}[1] &= \mathtt{01} \cdot \Delta S^{\mathsf{SB2}}[0] \\ &= \mathcal{S}\big(A_4^{\mathsf{AC}}[0]\big) \oplus \mathcal{S}\big(A_4^{\mathsf{AC}}[0] \oplus \Delta S^{\mathsf{AC}}[0]\big).\end{aligned}$$

By using the previously established identities between $F(A_4)$ and $F(D_3)$, and observing $\Delta S^{\mathsf{AC}}[0] = \mathtt{07} \oplus \mathtt{0A} = \mathtt{0D}$, we finally obtain all our target match bytes in D_5^{MC2} directly from $F(D_3)$:

$$\begin{aligned}
D_5^{\mathsf{MC2}}[1] &= A_4^{\mathsf{MC2}}[1] \oplus \Delta S^{\mathsf{MC2}}[1] \\
&= A_4^{\mathsf{MC2}}[1] \oplus \mathcal{S}\big(A_4^{\mathsf{AC}}[0]\big) \oplus \mathcal{S}\big(A_4^{\mathsf{AC}}[0] \oplus \mathtt{0D}\big) \\
&= D_3^{\mathsf{MC2}}[1] \oplus \mathcal{S}\big(D_3^{\mathsf{AC}}[0]\big) \oplus \mathcal{S}\big(D_3^{\mathsf{AC}}[0] \oplus \mathtt{0D}\big), \\
D_5^{\mathsf{MC2}}[6] &= D_3^{\mathsf{MC2}}[6], \\
D_5^{\mathsf{MC2}}[11] &= D_3^{\mathsf{MC2}}[11], \\
D_5^{\mathsf{MC2}}[12] &= D_3^{\mathsf{MC2}}[12].
\end{aligned}$$

Complexity of generating initial structures. Summarizing, we can now generate a large number of initial structures as follows. First, fix the diagonals in C_3 and C_5 to any valid diagonal. Fix all remaining bytes of C_3 and B_5 to arbitrary values. Copy the valid diagonal and first column of C_3 to A_5. Vary the remaining 9 bytes of A_5, storing the resulting values of the diagonal of A_6^{MC2} in a list. Independently vary the 12 bytes of C_5, derive the diagonal of D_5^{MC2}, and store it in a second list. Any match between the two lists gives a valid initial structure that follows the differential characteristic up to round 8.

If we only wanted one match on the 4 bytes of the diagonal, we could try 2^{16} values each for A_5 and C_5, and would expect roughly $2^{2\times 16-32} = 1$ match due to the birthday effect. However, consider using 2^{32} values each instead. The expected number of 4-byte matches is roughly $2^{2\times 32-32} = 2^{32}$. Now we evaluate the complexity for generating these 2^{32} solutions. Computing the match bytes requires evaluating $2 \times 2 \times 2^{32} = 2^{34}$ F-functions. Since 16-round Simpira-4 evaluates more than $16 = 2^4$ F-functions, this corresponds to a complexity of about $2^{32-4} = 2^{30}$ Simpira-4 evaluations. Thus, we were able to produce solutions with amortized complexity less than 1. With this initial structure, we obtain a 16-round collision with computational complexity about $2^{4\times 27.54} = 2^{110.16}$. The memory requirements are only about $2^{32} \times 2$ AES states.

4.4.2. Collision Attack on 15 Rounds with Truncation

In Section 4.4.1, we attacked more than the recommended number of 15 rounds for Simpira-4. In the following, we discuss the applicability of the analysis to the original 15-round design.

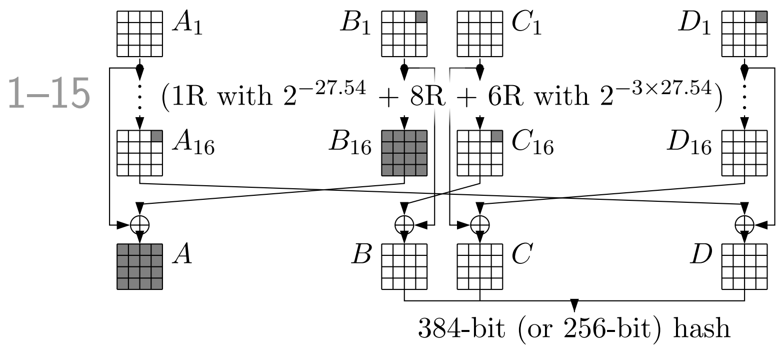
Permutation distinguisher

Clearly, the 16-round characteristic of Figure 4.8 also immediately leads to a 15-round permutation distinguisher. With a computational complexity of $2^{110.16}$, we can find pairs of inputs with a fixed input difference such that the permutation outputs collide in 62 of 64 bytes, or actually in 510 of 512 bits, since we use the 1-byte differences of Figure 4.7. This property implies, for example, second-order collisions for the hash construction with complexity $2 \times 2^{110.16}$, whereas the generic complexity bound is at least about $2^{512/4} = 2^{128}$. This distinguisher violates the security claims for Simpira-4.

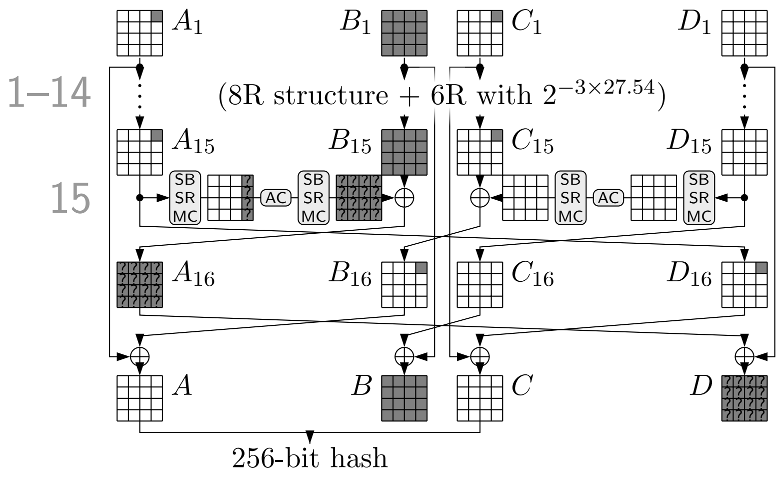
Furthermore, if we impose no constraints on the active F-function in round 15 by allowing arbitrary constraints in A_{15}^{MC2} and thus in A_{16}, we still get a collision on at least 46 of 64 bytes, or in at least 382 of 512 bits, with a fixed input difference. Then, only the 3 active F-functions in rounds 9, 11, and 13 need to be satisfied probabilistically. The probability for this characteristic is $2^{-3\times 27.54} = 2^{-82.62}$.

Truncated collisions

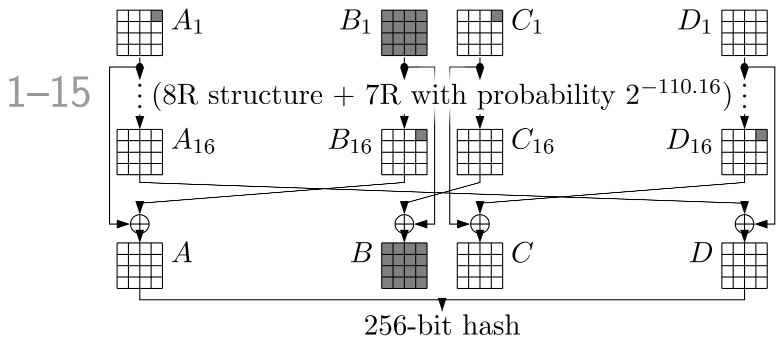
The characteristic no longer automatically leads to full-state collisions for the hash construction, since the 2 active state words we get after an odd number of rounds cannot cancel all 3 active state words at the input. However, we can consider truncated versions of the hash construction. Since the permutation-based Simpira-4 hash construction claims only 128-bit security, but the state size is 512 bits, Simpira's designers comment that "truncation of the output of Simpira may be required [. . .] to match the intended application". An obvious choice would be to truncate the state to 256 bits, so that the security claim matches the generic bound. The details and complexity of the collision attack then vary depending on the implementation of this truncation. Below, we consider 3 natural choices for truncation.


A_1
B_1
C_1
D_1
1–15
(1R with $2^{-27.54}$ + 8R + 6R with $2^{-3\times27.54}$)
A_{16}
B_{16}
C_{16}
D_{16}
A
B
C
D
384-bit (or 256-bit) hash

(a) Truncation variant 1: 384-bit collisions with complexity $2^{110.16}$



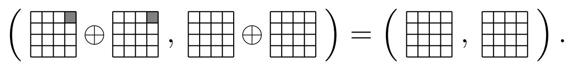
(b) Truncation variant 2: 256-bit collisions with complexity $2^{82.62}$



(c) Truncation variant 3: 256-bit collisions with complexity $2^{110.16}$

Figure 4.9.: Collisions for truncated 15-round Simpira-4 hash.

Truncation variant 1: Left/right half. The most intuitive choice is to simply truncate to the right (or left) half of the final state. Consider the rightmost 256 bits. If we re-use the previous 16-round characteristic of Figure 4.8 and simply cut the last round, as illustrated in Figure 4.9c, then the permutation of the output words means that this conveniently corresponds to a hash output of

$(C_1 \oplus D_{16},\ D_1 \oplus A_{16}) =$ 

The complexity for finding such collisions is $2^{110.16}$, as before.

In fact, we can extend this to collisions up to the rightmost 384 bits if we just shift our iterative characteristic down by 1 round, as illustrated in Figure 4.9a. The probabilistic part of the characteristic is then moved to rounds 1 (input D_1) and rounds 10, 12, and 14 (inputs A). For the same complexity of $2^{110.16}$, we get a 384-bit hash collision of the output

$$(B_1 \oplus C_{16},\ C_1 \oplus D_{16},\ D_1 \oplus A_{16}).$$

Truncation variant 2: Every second word. Assume the truncation function selects every second word, that is, the 256-bit hash output is

$$(A_1 \oplus B_{16},\ C_1 \oplus D_{16}).$$

Then, we can even take advantage of the improved permutation distinguisher with complexity $2^{82.62}$, as in Figure 4.9b.

Truncation variant 3: Updated words. In the previous truncation variants, we took advantage of the fact that the output of one of the last round's two F-functions was truncated. Consequently, another good candidate for a truncation function is to select exactly the words that depend on the last round's F-outputs, A_{16} and B_{16}, so the hash output is

$$(A_1 \oplus B_{16},\ D_1 \oplus A_{16}).$$

Nevertheless, the characteristic of Figure 4.9c still provides hash collisions with complexity $2^{110.16}$.

4.5. Discussion

Since the first publication of the results in this chapter, there have been some noteworthy updates regarding Simpira, in particular another attack by Rønjom exploiting similar properties and, in response, Simpira v2.

4.5.1. Rønjom's Distinguisher

Rønjom [Røn16] independently analyzed Simpira-4 v1 and similarly exploits the Feistel structure and F-function. He describes invariant subspaces for any even number of rounds by partitioning the input space into invariant cosets of dimension 56 over $\mathbb{F}_{2^8}^{64}$. In fact, his result assumes that the round constants differ randomly in 8 bytes, when in reality, only byte s_0 differs for Simpira-4. With this in mind, we actually get invariant cosets of dimension 49. More simply speaking, assume the input state satisfies $D_i = B_i \oplus \lambda \cdot \mathsf{MC}(E_{0,0})$ for some $\lambda \in \mathbb{F}_{2^8}$, where $E_{0,0}$ is the standard matrix with 1 in position s_0 and 0 else. Then, after two rounds of Simpira-4, these two words will again be identical except for the first column, or more precisely, $D_{i+2} = B_{i+2} \oplus \lambda' \cdot \mathsf{MC}(E_{0,0})$ for some $\lambda' \in \mathbb{F}_{2^8}$. More generally, any even number of Simpira-4 rounds preserves any difference $\Delta \in \mathbb{F}_{2^8}$: If $D_i = B_i \oplus \Delta \oplus \lambda \cdot \mathsf{MC}(E_{0,0})$, then $D_{i+2} = B_{i+2} \oplus \Delta \oplus \lambda' \cdot \mathsf{MC}(E_{0,0})$.

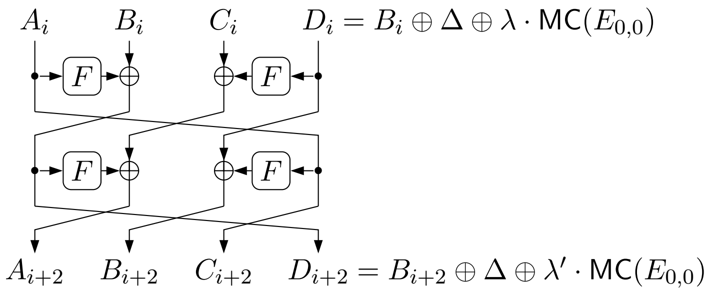

A_i
B_i
C_i
$D_i = B_i \oplus \Delta \oplus \lambda \cdot \mathsf{MC}(E_{0,0})$
F
A_{i+2}
B_{i+2}
C_{i+2}
$D_{i+2} = B_{i+2} \oplus \Delta \oplus \lambda' \cdot \mathsf{MC}(E_{0,0})$

Figure 4.10.: Invariant coset of dimension 49, similar to Rønjom [Røn16].

4.5.2. Simpira v2

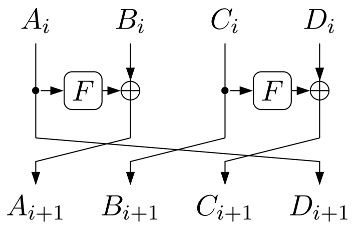
In response to these two attacks, Simpira's designers proposed a new version of the design, Simpira v2, which was published at ASIACRYPT 2016 [GM16b]. Functional changes include new round constants and different Generalized Feistel Networks.

Compared to Simpira v1, the round constants $C_{c,b}$ of the F-functions are changed to the denser, less structured values given in Figure 4.11a, where $c = (c_3, c_2, c_1, c_0)$ is a counter for F-function calls and $b = (b_3, b_2, b_1, b_0)$ specifies the Simpira family member. As before, only the first state row will vary for reasonably small b. This means that on identical inputs to two different F-functions, 3 to 4 (out of 32) S-boxes will receive different input values, and 3 to 4 columns of the function output will differ accordingly due to MixColumns. If these constants were combined with the network of Simpira v1, they would only slightly influence the previously described attacks: The differential probability would decrease slightly, and the invariant subspace would grow to dimension 52.

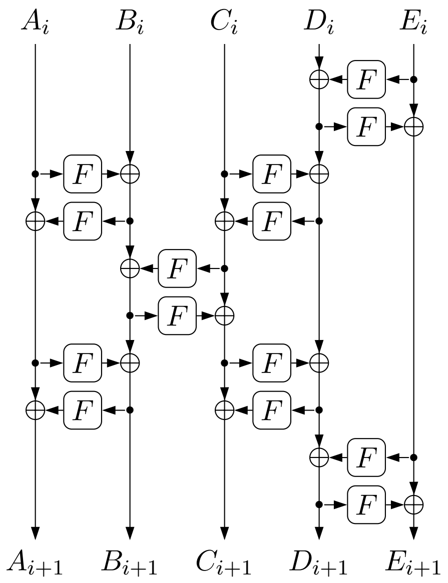
Additionally, to prevent easily exploitable identical inputs entirely, the affected Feistel constructions for Simpira-b with $b \geq 4$, $b \neq 6, 8$ are replaced by different constructions. The new constructions evaluate the same overall number of F-functions for each b. For $b = 4$, the Type-1.x GFS is replaced by the much more well-established Type-2 GFS by Zheng et al. [ZMI89] in Figure 4.11b, the same as used for example in the block cipher CLEFIA by Shirai et al. [SSA+07]. For $b > 4$, $b \neq 6, 8$, the designers propose a novel network construction that is not strictly round-based but scales

$$\begin{array}{|c|c|c|c|}\hline b_0 & b_0 & b_0 & b_0 \\ \hline b_1 & b_1 & b_1 & b_1 \\ \hline b_2 & b_2 & b_2 & b_2 \\ \hline b_3 & b_3 & b_3 & b_3 \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|}\hline c_0 & c_0 & c_0 & c_0 \\ \hline c_1 & c_1 & c_1 & c_1 \\ \hline c_2 & c_2 & c_2 & c_2 \\ \hline c_3 & c_3 & c_3 & c_3 \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|}\hline \mathsf{00} & \mathsf{10} & \mathsf{20} & \mathsf{30} \\ \hline \mathsf{00} & \mathsf{00} & \mathsf{00} & \mathsf{00} \\ \hline \mathsf{00} & \mathsf{00} & \mathsf{00} & \mathsf{00} \\ \hline \mathsf{00} & \mathsf{00} & \mathsf{00} & \mathsf{00} \\ \hline \end{array}$$

(a) $C_{c,b}$ for AddConstant



(b) $b = 4$ $(1 \leq i \leq 15)$



(c) $b = 5$, similarly $b \geq 7$ $(1 \leq i \leq 3)$

Figure 4.11.: Simpira-b v2: New AddConstant and GFS constructions.

easily for different b. This core network is repeated three times for any b. An example of the core network for the smallest variant with $b = 5$ is illustrated in Figure 4.11c. Each new variant executes a total of $12b - 18$ F-functions, the same as in Simpira v1. The new constructions prevent the straightforward re-use of F-inputs. However, the novel network for $b \geq 5$ would profit from more detailed analysis. For example, by choosing the input differences of the two leftmost or rightmost input words similar to the $1 \to 16$ differential of Figure 4.7, it is possible to differentially bypass all except 3 of the 4 leftmost or rightmost F-functions with a probability of less than 2^{30}, even if random round keys were used.

4.6. Conclusion

In this chapter, we analyzed the permutations Simpira-b, $b \geq 4$, $b \neq 6, 8$, of the Simpira v1 family, with a focus on Simpira-4. Due to properties of the underlying Type-1.x Generalized Feistel Structure and the sparse round constants, the computer-aided bounds given by the designers for the minimum number of active S-boxes are invalid. The count includes many pairs of S-boxes whose inputs are not independent, in particular, many actually share identical inputs. Based on differential characteristics that exploit this property, we present full-round collision attacks on the proposed Simpira-4 Davies-Meyer hash construction, with complexities down to $2^{82.62}$ for the recommended full 15 rounds and the truncated 256-bit hash value, depending on the truncation rule, and complexity $2^{110.16}$ for 16 rounds and the full 512-bit hash value.

The attacks exploit Generalized Feistel Structures which apply multiple F-functions to a Feistel branch without xoring other F-outputs in between, as would be the case in a standard Feistel construction. While it is not clear whether this property could be exploited in general for independent F, it certainly becomes a problem when the F-functions differ only by using different, sparse round constants. In Simpira v1, this is the case for all family members $b \geq 4$, $b \neq 6, 8$. The consequence is that two branches of the state will be updated with two closely related F-outputs.

To address the problems described in this chapter and by Rønjom [Røn16], Gueron and Mouha subsequently tweaked their design [GM16b]. The new Simpira v2, published at ASIACRYPT 2016, fixes the issue by replacing both the Feistel construction, to ensure disjoint F-inputs, and the round constants with denser values.

5
Key Recovery for LowMC

In this chapter, we analyze the block cipher LowMC, published at EUROCRYPT 2015 by Albrecht et al. [ARS+15]. The designers aim to minimize the multiplicative complexity of the cipher by using incomplete S-box layers and small, low-degree S-boxes, but compensate with dense and unstructured linear layers. We show that the linear layers alone cannot adequately protect against higher-order attacks, and find that the designers' analysis of the necessary number of rounds, which proposes a 5-round security margin, is too optimistic. In particular, we demonstrate that basic zero-sum distinguishers based on higher-order differential properties can be extended by 4 rounds. We construct structured subspaces to bridge the incomplete S-box layers of initial and final rounds at little additional cost. This allows us to recover the secret key for almost the full number of rounds, with complexities significantly below the security claim. In reaction to this attack and results by Dinur et al. [DLMW15], the designers informally proposed an updated LowMC v2 [ARS+16].

The results in this chapter are based on joint work with Christoph Dobraunig and Florian Mendel. I am the main author and developed significant parts of the attack, in particular the analysis to exploit the incomplete S-box layer with suitable linear subspaces. The following text is a modified, restructured version of the paper published at ICISC 2015 [DEM15c].

5.1. Introduction

Block ciphers are not only the workhorse at the core of most cryptographic protocols for classical two-party communication. More recently, they have also been discovered as a valuable cryptographic primitive to improve the efficiency of complex cryptosystems in modern public-key cryptography.

For example, in a fully homomorphic cryptosystem, homomorphic ciphertexts are usually subject to significant ciphertext expansion, which makes the transmission of homomorphic ciphertexts to the cloud a bottleneck in practice. A much more efficient approach is to encrypt a long plaintext symmetrically with a random key, then encrypt only the key homomorphically. The recipient can then homomorphically evaluate the symmetric decryption algorithm and recover the homomorphically encrypted plaintext [NLV11]. In a similar vein, block ciphers can be evaluated in a secure multi-party communication (SMC) or zero-knowledge proof (ZK) setting.

In all these applications, the block cipher's operations are not executed "bare-metal", but rather in a complex computational framework. The computational cost of evaluating a circuit in these frameworks typically depends primarily on the number of nonlinear operations (e.g., Boolean "and") or the corresponding circuit depth, whereas linear operations (e.g., Boolean "xor") are essentially for free. Moreover, the overall computational cost can be so high that the question of nonlinear complexity of a block cipher is not one of competitive efficiency, but of practicability.

One of the first symmetric designs to address this need is the block cipher LowMC, which was published at EUROCRYPT 2015 by Albrecht, Rechberger, Schneider, Tiessen, and Zohner [ARS+15]. The designers aim for a low nonlinear complexity, both in terms of circuit depth and number of operations. To achieve low depth, they choose an SP network with a relatively low number of rounds with S-boxes of degree 2. To minimize the number of nonlinear operations, the S-box layer is incomplete and applies S-boxes only to part of the state. A very strong, unstructured linear layer compensates for the weak S-box layer. To determine the necessary number of rounds, the designers derive probabilistic bounds against differential and linear cryptanalysis. These bounds also serve as estimates for the security margin against other attacks, in particular higher-order attacks.

Our contributions

We propose a key-recovery attack for 9 out of 11 (or 10) rounds of LowMC-80 with complexity $2^{58.2}$, and also target close to the full number of rounds for several other LowMC variants. Our attacks build on higher-order differential distinguishers, as introduced by Lai [Lai94] and Knudsen [Knu94]. By exploiting the low degree of the LowMC S-boxes, we can define a vector space of input values such that the corresponding outputs after several rounds of LowMC are balanced, that is, they will sum to zero [KR07; AM09]. The designers analyze the number of rounds that can be covered with such distinguishers without violating the data complexity limits of LowMC, but conclude that an additional security margin of 5 rounds is sufficient.

We show that the incomplete S-box layers facilitate the extension of the basic higher-order differential over several more rounds. In particular, we construct carefully structured subspaces and linear relations to bridge the incomplete S-box layers of initial and final rounds at little additional cost. Since we do not exploit any specific properties of the linear layers, the attacks are applicable even for strong, randomly chosen linear layers. For several recommended LowMC variants, we can extend the basic distinguisher by up to 4 rounds, so the designers' security margin seems too optimistic. We conclude that more analysis is necessary to fully understand the security of such novel design approaches.

Related work

In independent research, Dinur et al. [DLMW15] also investigated the security of LowMC against high-order differential cryptanalysis. By developing an optimized variation of interpolation attacks for key recovery, they can identify large classes of weak keys for LowMC-80, and also demonstrate attacks on up to 10 of 11 rounds of LowMC-80 and on full-round LowMC-128. In reaction to this attack and our results, the designers informally proposed an updated LowMC v2 [ARS+16].

Following the publication of LowMC, several other ciphers were proposed for related settings. Canteaut et al. [CCF+16] highlight the potential of IV-based stream ciphers, and show that well-analyzed stream ciphers like Trivium and its 128-bit variant Kreyvium can compete with the performance of LowMC. Méaux et al. [MJSC16] proposed FLIP to combine the advantages of block ciphers and stream ciphers, but Duval et al. [DLR16] quickly identified security flaws in the design. Albrecht et al. [AGR+16] aim to minimize the number of multiplications over a prime field $\mathbb{F}_p$, but ignore the multiplicative depth in their design MiMC.

Other cipher designs have also targeted similar optimization goals, though in a different context. A low number of nonlinear binary operations reduces the cost of side-channel countermeasures, such as masking. Most notably, the block cipher Zorro by Gérard et al. [GGNS13] also implements the idea of partial nonlinear layers in an AES-like cipher. In contrast to LowMC, Zorro features a highly structured linear layer, which was subsequently exploited in a full-round break [WWGY14]. Bar-On et al. [BDD+15] conclude that this is not an inherent flaw of the design approach, and propose a tweaked version of Zorro.

Outline

We briefly describe the LowMC family of block ciphers in Section 5.2. In Section 5.3, we propose a higher-order differential distinguisher that exploits the low degree and the incomplete S-box layer to extend the straightforward zero-sum by 3 rounds. In Section 5.4, we turn this distinguisher into a key-recovery attack and extend it by another round, for a total of 9 rounds of LowMC-80. Finally, we discuss the applicability of our approach to other parameter sets of LowMC, and briefly show how Dinur et al. attack even more rounds with interpolation attacks in Section 5.5.

5.2. Description of LowMC

5.2.1. The LowMC Family of Block Ciphers

LowMC is a family of block ciphers published at EUROCRYPT 2015 by Albrecht et al. [ARS+15]. The algorithms of the family are parametrized by a wide range of block sizes n, key sizes k, and number of rounds r, but also by a more unusual parameter: the number m of S-boxes per substitution layer, which is independent of the block size. Additionally, neither the concrete instantiation of the linear layers f_L, nor the linear key derivation functions used in f_K are fixed, but both are to be selected pseudorandomly for each family member. We will denote LowMC with key size k, state size n and m S-boxes per round as LowMC-$k^{n,m}$. The target security level of each family member is k bits, subject to a (logarithmic) data complexity limit d.

The designers propose two primarily recommended instances, as well as a number of secondary suggestions [ARS+15]. The first instance is intended to provide "PRESENT-like" security using an 80-bit key, while the second targets "AES-like" security using a 128-bit key. Table 5.2 (p. 133) lists the primary recommendations. We abbreviate the recommended parameter sets as LowMC-80 = LowMC-$80^{256,49}$ and LowMC-128 = LowMC-$128^{256,63}$.

5.2.2. The LowMC Round Function

LowMC is a key-alternating cipher and iteratively applies r keyed rounds to the initial n-bit message block. Encryption starts with a whitening key addition $f_K^{(0)}$, followed by r applications of the round function

$$f^{(i)} = f_K^{(i)} \circ f_L^{(i)} \circ f_S, \qquad 1 \le i \le r,$$

which applies an incomplete substitution layer f_S (identical for each round), a dense, unstructured linear layer $f_L^{(i)}$, and the round-key addition $f_K^{(i)}$, as illustrated in Figure 5.1.

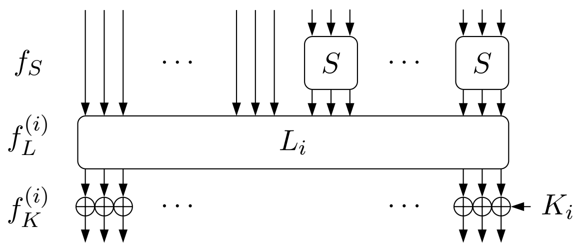


Figure 5.1.: The round function of LowMC: $f^{(i)} = f_K^{(i)} \circ f_L^{(i)} \circ f_S$.

- f_S: The substitution layer applies the 3-bit S-box $\mathcal{S}$, where

  $$\mathcal{S}(a, b, c) = (a \oplus b \cdot c, \quad a \oplus b \oplus a \cdot c, \quad a \oplus b \oplus c \oplus a \cdot b).$$

  However, to minimize the number of multiplications, this layer is incomplete and only applies $\mathcal{S}$ to the $3m$ rightmost (least significant) bits of the n-bit state. The other $n - 3m$ bits remain unchanged. The maximum differential and linear probability of $\mathcal{S}$ are 2^{-2}, and the algebraic degree is 2, which is optimal for a 3-bit S-box.

- $f_L^{(i)}$: The linear layer multiplies the n-bit state with a pseudorandom, round-dependent invertible $n \times n$ matrix over $\mathbb{F}_2$.

- $f_K^{(i)}$: The key layer adds an n-bit round key K_i, which is derived from the k-bit master key by a pseudorandom affine linear function.

5.3. Higher-Order Differential Distinguisher

In this section, we propose distinguishers for up to 8 rounds of the cipher. More specifically, we will focus on (families of) zero-sum distinguishers: finding sets of inputs to the permutation such that both the sum (over $\mathbb{F}_2^n$) of the inputs, as well as the sum of their outputs, equal zero. We will start with a straightforward distinguisher based on the low degree, which is also discussed by LowMC's designers. Then, we exploit the incomplete S-box layer and guess a few key bits in order to extend the zero-sum by up to 3 rounds.

5.3.1. Designers' Considerations

A well-known result from the theory of Boolean functions is that if the algebraic degree of a vectorial Boolean function (like a permutation) is d, then the sum over the outputs of the function applied to all elements of a vector space of dimension $\geq d+1$ is zero [Lai94] (as is the sum of all inputs, i.e., the elements of the vector space). The same property holds for affine vector spaces of the form $V + c = \{v + c \mid v \in V\}$ for some vector space V and constant c. Therefore, in the remaining text, we also refer to affine vector spaces as vector spaces for simplicity. This property allows exploiting a low algebraic degree of cryptographic functions to create zero-sum distinguishers and has been applied, for example, to Keccak [BC10; BCD11].

For this reason, the design paper of LowMC [ARS+15] includes (upper) bounds for the algebraic degree of multiple rounds of the permutation. Their analysis is based on the bounds by Boura et al. [BCD11]. One round of LowMC has degree $d_1 = 2$. So, if the degree after r rounds (with m S-boxes per round of the n-bit permutation) is $d_r^{(n,m)}$, then the degree $d_{r+1}^{(n,m)}$ after $r+1$ rounds is upper-bounded by

$$d_{r+1} \leq \min\left\{2 \cdot d_r, \quad m + d_r, \quad \tfrac{1}{2} \cdot (n + d_r)\right\}.$$

The resulting bounds for up to 15 rounds are given in Table 5.1.

Based on these numbers, the designers recommend that the number of rounds r satisfies $r \geq r_{\text{deg}} + r_{\text{outer}}$, where r_{deg} is the number of rounds necessary for a sufficiently high degree ($d_{r_{\text{deg}}} \geq d - 1$ for the logarithmic data complexity limit d), and $r_{\text{outer}} = 5$ is a heuristic estimate for the

Table 5.1.: Upper bounds for the algebraic degree $d_r^{(n,m)}$ after r rounds of the LowMC permutation on $n = 256$ bits with $m \in \{49, 63\}$ S-boxes.

r	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$d_r^{(256,49)}$	2	4	8	16	32	64	113	162	209	232	244	250	253	254	255
$d_r^{(256,63)}$	2	4	8	16	32	64	127	190	223	239	247	251	253	254	255

number of rounds that can be "peeled off" in the beginning and end of the cipher, based on the bounds for linear and differential cryptanalysis. This leads to the round numbers stated in Table 5.2 for the recommended parameter sets.

Table 5.2.: Recommended number of rounds $r \geq r_{\text{deg}} + r_{\text{outer}}$ for LowMC with key size k, block size n, m S-boxes, data limit 2^d [ARS+15].

Cipher	k	n	m	d	r_{deg}	r_{outer}	r
LowMC-80	80	256	49	64	6	5	11
LowMC-128	128	256	63	128	7	5	12

The degree bounds from Table 5.1 show that 11 or 12 rounds of the unkeyed round function cannot be considered an ideal random permutation, although the complexity of a straightforward zero-sum distinguisher is far beyond the claimed security level. Let f denote one (keyed) round of LowMC-80 (resp. LowMC-128). If we choose any subspace $V \leq \mathbb{F}_2^{256}$ with dimension ≥ 245 (resp. ≥ 252), we get

$$\bigoplus_{v \in V} v = \bigoplus_{v \in V} f^{11}(v) = 0 \qquad \text{(resp. } \bigoplus_{v \in V} f^{12}(v) = 0\text{)}.$$

Conversely, if we want to stay strictly below the data complexity limit of 2^d queries, we can target up to 5 (resp. 7) rounds with V of dimension 33 (resp. 128) with the bounds from Table 5.1. However, we will show that we can obtain distinguishers with a much lower complexity. In the following, we focus on LowMC-80.

5.3.2. Direct-Sum Construction

We want to add a free round in the beginning of the distinguisher by choosing an input vector space V of a particular structure as a starting point. Assume that V is the direct sum of any subspace $V_{\mathrm{id}} \leq \mathbb{F}_2^{256-3\cdot m}$, and m special subspaces $V_s \leq \mathbb{F}_2^3$ for each $1 \leq s \leq m$:

$$V = V_{\mathrm{id}} \times V_1 \times V_2 \times \cdots \times V_m.$$

If we choose each special subspace $V_s \leq \mathbb{F}_2^3$ such that

$$\mathcal{S}(V_s) = \{\mathcal{S}(v) \mid v \in V_s\} = V_s,$$

then the entire space V is invariant under f_S, that is, $f_S(V) = V$. For example, the bijective 3-bit S-box maps any trivial subspace V_s of $\mathbb{F}_2^3$ to itself, that is, $V_s = \mathbb{F}_2^3$ or $V_s = \{(0,0,0)\}$.

More generally, if V_s is such that $\mathcal{S}$ maps any coset $V_s + c$ to another affine space $V_s' + c'$ of the same dimension, then the entire first round f (with or without initial key whitening layer f_K) will map the structured space V to another space V'' of the same dimension. We refer to any such V_s as $\mathcal{S}$-compatible, and to any V with $f_S(V + c) = V' + c'$ for all c as f_S-compatible. Note that not only the trivial subspaces satisfy these requirements, but also all one-dimensional spaces $V_s + c = \{c, v + c\}$, since

$$\mathcal{S}(V_s + c) = \{v_1', v_2'\} = \{v_1', (v_2' - v_1') + v_1'\} = V_s' + v_1'.$$

Thus, any such structured initial space V of dimension ≥ 33 with subspaces V_s, $\dim V_s \in \{0, 1, 3\}$ for $1 \leq s \leq m$, is f_S-compatible and allows extending the zero-sum distinguisher for LowMC-80 by one round (Figure 5.2), since

$$\bigoplus_{v \in V} v = \bigoplus_{v' \in V' = f_S \circ f_K(V)} (f^5 \circ f_K \circ f_L)(v') = 0.$$

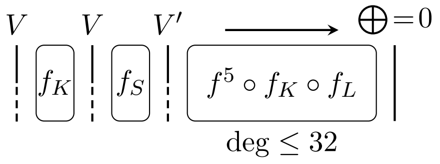


Figure 5.2.: Zero-sum of size 2^{33} for $r = 6$ rounds of LowMC-80.

5.3.3. Exploiting the Incomplete S-Box Layer

In addition to the generic attack strategy applied so far, we can take advantage of the special structure of LowMC's S-box layer to add another round. This is easy for low-dimensional zero-sums, but building higher-dimensional zero-sums will incur some (pre-)computational overhead.

Constructing f^2-compatible spaces by solving equations

We will now try to define an f^2-compatible initial structure that yields an affine space after the initial 2 rounds of LowMC-80, or more precisely, after $f_S \circ f_K \circ f_L \circ f_S \circ f_K$. We need to find spaces V and $V' = f(V)$ such that both V and V' are f_S-compatible. Let $W = \mathbb{F}_2^{109} \times \{(0, \dots, 0)\} \leq \mathbb{F}_2^{256}$. Clearly, any subspace $V \leq W$ is invariant under f_S and thus f_S-compatible. The input space $V' = f(V)$ to the second f_S is then an affine transformation of V, $V' = (f_K \circ f_L)(V)$. We will try to identify sets of linear equations that define the subspace $V \leq W$ and guarantee that V' is also f_S-compatible.

In the simplest case, V' would also be a subspace of W. However, this requirement would impose 147 linear constraints (one for each S-box input bit) on the 109-dimensional space W, so we cannot expect any nontrivial solution space V. Fortunately, an observation similar to the case $\dim V_s = 1$ in Section 5.3.2 allows us to reduce the number of equations without violating f_S-compatibility. Assume we drop one of the 147 constraints for V', so the input to one S-box $\mathcal{S}$ is no longer constant, but one input bit may take different values in different elements of V' (though V' is not necessarily a direct sum). This will also toggle some of the output bits of $\mathcal{S}$, that is, the behavior of $\mathcal{S}$ is linear with respect to the one input bit. Thus, if we drop one equation per S-box, V' is still f_S-compatible, and we get an f^2-compatible solution space V with dimension at least $109 - 2 \cdot 49 = 11$ (Figure 5.3).

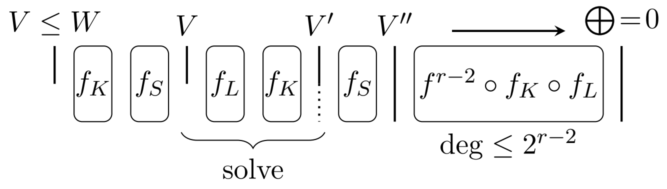


Figure 5.3.: Zero-sum of size $2^{2^{r-2}} \leq 2^{11}$ for $r \leq 5$ rounds of LowMC-80.

V can be precomputed and depends only on the matrix of the linear layer of the first round of LowMC-80. Note that due to the key addition f_K just before the second S-box layer, we cannot know or control the exact values of the two constant input bits per S-box, and thus do not know the linear behavior of $\mathcal{S}$ or the structure of V''. Unfortunately, the dimension of 11 is only sufficient for zero-sums up to $1 + 1 + 3 = 5$ rounds with size 2^9, and additional considerations are necessary for our target size of 2^{33} for $1 + 1 + 5 = 7$ rounds.

Increasing the dimension by guessing key bits

To attack 6 or 7 rounds, we would require input spaces of 2^{17} and 2^{33} elements, respectively. To increase the dimension of V accordingly, we need to allow for more freedom either in the first or in the second substitution layer f_S. First, consider the first substitution layer. If we want to choose our inputs so as to ensure a specific vector space structure after the first substitution layer, we can achieve this trivially if the target vectors are non-constant only in the identity part. If we want specific values at the output of an S-box, we need to guess the corresponding 3 bits of the first whitening key, which is added right before the substitution layer. By guessing these 3 bits, we can increase the dimension of V by 3. Note that if we "activate" an S-box this way, the required message input set S to produce V is no longer necessarily a vector space. In particular, its elements no longer necessarily sum to zero. However, this will not be required for our following key recovery attacks, so the loss of the input zero-sum property is not a problem. To apply the technique to attack 6 rounds with $\dim V = 17$, we need to activate 2 S-boxes and thus guess 6 key bits. This increases the attack complexity to $2^{17} \cdot 2^6 = 2^{23}$. To attack 7 rounds, we need $\dim V = 33$ and thus need to activate 8 S-boxes with 24 guessed key bits, leading to an attack complexity of $2^{33} \cdot 2^{24} = 2^{57}$ (Figure 5.4).

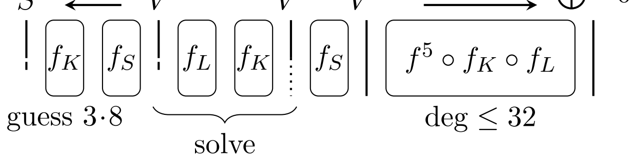


Figure 5.4.: Zero-sum of size 2^{33} for $r = 7$ rounds of LowMC-80.

Reducing the complexity by being lucky (or precomputation)

We can decrease the necessary number of activated S-boxes in the first f_S-layer a bit by considering additional freedom in the second f_S-layer. We previously chose a fixed bit per S-box of the second f_S which was allowed to toggle, while the other two bits needed to remain constant. But this is not actually necessary: we have the freedom to choose any of the 3 bit positions of each S-box as the toggle-bit, so we have a total of $3^{49} \approx 2^{77.7}$ options to choose the 98 (out of the total 147) constraints imposed by the second layer. The 147 available constraints are specified by the (roughly uniformly randomly generated) rows of the linear layer matrix. In addition, we have the freedom to select the activated S-boxes of the first layer. For each option, we have a very small chance that the selection of 98 constraints is redundant (with respect to the $109+3s$-dimensional V, if we guess s S-box keys in the first substitution layer), and the remaining solution space has a dimension larger than $11+3s$.

Consider again the 7-round attack, with its required input space of $2^{33} = 2^{11} \cdot 2^{22}$ elements. To increase the dimension by 22, we had to activate $s = 8$ S-boxes. We only needed 1 bit of freedom from the last of the 8 S-boxes, but still had to guess all the corresponding 3 key bits. There is a reasonable chance that if we activate only $s = 7$ S-boxes (and start with V of dimension $109 + 7 \cdot 3 = 130$) and add the 98 constraints of the second layer, the remaining solution space has the required dimension 33 instead of the expected $130 - 98 = 32$. This is equivalent to the event that a randomly selected 130×98 matrix over $\mathbb{F}_2$ has rank 97. The probability of picking a rank-r matrix uniformly random from $\mathbb{F}_2^{n\times m}$, $n \geq m$ [vW92] is given by

$$P(n,m,r) = \frac{\prod_{i=0}^{r-1}(2^m - 2^i) \cdot \prod_{i=0}^{r-1}(2^n - 2^i)}{\prod_{i=0}^{r-1}(2^r - 2^i) \cdot 2^{n\cdot m}},$$

so our success chance for one try is at least $P(130,98,97) \approx 2^{-32.0}$.

Even though the available selections of constraints are not independent, we verified experimentally that the measured distribution of the rank of random selections closely matches the theoretic expectations. Thus, it is reasonable to expect that a suitable selection exists among the available choices, and that it can be efficiently found (e.g., after trying about 2^{32} random selections). Since the selection depends only on the corresponding matrix of the linear layer, it can be precomputed in advance. The final attack complexity for 8 rounds is about $2^{33} \cdot 2^{21} = 2^{54}$.

5.3.4. Partial Zero-Sums

Finally, we can add another round at the end of the distinguisher by again taking advantage of the incomplete S-box layer, at the cost of a slightly weaker distinguishing property. For this purpose, we need to rewrite the last round $f = f_K \circ f_L \circ f_S$ using an equivalent round key $K' = f_L^{-1}(K)$. Then, we can swap the order of f_K and f_L to $f = f_L \circ f_{K'} \circ f_S$. If we use the input set S of size 2^{33} as defined in Section 5.3.3, we know that the output of f^7 sums to zero in each output bit. Consider the output of $f_{K'} \circ f_S \circ f^7$ (Figure 5.5). The first $\ell = n - 3 \cdot m = 109$ bits of the output state are unaffected by the S-boxes of the final f_S, so they will still sum to zero even after $f_{K'}$. The result is a generalized partial zero-sum property on some fixed (linearly independent) linear combinations of the cipher's output bits as summarized in Algorithm 1, where $\{e_0, \dots, e_{255}\}$ is the standard basis of $\mathbb{F}_2^{256}$, and $\lfloor x \rfloor_{\mathrm{id}}$ denotes the $\ell = 109$ most significant bits of x. Since ℓ is relatively large ($\ell > k$ for LowMC-80), even a zero-sum distinguisher only for ℓ bits of the linearly combined output gives us a detectable distinguishing property that we can then use for key recovery.

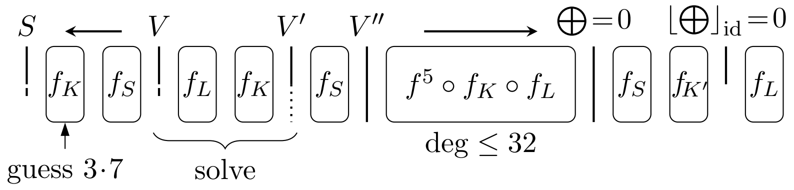


Figure 5.5.: Partial zero-sum of size 2^{33} for $r = 8$ rounds of LowMC-80.

Algorithm 1 Partial zero-sum distinguisher for 8 rounds of LowMC-80.

```
Output: Initial structure $S$ of size $2^{33}$ with partial zero-sum property
  repeat
    Choose $\{i_0, \dots, i_6\} \subseteq \mathbb{Z}_{49}$ and $(o_0, \dots, o_{48}) \in \mathbb{Z}_3^{49}$
    $C_i = \{e_{109+3 \cdot i+j} \mid i \in \mathbb{Z}_{49} \setminus \{i_0, \dots, i_6\},\ j \in \mathbb{Z}_3\}$
    $C_o = \{f_L^{-1}(e_{109+3 \cdot i+j}) \mid i \in \mathbb{Z}_{49},\ j \in \mathbb{Z}_3 \setminus \{o_i\}\}$
    $V = \ker(C_i) \cap \ker(C_o)$
  until $\dim V \geq 33$
  for all $2^{3 \cdot 7}$ key guesses $\kappa$ of the active S-boxes $\{i_0, \dots, i_6\}$ do
    $S \leftarrow f_\kappa^{-1}(f_S^{-1}(V)) + c$
    if $\lfloor \bigoplus_{s \in S} f_L^{-1}(\text{LowMC-80}(s)) \rfloor_{\mathrm{id}} = 0$ then
      return $S$, $\kappa$
```

5.4. Key Recovery Attack

In this section, we use the 8-round distinguisher of Section 5.3 to construct 8-round and 9-round key recovery attacks on LowMC-80.

5.4.1. Basic Zero-Sum Key Recovery for 8 Rounds

The zero-sums and partial zero-sums of Section 5.3.3 and 5.3.4 already give rise to straightforward 8-round key-recovery attacks. In Algorithm 1, we already guessed $3 \cdot 7$ bits of the initial whitening key. The correct value of these bits is necessary to construct a suitable vector space V, which we can detect by checking the partial zero-sum property after the final key addition layer. If we recover the remaining $80 - 21 = 59$ bits of key information by brute-force testing, the overall complexity is dominated by about 2^{59} trial encryptions, and we need at most $2^{21} \cdot 2^{33} = 2^{54}$ queries.

Alternatively, the zero-sum property after $r - 1 = 7$ rounds can be used to recover the final round key K_r in 3-bit chunks after identifying the correct set S and initial round key bits, which in turn allows to easily recover the original key K. Let $C_i^{(r-1)}$ denote the state after $r-1$ rounds applied to input $P_i \in S$, and $C_i^{(r)} = C_i$ the corresponding ciphertext obtained by the attacker, so $C_i = (f_L^{(r)} \circ f_S)(C_i^{(r-1)}) \oplus K_r$ (Figure 5.6). Since the key addition $f_K^{(r)}$ and linear layer $f_L^{(r)}$ can be swapped (replacing the original K_r with a transformed K'_r), the zero-sum property translates to

$$\bigoplus_{i=1}^{2^{33}} C_i^{(r-1)} = \bigoplus_{i=1}^{2^{33}} f_S^{-1}\big(K'_r \oplus f_L^{(r)-1}(C_i)\big) = 0.$$

For each of the $m = 49$ S-boxes, this property can be checked independently for each possible value of the corresponding 3 bits of K'_r. Since we expect to require about $80 - 21 = 59$ bits of K'_r to recover K_r and consequently K, we guess the keys for 20 S-boxes, which leads to an overall complexity dominated by the 2^{54} queries for identifying the correct initial key bits.

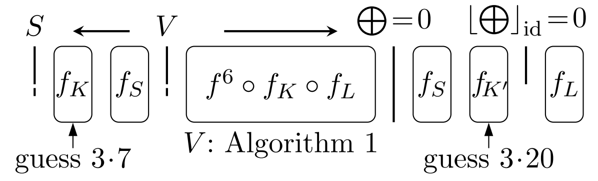


Figure 5.6.: Key recovery for $r = 8$ rounds of LowMC-80 with $|S| = 2^{33}$.

5.4.2. Key Recovery with Linear Masks for 9 Rounds

We extend the previous key recovery attack to 9 rounds of LowMC-80 by combining the higher-order differential zero-sum approach with linear masks for an additional round. This combination will allow us to derive 1 bit of key information per input set, and can be repeated to learn more.

For an attack on r rounds, assume we have constructed a zero-sum attack for $r-2$ rounds, that is, we can generate sets of inputs such that their corresponding outputs after $r-2$ rounds sum to zero. We name the intermediate states and rearrange the key addition layer as in Section 5.4.1:

$$C_i = C_i^{(r)} = \big(f_L^{(r)} \circ f_{K'}^{(r)} \circ f_S \circ f_L^{(r-1)} \circ f_{K'}^{(r-1)} \circ f_S\big)\big(C_i^{(r-2)}\big).$$

Since $\bigoplus_i C_i^{(r-2)} = 0$, we also get the partial zero-sum as in Section 5.3.4,

$$\Big\lfloor \bigoplus\nolimits_{i=1}^{2^{33}} \big(f_{K'}^{(r-1)} \circ f_S\big)\big(C_i^{(r-2)}\big)\Big\rfloor_{\mathrm{id}} = 0,$$

where $\lfloor x \rfloor_{\mathrm{id}}$ is the value x truncated to the most significant $\ell = 109$ bits, i.e., the identity part of the S-box layer. Let x_i and y_i denote the states right before and after the linear layer of the second-to-last round,

$$x_i = \big(f_{K'}^{(r-1)} \circ f_S\big)\big(C_i^{(r-2)}\big), \quad y_i = \big(f_S^{-1} \circ f_{K'}^{(r)-1} \circ f_L^{(r)-1}\big)\big(C_i\big) = f_L^{(r-1)}(x_i).$$

Now let (a, b) be a pair of consistent linear masks for $f_L^{(r-1)}$, that is,

$$\langle a, x\rangle = \langle b, f_L^{(r-1)}(x)\rangle \qquad \forall\, x \in \mathbb{F}_2^{256}.$$

We will call the mask pair (a, b) suitable if a is zero on its 147 least significant bits (i.e., all bits except the identity part of f_S), and b is zero on (most of) its 147 least significant bits. We refer to the S-boxes where b is non-zero on at least one of the corresponding 3 input bits as active.

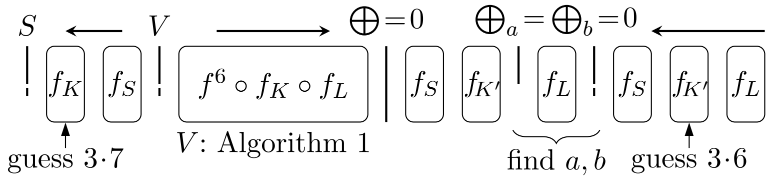


Figure 5.7.: Key recovery for $r = 9$ rounds of LowMC-80 with $|S| = 2^{33}$, based on 1-bit sums $\bigoplus_a = \bigoplus_i \langle a, x_i\rangle$ and $\bigoplus_b = \bigoplus_i \langle b, y_i\rangle$.

Bitwise key recovery with suitable masks

We target mask pairs (a, b) with at most 6 active S-boxes. For a random matrix, the probability that an input mask a is mapped to an output mask b in which at most 6 of 49 S-boxes are active is given by the cumulative distribution of a Binomial distribution with $p = 1 - 2^{-3}$ and $n = 49$ as

$$P[\leq 6 \text{ S-boxes active}] = \sum_{i=0}^{6} \binom{49}{i} \cdot \left(1 - 2^{-3}\right)^{i} \cdot \left(2^{-3}\right)^{49-i} \approx 2^{-106.4}.$$

Since we have a total of 2^{109} possible input masks a available, we expect several suitable mask pairs that can be found easily by solving at most $\binom{49}{6} \approx 2^{23.7}$ linear equation systems. In practical experiments, we were able to find masks with 6 or even fewer active S-boxes in reasonable time.

Observe that if (a, b) is a suitable mask pair, then

$$\bigoplus_i \langle b, y_i \rangle = \bigoplus_i \left\langle b, f_L^{(r-1)}(x_i) \right\rangle = \bigoplus_i \langle a, x_i \rangle = \left\langle a, \bigoplus_i x_i \right\rangle = 0,$$

since a only selects from the 109 most significant bits, and the x_i have the partial zero-sum property $\lfloor \bigoplus_i x_i \rfloor_{\mathrm{id}} = 0$. This modified zero-sum property of the y_i depends only on the last-round key bits (of the equivalent key K') added to the active S-boxes, i.e., for 6 active S-boxes, on 18 key bits. The other key bits are either not selected by b (inactive S-boxes), or cancel out during summation (identity part), see Figure 5.7. The probability of the 1-bit property to hold for a random key guess is $\frac{1}{2}$, so applying the attack to one zero-sum input set will eliminate half of the key guesses for the 18 key bits, or win 1 bit of key information. By repeating the attack for 18 input sets S (e.g., by adding 18 different constants to the original input set), we expect to recover all 18 round key bits.

To learn more key bits, we need to find more linear mask pairs (a, b), with different active S-boxes. Since the previously active S-boxes with previously recovered key bits can now be active for free, finding such masks becomes easier. In addition, we can re-use the same ciphertexts for different masks, so the data complexity does not increase. In summary, after precomputing suitable mask pairs, this attack described so far allows recovering the complete key for r instead of $r - 1$ rounds at an additional cost factor of $18 \approx 2^{4.2}$ data complexity and about $2^{18} \cdot 59 \approx 2^{24}$ computational complexity. The overall complexity for 9 rounds is $2^{54} \cdot 2^{4.2} = 2^{58.2}$ queries and about $2^{54} \cdot 2^{24} = 2^{78}$ round computations.

Reducing the complexity with FFT summation

The computational complexity can be further reduced by optimizing the repeated evaluation of the modified zero-sum check. Instead of summing over all inputs for each of the 2^{18} key guesses, we can precompute partial bit sums, and only combine those to compute the final sum for each of the 2^{18} key candidates. The idea is to decompose the target sum into its S-box-wise components as

$$\bigoplus_i \langle b, y_i\rangle = \bigoplus_i \left\langle \bigoplus_{s=0}^{49} b_s, y_i \right\rangle = \bigoplus_{s=0}^{49} \bigoplus_i \langle b_s, y_i\rangle,$$

where b_s equals b on the 3 bit positions corresponding to S-box s, $1 \leq s \leq 49$ (or the 109 bits of the identity part for $s = 0$), and is zero otherwise. Then $\bigoplus_i \langle b_s, y_i\rangle$ depends only on the 3 round key bits corresponding to S-box s (and 3 bits of $f_L^{-1}(C_i)$, see the definition of y_i), and can be precomputed in a first phase for all 2^3 possible values of these key bits, for each active S-box s. Then, in the second phase, to determine the test bit for each of the 2^{18} key candidates, it suffices to sum the 6 corresponding partial sums (of the active S-boxes). Considering that each linear layer alone needs about 2^{16} xor operations, the complexity of both phases is significantly smaller compared to the computational effort of generating all the required ciphertexts C_i. This step can be repeated 4 times with different mask pairs (a, b) to recover about $4 \cdot 18 = 72$ key bits; the remaining bits can easily be determined by brute force testing.

The final key recovery attack is summarized in Algorithm 2. The notation is similar to Algorithm 1 in Section 5.3.4: $\{e_0, \ldots, e_{255}\}$ is the standard basis of $\mathbb{F}_2^{256}$. Parts of the state 256-bit state $x \in \mathbb{F}_2^{256}$ are addressed by $\lfloor x \rceil$, where $\lfloor x \rceil_{\mathrm{id}}$ denotes the $\ell = 109$ most significant bits of x that correspond to the identity part of the S-box layer, and $\lfloor x \rceil_s$ the 3 bits that correspond to S-box index s (bits $109 + 3 \cdot s, \ldots, 109 + 3 \cdot s + 2$). For any set $C \subseteq \mathbb{F}_2^{256}$, $\ker(C)$ denotes the set of solutions $\{x \in \mathbb{F}_2^{256} \mid \forall c \in C : \langle x, c\rangle = 0\}$.

The offline phase is expected to require solving roughly 2^{32} binary linear equation systems over the state size to find the space V, and another roughly 2^{25} to find the masks (a_m, b_m). Overall, the complexity of the offline phase is expected to be practical.

The online phase is dominated by querying about $2^{21} \cdot 18 \cdot 2^{33} \approx 2^{58.2}$ chosen plaintexts, and computing the values σ_{b,κ'_s} with $2^{21} \cdot 18 \cdot 24 \cdot 2^3 \cdot 2^{33} \approx 2^{65.8}$ S-box lookups, equivalent to computing about $2^{65.8}/(49 \cdot 9) \approx 2^{57}$ 9-round encryptions of LowMC-80.

Algorithm 2 Key recovery attack for 9 rounds of LowMC-80.

```
Offline phase: Find vector space $V$ and masks $(a_m, b_m)$
  repeat                                            {Find spaces $(V, V')$ for $f_L^{(1)}$}
    Choose $\{i_0, \ldots, i_6\} \subseteq \mathbb{Z}_{49}$ and $(o_0, \ldots, o_{48}) \in \mathbb{Z}_3^{49}$
    $C_i \leftarrow \{e_{109+3 \cdot i+j} \mid i \in \mathbb{Z}_{49} \setminus \{i_0, \ldots, i_6\},\ j \in \mathbb{Z}_3\}$
    $C_o \leftarrow \{f_L^{(1)-1}(e_{109+3 \cdot i+j}) \mid i \in \mathbb{Z}_{49},\ j \in \mathbb{Z}_3 \setminus \{o_i\}\}$
    $V \leftarrow \ker(C_i) \cap \ker(C_o)$
  until $\dim V \geq 33$
  $B_0 = \emptyset$
  for $m = 1$ to 4 do
    repeat                                          {Find masks $(a_m, b_m)$ for $f_L^{(r-1)}$}
      Choose $B_m = \{o_0, \ldots, o_5\} \subseteq \mathbb{Z}_{49} \setminus B_{m-1}$
      $C_i \leftarrow \{e_{109+3 \cdot i+j} \mid i \in \mathbb{Z}_{49},\ j \in \mathbb{Z}_3\}$
      $C_o \leftarrow \{f_L^{(r-1)-1}(e_{109+3 \cdot i+j}) \mid i \in \mathbb{Z}_{49} \setminus B_m,\ j \in \mathbb{Z}_3\}$
      $A_m \leftarrow \ker(C_i) \cap \ker(C_o)$
    until $\dim A_m \geq 1$
    $B_m \leftarrow B_{m-1} \cup \{o_0, \ldots, o_5\}$
    Select $a_m \in A_m$ and let $b_m \leftarrow f_L^{(r-1)}(a_m)$
Online phase: Recover 80-bit key $K$
  for all $2^{3 \cdot 7}$ initial key guesses $\kappa$ of the active S-boxes $\{i_0, \ldots, i_6\}$ do
    for $b = 1$ to 18 do                            {Query and precompute partial sums}
      $S_b \leftarrow f_\kappa^{-1}(f_S^{-1}(V)) + c_b$
      Query ciphertext set $C_b \leftarrow$ LowMC-80$(S_b)$
      $\sigma_{b,\mathrm{id}} \leftarrow \left\lfloor \bigoplus_{c \in C_b} f_L^{(r)-1}(c) \right\rfloor_{\mathrm{id}}$
      for all $s \in B_m$ do
        for all $2^3$ last-round key guesses $\kappa'_s$ of the active S-box $s$ do
          $\sigma_{b,\kappa'_s} \leftarrow \left\lfloor \bigoplus_{c \in C_b} \left(f_S^{-1} \circ f_{\kappa'_s}^{(r)-1} \circ f_L^{(r)-1}\right)(c) \right\rfloor_s$
    for $m = 1$ to 4 do                             {Recover last-round key}
      $K'_m \leftarrow$ set of $2^{3 \cdot 6}$ last-round key guesses $\kappa' = \bigoplus_{s \in B_m} \kappa'_s$
      for $b = 1$ to 18 do
        for all $\kappa' \in K'_m$ do
          if $\langle \sigma_{b,\mathrm{id}}, \lfloor b_m \rfloor_{\mathrm{id}} \rangle \oplus \bigoplus_{s \in B_m} \langle \sigma_{b,\kappa'_s}, \lfloor b_m \rfloor_s \rangle \neq 0$ then
            $K'_m \leftarrow K'_m \setminus \{\kappa'\}$
      Fix values of 18 bits $\kappa'_s$, $s \in B_m \setminus B_{m-1}$, according to $K'_m$
    if $\kappa'$ (72 bits) and $\kappa$ (21 bits) are compatible then
      return recovered $K$ from $\kappa'$ and $\kappa$
```

5.5. Discussion

5.5.1. Application to Other Parameter Sets

Besides the recommended versions LowMC-80 and LowMC-128, the designers also propose several alternative parameter sets for the 80-bit and 128-bit security level. For 128-bit security, the design document discusses the performance of LowMC-$128^{256,63}$ ($r = 12$ rounds, main variant) and LowMC-$128^{512,86}$ ($r = 11$ or 12 rounds), all with data complexity limit $d = 128$; for 80-bit security, LowMC-$80^{256,49}$ ($r = 11$ rounds, main variant, or $r = 10$) and LowMC-$80^{128,34}$ ($r = 11$ rounds), all with data complexity limit $d = 64$. Below, we discuss the applicability of our attack techniques to the individual instances. The results are summarized in Table 5.3.

Table 5.3.: Key recovery attacks for different LowMC-$k^{n,m}$ variants.

Variant	Rounds	$\dim V$	κ	κ'	Data, Time
LowMC-$80^{256,49}$	9 / 10, 11	33	21-bit	$4\times$18-bit	$2^{58.2}$
LowMC-$80^{128,34}$	8 / 11	33	—	42-bit	$2^{38.4}$
LowMC-$128^{256,63}$	9 / 12	65	—	72-bit	$2^{71.2}$
LowMC-$128^{512,86}$	10 / 11, 12	65	0-bit	$24\times$3-bit	$2^{66.6}$

For LowMC-$128^{256,63}$, the basic 7-round key recovery with direct-sum inputs as in Section 5.3.2 and partial zero-sum key recovery as in Section 5.4.1 applies for the same number of rounds, with the same complexity. Furthermore, due to the increased logarithmic data complexity limit, an additional round can be added here (for a total of 8 rounds), and the data complexity increased accordingly. However, the size of the identity part, $\ell = 67$, is too small to append rounds with initial-key-guessing as in Section 5.3.3: the necessary number of about $3 \cdot 40$ guessed s-box key bits becomes prohibitive. Final-key-guessing as in Section 5.4.2, on the other hand, is applicable in a similar way. Again, the smaller identity part increases the complexity: instead of masks b with 6 active S-boxes, about 24 active S-boxes are necessary for a reasonably high probability. If the correct $3 \cdot 24$-bit subkey is recovered as described in Section 5.4.2, the computational complexity is about $2^{71.2}$ encryptions (for up to 9 rounds). however, it is possible to optimize this step at the cost of a slightly higher data complexity.

For LowMC-128^{512,86}, on the other hand, the size of the identity part $\ell = 254$ is almost as large as the s-box part of $3 \cdot m = 258$ bits. This allows the application of initial-key-guessing for free, and 1 active S-box is expected to be sufficient for final-key-guessing. Additionally, due to the higher logarithmic data complexity limit of $d = 128$, the core cube degree can be increased to 64 (f^6) to add another round, for a total of 10 attacked rounds (out of 11 or 12).

For LowMC-80^{128,34}, $\ell = 26$, so the same problems as for LowMC-128^{256,63} apply. For the final-key-guessing, about 14 active S-boxes would be required to find suitable a, b, to attack a total of 8 rounds.

We want to stress that all described attacks are generic for the design of LowMC, without requiring specific instances of the linear layer f_L or the key schedule matrices. For specific "weak" choices of the random matrices, it is likely that attacks on more rounds are feasible.

5.5.2. Dinur et al.'s Interpolation attacks

In independent research, Dinur et al. [DLMW15] also investigate the security of LowMC against high-order differential cryptanalysis. They start with a similar basic 5-round higher-order differential distinguisher with 2^{32} chosen plaintexts for LowMC-80 (or 6-round LowMC-128), similarly extended by 1 initial round and 1 final round that exploit the incomplete S-box layer, as illustrated in Figure 5.8. In Section 5.3.3 and Section 5.4.2, we showed how V and linear masks can be carefully constructed such that guessing a handful of S-box keys allows bridging two additional rounds to attack a total of 9 rounds. Dinur et al. [DLMW15] propose a different approach that allows covering several more final rounds during the key recovery. They apply the interpolation attack by Jakobsen and Knudsen [JK97], which recovers the key by interpolating the algebraic normal form of a bit b in terms of the ciphertext and key bits. The low degree of

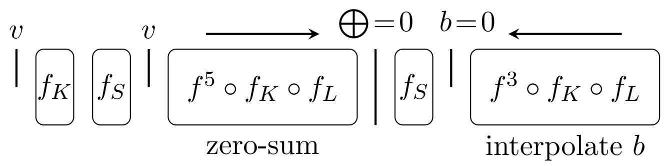


Figure 5.8.: Interpolation attack for 10-round LowMC-80 with $|V| = 2^{32}$.

2 of the inverse round function f^{-1} and the incomplete S-box layer of LowMC allow to combine the advantages of the two dual variants of the interpolation attack. This gives rise to an optimized interpolation attack with significantly fewer variables that can cover up to 3 (for LowMC-80) or 4 rounds (for LowMC-128). In total, this approach can cover up to 10 rounds of LowMC-80 and the full 12 rounds for LowMC-128, see Table 5.4.

If additionally the linear layer f_L right after the output of the zero-sum distinguisher is "weak" for the specific instance of LowMC, the attacks can be extended by 1 round for LowMC-80, or be optimized for LowMC-128 by interpolating only 3 rounds. A linear layer is called weak if there is a linear dependency between the ℓ bits of the identity part at the input and output. This corresponds to linear masks as in Section 5.4.2 with zero activated S-boxes. The probability that a random linear layer is weak is 2^{-38} for LowMC-80, and 2^{-122} for LowMC-128.

Table 5.4.: Optimized interpolation attacks by Dinur et al. [DLMW15].

Variant	Weak	Rounds	$\dim V$	Data	Time
LowMC-80^{256,49}	all	10 / 10, 11	32	2^{39}	2^{57}
LowMC-80^{256,49}	2^{-38}	11 / 10, 11	32	2^{39}	2^{57}
LowMC-128^{256,63}	all	12 / 12	64	2^{73}	2^{118}
LowMC-128^{256,63}	2^{-122}	12 / 12	64	2^{70}	2^{86}

5.6. Conclusion

We analyzed LowMC, which targets applications from asymmetric cryptography with its low multiplicative complexity. Our analysis shows that the dense linear layers are not sufficient to compensate for the incomplete 3-bit S-box layer and its low degree in both directions. By combining higher-order differential attacks with carefully constructed vector spaces, we can recover the secret key for up to 9 (out of 11 or 10) rounds of LowMC-80, and show that the security margin is similarly small for the other proposed LowMC variants. Our attacks are more efficient for the variants with larger identity parts of the incomplete S-box layers, even if these also feature larger state sizes and more S-boxes in total. Dinur et al. [DLMW15] demonstrated that higher-order differential attacks can even be extended to full-round LowMC with optimized interpolation attacks for key recovery. The designers proposed an updated LowMC v2 [ARS+16].

6 Related-Key Forgeries for Prøst

In this chapter, we analyze the authenticated cipher Prøst, submitted to Round 1 of the CAESAR competition by Kavun et al. [KLL+14]. We show that using a permutation in Even-Mansour construction as a block cipher in higher-level constructions, as proposed in Prøst-OTR, can lead to significant problems in simple related-key settings. Such settings are not covered by the security notions of the building blocks' security proofs, but we argue that they may nevertheless be relevant in practical implementations. We propose relatively generic forgery attacks that work for almost any message, key K, and permutation, assuming that the attacker knows some ciphertext encrypted under a different key K' with known difference $\Delta = K \oplus K'$.

The results in this chapter are based on joint work with Christoph Dobraunig and Florian Mendel. I am the main author and developed significant parts of the attack. The following text is a version with minor modifications of the paper published at FSE 2015 [DEM15d].

6.1. Introduction

The Even-Mansour scheme [EM91; EM97] is one of the earliest and simplest proposals for permutation-based cryptography, that is, creating different cryptographic primitives from a publicly known building block, a permutation. The scheme creates a block cipher $E_K(M)$ from a public pseudorandom permutation P as $E_{K_1\|K_2}(M) = P(M \oplus K_1) \oplus K_2$. Since its original proposal, its security has been studied extensively [Dae91; BW00; GR04], and several more [DKS12] or less [BKL+12] simple variants have been proposed and analyzed as well. However, it has also been criticized for its non-ideal properties, such as its birthday-level security [Dae91].

Prøst, designed by Kavun, Lauridsen, Leander, Rechberger, Schwabe, and Yalçın [KLL+14], is one of the candidates submitted to Round 1 of the CAESAR competition for authenticated encryption [CAE13]. It combines a newly designed permutation, the Prøst permutation, with several modes of operation. The resulting Prøst family of authenticated ciphers consists of three variants: Prøst-COPA, Prøst-OTR, and Prøst-APE, each with its own advantages and features. The Prøst-OTR variant uses the Prøst permutation in a single-key Even-Mansour construction [DKS12] as a block cipher in Minematsu's provably secure, Feistel-based OTR mode of operation [Min14].

We present a forgery attack on Prøst-OTR in a related-key setting. The scenario is that an attacker is given ciphertexts and tags of two messages: one under the target key K, and one under a related key $K \oplus \Delta$ for some arbitrary Δ. Both keys are secret, but their difference Δ is known to the attacker. The nonces used for encrypting the two messages are also related in a similar way. Then, with negligible computational complexity, the attacker can forge the ciphertext and authentication tag for a third message under the target key K. In fact, depending on the length of the original messages, forgeries for a large number of fake messages can be obtained. In addition, in case the attacker has control over one of the two originally encrypted messages, he can even control the content of the third, forged message.

Our attack is generic and exploits the combination of the OTR mode of operation with an Even-Mansour block cipher construction. It is independent of the used permutation, and thus does not use any particular properties or weaknesses of the Prøst permutation. Consequently, the other members of the Prøst family, Prøst-COPA and Prøst-APE, are not affected or endangered by the attack. However, the attack demonstrates the possible complications of using an Even-Mansour construction as a block cipher in otherwise secure modes of operation. Although this construction has been studied extensively and proven secure under different notions of security, it is inherently susceptible to related-key attacks. The OTR mode of operation allows lifting this property to the full encryption and authentication scheme and thus create forgeries. Karpman [Kar15] showed how to extend such observations to key recovery attacks. This unfortunate combination of otherwise secure building blocks shows two things: that the Even-Mansour construction should only be used cautiously in higher-level constructions, and that related-key properties are not well covered by the classical security notions, although they can lead to powerful attacks in practically relevant scenarios. Cogliati and Seurin [CS15] and Farshim and Procter [FP15] investigated under which conditions variants of the Even-Mansour construction with several rounds or nonlinear key schedule can be proved secure in the related-key model.

Related-key attacks, introduced independently by Knudsen [Knu91] and Biham [Bih93], are a relatively strong attack setting. They allow an attacker to query not only encryptions under the target key K, but also under related keys $K' = \varphi(K)$ for relations chosen by (or known to) the attacker from some set $\varphi \in \Phi$ [BK03]. A prominent example is the related-key attack on AES by Biryukov et al. [BKN09], which makes very strong assumptions about the relations between subkeys. Nevertheless, depending on the exact requirements and set Φ, they can be quite relevant in practical scenarios. In particular, scenarios where only a known (but arbitrary) xor difference $\Delta = K \oplus K'$ between any two unknown keys is required, like in our attack, are quite realistic, and occur as side effects of several published protocols. The only limitation that our attack imposes on Δ is that it does not affect the less significant half of the key bits. For compatibility with the nonce difference, the modified part of the key must not be longer than the nonce length (half the key size in Prøst-OTR).

As an example for related keys in practice, consider the WEP standard [IEE97]. There, the keys for the individual communication links are derived by concatenating (public, random) IVs with the fixed secret WEP key. Clearly, any two keys constructed this way have a publicly known differential relation. Another example for the relevance of security against related-key attacks using only two keys related via a fixed known xor difference is the corrected security proof for the 3GPP schemes f8 and f9, which requires the underlying block cipher to be secure in this setting [IK04]. Similar scenarios could be imagined in any other network of resource-constrained devices (e.g., of sensor nodes), where individual encryption keys need to be derived in a cheap way from some master secret (e.g., by xoring individual IDs, nonces or challenge values to the key). Despite its inherent susceptibility to birthday attacks, the idea to "xor nonce to key" was proposed as a generic approach to building tweakable block ciphers, and is also incorporated in several CAESAR candidates, such as Round-1 submissions AVALANCHE [Alo14] and Calico [Tay14]. A completely different scenario is that related keys might be caused by injecting faults in stored key material.

The additional requirement of related nonces is not as strong as the related keys. In many applications, nonces are generated in a very predictable pattern (typically a simple counter as a message sequence number). In some cases, the attacker may even be able to influence the nonce counter: a simple example is by triggering encryptions until the nonce counter arrives at the desired value, or by somehow causing the device to jump the unwanted nonce values. For this reason, standard security notions for authenticated ciphers assume the nonce to be under full control of the attacker, except that they must not be repeated [Rog04b]. We note that our attack does not require "nonce misuse" in the sense that the attacker requests repeated encryptions under the same nonce. The combination of related keys with related nonces has previously been applied primarily to stream ciphers, in particular in the context of the eSTREAM project. Examples include the key recovery attacks on Grain-v1 and Grain-128 by Lee et al. [LJSH08], or the analysis of generic chosen-IV attacks with applications to Trivium by Pasalic and Wei [PW13].

Outline

We first describe the Prøst family of authenticated ciphers in Section 6.2. In Section 6.3, we derive a first basic related-key attack on Prøst-OTR. In Section 6.4, we propose a few possible improvements to the attack and extended attack scenarios. Finally, in Section 6.5, we conclude with a discussion of the applicability of the Prøst-OTR attack to other authenticated encryption modes.

6.2. Description of Prøst

6.2.1. The Prøst Family of Authenticated Ciphers

Prøst is a family of authenticated encryption algorithms. Kavun et al. [KLL+14] proposed the cipher family as a candidate in the currently ongoing CAESAR competition [CAE13] for authenticated ciphers. Prøst comes in three flavors: Prøst-COPA, Prøst-OTR, and Prøst-APE. All flavors share the same core permutation Prøst, designed by Kavun et al. [KLL+14], but use it in different modes of operation.

Prøst-APE uses the Prøst permutation in Andreeva et al.'s sponge-based APE mode [ABB+14]. The other two flavors, Prøst-OTR and Prøst-COPA, use modes of operation that are originally not permutation-based, but block-cipher-based: Andreeva et al.'s COPA mode [ABL+13], and Minematsu's OTR mode [Min14]. In these variants, the Prøst permutation is used in a single-key Even-Mansour construction [DKS12] to provide the required block cipher.

Each of the three flavors is available in two security levels, specified by a parameter $n \in \{128, 256\}$, resulting in a total of six proposed cipher family members. The designers rank the COPA variants as their primary recommendations, the OTR variants second, and the APE variants last.

Throughout this paper, we use essentially the same notation as Prøst's designers [KLL+14]. Unless noted otherwise, all operations are performed in $\mathbb{F}_{2^{2n}}$ with respect to Prøst's irreducible polynomial, where $n \in \{128, 256\}$ defines the security level. For convenience of notation, elements in $\mathbb{F}_{2^{2n}}$ are often represented interchangeably as elements of $\mathbb{F}_2^{2n}$.

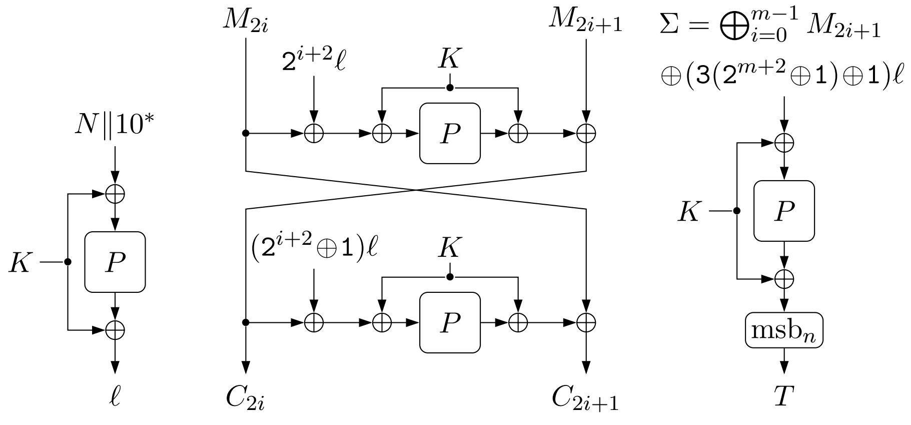
6.2.2. Prøst-OTR-n

Prøst-OTR-n uses the block cipher $\tilde{P}_K$, built from the permutation P in a single-key Even-Mansour construction [DKS12], in Minematsu's OTR mode of operation [Min14]. The result is a nonce-based authenticated encryption scheme with online encryption and decryption that is fully parallelizable [KLL+14]. Prøst-OTR-n is proposed in two security levels, $n \in \{128, 256\}$. The security level defines the permutation size $2n$ and block size $2n$, the key size $2n$ and nonce size n, and the tag size n. The claimed security for Prøst-OTR-n is $\frac{n}{2}$ bits (confidentiality and integrity of plaintext and integrity of associated data). No particular claims are made for or against the related-key security of the cipher.

Since our attack does not exploit any particular properties of the Prøst permutation $P : \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$, we do not include the definition of P in this description. The design of the permutation-based block cipher $\tilde{P}_K$, however, is essential for the attack. For a key $K \in \mathbb{F}_2^{2n}$, the block cipher $\tilde{P}_K : \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$ is defined as follows:

$$\tilde{P}_K(x) = K \oplus P(x \oplus K).$$

In OTR, message blocks M_j are encrypted in pairs in 2-round Feistel networks to get the ciphertext blocks C_j. The Feistel round function first adds a counter-like value, then applies the block cipher $\tilde{P}_k$. For the counter-like value, a helper value ℓ is computed in an initialization phase



(a) Initialization **(b)** Encrypting M_{2i}, M_{2i+1}, $0 \le i < m$ **(c)** Finalization

Figure 6.1.: Encrypting $2m$ message blocks M_j with Prøst-OTR-n under key K and nonce N. All values are $2n$ bits, with $n \in \{128, 256\}$, except the n-bit tag T.

by encrypting the padded nonce $N\|10^*$ under $\tilde{P}_K$. After processing all block pairs, the tag T is finally computed by encrypting a function of the checksum Σ, which is the xor of all odd-indexed message blocks M_{2i+1}. The detailed algorithm is listed in Algorithm 3 and illustrated in Figure 6.1. For simplicity, we only describe the mode for empty associated data, and only for padded messages with an even (rather than odd) number of message blocks.

Algorithm 3 Prøst-OTR-n encryption

```
Input: padded message $M\|01^* = M_0 \cdots M_{2m-1}$, padded nonce $N\|10^*$
Output: ciphertext $C = C_0 \cdots C_{2m-1}$, tag $T$
  $\Sigma \leftarrow 0$
  $\ell \leftarrow \tilde{P}_K(N\|10^*)$
  for $i = 0, \dots, m-1$ do
    $C_{2i} \leftarrow \tilde{P}_K(2^{i+2}\ell \oplus M_{2i}) \oplus M_{2i+1}$
    $C_{2i+1} \leftarrow \tilde{P}_K(2^{i+2}\ell \oplus \ell \oplus C_{2i}) \oplus M_{2i}$
    $\Sigma \leftarrow \Sigma \oplus M_{2i+1}$
  $T \leftarrow \mathrm{msb}_n(\tilde{P}_K(3(2^{m+2}\ell \oplus \ell) \oplus \ell \oplus \Sigma))$
```

6.3. Forgery Attack

In this section, we describe our basic forgery attack on Prøst-OTR. The attack exploits the combination of the OTR mode with the Even-Mansour block cipher construction, and is independent of the concrete permutation P used. We consider a related-key scenario where encrypted messages of two different keys K and K' can be observed. Both K and K' are secret, but we assume the attacker knows the difference $\Delta = K \oplus K'$ (i.e., $K' = K \oplus \Delta$). In addition, we assume that the attacker can observe encrypted messages for related nonces N, N', such that $\Delta = (N \| 10^*) \oplus (N' \| 10^*)$. Since the last n bits of the padded nonces are identical, this means that the n least significant bits of Δ must be 0.

The basic idea of the proposed forgery attack is to combine information from the encryption of the same message M under the two related keys K, K' to forge a ciphertext and tag for a modified message M^* under one of the two keys, K. More specifically, we will first show how to use the ciphertext from the related key $K' = K \oplus \Delta$ to forge ciphertexts for modified messages under the target key K. Then, we will combine original and forged ciphertexts in a way such that the original tag remains valid for the resulting modified plaintext under K. The attack works for any plaintext of sufficient length (≥ 514 message blocks for Prøst-OTR-128, ≥ 1026 blocks for Prøst-OTR-256).

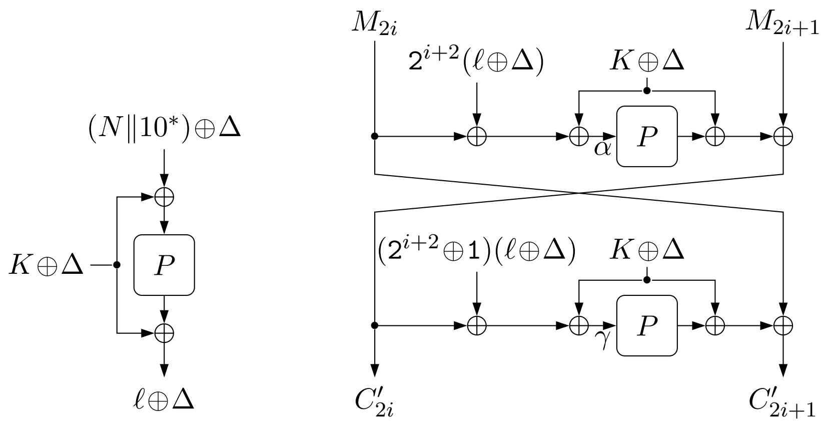
6.3.1. Forging the Ciphertext

Assume that the attacker obtains the ciphertext for the same message $M = M_0 \cdots M_{2m-1}$ (from Figure 6.1) under a related key $K' = K \oplus \Delta$ and a related nonce $N' \| 10^* = (N \| 10^*) \oplus \Delta$, as illustrated in Figure 6.2. Note that since the nonce only has length n (instead of $2n$ like the other values), Δ must only modify the most significant n bits, i.e., $\Delta = \Delta_n \| 0^n$. Then, in the initialization phase illustrated in Figure 6.2a, the differences in K' and N' cancel out right before the call to the permutation P in the initialization. Thus, we receive a related counter value ℓ' with a simple relation to the original ℓ:

$$\ell' = P_{K'}(N' \| 10^*) = K \oplus \Delta \oplus P(K \oplus \Delta \oplus (N \| 10^*) \oplus \Delta) = \ell \oplus \Delta.$$

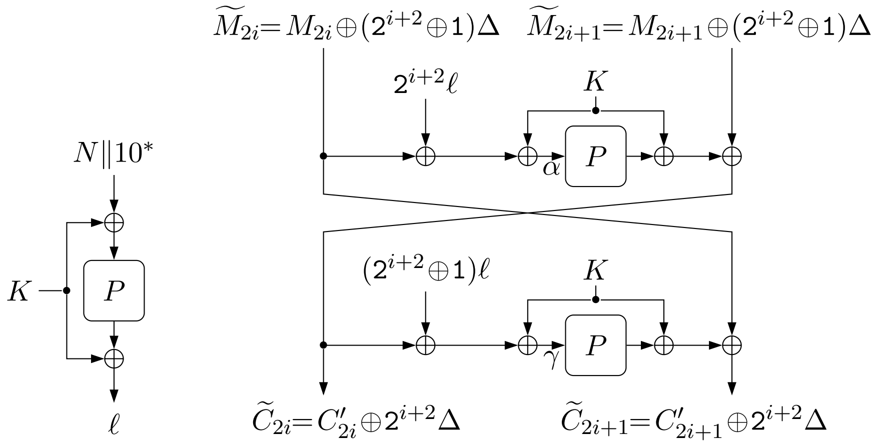
Now consider the encryption of a modified message with message blocks

$$\widetilde{M_j} = M_j \oplus (\mathsf{2}^{\lfloor j/2 \rfloor + 2} \oplus \mathsf{1})\Delta$$



(a) Initialization **(b)** Message blocks M_{2i}, M_{2i+1}, $0 \leq i < m$

Figure 6.2.: Encrypting the original message blocks M_j under a related key $K \oplus \Delta$ and nonce.


$N\|10^*$
K
P
ℓ
$\widetilde{M}_{2i}= M_{2i}\oplus(2^{i+2}\oplus 1)\Delta$
$\widetilde{M}_{2i+1}= M_{2i+1}\oplus(2^{i+2}\oplus 1)\Delta$
$2^{i+2}\ell$
K
α
P
$(2^{i+2}\oplus 1)\ell$
K
γ
P
$\widetilde{C}_{2i}= C'_{2i}\oplus 2^{i+2}\Delta$
$\widetilde{C}_{2i+1}= C'_{2i+1}\oplus 2^{i+2}\Delta$

(a) Initialization **(b)** Message blocks $\widetilde{M}_{2i}, \widetilde{M}_{2i+1}$, $0 \leq i < m$

Figure 6.3.: Encrypting modified message blocks $M_j \oplus (2^{\lfloor j/2 \rfloor+2} \oplus 1)\Delta = \widetilde{M}_j$ under the original key K and nonce N.

under the original key K and nonce N. As Figure 6.3 illustrates, the message differences “cancel out” with the corresponding difference in the ℓ values from the encryption under the related key in Figure 6.2. Thus, in both Figure 6.2 and Figure 6.3, the inputs α and γ to the permutations are the same:

$$\alpha = \widetilde{M}_{2i} \oplus 2^{i+2}\ell \oplus K = M_{2i} \oplus 2^{i+2}\ell \oplus 2^{i+2}\Delta \oplus \Delta \oplus K,$$
$$\gamma = \widetilde{M}_{2i+1} \oplus P(\alpha) \oplus (2^{i+2}\oplus 1)\ell = M_{2i+1} \oplus P(\alpha) \oplus 2^{i+2}\ell \oplus 2^{i+2}\Delta \oplus \ell \oplus \Delta.$$

For this reason, the ciphertext $\widetilde{C}_j$ of the modified message block $\widetilde{M}_j$ under the original key K can be derived from the ciphertexts C'_j of the original message M_j under the related key $K \oplus \Delta$:

$$\begin{aligned}\widetilde{C}_{2i} &= \widetilde{M}_{2i+1} \oplus P(\alpha) \oplus K = C'_{2i} \oplus 2^{i+2}\Delta,\\ \widetilde{C}_{2i+1} &= \widetilde{M}_{2i} \oplus P(\gamma) \oplus K = C'_{2i+1} \oplus 2^{i+2}\Delta,\end{aligned}$$

since

$$\begin{aligned}C'_{2i} &= M_{2i+1} \oplus P(\alpha) \oplus K \oplus \Delta,\\ C'_{2i+1} &= M_{2i} \oplus P(\gamma) \oplus K \oplus \Delta.\end{aligned}$$

Now, we know the correct ciphertexts for a modified message. However, we still need to find the corresponding authentication tag. We will try to re-use the original tag T for our forged message.

6.3.2. Forging the Tag

For a fixed key K and nonce N, the authentication tag only depends on the xor sum of all message blocks with odd index,

$$\Sigma = \bigoplus_{i=0}^{m-1} M_{2i+1}.$$

Thus, if we want to re-use the original tag T for our forged message, we need to make sure that any induced differences cancel out when summing up the message blocks. We want to combine original and modified message M and $\widetilde{M}$ to construct the final forged message M^* that satisfies this property.

For each message block pair M^*_{2i}, M^*_{2i+1} of the forged message M^*, we can decide to use either the original message block pair M_{2i}, M_{2i+1}, or the modified blocks $\widetilde{M}_{2i}, \widetilde{M}_{2i+1}$. Let λ_i denote whether we use the original ($\lambda_i = 0$) or modified ($\lambda_i = 1$) block pair for $0 \leq i < m$. Then, we get the message sum

$$\Sigma^* = \bigoplus_{i=0}^{m-1} M^*_{2i+1} = \Sigma \oplus \bigoplus_{i=0}^{m-1} \lambda_i (\mathsf{2}^{i+2} \oplus \mathsf{1})\Delta.$$

Note that if Σ would sum up all message blocks (not only every second), then any choice of λ_i would create a successful forgery, since $M_{2i} \oplus M_{2i+1} = \widetilde{M}_{2i} \oplus \widetilde{M}_{2i+1}$. As it is, however, we need to select suitable coefficients $\lambda_i \in \mathbb{F}_2$ such that at least one coefficient λ_{i^*} is non-zero and

$$\bigoplus_{i=0}^{m-1} \lambda_i (\mathsf{2}^{i+2} \oplus \mathsf{1})\Delta = \mathsf{0} \quad \Leftrightarrow \quad \bigoplus_{i=0}^{m-1} \lambda_i (\mathsf{2}^{i+2} \oplus \mathsf{1}) = \mathsf{0}. \tag{6.1}$$

Since $\{(\mathsf{2}^{i+2} \oplus \mathsf{1})\Delta\} \subseteq \mathbb{F}_2^{2n}$, a vector space with dimension $2n$, any $2n+1$ such vectors are linearly dependent, and suitable coefficients λ_i exist. Thus, for any given key difference Δ and known plaintext M with $2m \geq 4n+2$ message blocks, we can solve this system of equations to find suitable coefficients λ_i. The ciphertext blocks C^* for the resulting forged message M^* can be computed as in Section 6.3.1, while the correct tag $T^* = T$ can be copied from M.

As an example that can be easily verified with the reference implementation, we target Prøst-OTR-128 with $n = 128$ and field modulus $f(x) = x^{256} + x^{10} + x^5 + x^2 + 1$. Assume any message M with at least 514 blocks of 256 bits was encrypted under K and nonce N to ciphertext C and tag T, and under K' and N' to C' and T' for any $\Delta = \Delta_n \| 0^n$. We can now forge tag T^* and ciphertext C^* for the modified message M^*, which differs from M in blocks indices $j \in J$:

$$\begin{aligned}
J &= \{4, 5, 6, 7, 10, 11, 16, 17, 20, 21, 508, 509, 512, 513\}, \\
M^*_j &= \begin{cases} M_j \oplus (\mathsf{2}^{\lfloor \frac{j}{2} \rfloor + 2} \oplus \mathsf{1})\Delta & j \in J, \\ M_j & \text{else;} \end{cases} \\
C^*_j &= \begin{cases} C'_j \oplus \mathsf{2}^{\lfloor \frac{j}{2} \rfloor + 2}\Delta & j \in J, \\ C_j & \text{else;} \end{cases} \\
T^* &= T.
\end{aligned}$$

Summarizing, from observing the ciphertext and tag for encryptions of the same message M under two related keys K and $K' = K \oplus \Delta$, the attacker has forged the ciphertext C^* and tag T^* for a different message M^* of the same block length with negligible computational effort. The attacker knows this forged message but has almost no control over its contents. The attack nonce is the same as the original nonce N. We discuss some remarks and improvements to this attack in Section 6.4.

6.4. Remarks and Attack Variants

6.4.1. Remarks on the Message Length

If an attacker carries out the basic attack as in Section 6.3, the modified message may have a slightly modified bit length. This is because the modification can shift the last non-zero bit, which marks the beginning of the message padding. This is not a problem since the message bit-length is not encoded anywhere else in the encryption process – except in the rare case that the last non-zero bit moves to the second-to-last block or earlier, which is not a valid format for the padded plaintext. This can be avoided by not including the last block pair in the modification process.

The attack is also applicable to messages $M = M_0 \cdots M_{2m-1} M_{2m}$ with an odd number of blocks: simply do not include the last block M_{2m} in the modification process, and copy it directly to M^*_{2m}. The same holds true for messages that include associated data A: simply copy the same associated data to the forged message.

6.4.2. Unknown Messages

The description in Section 6.3 assumes that the same message M is encrypted under both keys, K and $K' = K \oplus \Delta$, and that M is known to the attacker. This is, however, not necessarily required. Even without knowing M, the attacker can compute forged ciphertext blocks and the tag. In this case, he will not know the modified message M^*, but only the induced difference $M^* \oplus M$.

Neither is it necessary that the same message M be encrypted under both K and $K \oplus \Delta$. In fact, it is sufficient that the attacker has access to the ciphertexts for any two (not necessarily known, not necessarily equal-length) messages M (under K) and M' (under $K' = K \oplus \Delta$), and knows the difference $M_{2i+1} \oplus M'_{2i+1}$ for at least $2n+1$ values of i. Let I be the set of indices i with known message differences, with $|I| \geq 2n+1$. Then, the attacker solves

$$\bigoplus_{i \in I} \lambda_i \left(M_{2i+1} \oplus M'_{2i+1} \oplus (2^{i+2}+1)\Delta\right) = 0.$$

The forged message M^* (not known to the attacker, same block length as M), ciphertext C^* and tag T^* are then given by

$$\begin{aligned}
(M^*_{2i}, M^*_{2i+1}) &= \begin{cases} (M'_{2i} \oplus (2^{i+2}+1)\Delta, M'_{2i+1} \oplus (2^{i+2}+1)\Delta) & \lambda_i = 1, \\ (M_{2i}, M_{2i+1}) & \text{else;} \end{cases} \\
(C^*_{2i}, C^*_{2i+1}) &= \begin{cases} (C'_{2i} \oplus 2^{i+2}\Delta, C'_{2i+1} \oplus 2^{i+2}\Delta) & \lambda_i = 1, \\ (C_{2i}, C_{2i+1}) & \text{else;} \end{cases} \\
T^* &= T.
\end{aligned}$$

6.4.3. Multiple Forgeries

As described in Section 6.3 and 6.4.2, an attacker can forge one message from $4n+2$ original message blocks. This can be extended to 2^s-1 different forgeries from $4n+2s$ blocks (i.e., $|I| \geq 2n+s$). Then, the homogeneous linear system

$$\bigoplus_{i \in I} \lambda_i \left(M_{2i+1} \oplus M'_{2i+1} \oplus (2^{i+2}+1)\Delta\right) = 0$$

is underdetermined with $\geq 2n+s$ variables for $2n$ equations. Thus, the solution space has dimension $\geq s$, containing $\geq 2^s - 1$ different non-zero solutions for λ.

In the case $M_j = M'_j$, different values λ, λ' produce different plaintexts as long as

$$\max\{i \in I : \lambda_i \neq \lambda'_i\} < \operatorname{ord}(2) - 2,$$

where $\operatorname{ord}(2)$ denotes the multiplicative order of 2 in $\mathbb{F}^*_{2^{2n}}$. For Prøst's irreducible polynomials, $\operatorname{ord}(2) = 2^{256}-1$ for $n = 128$ and $\operatorname{ord}(2) = 2^{512}-1$ for $n = 256$. In general, if

$$M_{2i+1} \oplus M'_{2i+1} \oplus (2^{i+2}+1)\Delta \neq 0 \qquad \forall i \in I,$$

then all different λ produce different forgeries.

6.4.4. Almost Universal Forgery with Related-Key Queries

Assume that the attacker can query the encryption of a chosen message under one of the two keys, $K' = K \oplus \Delta$. He wants to forge the ciphertext and tag for a meaningful message M^* (chosen beforehand or provided externally) under the original key K. He can achieve this goal if (a) M^* has an even number of blocks, (b) he has access to the tag T of a known message M with the same number of blocks as M^* under the key K, and (c) he can modify one $2n$-bit block with odd index of M^* (or, alternatively, of M). The attack works as follows:

1. Fix the target message length $|M^*| = 2m$ (in blocks).
2. Obtain tag T for any known message M with $|M| = 2m$ under key K and any nonce N.
3. Fix the preliminary target (challenge) message M^*.
4. Let $j^* = 2i^* + 1$ be the modifiable block of M^*. Modify
$$M^*_{2i^*+1} = M_{2i^*+1} \oplus \bigoplus_{i \neq i^*} \left(M_{2i+1} \oplus M^*_{2i+1}\right).$$
5. Construct the query message M' for $i = 0, \dots, m-1$ as
$$(M'_{2i}, M'_{2i+1}) = (M^*_{2i} \oplus (2^{i+1} \oplus 1)\Delta, M^*_{2i+1} \oplus (2^{i+1} \oplus 1)\Delta).$$
6. Request the ciphertext C' for the query message M' under $K' = K \oplus \Delta$ with nonce $N' \| 10^* = (N \| 10^*) \oplus \Delta$.
7. The forged ciphertext C^* and tag T^* for message M^* and nonce $N^* = N$ can be computed as
$$\begin{aligned}(C^*_{2i}, C^*_{2i+1}) &= (C'_{2i} \oplus 2^{i+2}\Delta, C'_{2i+1} \oplus 2^{i+2}\Delta) \qquad i = 0, \dots, m-1, \\ T^* &= T.\end{aligned}$$

This is essentially the same strategy as in Section 6.4.2, except that instead of using fixed M, M' and adapting M^*, we fix M, M^* and adapt M'. To avoid solving the equation system for the correct λ_i (which would require relatively long message lengths $2m$, and force us to have $M^*_j = M_j$ for many j), we modify one block $M^*_{j^*}$ to make $\forall i: \lambda_i = 1$ a valid solution.

6.5. Discussion

6.5.1. Applicability to Other Modes

The core of our attack is the following observation: If an authenticated encryption mode applies the block cipher to variable (controllable) inputs, an attacker can lift the inherent related-key weaknesses of the Even-Mansour construction to the entire mode. Then, he can use information from encryptions under a related key to forge ciphertext and tag for the target key.

A question that suggests itself is whether similar attacks are possible on other Prøst modes. In addition, other authenticated encryption modes might display similar problems when combined with an Even-Mansour block cipher.

Prøst-APE does not use the Even-Mansour construction at all, but plugs the permutation into a sponge construction. Thus, the attack is clearly not applicable. Prøst-COPA does use the permutation in an Even-Mansour construction. However, it seems to defy the attack by including $E_K(0)$, the encryption of the value 0, in the definition of the helper value L (which plays a role similar to ℓ in Prøst-OTR). Since a constant instead of the variable nonce N serves as input to the encryption, the input cannot be controlled to produce (differentially) predictable outputs of L. In fact, Mennink [Men16] proves that Prøst-COPA uses an instance of XPX, a generalized tweakable Even-Mansour construction, and is thus secure in related-key settings. The situation is similar, for example, for the OCB mode of operation [KR14]: while the message could be used to cancel out differences in the helper counter value, this value is also derived from the encryption $E_K(0)$ of the zero value and thus unpredictable.

On the other hand, other popular modes show significant weaknesses when combined with Even-Mansour ciphers. Of course, unlike Prøst, these modes are usually not recommended for use with an Even-Mansour block cipher, but with AES. Consider, for example, the CCM mode of operation [Dwo04; WHF03], an ISO/IEC-standardized combination of CBC-MAC with CTR encryption, as illustrated in Figure 6.4. CCM allows a much simpler related-key attack. Assume that an attacker knows the ciphertext (including the tag) $C = C_1 \cdots C_\ell C_{\ell+1}$ of a message $M = M_1 \cdots M_\ell$ under key $K \oplus \Delta$ and padded nonce $(N \| 0) \oplus \Delta$ (in the format used as counter input to

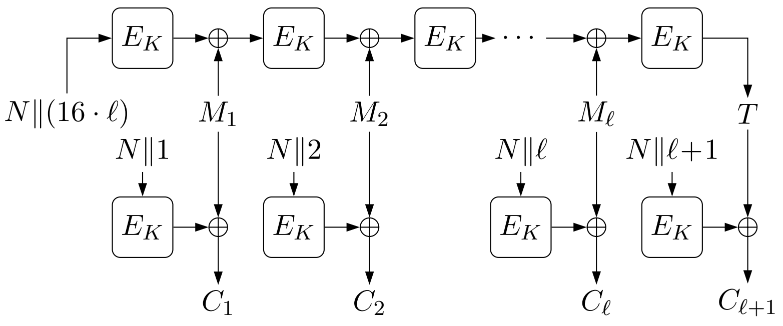


Figure 6.4.: CCM encryption.

the CTR encryption). Then, the ciphertext C' for M under key K and padded nonce $N\|0$ is simply

$$C'_i = \begin{cases} C_i \oplus \Delta & 1 \leq i \leq \ell, \\ C_i & i = \ell + 1. \end{cases}$$

As can be observed from Figure 6.4, all differences Δ during the CCM computation cancel out either with the nonce difference fed to the Even-Mansour block encryptions $E_{K\oplus\Delta}$, or with neighboring block cipher calls in the CBC-MAC computation. The final differences in the block cipher outputs from the CTR encryption can simply be added to the ciphertext blocks.

6.5.2. Karpman's Key Recovery Attack

Karpman [Kar15] shows how a related-key distinguisher on an Even-Mansour construction can be converted to a related-key key-recovery attack for a different class of relations. He illustrates the approach by converting the same distinguisher we use for our forgeries into a key-recovery attack on Prøst-OTR.

The approach is based on the work of Bellare and Kohno [BK03], who show the impossibility of achieving related-key security against certain classes of key relations, in particular $\Phi^+ \cup \Phi^\oplus$, where Φ^+ modifies keys by modular addition of constants, whereas $\Phi^\oplus$ adds them with bitwise xor. The reason for this is that relations in Φ^+ and $\Phi^\oplus$ collide on certain inputs, depending on the exact bit values, and detecting these collisions allows deriving the corresponding bit values. The same idea can be applied by testing our $\Phi^{\oplus}$-based distinguisher systematically for Φ^{+}-related keys. This way, the more significant half of the key bits can be recovered. Recovering the less significant half is a bit more challenging since the corresponding padded nonce bits are always free of differences. By taking advantage of carry-propagation and knowledge of the more significant half, the full key can be recovered with a number of related-key queries linear in the number of key bits.

Overall, this leads to a much more powerful key recovery attack. The downside is that the attack scenario also comes with stronger requirements for the attacker's capabilities, who needs to be able to query chosen messages under many adaptively chosen Φ^{+}-related keys.

6.6. Conclusion

In this chapter, we analyzed the related-key security of the authenticated cipher Prøst-OTR. The design is an example of how even more complex modes can allow some undesirable properties of the Even-Mansour construction to be lifted to the complete authentication mode, in this case, to generate related-key forgeries. The Even-Mansour construction is not well-suited as a general-purpose block cipher construction for all modes of operation. The rising popularity of sponge modes, and permutation-based encryption in general, may lead to interesting new observations in this direction.

We stress again that the presented attack only concerns the OTR variant of Prøst. The security of the other modes, Prøst-COPA and Prøst-APE, and in particular of the Prøst permutation itself, remains unaffected. However, Prøst did not advance to Round 2 of the CAESAR competition. It may be possible to tweak OTR to prevent the specific attack, for example by adapting the initialization of ℓ to include $\tilde{P}_K(0)$, similar to COPA and OCB. However, the general interactions of modes like OTR or CCM with the single-key Even-Mansour construction remains a reason for concern.

Part II.

Automating Differential Cryptanalysis

7 Practical Collision Search for Round-Reduced SHA-2

In this chapter, we propose techniques to improve the automated, practical collision search for round-reduced SHA-2. We focus in particular on the challenges of applying previous methods developed for SHA-256 [MNS11b; MNS13b] to SHA-512, which arise from the increased state size of the compression function. We extend a dedicated search tool for differential characteristics with several techniques for faster propagation and more targeted search heuristics to improve the efficiency of the search for larger primitives. With the improved tool, we can find practical semi-free-start collisions for the round-reduced hash standard SHA-512 and its truncated variants, including collisions for 27 (out of 80) steps and semi-free-start collisions for 39 steps of all SHA-512 variants, as well as free-start collisions for up to 44 steps of selected truncated SHA-512 variants. Additionally, we show how the tool can be used to insert collision backdoors in malicious variants of SHA-1 with a few modified round constants.

The results in this chapter are based on several collaborations with Florian Mendel, Christoph Dobraunig, Martin Schläffer, Tomislav Nad, Vincent Rijmen, Ange Albertini, and Jean-Philippe Aumasson. They also build substantially on previous work of these authors. The following text covers the main contributions of papers published at WCC 2013 [EMN+13], FSE 2014 [EMS14], and ASIACRYPT 2015 [DEM15a], as well as selected aspects of a paper at SAC 2014 [AAE+14] and a report for CRYPTREC [DEM15e]. I am the main author of the first three texts, developed the proposed search techniques [EMN+13; EMS14], and conducted some of the practical searches building on previous strategies by Florian Mendel.

7.1. Introduction

Cryptographic hash functions are one of the essential building blocks in cryptography with applications in many higher-level security protocols. They are used to map any message of arbitrary length to a short, fixed-length fingerprint, the hash value. This fingerprint can be used as a placeholder for the full input message in applications where processing the full message would be inefficient (such as electronic signatures), or where storing the full message would incur security risks or defeat the security purpose entirely (for example as a preliminary commitment to a secret value, or for password systems), or, most prominently, as a short cryptographic checksum for verifying integrity. In order to serve in these applications, cryptographic hash functions must on the one hand be very efficient to evaluate, and on the other hand satisfy three essential security properties: Preimage resistance (infeasibility of finding a message that hashes to a given hash value), second-preimage resistance (infeasibility of producing a second message with the same hash value as a given message), and collision resistance (infeasibility of finding two messages that will hash to the same hash value).

The lack of secret inputs for hash functions that can serve, for example, as round keys or whitening keys is a significant additional design challenge. Compared to the steady advances in research on block ciphers and other keyed primitives, hash functions were relatively neglected and significantly less well understood at the end of the last century. While the academic efforts in analyzing the block cipher DES had lead to systematic analysis approaches like differential [BS91] and linear [Mat93; MY92; TG91] cryptanalysis, only few results were published around hash functions – in spite of the widespread use of the standards MD5 and SHA-1.

The situation changed with the breakthrough results of Wang et al. in 2005 [WY05; WYY05b], who were able to demonstrate practical collision attacks on the hash standard MD5. Their attack technique was closely related to the differential cryptanalysis of block ciphers, but required sophisticated (and less-than-well understood) manual work. The success of their approach inspired significant advances in the analysis of hash functions, in particular for several other widely-deployed members of the extensive MD4 family, like RIPEMD [WLF+05; LP13; MNSS12], SHA-1 [WYY05b; Ste13; KPS15; SKP16; SBK+17], and also the current standard SHA-2. The fact that SHA-2 may be significantly stronger than

MD4 and relatives, but is still based on the same general skeleton structure and the same design approaches, motivated the SHA-3 competition. The resulting standard SHA-3/Keccak [BDPV11d; Dwo15] did not yet replace SHA-2, but rather complements it by providing much more flexible interface and implementation options [Dwo15; KCP16], and serves as a safe backup in case the analysis of SHA-2 should advance too drastically. It is likely that the SHA-2 family will remain as ubiquitously deployed in the foreseeable future as it is now. Therefore, the continuous application of state-of-the-art cryptanalytic techniques for quantifying the security margin of the SHA-2 family is of significant practical importance.

Due to the more complex definition of SHA-2, compared to MD5 and other family members, manually-found differential attacks like those by Wang no longer seem to be feasible beyond a certain number of rounds. As a consequence, several automated search techniques have been proposed to apply search strategies similar to Wang et al.'s to newer hash functions [SO06; DR06; Leu13; Leu12; MNS11b; MNS13b].

One of the most successful approaches for finding practical hash collisions is the guess-and-determine search tool `nltool` developed by Mendel, Nad, and Schläffer [MNS11b] for practical collision attacks on round-reduced SHA-256 [MNS11b; MNS13b] and other targets [MNS11a; MNS12; MNSS12; KMNS13; MNS13a; MPS+13; DEM15b; DEMS15]. It is based on a method proposed for automatically finding characterstics for SHA-1 by De Cannière and Rechberger [DR06]. This tool takes as input the specification of a compression function or other primitive, a specification of initial constraints, and a specification of strategic search phases. Based on these bitwise specifications and constraints, the tool successively finds a compatible and consistent assignment of differences and/or values for all variables required in the respective phase. The search algorithm itself is an instance of the guess-and-determine approach widely used in combinatorial constraint solvers, such as DPLL-based SAT solvers [DLL62]: It repeatedly picks an undetermined variable, assigns a tentative value ("guess"), and applies a form of constraint propagation based on the circuit of basic operations to derive the implications of this assignment, as well as potential contradictions ("determine"). In terms of SAT solvers, the basic circuit operations play a role similar to the clauses of SAT formulas, the bit position selected for guessing would be referred to as branching literal, and the propagation is a generalization of SAT unit propagation for general Boolean functions.

So far, the practical results on SHA-2 are mostly on the family member SHA-256. The best previous practical collisions found for SHA-512 are those for 24 of 80 steps, proposed independently by Sanadhya and Sarkar [SS08] and Indesteege et al. [IMPR08], together with 24-step collisions for SHA-256. While the results for SHA-256 have since been improved to collisions on 27 [MNS11b], 28 [MNS13b] (both practical), and finally 31 steps [MNS13b] (theoretical attack with almost practical complexity), as well as semi-free-start collisions on up to 38 of 64 steps found by Mendel, Nad, and Schläffer [MNS13b], no such improvements have been proposed for SHA-512 so far. The main reason for this seems to be the doubling in state size from SHA-256 to SHA-512; this larger search space increases the difficulty of the problem for the search tools. In summary, SHA-512 was a much less popular analysis target, even though in practice, there are many situations where SHA-512 and its variants may be a better choice than SHA-256. In particular, SHA-512 is significantly faster than both SHA-256 and SHA-3/Keccak on many 64-bit platforms [BL11]. For these reasons, it has been suggested to use a truncated version of SHA-512 even for 256-bit hash values [GJW11]. NIST also defines this variant, called SHA-512/256, in FIPS 180-4 [Dan12]. In addition to performance advantages, unlike the narrow-pipe Merkle-Damgård structure of SHA-256, the generic security of SHA-512/256 profits from the wide-pipe structure of this chop-MD [CDMP05] design with is wide-pipe structure, which prohibits generic attacks like Joux' multicollision attack [Jou04], Kelsey and Kohno's herding and Nostradamus attacks [KK06], and Kelsey and Schneier's second preimages for long messages [KS05]. However, no dedicated cryptanalysis of SHA-512/224 and SHA-512/256 has been published before those summarized in this chapter.

Our contributions

We adapted several aspects of the search tool in order to improve the performance for primitives with larger state sizes, such as those of SHA-512 and SHA-3/Keccak. Two particular problems when applying the previous search algorithm to larger primitives are related to detecting contradictions soon enough: First, the search algorithm has to select and guess the critical bits that reveal the contradiction within a much larger search space; and second, it has to determine the resulting contradiction in a larger circuit. We developed, implemented, and evaluated several techniques to address these two problems. The first is a look-ahead branching heuristic that aims to identify critical bits by peeking at the implications of several candidate bits and picking the most impactful bit [EMS14]. The second is an approach for efficiently propagating partial information through large linear layers [EMN+13]. Together, they significantly decrease the time spent in “dead ends” of the search space. This allows tackling SHA-512 collision challenges with comparable or even higher round numbers than SHA-256. The best results include semi-free-start collisions for up to 39 out of 80 steps of SHA-512. We also show how to extend these results to even more steps of the truncated, wide-pipe variants SHA-512/t.

In some cases, it is even easier to find results for the wide-pipe variants SHA-512/224 and SHA-512/256 compared to the narrow-pipe designs SHA-224 and SHA-256: If characteristics with very sparse constraints are available, the higher available degrees of freedom from the larger message space may permit solutions where none exist in the narrow-pipe case. Additionally, in a free-start collision setting, the truncation of parts of the state in the wide-pipe variants allows extending characteristics by a few steps and “hide” the resulting differences [DEM15a; DEM15e].

We show that due to this truncation, practical free-start collision for 43-step SHA-512/256 and 44-step SHA-512/224 are possible. Moreover, we improve upon the previous best collisions for 24-step SHA-512 [IMPR08; SS08] and show collisions for 27 steps of SHA-512, SHA-512/224, and SHA-512/256. We also include results for the older truncated SHA-2 variants, SHA-224 and SHA-384. Since all of our results are practical, we provide examples of colliding message pairs for every attack. Our results are summarized in Table 7.1 together with previously published collision attacks.

Finally, we explore how this tool can be used not only by a cryptanalyst, but by a malicious designer who wishes to include a collision backdoor in a malicious SHA-1 variant with tweaked round constants, and give examples of meaningful collisions for such a backdoored hash function [AAE+14].

Outline. In Section 7.2, we describe the SHA-2 family and provide an overview of the existing analysis of this hash family, in particular the tool for guess-and-determine attacks that we build on. In Section 7.3 and Section 7.4, we introduce and evaluate possible approaches to improve this search tool. We present the resulting collision attacks on SHA-512 variants in Section 7.5. Finally, in Section 7.6, we show a different application for a malicious designer with backdoored variants of SHA-1.

7.2. Background

7.2.1. Description of SHA-2

The SHA-2 family of hash functions is specified by NIST as part of the Secure Hash Standard (SHS) [Dan12]. The standard defines two main algorithms, SHA-256 and SHA-512, with truncated variants SHA-224 (based on SHA-256) and SHA-512/224, SHA-512/256, and SHA-384 (based on SHA-512). In addition, NIST defines a general truncation procedure for arbitrary output lengths up to 512 bits. Below, we first describe the two main variants SHA-256 and SHA-512, followed by a brief discussion of the truncated variants.

SHA-256 and SHA-512. The two main SHA-2 variants follow the design principles established in the previous designs of the family, such as MD4, MD5, and SHA-1, but increase the state size and complexity of the round update function. They instantiate a narrow-pipe Merkle-Damgård mode [Dam89; Mer89] with a compression function obtained from a Davies-Meyer construction with a dedicated block cipher. The block ciphers of SHA-256 and SHA-512 are closely related, but differ in their word sizes of $w = 32$ and $w = 64$ bits, respectively, and their resulting interface sizes.

The Merkle-Damgård mode of SHA-2 pads the input message M to ℓ message blocks m_j, $0 \leq j < \ell$, of $b = 16w$ bits. The appended padding consists of the bit "1", as many bits "0" as necessary to fill the length (mod b) to $b - 2w$ bits, and finally the bit-length of the original input message encoded as a $2w$-bit integer. The final t-bit hash value $T = \mathsf{SHA\text{-}2}(M) = h_\ell$ is then computed by iterative application of the compression function F to these message blocks and the intermediate chaining variables h_i, starting from a pre-defined, constant initial value IV. SHA-256 operates on message blocks m_j of $b = 16 \times 32 = 512$ bits and produces chaining variables h_j and a hash output T of $t = 8 \times 32 = 256$ bits. SHA-512 uses message blocks of $b = 16 \times 64 = 1024$ bits and produces $t = 8 \times 64 = 512$ bits. The compression function F is essentially a block cipher $E_K(\cdot)$ used in a Davies-Meyer construction, with the $8w$-bit chaining value h_j as plaintext and feed-forward, and the $16w$-bit message block m_j as key (Figure 7.1):

$$\begin{aligned} h_0 &= \mathrm{IV}, \\ h_{i+1} &= F(h_j, m_j) = E_{m_j}(h_j) \boxplus h_j \qquad \text{for } 0 \leq j < \ell, \\ T &= h_\ell\,. \end{aligned}$$

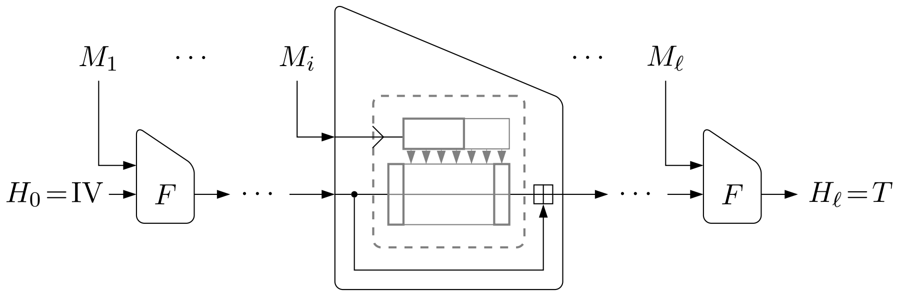


Figure 7.1.: SHA-2's mode of operation of the block cipher E (dashed).

Below, we first describe the padding and message expansion of F (i.e., the key schedule of E), followed by the state update transformation of F. A full specification, including the round constants, initial values, and other details, is given in the standard document [Dan12]. We use the notation of Section 2.1.1 for word-oriented operations on w-bit words.

Padding and message expansion. The message expansion of SHA-2 splits each message block m_j into 16 w-bit words M_i, $i = 0, \ldots, 15$, where $w = 32$ (SHA-256) or $w = 64$ (SHA-512), and expands these words into $R = 64$ (SHA-256) or $R = 80$ (SHA-512) expanded message words W_i:

$$W_i = \begin{cases} M_i & 0 \le i < 16, \\ \sigma_1(W_{i-2}) \boxplus W_{i-7} \boxplus \sigma_0(W_{i-15}) \boxplus W_{i-16} & 16 \le i < R. \end{cases} \tag{7.1}$$

The $\mathbb{F}_2$-linear functions $\sigma_0(x)$ and $\sigma_1(x)$ depend on the word size w:

$$\sigma_0(x) = \begin{cases} (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3) & \text{for } w = 32, \\ (x \ggg 1) \oplus (x \ggg 8) \oplus (x \gg 7) & \text{for } w = 64, \end{cases}$$

$$\sigma_1(x) = \begin{cases} (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10) & \text{for } w = 32, \\ (x \ggg 19) \oplus (x \ggg 61) \oplus (x \gg 6) & \text{for } w = 64. \end{cases}$$

State update transformation. The intermediate state of SHA-2 consists of 8 w-bit words $(A_i, B_i, \ldots, H_i)$. In each step, the state is updated by a generalized Feistel construction, where two of these words, A_i and E_i, are updated with new values, while the other words are shifted. We use the alternative description of the state update function introduced by Mendel, Nad, and Schläffer [MNS11b] and illustrated in Figure 7.2.

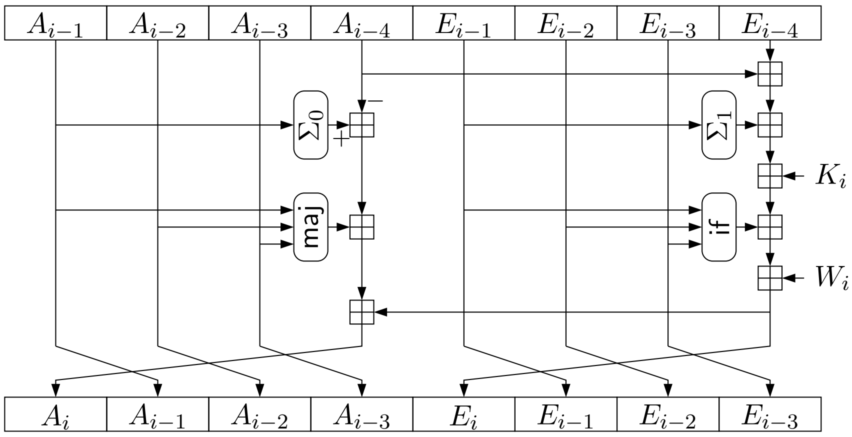


Figure 7.2.: The state update transformation of SHA-2 [MNS11b].

The state update transformation starts from the previous $8w$-bit chaining value $h_j = (A_{-1}, \ldots, A_{-4}, E_{-1}, \ldots, E_{-4})$ and updates it by applying the step functions $R \in \{64, 80\}$ times. In each step i, $0 \leq i < R$, the expanded message word W_i is used to update the two state words E_i and A_i:

$$E_i = A_{i-4} \boxplus E_{i-4} \boxplus \Sigma_1(E_{i-1}) \boxplus \mathsf{if}(E_{i-1}, E_{i-2}, E_{i-3}) \boxplus K_i \boxplus W_i, \quad (7.2)$$

$$A_i = E_i \boxminus A_{i-4} \boxplus \Sigma_0(A_{i-1}) \boxplus \mathsf{maj}(A_{i-1}, A_{i-2}, A_{i-3}), \quad (7.3)$$

where the step constants K_i are defined as nothing-up-my-sleeve numbers using the fractional part of $\sqrt[3]{p_i}$, where p_i is the i-th prime number [Dan12], and the $\mathbb{F}_2$-linear functions Σ_0 and Σ_1 are defined as follows:

$$\Sigma_0(x) = \begin{cases} (x \ggg \ \ 2) \oplus (x \ggg 13) \oplus (x \ggg 22) & \text{for } w = 32, \\ (x \ggg 28) \oplus (x \ggg 34) \oplus (x \ggg 39) & \text{for } w = 64, \end{cases}$$

$$\Sigma_1(x) = \begin{cases} (x \ggg \ \ 6) \oplus (x \ggg 11) \oplus (x \ggg 25) & \text{for } w = 32, \\ (x \ggg 14) \oplus (x \ggg 18) \oplus (x \ggg 41) & \text{for } w = 64. \end{cases}$$

After the last step of the state update transformation, the previous chaining value is added as feed-forward to the output of the state update to produce the next chaining value h_{j+1} (or the final hash value h_ℓ):

$$h_{j+1} = (A_{R-1} \boxplus A_{-1}, \ldots, A_{R-4} \boxplus A_{-4}, E_{R-1} \boxplus E_{-1}, \ldots, E_{R-4} \boxplus E_{-4}).$$

SHA-224, SHA-512/224, SHA-512/256, SHA-384, and SHA-512/t
The truncated variants differ from the main variants only in their initial values IV and by truncating the final hash output h_ℓ to the first t bits.

7.2.2. Published Analysis of SHA-2

The main focus of cryptanalytic attacks so far was on the main family members SHA-256 and, to a much lesser extent, on SHA-512. Only few results are available on the original truncated variants SHA-224, SHA-384, and to the best of our knowledge, no dedicated cryptanalysis of SHA-512/224, SHA-512/256 was published before the results discussed in this chapter.

Generic attacks. The main SHA-2 variants, SHA-256 and SHA-512, are susceptible to several generic attacks on the narrow-pipe Merkle-Damgård [Dam89; Mer89] structure. This includes Joux' multicollision attack [Jou04], Kelsey and Kohno's herding and Nostradamus attacks [KK06], and Kelsey and Schneier's second preimages for long messages [KS05]. The chop-MD [CDMP05] structure of the new truncated variants SHA-512/224 and SHA-512/256 with its wide-pipe structure, on the other hand, prohibits such attacks.

For the sake of completeness, we mention that generic quantum preimage attacks on SHA-256 were analyzed by Amy et al. [AMG+16] to cost $2^{166.4}$ logical-qubit-cycles, or about 2^{128} black-box queries.

Dedicated attacks. The security of SHA-2 against preimage attacks was first studied by Isobe and Shibutani [IS09], who presented attacks on 24-step SHA-256. This was drastically improved and extended to SHA-512 by Aoki et al. [AGM+09], who presented MitM preimage attacks on 43-step SHA-256 and 46-step SHA-512, which were later improved by Guo et al. [GLRW10] and then further extended using bicliques to 45-step SHA-256 and 50-step SHA-512 by Khovratovich et al. [KRS12]. Due to their wide-pipe structure, these MitM preimage attacks do not carry over to SHA-512/224 and SHA-512/256. Li et al. [LIS12] showed that particular preimage attacks can also be used to construct a free-start collision attack for up to 57 steps of SHA-512 and 40-step SHA-384. The attacks are only slightly faster than the respective generic attack complexities. In a MAC setting, Yu and Wang [YW09] provide distinguishers for 39-step SHA-256.

Practical attacks. Local collisions as a basis for collision attacks on SHA-2 were first studied for simplified variants of SHA-2, starting with 9-step local collisions [GH03; HPR04; SS07]. This was adapted to find a first 18-step collision for unmodified step-reduced SHA-256 by Mendel et al. [MPRR06a]. Nikolić and Biryukov [NB08] first worked with modular differences and proposed 9-step local collisions which can be extended to attacks on 24 steps of SHA-256 and SHA-512, found independently by Indesteege et al. [IMPR08] and by Sanadhya and Sarkar [SS08; SS09]. For SHA-256, these results were significantly improved by constructing longer local collisions spanning up to 18 steps with the help of an automatic search tool by Mendel, Nad, and Schläffer [MNS11b], resulting in 27-step collisions and 32-step semi-free-start collisions [MNS11b] and later 28-step collisions and 38-step semi-free-start collisions for SHA-256 [MNS13b].

Furthermore, practical second-order differential collisions were demonstrated for 46-step SHA-256 by Lamberger and Mendel [LM11], 47-step SHA-256 by Biryukov et al. [BLMN11], and for up to 48-step SHA-512 by Yu et al. [YB14; YHB16], who evaluated the security margin of the SHA-256 and SHA-512 compression functions against boomerang attacks.

Table 7.1.: Best published collision attacks on the SHA-2 family, including semi-free-start (SFS) and free-start (FS) collisions without padding (WP). [EMS14; DEM15a] are covered in this thesis.

SHA-b /	t	Type	Steps	Complexity	Reference
SHA-512	all	collision	24/80	—	[IMPR08; SS08]
		collision	27/80	—	[DEM15a]
		SFS collision	38/80	—	[EMS14]
		SFS collision	39/80	—	[DEM15a]
	512	FS collision	57/80	$2^{255.5}$	[LIS12]
	384	FS collision	40/80	$2^{183.0}$	[LIS12]
		FS collision WP	41/80	—	[DEM15a]
	256	FS collision WP	43/80	—	[DEM15a]
	224	FS collision WP	44/80	—	[DEM15a]
SHA-256	all	collision	28/64	—	[MNS13b]
		collision	31/64	$2^{65.5}$	[MNS13b]
		SFS collision	38/64	—	[MNS13b]
	256	FS collision	52/64	$2^{127.5}$	[LIS12]
	224	FS collision	40/64	$2^{110.0}$	[LIS12]
		FS collision WP	39/64	—	[DEM15a]

7.2.3. Collision Attack Strategy for SHA-2

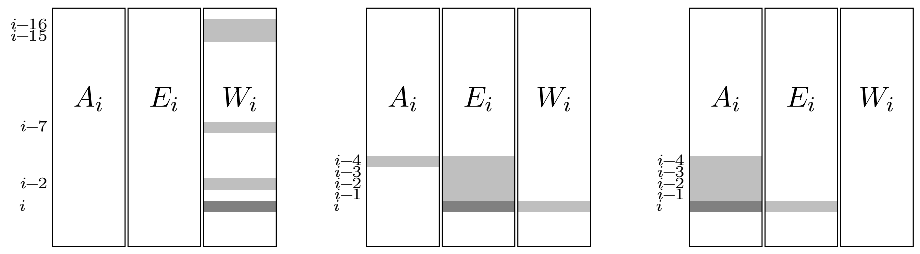
Since the ground-breaking results of Wang et al. [WYY05b; WY05; WLF+05], the search techniques used for practical collisions have been significantly improved, but the top-level attack strategy has remained essentially the same. We start from a suitable starting point which usually specifies a selection of active words in the expanded message to define a local collision. The starting point defines the search space for the (automatic or manual) search. The search itself is divided into two phases:

- **Find a differential characteristic**
  1. Construct the high-probability part of a characteristic
  2. Determine the low-probability part of a characteristic
- **Find a conforming message pair**
  3. Use message modification in low-probability part
  4. Perform random trials in high-probability part

Constructing the differential characteristic for the low-probability part is one of the most difficult tasks in a differential attack. The main reason is that such low-probability characteristics are usually very dense and have many (hidden) relations which need to be taken into account. Wang et al. found the dense low-probability characteristics for the attacks on MD4, MD5, RIPEMD, SHA-0 and SHA-1 mostly by hand [WY05; WYY05b; WLF+05]. However, for more complex hash functions, such an approach is infeasible. We use the heuristic guess-and-determine search tool previously developed for SHA-256 by Mendel, Nad, and Schläffer [MNS11b; MNS13b], which we briefly describe in the following. Before, we discuss the choice of suitable starting points in Section 7.5.1.

Finding starting points for SHA-2

To model SHA-2 as a satisfiability problem for the search tool, we need to introduce suitable intermediate variables. Based on the alternative description given in Section 7.2.1, we only use the words A_i and E_i of the state, plus the words W_i of the message expansion. Figure 7.3 illustrates the update rules for A, E and W given in equations (7.3), (7.2), (7.1) by highlighting the input words for updating each word: Each row represents one of the $R \in \{64, 80\}$ step iterations, with its three state words A_i, E_i, and W_i.



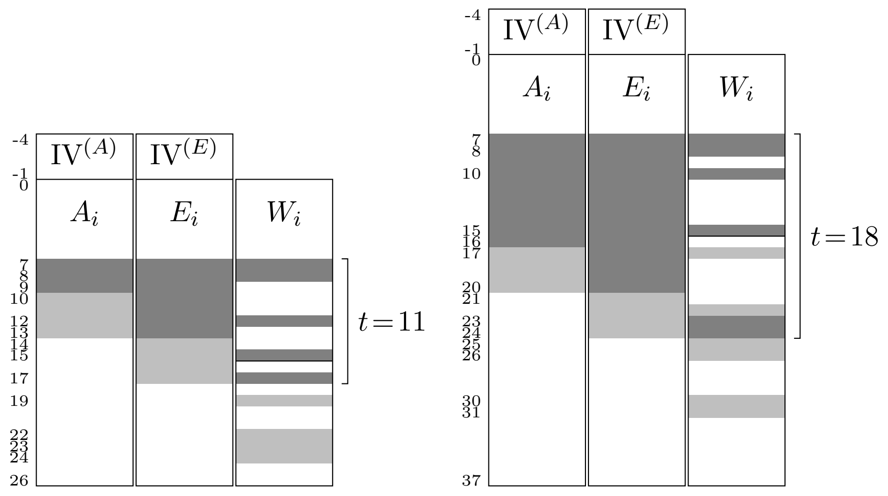
(a) Updating W_i (7.1) **(b)** Updating E_i (7.2) **(c)** Updating A_i (7.3)

Figure 7.3.: Update rules to compute A_i, E_i, and W_i (■) recursively from other state words (▨).

Local collisions. All our results are based on "local collisions" in the message expansion: by carefully selecting (expanded) message words in the middle steps so that the differences can cancel out in as many consecutive steps as possible in the forward and backward expansion, i.e., the first and last few expanded message words contain no differences. The t middle steps with differences can induce differences in the A_i and E_i words. However, the W_i words can be used to achieve zero difference in the last 4 of the t words E_i, and in the last 8 of the t words A_i. This is necessary to obtain words with zero difference in the very last 4 steps of the state update and thus in the output chaining value.

As an example, the starting point used to find a 27-step collisions for SHA-256 [MNS11b] allows differences in the five expanded message words $W_7, W_8, W_{12}, W_{15}, W_{17}$ and state words $E_7, \dots, E_{13}$ and $A_7, \dots, A_{10}$. The exact bitwise signed differences are chosen during the search such that any potential differences in $W_{19}, W_{22}, W_{23}, W_{24}$, as well as $E_{14}, \dots, E_{17}$ and $A_{10}, \dots, A_{13}$ cancel out. The resulting starting point is illustrated in Figure 7.4a. We show in Section 7.5.2 how the same starting point can be used for SHA-512. In addition, we use the closely related starting point for 28-step SHA-256 collisions [MNS13b], illustrated in Figure 7.6b (p. 194), to also find collisions for SHA-224.

The semi-free-start collision starting point covering the most steps before the results covered in this chapter is for 38 steps of SHA-256 [MNS13b] with a local collision spanning $t = 18$ steps, as illustrated in Figure 7.4b. Considering the large number of steps, the number of expanded message words with differences and cancellations is remarkably low: only 6 words with differences, and 6 words imposing cancellation conditions.



(a) 27-step collision [MNS11b]. **(b)** 38-step SFS collision [MNS13b].

Figure 7.4.: SHA-256 starting points: Differences ■ and cancellations ■.

Semi-free-start collisions and collisions. The discussed starting points are targeted to find semi-free-start collisions, that is, different messages m, m' and an IV h_0 such that $f(h_0, m) = f(h_0, m')$. However, they can also be used for hash function collisions with the original IV h_0 by trading the freedom of the IV for freedom in the message words.

In order to find hash function collisions, the first few message words W_i must retain sufficient freedom (i.e., they should not be constrained by conditions from the message expansion for cancelling differences) to allow to match the correct IV value. Ideally, this means that the first 8 message words $W_0, \ldots, W_7$ are free of any conditions (no differences, but also not constrained by conditions from other message words connected via the message expansion). If the W_i differences are sparse enough overall, it can also be sufficient to have at least 5 words $W_0, \ldots, W_4$ free of conditions by providing the remaining freedom with a two-block approach [MNS13b].

After a starting point has been fixed, we need to find a suitable differential characteristic for the active words such that the cancellations actually work out. The core of the local collision in the first active words A_i and E_i is usually very dense and only has low differential probability. In Wang et al.'s original attacks [WY05; WYY05b; WLF+05], these conditions were analyzed manually, but this is infeasible for SHA-2.

7.2.4. Dedicated Guess-and-Determine Search Tools

As a result, (semi-)automatic approaches to find such dense characteristics were published soon afterwards, first for Wang et al.'s original targets like MD4 [SO06] and SHA-1 [DR06] and later refinements to target more complex hash functions such as SHA-256 and related ciphers [MNS11b; MNSS12; MNS13b; MNS13a; LP13] or ARX-based SHA-3 candidates [Leu12; Leu13]. All these approaches essentially follow the guess-and-determine constraint-propagation strategy.

Differential model. As discussed in Section 2.2.4, the natural difference notion for generalized ARX designs with Boolean functions is the signed difference introduced by Wang et al. [WY05; WLF+05]. A bitwise signed difference $\Delta^{\pm} \in \{0, +1, -1\}^w$ for w-bit words uniquely determines both the modular difference $\Delta^{\boxminus} = \sum_i 2^{\Delta_i^{\pm}} \in \mathbb{Z}_{2^w}$ as used for MD4 by Dobbertin [Dob96; Dob98] and the bitwise xor difference $\Delta^{\oplus} = (|\Delta_i^{\pm}|)_i$.

The aim of the search is to find first a consistent signed characteristic and then a confirming message pair. In order to represent all stages of the evolution from a starting point (where only some zero-differences are fixed) via the characteristic (of signed differences) to the message pair (of fixed bit values), De Cannière and Rechberger [DR06] introduced generalized bit conditions in their analysis of SHA-1. Let (x_j, x_j^*) be a pair of bits. The *generalized condition* $\nabla(x_j, x_j^*)$ constrains the possible values of (x_j, x_j^*) to a subset of all pairs $\{(0,0),(0,1),(1,0),(1,1)\}$, i.e., it specifies an element of the power set $\mathcal{G} = \mathcal{P}(\{0,1\}^2)$. In total, we get 16 possible generalized conditions. We use the same notation [DR06], which denotes each condition by a descriptive character (see Table 7.2 on p. 182). For instance, xor differences can be denoted by `-` (for $x_j = x_j^*$) and `x` (for $x_j \neq x_j^*$). For a pair of w-bit words $x, x^* \in \{0,1\}^w$ with generalized conditions $\nabla(x, x^*) = \nabla(x_{w-1}, x_{w-1}^*) \cdots \nabla(x_0, x_0^*) = c_{w-1} \cdots c_0$, we denote the number of solutions to these generalized conditions by $|\nabla(x, x^*)| = \prod_{i=0}^{w-1} |c_i|$.

For any set $S \subseteq (\{0,1\}^w)^2$ of pairs of w-bit words, we can consider the minimal w-bit generalized condition $\nabla(x, x^*) \in \mathcal{G}^w$ that covers S, i.e., $S \subseteq \nabla(x, x^*)$. We say that any strict subset of $\nabla(x, x^*)$ in $\mathcal{G}^w$ constrains (or refines) $\nabla(x, x^*)$ (and also S). We can describe the differential behavior of operations (or other relations) with some input and output words $x, y, \ldots$ with generalized conditions $\nabla(x, x^*), \nabla(y, y^*), \ldots$ by considering all pairs covered by the generalized condition, testing which ones are consistent with the definition of the operation (or relation), and constraining $\nabla(x, x^*), \nabla(y, y^*), \dots$ accordingly to the minimal condition in $\mathcal{G}$ that covers the solutions. We refer to this as (perfect) propagation for this operation. In practice, if w and $\nabla(x, x^*), \nabla(y, y^*), \dots$ are large, this cannot be implemented efficiently, and the differential behavior must be modeled on a smaller granularity, for example on bit-slices with one output bit and a few input bits ("bit-sliced propagation") [DR06].

Guess-and-determine search. The search process aims to refine a starting point given in generalized input conditions (usually `-`,`?`) first to a signed characteristic (output conditions `u`, `n`, `-`) and then a confirming message pair (`u`, `n`, `0`, `1`). We refer to the desired output conditions as "determined", and the other input conditions as "undetermined". The guess-and-determine search approach iteratively picks and guesses (constrains) bits, and propagates the constraints to neighboring bits via the connected operations. If an inconsistency occurs, the algorithm backtracks to an earlier state of the search and tries to correct it. [MNS11b] denote these three parts of the search by decision, deduction, and backtracking (Algorithm 4). This procedure can be visualized by a search tree, which is traversed by depth-first search. The guessing strategy defines the tree's shape and which branches are visited first. The backtracking algorithm can skip parts of the tree to improve exploration. We discuss some more detailed considerations on the individual steps in the following.

Algorithm 4 Guess-and-determine search algorithm

```
while there are undetermined bits do
  Decision (Guessing)
   1.  Pick an undetermined bit
   2.  Constrain this bit
  Deduction (Propagating)
   3.  Propagate the new information to other variables and equations
   4.  if no inconsistency is detected, goto step 1
  Correction (Backtracking)
   5.  if possible, apply a different constraint to this bit, goto step 3
   6.  else undo guesses until this critical bit can be resolved
```

The *guessing* step needs to select and constrain the target bits. The strategy applied in this step crucially influences the shape of the resulting characteristic, as well as the search effort necessary to complete the result. On a high level, this strategy must be defined by the cryptanalyst in order to determine which parts of the characteristic should be sparse to fulfill probabilistically, which parts are dense and should be determined first, and in which order related bits should be guessed (e.g., ordered guesses from LSB to MSB for modular additions). This is done by specifying several "phases" that define which general parts of the characteristic and which generalized conditions (input and desired output conditions) are targeted to determine next [MNS11b]. A phase may cover several "settings" to choose from randomly. On a lower level, within the set of candidates defined this way, the target bit may be predetermined (for ordered guesses), selected uniformly at random, or selected based on a heuristic such as the number of two-bit conditions [MNS11b].

The *propagation* step needs to identify contradictions and make implicit constraints explicit by propagation. For bitwise Boolean functions (if, maj), bit-sliced propagation is perfect and can be efficiently applied by pre-computing the propagation result for any generalized input condition. For modular sum with two or more summands, the same approach was successfully applied [DR06; MNS11a; MNS11b] and extended with multi-bit checks by Leurent [Leu12; Leu13]. For SHA-2, additional, more expensive checks can be performed at the end of phase, including linear 2-bit conditions - and "complete" checks [MNS11b]. These 2-bit conditions are particularly relevant for the linear functions σ_i, Σ_i in SHA-2, where input bits are shared between several bit-slices each. Generally speaking, bit-sliced propagation is not well-suited for typical linear diffusion layers, in particular with large state sizes as, e.g., in SHA-3/Keccak.

The *backtracking* step needs to recover after detecting an inconsistency in the current characteristic. If all alternatives for constraining a target bit fail, one of the previous guesses must already have been inconsistent, so the last few guesses are iteratively undone and alternative constraints for the previous bits are tested. It is useful to mark the original target bit as critical and undo guesses until it can be constrained without detecting any more inconsistencies. For undoing guesses, snapshots of previous characteristics and lists of intermediate guesses since then are kept. After a certain threshold of contradictions, the entire search tree is discarded and the search restarted from scratch [MNS11b].

The tool is publicly available: `https://github.com/iaikkrypto/nltool`.

7.3. Improving Deductions with Linear Propagation

Bitsliced propagation and the extensions discussed in Section 7.2.4 work well for modular sum and the bitwise Boolean functions in SHA-2: The approach is both computationally very efficient and effective in detecting contradictions and propagated information. However, for classical linear diffusion layers, this method is not ideal.

As an example, consider the Σ_i functions in the state update function of SHA-2 (Section 7.2.1). Each output bit y_i, $0 \leq i < w$ is computed as the xor of three input bits $x_{i+r_1}, x_{i+r_2}, x_{i+r_3}$ (indices mod w), i.e., Σ_i is an example of an invertible linear shift-invariant transformation based on cellular automata [DGV94b; Dae95, Chapter 6]. It is thus easy to model with bit-sliced propagation steps; however, this approach does not model the implicit conditions arising from shared input variables between bit-slices. As a simple example, if we start with conditions $\nabla(x, x^*) = \texttt{?}^w$ and $\nabla(y, y^*) = \texttt{-}^w$, bit-sliced propagation will be unable to refine this to $\nabla(x, x^*) = \texttt{-}^w$. Adding a bit-sliced model of the inverse Σ_i^{-1} could help in this specific example, but as in many cases, the inverse has a higher implementation footprint and requires significantly larger bit-slices to compute x_i from y.

For this reason, we suggest to combine bit-sliced propagation with linear propagation based on Gaussian elimination. With an incremental solving approach and a suitable matrix layout for extraction of partial information, the linear propagation approach integrates efficiently with the existing search procedure. We evaluate the improved propagation quality for the linear layers of SHA-2 and SHA-3/Keccak. The approach was used by Kölbl et al. [KMNS13] to find 4-round differential characteristics for Keccak.

7.3.1. Linear Propagation Approach

Linear constraints

Let $f : \mathbb{F}_2^m \to \mathbb{F}_2^n$, $x \mapsto y$ be an affine function, with $y = F \cdot x \oplus a$ for some matrix $F \in \mathbb{F}_2^{n \times m}$ and $a = f(0) \in \mathbb{F}_2^n$. We rewrite this using the $n \times n$ identity matrix I in block matrix notation as

$$y = f(x) \;\Leftrightarrow\; \begin{bmatrix} I & F \end{bmatrix} \cdot \begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} a \end{bmatrix} .$$

When considering input pairs x and x^*, the same affine function is applied to both inputs. We denote the combined, interleaved system with $\bar{F}, \bar{z}, \bar{a}$ defined using the Kronecker product $\otimes$ by

$$\bar{F} \cdot \bar{z} = \bar{a} : \qquad (7.4)$$

$$\begin{aligned}
\bar{F} &= \begin{bmatrix} I & F \end{bmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} && \in \mathbb{F}_2^{2n \times (2n+2m)} \\
\bar{z} &= \left(\begin{bmatrix} y \\ x \end{bmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right) \oplus \left(\begin{bmatrix} y^* \\ x^* \end{bmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right) && \in \mathbb{F}_2^{2n+2m} \\
\bar{a} &= \begin{bmatrix} a \end{bmatrix} \otimes \begin{pmatrix} 1 \\ 1 \end{pmatrix} && \in \mathbb{F}_2^{2n} .
\end{aligned}$$

Consider the generalized condition $\nabla(z_j, z_j^*)$ for some bit z_j of $z = [y \ x]^T$. We call $\nabla(z_j, z_j^*)$ a linear generalized condition if the set of bit pairs constitutes an affine space over $\mathbb{F}_2$, i.e., if it is any generalized condition except 7, B, D, E with 3 possible pairs each. Any linear generalized condition can be defined by $k_j \in \{0, 1, 2\}$ affine equations with $C_j \in \mathbb{F}_2^{k_j \times 2}$, $c_j \in \mathbb{F}_2^{k_j}$:

$$C_j \cdot \begin{pmatrix} z_j \\ z_j^* \end{pmatrix} = c_j .$$

Table 7.2 lists the matrices $(C_j | c_j)$ for all conditions. We denote the combined generalized condition system with $k = \sum_j k_j$ rows by

$$\bar{C} \cdot \bar{z} = \bar{c} : \qquad \begin{aligned} \bar{C} &= \operatorname{diag}(C_j) && \in \mathbb{F}_2^{k \times (2n+2m)} \\ \bar{c} &= \begin{bmatrix} c_0 & \cdots & c_{n+m-1} \end{bmatrix}^T && \in \mathbb{F}_2^{k} . \end{aligned} \qquad (7.5)$$

Table 7.2.: Linear representation of generalized conditions $\nabla(z_j, z_j^*) \in \mathcal{G}$. $\nabla(z_j, z_j^*)$ given wrt. $\{(1,1), (0,1), (1,0), (0,0)\}$. ●,■ $= 1$ and ○,□ $= 0$.

$\nabla(z_j, z_j^*)$	$(C_j \| c_j)$	$\nabla(z_j, z_j^*)$	$(C_j \| c_j)$
`0` = ○○○●	■□□ □■□	`3` = ○○●●	□■□
`u` = ○○●○	■□■ □■□	`5` = ○●○●	■□□
`n` = ○●○○	■□□ □■■	`A` = ●○●○	■□■
`1` = ●○○○	■□■ □■■	`C` = ●●○○	□■■
`-` = ●○○●	■■□	`7` = ○●●●	—
`x` = ○●●○	■■■	`B` = ●○●●	—
`#` = ○○○○	□□■	`D` = ●●○●	—
`?` = ●●●●	—	`E` = ●●●○	—

Linear propagation of information

To propagate linear information, we perform the following three steps:

1. Construct the combined $(2n + k) \times (2m + 2n)$ system of (7.4),(7.5):
$$\begin{bmatrix} \bar{L} \\ \bar{C} \end{bmatrix} \cdot \bar{z} = \begin{bmatrix} \bar{a} \\ \bar{c} \end{bmatrix} .$$
2. Apply Gauss-Jordan elimination to create sparse equations.
3. Convert equations only on z_j and z_j^* back to generalized conditions: If the system is inconsistent, we know that the generalized conditions at the input and output of the affine function contradict each other. If the system is consistent, we can extract new information in the form of generalized conditions. Since a generalized condition consists of equations involving only z_j and z_j^*, we get this information by a linear combination of at most two adjacent rows with pivot elements z_j, z_j^*, so we can simply linearly scan the final matrix row by row.

Incremental elimination. During a guess-and-determine attack, the equation system for a particular linear layer is processed many times with only slightly different generalized conditions. In particular, most changes only consist of refining generalized conditions, i.e., adding simple individual equations with at most two set bits to the system. For this reason, we do not implement a full Gaussian elimination procedure, but maintain the entire system in reduced row echelon form and only implement an incremental "ε-Gauß" step that adds a row with at most two set bits to the system. Note that the initial system $\bar{L} \cdot \bar{z} = \bar{a}$ is already in reduced row echelon form, and during runtime, we only add individual rows from $\bar{C} \cdot \bar{z} = \bar{c}$ to the system. Furthermore, we know that within one branch of the search tree, conditions will only be refined and thus constraints added, never changed. Thus, those new rows are not actually added, but only used to eliminate one or two columns. By storing meta-information about which rows were recently updated, it is possible to minimize the complexity of adding and extracting new, updated information.

If the input contains nonlinear conditions (`7`, `B`, `D`, `E`), these are ignored (i.e., treated as `?`) in the equation system. After extracting updated conditions from the equation system, recombining the results with the original input may induce another update. For example, if the input `7` was treated as `?`, but propagated to `-`, this information is combined to propagate to `0`.

7.3.2. Discussion

To compare and evaluate the different propagation methods, we need to measure how well they propagate. Propagation corresponds to narrowing down the solution space, or gaining information about the solution. Let $\nabla(z, z^*)'$ denote the generalized conditions obtained by propagating $\nabla(z, z^*)$ by means of propagation method M. Then we take as figure of merit for M:

$$I_M(z) = \log_2 \frac{|\nabla(z, z^*)|}{|\nabla(z, z^*)'|} \ .$$

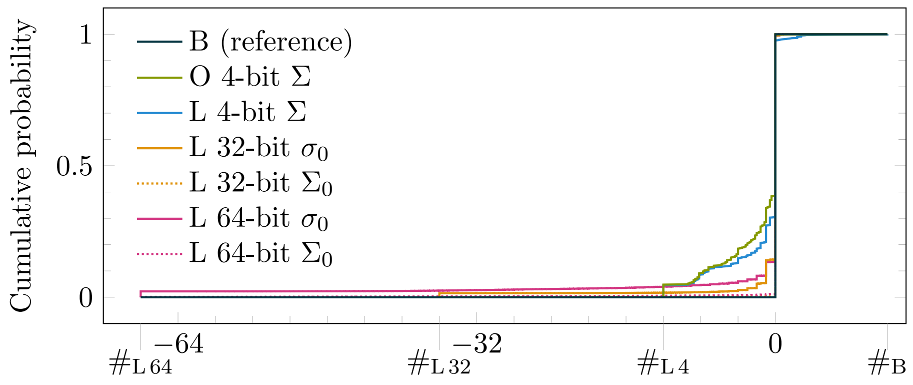
If $\nabla(z, z^*)$ is a contradiction, then $|\nabla(z, z^*)| = |\nabla(z, z^*)'| = 0$ and we set $I_M(z) = 0$. If $|\nabla(z, z^*)'| = 0$ but $|\nabla(z, z^*)| \neq 0$, then $I_M(z)$ is undefined, which we denote by $I_M(z) = \#$. To compare the two propagation methods and measure the gain of the linear method (L) over the bit-sliced method (B) for one specific condition $\nabla(z, z^*)$, we use

$$I_{\text{diff}}(z) = I_{\text{B}}(z) - I_{\text{L}}(z) \ .$$

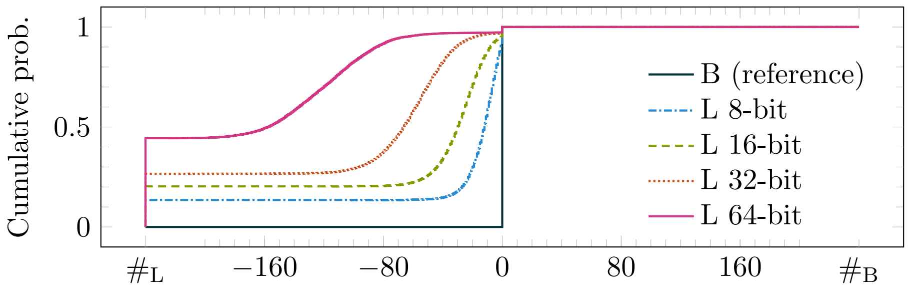
If $I_{\text{L}}(z) = \#$ but $I_{\text{B}}(z) \neq \#$, the linear method detects the contradiction but the bit-sliced method does not, and we set $I_{\text{diff}}(z) = \#_L$. If $I_{\text{L}}(z) \neq \#$ but $I_{\text{B}}(z) = \#$ we set $I_{\text{diff}}(z) = \#_B$. If both are $\#$, we set $I_{\text{diff}}(z) = 0$.

Results. We evaluated the propagation methods for different functions f by computing $I_{\text{diff}}(z)$ for many sample inputs $\nabla(z, z^*)$. We compare exhaustive search on bit-slices (B) with the linear propagation method (L) and, where possible, with optimal propagation (O). For the linear propagation, inputs with nonlinear condition are ignored but added again to the propagation result for computing I_{L}. We target the Σ_i, σ_i functions of SHA-2 for $w \in \{32, 64\}$, plus a toy variant with $w = 4$, and the linear layer of Keccak-f with lane sizes $w \in \{8, 16, 32, 64\}$.

Figure 7.5a illustrates the results for SHA-2's Σ_i, σ_i. For the 4-bit toy case, it is possible to exhaustively test all 15^8 generalized conditions without # and also compare with optimal propagation. For $w = 4$, in about a third of all samples, linear propagation performed better than bit-sliced propagation ($x < 0$ and $I_{\text{L}} > I_{\text{B}}$). In a very small number of cases, the bit-sliced approach is better due to nonlinear conditions ($x > 0$ and $I_{\text{L}} < I_{\text{B}}$). For $w \in \{32, 64\}$, for representative results, we used input conditions sampled from a practical search with the tool. The advantage of linear propagation appears quite limited, in particular for the Σ_i functions in the state update.



(a) Distribution of $I_B - I_L$ for Σ_i, σ_i of SHA-2.



(b) Distribution of $I_B - I_L$ for linear layer of Keccak-f.

Figure 7.5.: Comparison of propagation methods: $I_B - I_L$ for different linear functions. Higher values correspond to better propagation. Results for 4-bit Σ are exhaustive, the others are sampled during a search.

Figure 7.5b shows the results for the Keccak round function [BDPV11d] with different lane sizes. The linear layer updates 25 lanes, each of length $w = \{8, 16, 32, 64\}$. We observed that choosing uniformly random generalized conditions at the input and output results in impossible characteristics with a very high probability. Since this is probably not representative for an actual attack, we also extracted the samples from a 16-hour practical search with the tool. For $w = 64$, the linear approach performs better in more than 97 % of the samples, and almost 50 % of the samples were contradictions detected by linear, but not by bit-sliced propagation.

Performance Evaluation. Beside the propagation quality, the overall performance of each propagation strategy implemented in a practical

Table 7.3.: Performance evaluation of linear and bit-sliced propagation for round-reduced Keccak in terms of propagation complexity (iterations per second) and overall success (collisions found after 24 hours).

Rounds	Lane size	Hash size	Capacity	Propagation	Iter/s	Found
2	32 bits	128 bits	256 bits	bit-sliced	2828	538
				linear	770	1517
	64 bits	256 bits	512 bits	bit-sliced	1679	6
				linear	352	316
3	32 bits	128 bits	256 bits	bit-sliced	2094	0
				linear	365	113
	64 bits	256 bits	512 bits	bit-sliced	1165	0
				linear	182	42

search algorithm is significant. An improvement of the propagation quality needs to outweigh an expected increase in the runtime complexity of the propagation step. Bitsliced propagation for small bit-slices can be implemented very efficiently by a single table lookup per bit-slice, whereas linear propagation requires iterations of the incremental Gauss elimination and extraction of the generalized conditions from the equation system. To judge the practical impact of the propagation, it is necessary to evaluate the whole search process.

Table 7.3 shows the results of a naive collision search for Keccak with lane sizes $w = 32, 64$ for $r = 2, 3$ rounds, and a search runtime of 24 hours on a single CPU. The starting point for each search run was an undetermined state with only IV, zero difference in the output bits, and a difference in one bit of the input fixed. The results show that linear propagation takes 3 to 7 times longer per propagation iteration, but still performs far superior in terms of overall search performance. While the performance is still comparable for simpler problems like 2 rounds with $w = 32$, the bit-sliced approach becomes inefficient for larger problems. In case of the 3-round experiments, bit-sliced propagation had not progressed much further after 24 hours than linear propagation had reached after seconds. Note that the search strategy used for these results was optimized for bit-sliced propagation, which is a lot more sensitive to different parameters of the search algorithm. The results for the bit-sliced version quickly deteriorate in less optimal configurations, while linear propagation is more robust.

7.4. Improving Decisions with Branching Heuristics

Branching rules are one of the essential ingredients for guess-and-determine attacks. They define how the search algorithm selects the next variable to guess, and which guess values to try first for this variable. The branching rule aims to keep the search runtime as short as possible. Depending on whether the current partial assignment is correct (satisfiable) or contradictory, this means either that a satisfying solution is found as soon as possible, or that the contradiction is detected quickly. In the latter case, this corresponds to identifying a conflicting subset of unassigned variables and branching on these first in order to prune the search tree. The search trees traversed by different branching rules can vary drastically in size, from constant (for unsatisfiable problems) or linear (for satisfiable problems) to exponential in the number of variables [Ouy98].

To handle the larger search space of SHA-512, we propose a new branching heuristic for the guess-and-determine strategy used in these attacks. Our approach is inspired from related ideas in SAT solvers [LA97; HM09]. The heuristic performs a randomized look-ahead selection of candidates which should be guessed first. Our aim in using this approach is to detect contradictions earlier and reduce the search space faster. The actual effect varies significantly with the problem instance. As a relevant example, we speed up the search for 27-step SHA-512 by a factor of about 2^{20}, and are thus for the first time able to find practical semi-free-start collisions for 38-step (and later 39-step) SHA-512, with an exemplary runtime corresponding to about $2^{40.5}$ compression function evaluations.

In the following, we discuss branching heuristics used in SAT solvers and then propose a look-ahead branching heuristic for differential cryptanalysis.

7.4.1. Branching Heuristics in SAT Solvers

Most general-purpose SAT solvers are based on extensions of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [DLL62], a guess-and-determine approach for satisfiability problems given in conjunctive normal form (CNF). The problem of choosing optimal branching variables and corresponding assignments for DPLL algorithms has been proven to be both NP-hard and coNP-hard [Lib00]. However, there is a variety of commonly implemented branching rules based on different heuristics to evaluate the urgency or relevance of potential branching variables. In addition, meta-rules to select different branching rules depending on the situation and search history have been proposed [HB03].

SAT branching rules can be categorized according to their target heuristic (current properties, look-ahead or history analysis), their output (a single branching variable/literal or a preselection of candidate variables) and their randomness (deterministic or randomized). Popular heuristics include:

- **Uniformly random**. This computationally cheapest approach picks an unassigned variable uniformly at random. Many modern SAT solvers apply this rule with a small probability and otherwise use a more informed choice. In differential analysis, this is the default.

- **Small clauses**. The earliest heuristics greedily favor variables that appear in many small clauses. The rationale for this choice is twofold. First, smaller clauses need to be fulfilled “more urgently” since there are fewer options left that avoid contradictions. Second, even if the guessed literal evaluates to false in binary clauses, unit propagation ensues and curtails the search tree. Examples include Böhm’s rule [BK93], MOM [Fre95], and the Jeroslaw-Wang rules [JW90] where variables score according to the sum of weights inversely exponential in the clause length they appear in. Small clauses have also been used as a preselection heuristic for more expensive look-ahead rules. In differential guess-and-determine attacks, two-bit conditions [MNS11b] play a related role.

- **Literal count**. Heuristics like DLCS and DLIS [SS96; Sil99] simply count clauses irrespective of their lengths. This is useful in CNF problems, where satisfying one literal resolves the complete clause, but counterproductive for the xor-chains in hash functions. There, this heuristic would create a large number of hidden dependencies and reduce the freedom without useful propagation.

- **Conflict-driven**. A more popular variation of literal counting is VSIDS, first implemented in Chaff [MMZ+01] and later included in MiniSAT [ES03] and others. Here, the initial literal score of each variable decays over time, but scores are refreshed (bumped) by occurrences in newly learned clauses from the CDCL process. Effectively, critical variables with many recent contradictions are guessed first. The BerkMin solver extends this concept to bump not only variables from learned clauses, but from any clauses involved in the resolution process [GN02]. In differential attacks, the backtracking strategy [MNS11b] provides a similar behavior.

- **Look-ahead**. Instead of judging current properties of the formula or the previous search history, look-ahead heuristics analyse the actual effects of branching in a candidate variable [LA97; HM09]. For example, the Satz solver performs Unit Propagation Look-Ahead: both possible assignments for each free variable are tested for consequences of this decision and the caused unit propagations. If one of two assignments causes a contradiction, the other is fixed; if both are contradictory, backtracking is started; and if both seem valid, the variable v is assigned score $\mathcal{M}(v) = w(\neg v) \cdot w(v) \cdot 1024 + w(\neg v) + w(v)$, where $w(\ell)$ is typically the number of new binary clauses caused by the propagation of literal $\ell \in \{v, \neg v\}$.

- **Locality**. To limit the candidates for expensive look-ahead calculations, the candidate variables can be limited to those occurring in recently changed clauses, as implemented in marchdl [HM06].

Not all of these rules are suitable for general Boolean satisfiability problems that are not given in CNF format, as already indicated in the list above. In particular, if the propagation and learning process differs from the standard SAT case, the above rules can be counterproductive. On the positive side, dedicated solvers for specific applications can apply domain-specific knowledge to guide the search process.

7.4.2. The Look-Ahead Branching Heuristic

The branching strategy is one of the most promising areas for optimization in differential cryptanalysis tools based on tree search. Ideally, the branching strategy quickly navigates towards a valid assignment of variables and avoids subtrees without solutions. For detecting invalid subtrees, the branching strategy relies on the propagation method to detect contradictions as soon as possible. However, the propagation procedure can not only be used to decide whether previous guesses were contradictory. In addition, we also want to apply it to guide the branching strategy. The goal of this interaction is to minimize the size of the search tree in order to find solutions faster. The intuition of our approach can be summarized as follows.

- **Productive propagation** is good. Guessing a variable where propagation of the value determines many other variables can have multiple advantages compared to variables with less propagation. The most immediate effect is that the remaining search space is reduced. If more variables are determined right now, they will not create unnecessary subtrees for guessing later. The overall tree size and thus the complexity of the remaining search is reduced. On the downside, limiting the search space at the same time reduces the remaining degrees of freedom. If one value assigned to a specific bit propagates better than the second possible value, then, intuitively speaking, the probability for a solution in the remaining search space for the first option is lower than for the second value.

- **Contradictions** are even better. Of course, the overall search aims to find non-contradictory assignments. Nevertheless, discovering contradictory value assignments in the current subtree is consistently helpful for the remaining search. If only one of two possible value assignments is contradictory, the variable certainly needs to be fixed to the other value. If both values are contradictory, we must already have made an error with a previous guess and need to backtrack immediately. In both cases, it is better to address the conflicting bit sooner rather than later.

Implementation

In order to implement the criteria above in a practical branching heuristic, we use a look-ahead approach related to the Unit-Propagation Look-Ahead (UPLA) used in some SAT solvers. When the branching rule needs to select the next variable to guess, each candidate is in turn evaluated. For each candidate, a value is tentatively assigned and the propagation method is applied to determine the consequences of this assignment. If a contradiction occurs, this candidate is selected immediately. Otherwise, the number of propagated variables is calculated. If it is better than the previously favorite candidate, this variable becomes the new favorite.

There are two performance-related problems with this basic approach. First, performing the look-ahead propagation for all free variables is very costly. Second, the basic UPLA approach includes no randomization. However, we need randomization since a complete search of the tree is typically computationally infeasible in differential cryptanalysis. Instead, large tree parts are skipped and the search is restarted regularly. To avoid becoming lost in the same search branches over and over again, it is essential that the branching strategy is sufficiently randomized. We address both problems at once by selecting only a random subset of variables for closer evaluation. Our branching heuristic is summarized in Algorithm 5.

Algorithm 5 Look-ahead branching heuristic

```
Input: Set $U$ of undetermined bits, Look-ahead limit $s_{\max}$
Output: Branching variable $v$ to use in Steps 1 and 2 of Algorithm 4
  $L$ is an empty list
  while $U \neq \emptyset$ and $|L| < s_{\max}$ do
    Decision (Guessing)
     1.  Pick a bit $v \in U$ randomly
     2.  Constrain this bit $v$
    Deduction (Propagation)
     3.  Propagate the new information to other variables and equations
     4.  if an inconsistency is detected, return $v$ as the decision bit
     5.  $D_v := \{$bits determined in Step 3$\}$. Store $(v, |D_v|)$ in $L$.
    Update
     5.  Update $U := U \setminus D_v$
     6.  Undo all changes to restore the original assignment
  return $v^*$ from $L$ with the highest score $|D_{v^*}|$
```

The size of the randomly selected subset is an essential parameter for the success of the heuristic. To limit the look-ahead costs, we limit the maximum subset size by a constant number $s_{\max}$ that is chosen in the beginning of the search procedure, depending on the specific problem instance. In order to also provide sufficient randomization, we additionally bound the size $s_{\max}$ relative to the current number of unguessed variables.

Beside the subset size, the decision which individual variables to select for look-ahead plays a role. UPLA-based solvers use a pre-selection of interesting candidates, for example by locality criteria. We use similar pre-selection criteria as in the original Step 1 of Algorithm 4, based on the defined strategy, favoring bits with more two-bit conditions, and bits involved in recent conflicts [MNS11b]. The selection must remain sufficiently randomized.

Additionally, we do not explicitly evaluate variables that were already determined by the propagation procedure of one of the previous candidates. We mark these as evaluated without calculating a separate look-ahead, since their score is at most as good as the bit that triggered their propagation (at least with respect to one of the assignment options).

We have evaluated different variants of the heuristic and get the best results for a limit of $s_{\max} = 16$. Larger values of $s_{\max}$ further reduce the tree depth, but due to the additional cost for evaluating more candidates, this does not improve the overall runtime. Additionally, with larger subset sizes, the search tends to visit very similar subtrees again and again after each restart. This is particularly critical if the search space is limited to a few words, as in the focused search strategy described below. For other hash functions with larger states sizes or less focused strategies, the optimal value for $s_{\max}$ may be very different.

Results

Using the improvements in the branching heuristic proposed in the previous section together with the starting point and an adapted search strategy from the semi-free-start collision on 38-step SHA-256 [MNS13b], it is possible to find semi-free-start collisions for SHA-512 on up to 38 steps. Finding a differential characteristic together with a conforming message pair took 5441 seconds ($\approx$ 1.5h) on a cluster with 40 CPUs. This corresponds to a complexity of about $2^{40.5}$ evaluations of the SHA-512 compression function.

To show the benefit of our new look-ahead branching heuristic, we have performed some comparisons. Without look-ahead branching, we were able to find a semi-free-start collision for 27 steps of SHA-512 using 4 days on a cluster with 40 nodes, which corresponds to a complexity of about $2^{46.5}$. Using look-ahead branching with $s_{\max} = 16$ we can find differential characteristics with conforming message pairs within seconds on a standard PC (complexity $2^{26.5}$).

The heuristic can also be used to improve the search complexity for primitives with a smaller state to a certain extent. For example, experiments show a speedup of more than an order of magnitude for attacks on 27 or 38 steps of SHA-256. However, due to the heuristic nature of the improvement and the general sensitivity of the search procedure to different parameters, the effects are hard to quantify.

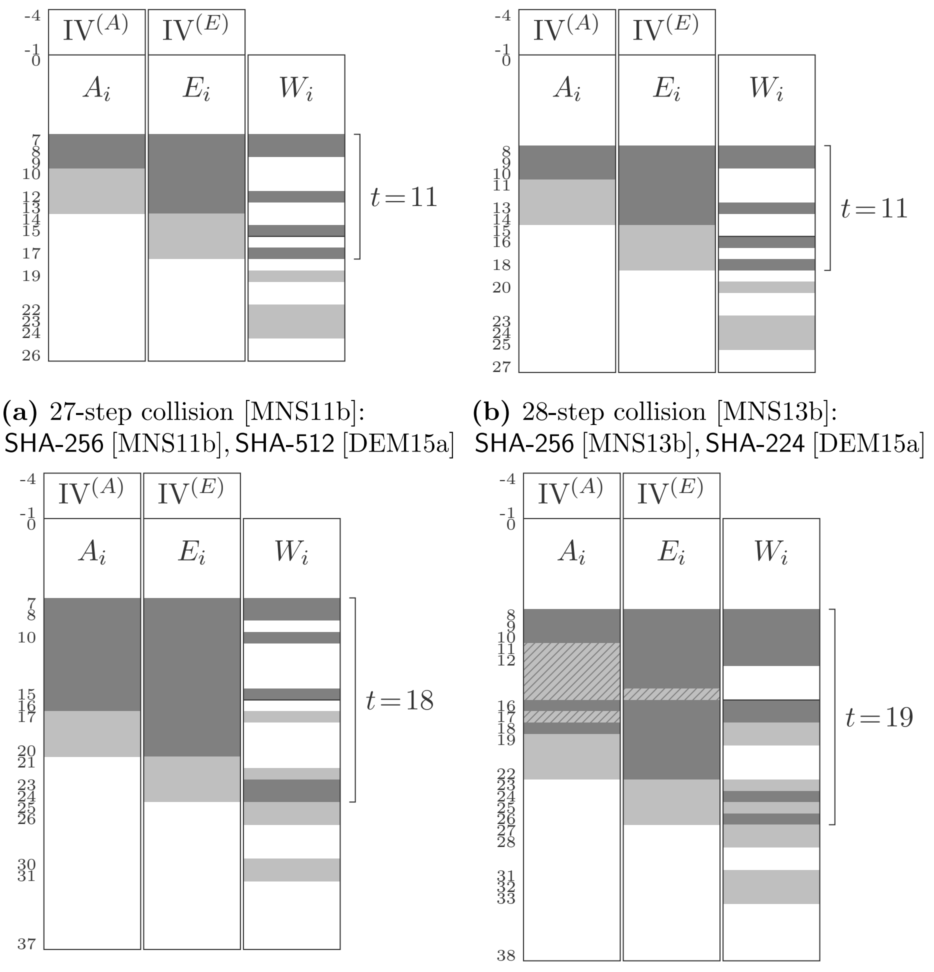
7.5. Application to the SHA-2 Family

We target SHA-512 and its truncated variants, in particular SHA-512/224 and SHA-512/256. The hash functions SHA-512/224 and SHA-512/256 differ from SHA-512 in their IV and a final processing step, which truncates the 512-bit state to 224 or 256 bits, respectively. Consequently, the semi-free-start collisions demonstrated for SHA-512 based on the SHA-256 starting points [EMS14] are also valid for these truncated versions (since the IV is non-standard anyway in this attack scenario). In this section, we first improve these results by providing 39-step semi-free-start collisions for SHA-512 and its variants. We then extend this result to free-start collisions for 43-step SHA-512/256 and 44-step SHA-512/224. By free-start collisions, we mean two messages m, m' and two IVs h_0, h'_0 such that the hash values of m (under IV h_0) and m' (under IV h'_0) collide. Note that free-start collisions are not equivalent to collisions of the compression function for truncated SHA-2 versions, since the truncated output bits of the last compression function call may contain differences. Additionally, we present collisions for 27 steps of SHA-512, SHA-512/224, and SHA-512/256.

7.5.1. Starting Points

To find candidates for a higher number of steps, we enumerated all possible selections of active message words (more precisely, of some $t \leq 20$ intermediate expanded message words, the “core words” of the local collision) and investigated the forward and backward expansion under certain assumptions: the t core words are chosen freely, according to the message expansion rule; in the forward and backward expansion, if at least 2 of the input words have differences, they are assumed to cancel out, while a single input word with difference never cancels out. Criteria for selecting suitable candidates then include a low number t of spanned steps and a low number of required cancellation constraints. The best (consistent) result for 39 steps, spanning $t = 19$ steps with 9 cancellations, is given in Figure 7.6d.

Semi-free-start collisions and collisions. The starting points of Figure 7.6a and Figure 7.6d both have at least 7 message words free of differences in the beginning. However, the local collision shown in Figure 7.6d spans over $t = 19$ steps. Thus, the first message words are constrained by many conditions, leaving not enough freedom to match



(a) 27-step collision [MNS11b]: SHA-256 [MNS11b], SHA-512 [DEM15a]

(b) 28-step collision [MNS13b]: SHA-256 [MNS13b], SHA-224 [DEM15a]

(c) 38-step SFS collision [MNS13b]: SHA-256 [MNS13b], SHA-512 [EMS14]

(d) 39-step SFS collision [DEM15a]: SHA-512 [DEM15a]

Figure 7.6.: SHA-2 starting points: Differences ■ and cancellations ■, ▨.

the correct IV. The starting point with $t = 18$ in Figure 7.6c is similarly unsuitable for IV matching. In contrast, the 11-step local collision shown in Figure 7.6a provides enough freedom in the first 7 message words to be used in a single-block collision attack [MNS11b].

7.5.2. Collision Attacks on SHA-512 and its Variants

Semi-free-start collisions

We use the 39-step starting point from Figure 7.6d. Previous work showed that sparse differences particularly in the A_i words are essential for the success probability of the message modification phase. For this reason, we additionally require that in 6 words between A_8 and A_{18}, namely $A_{11}, A_{12}, A_{13}, A_{14}, A_{15}$, and A_{17}, differences also cancel out. The five consecutive zero-difference words in A_i also force E_{15} to zero difference. These additional requirements are already marked in Figure 7.6d (hatched area).

The first task for the search procedure with the solving tool is to fix a suitable signed characteristic. Compared to the previously published 38-step SHA-512 semi-free-start collision [EMS14], the local collision for our starting point spans 19 steps (compared to previously 18) and has 9 (previously 6) active expanded message words. Cancellations are also required in 9 (previously 6) expanded message words. This increases the necessity for very sparse differences in A_i and W_i in steps 16–26. For this reason, we require a single-bit difference in W_{26}, W_{17} and A_{18}, and very low Hamming weights for the other words. We finally found a characteristic with at most two active bits in almost all words of A_i and W_i (except $A_9, A_{10}, W_{11}, W_{12}$), given in **??** in Table 7.6.

After the characteristic is fixed, we need to find a complying message pair. We start by guessing the dense parts in A_i and E_i, hoping that the sparser conditions in the later steps are fulfilled probabilistically. Since the dense parts are already almost fully determined by the characteristics and the sparse parts pose only so few conditions, a message pair is easily found. The result is a semi-free-start collision valid for all SHA-512 variants reduced to 39 rounds. We give an example in **??** in Table 7.19a.

Free-start collisions

Free-start collisions are a generalization of semi-free-start collisions, so the 39-step results obtained in the previous section give a first result for SHA-512/224 and SHA-512/256. However, we can take advantage of the truncated output bits to add several more steps. If we add another step in the beginning or in the end, the existing difference pattern remains unchanged, but there will be differences in the word W_0 (computable via backward expansion, which includes $W_{i+9} = W_9$, the previous W_8 from Figure 7.6d) or in the new word W_{39} (via the normal forward expansion, which includes $W_{39-15} = W_{24}$), respectively. These, in turn, can imply differences in E_{-4} or in A_{39} and E_{39}, which translates to differences in the IV (turning semi-free-start into free-start results, and included in the hash value via the feed-forward) or directly in the compression function output, respectively.

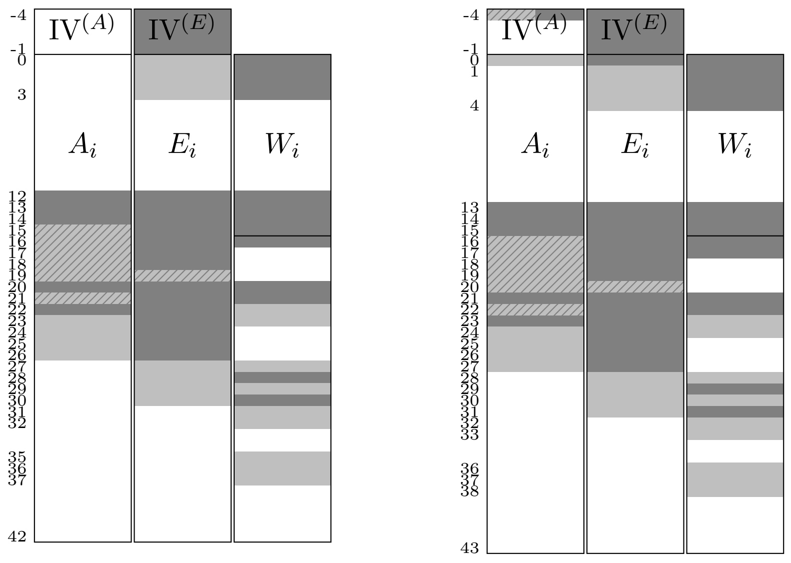
The advantage of adding steps in the beginning is that it is possible to limit the additional differences in the state update words to E, and keep A free of differences. Any differences in $E_{-1}, \ldots, E_{-4}$ will be added to the compression function output with the final feed-forward, but the corresponding words of the result are truncated, so the hash still collide.

Free-start collisions for 43-step SHA-512/256. Since SHA-512/256 truncates the last 4 output words of the compression function call ($E_{79} + E_{-1}$, $E_{78} + E_{-2}$, $E_{77} + E_{-3}$, and $E_{76} + E_{-4}$), differences in $E_{-1}, \ldots, E_{-4}$ are acceptable for a free-start collision. This observation allows us to add 4 additional steps in the beginning of the 39-step starting point from Figure 7.6d. Shifting the characteristic "downwards" by 4 steps causes the previous message words $W_{12}, \ldots, W_{15}$ to turn into new expanded message words $W_{16}, \ldots, W_{19}$; in particular, this affects the difference in the previous word W_{12}. To determine a compatible difference pattern for the new first 4 words, the message expansion can be computed backwards from the new words $W_4, \ldots, W_{19}$ via

$$W_i = W_{i+16} - \sigma_1(W_{i+14}) - W_{i+9} - \sigma_0(W_{i+1}).$$

It turns out that all 4 new words will contain differences (W_3 from $W_{3+9} = W_{12}$; W_2 from $W_{2+1} = W_3$ and $W_{2+14} = W_{16}$; W_1 from $W_{1+1} = W_2$ and $W_{1+14} = W_{15}$; and W_0 from $W_{0+1} = W_1$, $W_{0+14} = W_{14}$ and $W_{0+16} = W_{16}$). However, similar to steps 27–30, the state words A_i and E_i can be kept free of differences for 4 steps. To achieve this, the search tool needs to find differences in the IV words $E_{-4}, \ldots, E_{-1}$ to cancel out those in $W_0, \ldots, W_3$ when computing $E_0, \ldots, E_3$. The resulting starting point is given in Figure 7.7a.

For the search procedure with the solving tool, we fixed the signed differences of steps 12–30 to the same values as the 39-step SHA-512 semi-free-start collision of Section 7.5.2. Then, to complete the characteristic, we first search for a valid solution for the dense part of the middle steps



(a) 43-step SHA-512/256.   **(b)** 44-step SHA-512/224.

Figure 7.7.: Free-start starting points: Differences ■, cancellations ■, ▨.

(A_i and E_i in steps 13–16, and E_i in steps 17–27), and finally fix the corresponding message words W_i in steps 13–17, which determines the complete state, including dense differences in the prepended steps and IV.

The search only takes seconds on a standard computer; an example for a free-start collision is given in **??** in Table 7.15a. We also give a free-start collision for 41-step SHA-384 obtained with the same method (with 2 instead of 4 additional steps) in Table 7.17a, as well as one for 39-step SHA-224 (with 1 additional step added to the 38-step SHA-256 characteristic [MNS13b]) in Table 7.11a.

Free-start collisions for 44-step SHA-512/224. A very similar strategy can be employed to extend the previous 43-step free-start collision by another step for SHA-512/224. Prepending an additional step shifts the difference of previous word E_{-1} to E_0, which in turn requires a cancellation in A_0 and a difference in A_{-4}, as illustrated in Figure 7.7b. However, only the least significant 32 bits of the corresponding compression function output word are truncated. Furthermore, this output word is computed from A_{-4} via modular addition, so even differences only in the lower 32 bits can possibly cause differences in the untruncated output bits.

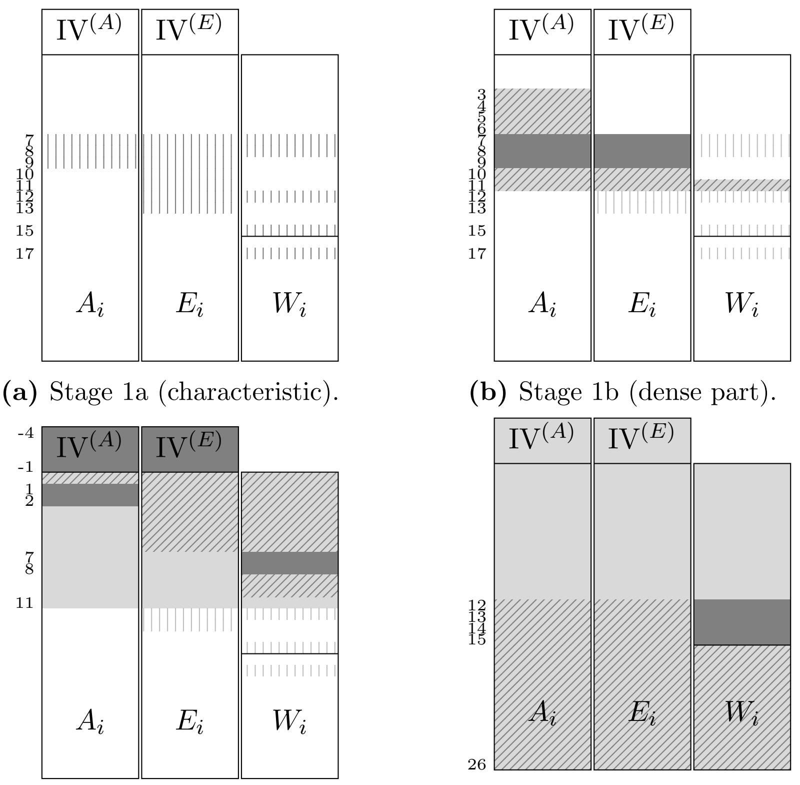
Fortunately, the underlying characteristic of signed differences as used for the 39-step SHA-512 semi-free-start collision is well compatible with our constraints: The difference in A_{-4} needs to cancel that in W_4 in a modular addition (via E_0, by equations (7.3) and (7.2) or Figure 7.3, since all other involved words have zero difference). This difference of W_4, in turn, is dictated by that in W_{13} (by the update rule for W_{20}, where again all other involved words have zero difference). None of these equalities involves any of the bitwise functions $\sigma, \Sigma, \mathsf{maj}$ or if. Thus, the modular difference in A_{-4} must be the same as that in W_{13}, which is already fixed by the underlying characteristic to a modular difference of $+32$. Written as bitwise differences, this will translate to a single-bit difference (in the sixth least significant bit) with probability $\frac{1}{2}$ (which does not carry over to the untruncated bits of the final output with overwhelming probability). Indeed, the example for a free-start collision given in **??** in Table 7.13a only displays this single-bit difference in A_{-4} (and no carries in the output bits).

Collisions

Starting point for SHA-512. Since the message expansion is essentially the same for all SHA-2 variants (except for different word sizes and rotation values, of course), the SHA-256 starting points can theoretically also be used for SHA-512. However, the resulting search complexity is different. For our results, we used the 27-step starting point (based on a local collision over the $t = 11$ steps 7–17), as illustrated in Figure 7.6a. Just as the 39-step semi-free-start starting point (Figure 7.6d), it requires that differences cancel in E in 4 of the t steps $(E_{14}, \ldots, E_{17})$ and in A in the 4 previous steps $(A_{10}, \ldots, A_{13})$, as well as in several steps of the message expansion.

Finding a solution from this starting point requires significantly more effort than for SHA-256. Of course, we also tried to expand our search to the closely related 28-step starting point, which adds an additional step in the beginning of the 27-step version. However, with the additional constraints imposed on the message expansion by this added step we could not find any suitable (reasonably sparse) characteristics.

In contrast to the results from Section 7.5.2, since the IV needs to exactly match the original IV, we were not able to take advantage of the final truncation to simplify the search process, or add additional steps. We first



(a) Stage 1a (characteristic).

(b) Stage 1b (dense part).

(c) Stage 2a (match IV).

(d) Stage 2b (sparse part).

Figure 7.8.: Stages of the 27-step collision search: guessed values ■ and differences ▥, derived values ▨, previously fixed values ■ and differences ▥

search a characteristic for SHA-512, and then try to use it to match the different IVs for SHA-512/224, SHA-512/256, SHA-384, and SHA-512.

Search strategy. The search progresses in several stages (Figure 7.8):

1. **Fix signed characteristic**:

    a) **Find candidate characteristic** (Figure 7.8a): First fix the signed differences of the message expansion W (5 words) and state update A (3 words). Since the word W_{17} poses conditions on the first few message words, whose freedom we will later need to match the IV, we focus on keeping its signed difference as sparse as possible, with only few difference bits. With much lower priority, also determine the differences in the state update words E (7 words) to complete the signed characteristic. The characteristic is very dense in E, but this only has limited influence on the success of the IV matching phase.

b) **Verify dense parts** (Figure 7.8b): Fully determine the values of A and E in the densest steps 7–9 to verify the validity of the candidate characteristic. If necessary, fix any remaining free bits of A and E in steps 10–11. This fully determines $A_3, \ldots, A_{11}$, $E_7, \ldots, E_{11}$ and W_{11}.

To maneuver the search process in the large search space and detect contradictions as soon as possible, we need to apply the look-ahead strategies previously employed for semi-free-start collisions on SHA-512 [EMS14] in this stage (with 16 look-ahead candidates per guess).

2. **Message modification to match IV**: Starting from the best signed characteristics of the previous stage, with the correct IV inserted, find a solution message pair step by step:

   a) **Match IV** (Figure 7.8c): Fix the values in the more difficult, heavily constrained words first (W_{10}, W_9, W_8, W_7). Choosing W_{10} and W_9 also determines A_2 and A_1 (via E_6 and E_5). Together with W_7, W_8, and the IV, this determines all values in steps 0–11.

   b) **Finalize message for sparse parts** (Figure 7.8d): choosing the 4 remaining message words $W_{12}, \ldots, W_{15}$ allows satisfying the remaining, sparse parts of the characteristic in steps 12–26 with high probability.

   Unlike the other stages, guesses are not made randomly here, but systematically word-by-word. Since most conditions are from modular additions, we always start from the least significant bits and proceed towards the more significant bits. This last stage needs to be repeated for each IV separately, which takes some hours on a single CPU per target IV.

Results. Our results for collisions for 27-step SHA-512/224, SHA-512/256, SHA-384, and SHA-512 are given in **??** in Table 7.13b, 7.15b, 7.17b, and 7.19b, respectively. We also include a 28-step collision for SHA-224, based on the SHA-256 characteristic [MNS13b], in Table 7.11b.

7.5.3. Search Strategy and Configuration

Below, we list all search parameters defining the respective search strategy. Each strategy is listed as an enumeration of several phases, and the algorithm only switches to the next phase when all bits of the current phase have been fixed (according to the defined settings). In each step, the algorithm selects one of the settings of the current phase randomly, distributed according to their specified weights w (e.g., a weight-5 setting is selected with 5 times the probability of a weight-1 setting). Each setting specifies which constraints of which words can be selected for guessing, and according to which guess pattern they are refined. In addition, it specifies whether (with which probability p) the choice is stored in the search tree for backtracking (i.e., on conflicts during backtracking, the alternative guess will be considered; otherwise, backtracking skips the alternative and proceeds resolving with the previous stored guess).

The sets of "undetermined" bits are defined by the settings of each phase as specified in the search strategy below. For picking the next bit of this set, we first randomly select one of the settings, and then either select the next least significant bit (if the setting specifies ordered guesses), or apply Algorithm 5 for look-ahead (in Phase I only), or pick a random bit (otherwise). The new constraints for the selected decision bit are also assigned probabilistically, as specified by the setting. We apply Algorithm 5 only in Phase I, with $s_{\max} = 16$, and the same settings in the Guessing and Propagation steps that are currently active in the main search in Algorithm 4. The basic settings follow [MNS13b; MNS11b] and are summarized in Table 7.4.

Table 7.4.: Basic guessing and backtracking strategy.

$\mathcal{G}$	Bit pairs	Description	Guessed to	Stack
`?`	Any	No constraints	`-` (100 %)	$p = 0$
`-`, `-`	$(0, 0)$, $(1, 1)$	Equal (`-` 2-bit c.)	`0` (50 %), `1` (50 %)	$p = 1$
`0`	$(0, 0)$	Fixed, equal bits		
`1`	$(1, 1)$	Fixed, equal bits		
`x`	$(0, 1)$, $(1, 0)$	Different bits	`u` (20 %), `n` (80 %) `u` (50 %), `n` (50 %)	$p = 1$
`u`	$(1, 0)$	Fixed, different bits		
`n`	$(0, 1)$	Fixed, different bits		

Table 7.5.: Starting point for semi-free-start collision of 39 steps of SHA-512 and SHA-512/t.

i	A_i	E_i	W_i
−4	--	--	
−3	--	--	
−2	--	--	
−1	--	--	
0	--	--	--
1	--	--	--
2	--	--	--
3	--	--	--
4	--	--	--
5	--	--	--
6	--	--	--
7	--	--	--
8	??	??	??
9	??	??	??
10	??	??	??
11	--	??	??
12	--	??	??
13	--	??	--
14	--	??	--
15	--	??	--
16	----------x---x---	??	??
17	--	??	--x-----
18	--x-----	??	--
19	--	??	--
20	--	??	--
21	--	??	--
22	--	??	--
23	--	--	--
24	--	--	----------x---x---
25	--	--	--
26	--	--	--x-----
27	--	--	--
28	--	--	--
29	--	--	--
30	--	--	--
31	--	--	--
32	--	--	--
33	--	--	--
34	--	--	--
35	--	--	--
36	--	--	--
37	--	--	--
38	--	--	--

Table 7.6.: Characteristic for semi-free-start collision of 39 steps of SHA-512 and SHA-512/t.

i	A_i	E_i	W_i
−4	--	--	
−3	--	--	
−2	--	--	
−1	--	--	
0	--	--	--
1	--	--	--
2	--	--	--
3	--	--	--
4	--	--	--
5	--	--	--
6	--	---10-----	--
7	--	-------------------------------01------------------------01-----	--
8	--n-----	-----------------1---0-------1-100-00-----101---1-----0--nu---1-	-0--n-----
9	---uuuu-u---u----------uuuuuu----unn-----------------unnn-nn----	nnn1-01nn11un1-------0--00--101nuuun00----000---0-1------u0u1---	--u--n----
10	nun-uuu-u-u-u-nu-----uun--n---n------uuu-n-nu---n-u-n--nun-uu-n-	101---0n00111000n1n-0nu-00--1n111u0un-100-nnn--0u-0u10-110u-1-0-	-1--u-----
11	--	n11---1u1un00uu1000n0u0010--1nu011u101-00n01n1-01-n-1nn0011n1u10	---------nnnnnn------n----------------------------nuuuuuu------n
12	--	1101nnn0n111n10u01000100nn1nn1u01101n11u10110-0101101n1u10n1n000	--nuuuuuuuuuuu--1u0--1111-
13	--	10-110n1u00nn00n100011110100n1110001uu11n1nn011110111uuuuu0u1011	------1--------------------------1------------------------------
14	--	1n1n10n0uu0nn00n00010u0u01n001n00000u10n1n01n101n1unu01n0uu0nnu0	---------------------1----------------1----------------1--------
15	--	-0-1-----1-----------011--1-011------10110110---01000---1001000-	--
16	----------u---u---	11-1--1---u---10-----100--1--00--11--10000--0--0100100011unnn11-	--u--n----
17	--	11111100111-101--0--00unn1010nu011nunuu000--u--00u00000111100011	-1--u-----
18	--u-----	0111100000n-00--01--001-----11101unnnnnnnnnnnnnnnnnnnnnnnnnnu100	--
19	--	uuuuuuuuuuu-nuuuuuuuuuu001-0uunnnn0100001100010nuuuuuuuuuuu0n01-	--
20	--	----101111u0--100000000-----0--011---101111101101101nuuuuuu1u---	--
21	--	1111111111--11111111111-----111111---1011111011000000111110-1---	--
22	--	0-0-0-0-0-1-1-0-0-1------------0-0unnnnnnnnnnnnnnnnnnnnnnnn-0-0-	--
23	--	----------------------------------1111111111111111111111111-----	--
24	--	----------------------------------1111111111111111111111111-----	----------n---n---
25	--	--	--
26	--	--	--n-----
27	--	--	--
28	--	--	--
29	--	--	--
30	--	--	--
31	--	--	--
32	--	--	--
33	--	--	--
34	--	--	--
35	--	--	--
36	--	--	--
37	--	--	--
38	--	--	--

Table 7.7.: Characteristic for free-start collision of 44 steps of SHA-512/224.

i	A_i	E_i	W_i
−4	--n-----	nu-unn-u--u--n----n----nnn-n--n--u--nn-uu-n--nn-nnnunnu--nn-u-u-	
−3	--	u-nuuuuu--uunnu---nn-1u1--n--u0-0nu-n-un-----nnuu-n1uu---uu1unn-	
−2	--	u-u0-n-n-uu-u-nn--u--uun----n-1nn--n-nn--------u--u0n--nun0u-nu-	
−1	--	--nu0n1n-0-1uunun-0uunnu-uuu11u0n1011--0--1-101-00un1nnnnnnnn0uu	
0	--	0-10---------1-000001010-0001-00---0-11--------1---0011-1nu00101	--u--n-----uu-n-uu---u--nu-nnnn-u-n----u-un---n-nn-n-uu-u-n---nn
1	--	--01-0-0---11101--01101-101--1-0------------------11-0010-010-10	nn-nnn-u--unu--u----unu-nnn---n-n---u--n----un--u-unnn--nn0-uu1n
2	--	---11-----	n01un1unuun01nn--n01uuun0nuunun11--u-uu-------n-uuuu-nuuu0u-1nnn
3	--	--	---uun-u---nunu-n--nnu-ununn-nu-u-------------------n-------n1nn
4	--	--	--u-----
5	--	--	--
6	--	--	--
7	--	--	--
8	--	--	--
9	--	--	---1
10	--	--	--
11	--	---10-----	--
12	--	-------------------------------01------------------------01-----	--
13	--n-----	-----------------1---0-------1-100-00-----101---1-----0--nu---1-	-0--n-----
14	---uuuu-u---u----------uuuuuu----unn-----------------unnn-nn----	nnn1-01nn11un1------0--00--101nuuun00----000---0-1------u0u1---	--u--n----
15	nun-uuu-u-u-u-nu-----uun--n---n------uuu-n-nu---n-u-n--nun-uu-n-	101---0n00111000n1n-0nu-00--1n111u0un-100-nnn--0u-0u10-110u-1-0-	-1--u-----
16	--	n11---1u1un00uu1000n0u0010--1nu011u101-00n01n1-01-n-1nn0011n1u10	00------0nnnnnn------n00------000----------1111111nuuuuuu------n
17	--	1101nnn0n111n10u01000100nn1nn1u01101n11u10110-0101101n1u10n1n000	--nuuuuuuuuuuu1--0101-----0111-----0------------110---1u0--1111-
18	--	10-110n1u00nn00n100011110100n1110001uu11n1nn011110111uuuuu0u1011	------------------------1---------------------------------------
19	--	1n1n10n0uu0nn00n00010u0u01n001n00000u10n1n01n101n1unu01n0uu0nnu0	--------0------------0----------------0----------------1--------
20	--	-0-1-----1-----------011--1-011------10110110---01000---1001000-	--
21	----------u---u---	11-1--1---u---10-----100--1--00--11--10000--0--0100100011unnn11-	--u--n----
22	--	11111100111-101--0--00unn1010nu011nunuu000--u--00u00000111100011	-1--u-----
23	--u-----	0111100000n-00--01--001-----11101unnnnnnnnnnnnnnnnnnnnnnnnnnu100	--
24	--	uuuuuuuuuuu-nuuuuuuuuuu001-0uunnnn0100001100010nuuuuuuuuuuu0n01-	--
25	--	----101111u0--100000000-----0--011---101111101101101nuuuuuu1u---	--
26	--	1111111111--11111111111-----111111---1011111011000000111110-1---	--
27	--	0-0-0-0-0-1-1-0-0-1------------0-0unnnnnnnnnnnnnnnnnnnnnnnn-0-0-	--
28	--	----------------------------------1111111111111111111111111-----	--
29	--	----------------------------------1111111111111111111111111-----	----------n---n---
30	--	--	--
31	--	--	--n-----
32	--	--	--
33	--	--	--
34	--	--	--
35	--	--	--
36	--	--	--
37	--	--	--
38	--	--	--
39	--	--	--
40	--	--	--
41	--	--	--
42	--	--	--
43	--	--	--

Table 7.8.: Starting point for collision of 27 steps of SHA-512 and SHA-512/t.

i	A_i	E_i	W_i
−4	--	--	
−3	--	--	
−2	--	--	
−1	--	--	
0	??	??	--
1	??	??	--
2	??	??	--
3	??	??	--
4	??	??	--
5	??	??	--
6	??	??	--
7	??	??	??
8	??	??	??
9	??	??	--
10	??	??	--
11	??	??	--
12	??	??	??
13	??	??	--
14	??	??	--
15	??	??	??
16	??	??	--
17	??	??	???x
18	??	??	--
19	??	??	--
20	??	??	--
21	??	??	--
22	??	??	--
23	--	--	--
24	--	--	--
25	--	--	--
26	--	--	--

Table 7.9.: Characteristic for collision of 27 steps of SHA-512 and SHA-512/t.

i	A_i	E_i	W_i
−4	--	--	
−3	--	--	
−2	--	--	
−1	--	--	
0	--	--	--
1	--	--	--
2	--	--	--
3	--	--	--
4	--	--	--
5	--	----0------0----------0---0------0------------------------------	--
6	--	----1------1-------0--10011--1---1---11-----------1-------------	--
7	u---u-n----n--------nu-uuuu------u------------------------------	-nnnn-n1--1n----1-11--n110n011-1-u0--00-----1-111-0---1---00-1--	u-unnnu--nuu---------nunuuu------u-----------------------1-1001-
8	------------------------------nn-----uuun----u----u---u-u--uu--u	-0u0u000n-1u----010un-nuunu0-u---u1--uuuuuuu1u000-n1u-1--010-0-0	----uuuuuu------n-unnnnnnn-----------------------------------11u
9	--un-----n------n-------un--unnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn	0n1n00100unn0u11n-n10000110u1001-01u1110011111nuuuu010n1-1u--1-1	--
10	--	100010010011011-100001u11u0u110uuuu1100nn110u00000uu01u1un00--1u	--
11	--	u1u1n-n1-01n-1010-111n0uuuu-u--11--000-0uu--0--100-100001---1011	-----------------------10---------------------------------------
12	--	nuuu0-01--11-0010-unnnnnnn100-0100-----100-000-111010-0110----00	n-----u--u----u-uu-uuuu-----u--u---n--------u---------------n---
13	--	---u1-1-n-uuuuuuu-------un1-1-1--------------1-1---1-----------u	---1
14	--	111-----0-0010011-1100---0	--
15	--	---0----1-1111111-------10-------------------------------------1	nnunuun------uuu-nnn------nnu----------nun----001---------------
16	--	--	--
17	--	--	---n-----u------u--------n-------------------------------------n
18	--	--	--
19	--	--	--
20	--	--	--
21	--	--	--
22	--	--	--
23	--	--	--
24	--	--	--
25	--	--	--
26	--	--	--

SHA-512/t 39-step semi-free-start collision: We follow the strategy in Section 7.5.2 to find the signed characteristic (Phase I) and pair; results are given below:

- **Phase I**: Guess `?`, `x` in $\mathbf{W}$ ($w = 5$) or $\mathbf{E}, \mathbf{A}$ ($w = 1$) (final 2-bit-condition check; look-ahead with $s_{\max} = 16$; biased guesses in `x`)
- **Phase II**: Guess `-` in $\mathbf{A}_{9\ldots12}, \mathbf{E}_{9\ldots12}$ (ordered guesses LSB→MSB)
- **Phase III**: Guess `-` in $\mathbf{E}_{13\ldots24}$ (ordered guesses LSB→MSB)
- **Phase IV**: Guess `-` in $\mathbf{W}_{9\ldots12}$ (ordered guesses LSB→MSB)

For the truncated free-start collisions of Section 7.5.2 for SHA-512/t, we can use essentially the same strategy, shifted by the right number of steps.

SHA-512/t 27-step collision: We use the strategy of Section 7.5.2:

- **Phase I**, Step **1(a)**: Guess `?`, `x` in $\mathbf{W}_{17}$ ($w = 20$) or $\mathbf{A}, \mathbf{W}$ ($w = 5$) or $\mathbf{A}, \mathbf{E}, \mathbf{W}$ ($w = 1$) (final 2-bit check; $s_{\max} = 16$; fair guesses for `x`)
- **Phase II**, Step **1(b)**: Guess `-` in $\mathbf{A}_{7\ldots9}, \mathbf{E}_{7\ldots9}$ (o.g. LSB→MSB)
- **Phase IIIa**, Step **2(a)**: Guess `-` in $\mathbf{A}_2$ (ordered guesses LSB→MSB)
- **Phase IIIb**, Step **2(a)**: Guess `-` in $\mathbf{A}_1$ (ordered guesses LSB→MSB)
- **Phase IIIc**, Step **2(a)**: Guess `-` in $\mathbf{W}_8$ (ordered guesses LSB→MSB)
- **Phase IIId**, Step **2(a)**: Guess `-` in $\mathbf{W}_7$ (ordered guesses LSB→MSB)
- **Phase IV**, Step **2(b)**: Guess `-` in $\mathbf{W}_{12\ldots15}$ (o.guesses LSB→MSB)

Table 7.10.: Results for SHA-224.

h_0	5594e74a	2234bcbd	635966b5	97a9e488	4a6a7d63	7c28ca05	f1384c84	d2c9753a
h_0^*	5594e74a	2234bcbd	635966b5	97a9e488	4a6a7d63	7c28ca05	f1384c84	d2c97532
Δh_0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000008
m	949722de	48501dcc	7fe48849	ba821c7a	5b5a1e6f	30c487c8	401134e4	2d9eacc7
	dd15d12f	9079b000	37c75e69	a47f0dde	8baaf91a	d348cc06	2b64ef59	011f91be
m^*	949722e6	48501dcc	7fe48849	ba821c7a	5b5a1e6f	30c487c8	401134e4	2d9eacc7
	e373cfef	9079affc	37c75e69	a4808dde	8baaf91a	d348cc06	2b64ef59	011f91be
Δm	00000038	00000000	00000000	00000000	00000000	00000000	00000000	00000000
	3e661ec0	00001ffc	00000000	00ff8000	00000000	00000000	00000000	00000000
h_1	1d6d980a	2aa5f9c0	9843296b	da4f8baa	09c36608	7e2bdad9	cb0f1654	

(a) Example of a free-start collision for 39 steps of SHA-224.

m	a3e2972c	73ba31f5	e9f85a00	c2cb8cbc	4dd15d27	d240b6f6	0c1243ec	07504bfd
	134d0e53	fb839dcc	e14f2cbc	53c3c1c6	6ac516a7	a19b112f	d844f621	939e7e53
m^*	a3e2972c	73ba31f5	e9f85a00	c2cb8cbc	4dd15d27	d240b6f6	0c1243ec	07504bfd
	e3f85ef3	f75b9934	e14f2cbc	53c3c1c6	6ac516a7	a3db19bf	d844f621	939e7e53
Δm	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
	f0b550a0	0cd804f8	00000000	00000000	00000000	02400890	00000000	00000000
h_1	869fd0b5	fb96d20b	28177d1e	c7af4885	06ecb162	88332da4	5022b6c7	

(b) Example of a collision for 28 steps of SHA-224.

Table 7.12.: Results for SHA-512/224.

h_0	fef65b64d3694995	959fbfb82ed84eb1	1d9e855642e62ef2	335cc6d027695d91
	921d197e5cfa2803	e26c6eb26163a692	9ff3cf4d26f1de78	5323942861d9139a
h_0^*	fef65b64d3694995	959fbfb82ed84eb1	1d9e855642e62ef2	335cc6d027695db1
	a712860cdcfa1ff8	470749bbf7628f44	20cdfd694df67216	8e07b5fa2c7fedf0
Δh_0	0000000000000000	0000000000000000	0000000000000000	0000000000000020
	350f9f72800037fb	a56b2709960129d6	bf3e32246b07ac6e	dd2421d24da6fe6a
m	7a19df6089d00684	03ed2a0d0c29e00e	36c91e35f681fbb8	bb2b47428aeff294
	dce94ccc981d39a3	44230f73cf56d9ef	e9d46b26b44950c8	550bed4b9419741c
	58a98894206e00de	f3448a6f761d384d	9ae59f3a3bcc5bba	ece85d5c77be431b
	6e3cf817e9376cc7	b74a2a43c0b96c93	7c5b51d6fe2a0c26	5a9868e5bf2e422d
m^*	5e031bbe28b2d027	ded424ef85255cc3	ad2f514be0830c1f	a635dab40aeffa9f
	dce94ccc981d3983	44230f73cf56d9ef	e9d46b26b44950c8	550bed4b9419741c
	58a98894206e00de	f3448a6f761d384d	9ae59f3a3bcc5bba	ece85d5c77be431b
	6e3cf817e9376cc7	b74a2a43c0b96cb3	5c5b51d6fe2a0c36	5a9868e5bf2e420d
Δm	241ac4dea162d6a3	dd390ee2890cbccd	9be64f7e1602f7a7	1d1e9df68000080b
	0000000000000020	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000020	2000000000000010	0000000000000020
h_1	e309edf68f4d89b8	5c356e0359eb0dab	76b4a45ec3c2cd25	8bd0955d

(a) Example of a free-start collision for 44 steps of SHA-512/224.

m	20dbf13a352116a9	295506e205afd435	abfe4826742c1a1a	279f07c7813dd9be
	47da77c701a98858	25aec1349d486501	37a992a15616ea31	e2b122ecf19e90d3
	2fff6025dc03dd67	032c261d740f459e	2e2599bd6e7e74df	d490bd22815eb494
	72fedf1f607df6e3	87fc91fcfb7397fd	e647b1b499eee17f	2dff8e493cbc8a4c
m^*	20dbf13a352116a9	295506e205afd435	abfe4826742c1a1a	279f07c7813dd9be
	47da77c701a98858	25aec1349d486501	37a992a15616ea31	5cc1250cb19e90d3
	203fdfe5dc03dd66	032c261d740f459e	2e2599bd6e7e74df	d490bd22815eb494
	f0bc01167075f6eb	87fc91fcfb7397fd	e647b1b499eee17f	d3f8fe713d7c8a4c
Δm	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	be7007e040000000
	0fc0bfc000000001	0000000000000000	0000000000000000	0000000000000000
	8242de0910080008	0000000000000000	0000000000000000	fe07703801c00000
h_1	65b11e66e48da563	1b70d12da92e2dba	8f338768bb95601b	60b995bb

(b) Example of a collision for 27 steps of SHA-512/224.

Table 7.14.: Results for SHA-512/256.

h_0	159b52516f10f30d	546b2042f240afee	f25339b24c441edf	d62c698666558242
	e5a9e39861fbd81d	d2138eacc20d5224	a332c16df23609fb	73f78341dfd7a4e5
h_0^*	159b52516f10f30d	546b2042f240afee	f25339b24c441edf	d62c698666558242
	e5a9e39861fbd83d	72e259ce420d5a0f	4db37906cc361264	ae579d9e0275b446
Δh_0	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000020	a0f1d7628000082b	ee81b86b3e001b9f	dda01edfdda210a3
m	cfbec86f1cf6821e	dd3343c25aad835a	2a08612b753f3d6b	b328d40d2c624ef7
	b3e51f8a3a63bd6f	4abdf96375bbf609	a8c5c1f784672e86	a78e2aa625830d4b
	169dcb5039bf3d9f	fbcc43ffebd8ae47	1b3eaefccf5c6a46	f668a2a728851b4e
	374601ea44422bdb	2ca290d26a23a02f	6685babbfdcb5e22	e000111457201fd4
m^*	ee37d77210586a56	b2a4122800ad72cf	89399609f53f3560	b328d40d2c624ed7
	b3e51f8a3a63bd6f	4abdf96375bbf609	a8c5c1f784672e86	a78e2aa625830d4b
	169dcb5039bf3d9f	fbcc43ffebd8ae47	1b3eaefccf5c6a46	f668a2a728851b4e
	374601ea44422bfb	0ca290d26a23a03f	6685babbfdcb5e02	e07e151457202055
Δm	21891f1d0caee848	6f9751ea5a00f195	a331f7228000080b	0000000000000020
	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000020	2000000000000010	0000000000000020	007e040000003f81
h_1	1d7041bbbffa676a	03d8c440d9246b9d	20ce2d17c5b0b2c4	7e6e4d33a7f54afd

(a) Example of a free-start collision for 43 steps of SHA-512/256.

m	306b0c2ebe7c1341	c8b55d4df1c5f4fe	b91a173aeceb818a	33b5977f9b46e58b
	6c6d5a4f87f1364f	1b7e33249d4acf4f	b7f784ecdcaefc1f	a33edafe7afc0452
	dfc0200932c2b9df	faec7d05e3518e56	ec2e19a7ee867396	d490bd22815eb494
	72fedf1f887df303	f95891f08483da25	c327d0afa2c4f902	2c5f0c0806a4e298
m^*	306b0c2ebe7c1341	c8b55d4df1c5f4fe	b91a173aeceb818a	33b5977f9b46e58b
	6c6d5a4f87f1364f	1b7e33249d4acf4f	b7f784ecdcaefc1f	1d4edd1e3afc0452
	d0009fc932c2b9de	faec7d05e3518e56	ec2e19a7ee867396	d490bd22815eb494
	f0bc01169875f30b	f95891f08483da25	c327d0afa2c4f902	d2587c300764e298
Δm	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	be7007e040000000
	0fc0bfc000000001	0000000000000000	0000000000000000	0000000000000000
	8242de0910080008	0000000000000000	0000000000000000	fe07703801c00000
h_1	fcba5c8faf05fd68	c676b8f17b5daae3	6233801174b7fd01	0ff72ab4a869c54f

(b) Example of a collision for 27 steps of SHA-512/256.

Table 7.16.: Results for SHA-384.

h_0	500c5f3c4787ba7d	a068fcb4011d5bf3	42e5d7f8a8b5379a	7e74fdcc5a87935c
	f5f415efc47d32d6	26dd61b02c6d0535	c7db5e89eefe94b2	428c2357ed063d18
h_0^*	500c5f3c4787ba7d	a068fcb4011d5bf3	42e5d7f8a8b5379a	7e74fdcc5a87935c
	f5f415efc47d32d6	26dd61b02c6d0535	c7db5e89eefe94d2	a76a0e166d06354d
Δh_0	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000060	e5e62d4180000855
m	e13bb65dd3648429	0c1f502f0462116a	537195824dd4d9c3	a79d8323b63058ee
	88821d5bb67c9042	d37ee7690e0f79f2	ebefa3e85f97bc7b	e9e2ac55d87d9d51
	881ea963df122650	210c7bf69ef999c7	b1e4fdb6c5e209ca	3eee8c4a06d943c4
	484835ab03c4a2ff	2001ba37b3a01fd8	5ffd223aff721f3f	398994124514202b
m^*	7c5dcb9f53648c14	0c1f502f0462114a	537195824dd4d9c3	a79d8323b63058ee
	88821d5bb67c9042	d37ee7690e0f79f2	ebefa3e85f97bc7b	e9e2ac55d87d9d51
	881ea963df122650	210c7bf69ef999c7	b1e4fdb6c5e209ea	1eee8c4a06d943d4
	484835ab03c4a2df	207fbe37b3a02059	6001223aff721e3f	398994124514202b
Δm	9d667dc28000083d	0000000000000020	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000020	2000000000000010
	0000000000000020	007e040000003f81	3ffc000000000100	0000000000000000
h_1	667b472d680391c2	9c41c2626b95724d	3e537e772da88bed	cfbd5a3b5037bfef
	e6d7e0d6b53df84d	f9667a25301c99e4		

(a) Example of a free-start collision for 41 steps of SHA-384.

m	3e7553f2cfb535ab	c6b10da716e10303	dacfae5c546a644d	f583858a09a07330
	80f9a52d3920a14d	5cdc569b9d21b864	b1a502c28b3d61d8	e2b122ece7ac4b72
	dfc0201101b358e7	8166c65680a5ac4f	ab8499afe6873554	d490bd22815eb494
	76fedf1f605ff2d3	d88056eb1a397147	aefff39c56655d5b	2d9f834e6cb4f200
m^*	3e7553f2cfb535ab	c6b10da716e10303	dacfae5c546a644d	f583858a09a07330
	80f9a52d3920a14d	5cdc569b9d21b864	b1a502c28b3d61d8	5cc1250ca7ac4b72
	d0009fd101b358e6	8166c65680a5ac4f	ab8499afe6873554	d490bd22815eb494
	f4bc01167057f2db	d88056eb1a397147	aefff39c56655d5b	d398f3766d74f200
Δm	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	be7007e040000000
	0fc0bfc000000001	0000000000000000	0000000000000000	0000000000000000
	8242de0910080008	0000000000000000	0000000000000000	fe07703801c00000
h_1	dc3c054014ee5e3b	4da7c81dba383c4e	d9997f5a26fe5f59	d987d2e5bc2f7e83
	46ba52d79f525f95	90c5db2cc94e87ee		

(b) Example of a collision for 27 steps of SHA-384.

Table 7.18.: Results for SHA-512.

h_0	eccf3da189dd9668	b1ec21a4fd53b8d8	609ce4465f772770	adf4e7738e2978f6
	8edd237ea50eebc9	231b3af0102a926d	db45e613e8d2fd52	ad384433420073f6
m	a0ec9872cfffe63c	df5c6a2b59f4c453	f2bea3763fc8fa7a	6a47e8ff0a995116
	fa59232e8b617048	4c9690984c084498	28bee8f5701eab16	8d57686ecbdce623
	3879318f901ff782	72644b0ca55a6142	6cb281dab11480b4	4a8198441f401ff2
	5ffd956ed11a2b5f	9a640988d68287d3	74942df792f2637f	b2819dc61f772d4f
m^*	a0ec9872cfffe63c	df5c6a2b59f4c453	f2bea3763fc8fa7a	6a47e8ff0a995116
	fa59232e8b617048	4c9690984c084498	28bee8f5701eab16	8d57686ecbdce623
	3879318f901ff7a2	52644b0ca55a6152	6cb281dab1148094	4aff9c441f402073
	6001956ed11a2a5f	9a640988d68287d3	74942df792f2637f	b2819dc61f772d4f
Δm	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000020	2000000000000010	0000000000000020	007e040000003f81
	3ffc000000000100	0000000000000000	0000000000000000	0000000000000000
h_1	3aa73bfae7b82789	711f2024cf0f636e	0c6965f707279a53	8227fba8617aa955
	fdd9e2ca8c4d0038	57db244560d7b70b	08ec5698343353c0	9e9b739ee307ea92

(a) Example of a semi-free-start collision for 39 steps of SHA-512.

m	537e7a4986aa2fce	11206ad0306c752b	90124a9e1c9b0ce2	8c14e0356fd26f5f
	fd3ef90ea3e4366f	35d8c2ba58abd92f	b23e476632eca1fd	e2b122ef46649b73
	dfc020070e628f37	7acf74d1d1007558	6c6359a6fe7fe2f0	d490bd22815eb494
	72fedf1f807df6f3	a8585af19b6dd9d1	3d2053b0c295522b	2d970e0e52a49081
m^*	537e7a4986aa2fce	11206ad0306c752b	90124a9e1c9b0ce2	8c14e0356fd26f5f
	fd3ef90ea3e4366f	35d8c2ba58abd92f	b23e476632eca1fd	5cc1250f06649b73
	d0009fc70e628f36	7acf74d1d1007558	6c6359a6fe7fe2f0	d490bd22815eb494
	f0bc01169075f6fb	a8585af19b6dd9d1	3d2053b0c295522b	d3907e3653649081
Δm	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	be7007e040000000
	0fc0bfc000000001	0000000000000000	0000000000000000	0000000000000000
	8242de0910080008	0000000000000000	0000000000000000	fe07703801c00000
h_1	d838f1d2ae4bf185	3fc837ae9bbc28d4	6b2f2977f58a9697	99c48839f0e8bdca
	c9c0a86fed1d921a	2f823b1fa1913751	3ba170b902c6da30	9c4e5807be51a7e7

(b) Example of a collision for 27 steps of SHA-512.

7.6. Application to Design of Malicious SHA-1

As the recent NSA revelations have shown, it is not sufficient to only demonstrate that it is hard for a malicious third-party attacker to break a cryptographic scheme. In fact, the designers themselves might be considered an even greater threat, since it is possible for them to embed backdoors that only they can exploit with some secret backdoor key – and it might be very hard to even detect that there is such a backdoor.

We present collisions for a version of SHA-1 with modified constants, where the colliding payloads are valid binary files (executables, archives, images). These collisions can readily be exploited in applications that use custom hash functions (e.g., for customers' segmentation) to ensure that a legitimate application or file can surreptitiously be replaced with a malicious one, while still passing the integrity checks such as secure boot or application code signing. In terms of Aumasson's malicious hashing framework [Aum11; AAE+14], Malicious SHA-1 is an instance of a static collision backdoor. Our malicious SHA-1 instances have round constants that differ from the original ones in about 40 bits on average. Modified versions of cryptographic standards are typically used on closed systems such as media platforms and aim to differentiate cryptographic components across customers or services. Our proof-of-concept thus demonstrates the exploitability of custom SHA-1 versions for malicious purposes, such as the injection of user surveillance features.

SHA-1 is an interesting target because it was one of the most widely deployed hash function and because of its background as an NSA/NIST design. We exploit the freedom of the four 32-bit round constants of SHA-1 to efficiently construct 1-block collisions such that two valid executables collide for this malicious SHA-1. Such a backdoor could be trivially added if a new constant is used in every step of the hash function. However, in SHA-1 only four different 32-bit round constants are used within its 80 steps, which significantly reduces the freedom of adding a backdoor. Our approach modifies at most 80 (or, on average, 40) bits of the constants.

This demonstrates not only the power of backdoors, but also shows how important it is to question the values and reasons behind every element in a cryptographic design. Unfortunately, national standards do not always follow these recommendations – for instance, the Russian hash standard Streebog comes filled with seemingly "high-entropy" constants and linear functions, without any further justification [AY15].

7.6.1. Background on Backdoors and Malicious Hashing

The NSA revelations of 2013 highlighted the apparent interest of intelligence agencies in cryptographic backdoors. A cryptographic backdoor aims to allow selected entities, such as the designer or parameter selector, to subvert the security goals of a cryptographic algorithm, whereas the algorithm is (relatively) secure against third-party cryptanalysis. The most prominent example is the easily backdoorable random number generator Dual_EC_DRBG [BK12]. Even since then, only relatively few academic publications have targeted this topic, for example with a formal treatment of backdoored pseudorandom generators [DGG+15] and proposals of "trustworthy" designs [LW17]. Schneier et al. [SFKR] provide a survey of attempts at surreptitiously weakening cryptographic systems.

Aumasson [Aum11; AAE+14] provides first definitions for different types of backdoor properties in hash functions. All definitions refer to the adversary, Eve, in terms of two algorithms: a *malicious generator* and an *exploit algorithm*. The first is essentially the malicious hash function designer who, after executing a probabilistic algorithm, returns a hash function (to be published) plus an additional piece of information to unlock the backdoor (secret). The second is a potentially separate entity that runs another algorithm using the backdoor information as input and attacks some security property of the hash function. If this exploit algorithm is essentially deterministic, the backdoor is said to be *static*, else for a probabilistic algorithm it is *dynamic*. Backdoor adversaries are further categorized according to their *undetectability*, referring to the hardness of detecting that there is a backdoor and recovering the according exploit algorithm, and their *undiscoverability*, referring to the hardness of recovering the backdoor information necessary for the exploit algorithm for an adversary who knows the exploit algorithm and H.

In particular, Aumasson defines a *static collision adversary* as a backdoor adversary where the probabilistic malicious generator returns a hash function H and backdoor information b, and the deterministic exploit algorithm, upon input of H and b, returns one colliding message pair $m \neq m'$ with $H(m) = H(m')$. A *dynamic collision adversary*'s probabilistic exploit algorithm samples such a pair (m, m') from a larger space.

We propose an undiscoverable static collision adversary for a modified, Malicious SHA-1. Since the first publication of Malicious SHA-1 [AAE+14], other designs of backdoored hash functions with modified round constants have been proposed [AY15; Mor15].

7.6.2. Background on SHA-1

SHA-1 is a hash function designed by the NSA and standardized by NIST in 1995 [Dan12]. The design is similar to, but simpler than SHA-2; we give a summary in the following. SHA-1 processes 512-bit message blocks m_j and produces a 160-bit hash value by iterating the compression function f with a similar Davies-Meyer construction as SHA-2. The 160-bit internal state is divided into 5×32-bit words A_i, B_i, C_i, D_i, E_i.

SHA-1's state update transformation consists of 4 rounds of 20 steps each. The step function is illustrated in Figure 7.9, where the bitwise Boolean function f_r and the round constant K_r depend on the round r, $1 \leq r \leq 4$:

$$\begin{array}{llll}
f_1 = \mathsf{if}(B,C,D) & K_1 = \lfloor\sqrt{2}\cdot 2^{30}\rfloor = \texttt{5a827999} & \text{for} & 0 \leq i \leq 19 \\
f_2 = \mathsf{xor}(B,C,D) & K_2 = \lfloor\sqrt{3}\cdot 2^{30}\rfloor = \texttt{6ed9eba1} & \text{for} & 20 \leq i \leq 39 \\
f_3 = \mathsf{maj}(B,C,D) & K_3 = \lfloor\sqrt{5}\cdot 2^{30}\rfloor = \texttt{8f1bbcdc} & \text{for} & 40 \leq i \leq 59 \\
f_4 = \mathsf{xor}(B,C,D) & K_4 = \lfloor\sqrt{10}\cdot 2^{30}\rfloor = \texttt{ca62c1d6} & \text{for} & 60 \leq i \leq 79
\end{array}$$

The expanded message word W_i for step i is derived from the 16×32-bit words M_i of message block m_j with the $\mathbb{F}_2$-linear message expansion:

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

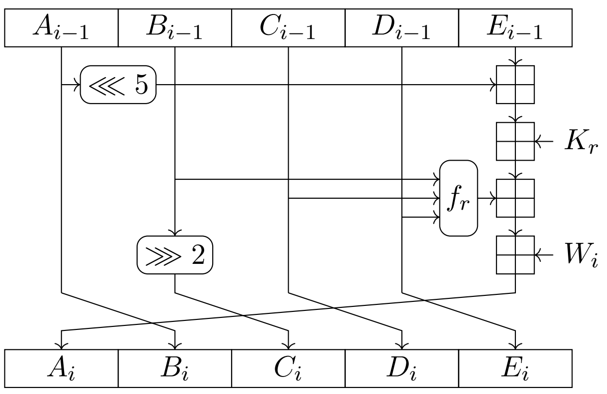


Figure 7.9.: The step function of SHA-1.

Analysis of SHA-1

SHA-1 has been considered theoretically broken since 2005, when Wang et al. [WYY05b] presented the first collision attack on full SHA-1 with an estimated complexity of about 2^{69}. Although deprecated, SHA-1 is still widely used in a variety of applications, such as GIT, and was accepted in SSL certificates by major browsers until 2017.

Wang et al. [WYY05b] included a practical collision example for round-reduced SHA-1 with 58 out of 80 steps with cost about 2^{33} SHA-1 calls. More practical round-reduced examples were published for 64 steps with 2^{35} [DR06], 70 steps with 2^{44} [DMR07], 73 steps with $2^{50.7}$ [Gre10], and 75 steps with $2^{57.7}$ [AG12]. Furthermore, practical semi-free-start collisions for and free-start collisions for 76-step [KPS15] and full SHA-1 [SKP16] were presented. The theoretical complexity for full SHA-1 collisions was improved to 2^{63} [WYY05a; Coc07] and later 2^{61} [Ste13].

Finally, in 2017, Stevens et al. [SBK+17] presented the first practical full-round collision example with a complexity of about $2^{63.1}$ SHA-1 evaluations (6500 CPU years and 110 GPU years).

All attacks are based on the differential attack strategy by Wang et al. [WYY05b] and its improvements: We first construct a high-probability differential characteristic that yields a zero (or nearly zero) output difference, i.e., a collision or near-collision. In the second stage, we probabilistically try to find a confirming message pair for this differential characteristic using message modification. The characteristic is divided into a denser, low-probability part at the beginning of the compression function (first round of SHA-1), referred to as "nonlinear (NL)" characteristic, and a sparse, high-probability "linear (L)" part covering the remaining rounds.

The high-probability part of the differential characteristic for SHA-1 covers Rounds 2 to 4. It has been shown [CJ98; WYY05b; Man11; Ste13] that for SHA-1, the best way to construct these high-probability characteristics is to interleave local collisions (one disturbing and a set of correcting differences). These characteristics can be constructed by using a linearized variant of the hash function and tools from coding theory [RO05; PRR05]. The probability of this characteristic determines the complexity of the attack. The low-probability part and message modification in Round 1 can be performed using automated non-linear equation solving tools [DR06].

For SHA-1, the best attacks construct two-block collisions whose first block produces a near-collision canceled by the second block [WYY05b].

7.6.3. Malicious SHA-1

We present an example of a *static collision backdoor*: The backdoor adversary Eve constructs a custom variant of SHA-1 that differs from the standardized specification only in the values of some of the round constants (up to 80 bits). Eve can use the additional freedom gained from choosing only part of the 4×32-bit constants to find a practical collision for the full modified SHA-1 function during its design. We show that Eve even has enough freedom to construct a meaningful collision block pair which she can, at a later point, use to build multiple colliding file pairs of a particular format (e.g., executable or archive format) with almost arbitrary content. This is stronger than required by a static collision backdoor, and possible due to the narrow-pipe Merkle-Damgård design of SHA-1.

The backdoor does not exploit any particular "weaknesses" of specific round constants, nor does it weaken the logical structure of the hash function. Instead, it only relies on the designer's freedom to choose the constants during the attack. This freedom can be used to improve the complexity of previous attacks [Ste13; WYY05b] and thus makes it feasible to find collisions for the full hash function.

For an attacker who only knows the modified constants but cannot choose them, collisions are as hard to find as for the original SHA-1.Thus, in terms of the definitions of the previous section, this backdoor is *undiscoverable*. It is, however, *detectable* since constants in hash functions are normally expected to be identifiable as nothing-up-your-sleeve numbers. This is not achievable in our attack.

SHA-1 uses only four round constants $K_1, \ldots, K_4$ of 32 bits each, one in each of the four 20-step rounds. In our malicious collision attack, we use the freedom of these four constants to reduce the complexity of the attack. Mendel and Schläffer suggested the following attack strategy [AAE+14]. Similar to message modification, we choose the constants during the search for a confirming message pair. We modify the constants in a round-by-round strategy, always selecting a round constant such that the differential characteristic for the steps of the current round can be satisfied. Since we have to choose the constants when processing the first block, we can only improve the complexity of this block. Hence, we need to use a differential characteristic that results in a single-block collision. Note that all the collisions attacks on SHA-1 so far use a 2-block characteristic.

To find the high-probability part of a differential characteristic for Rounds 2 to 4 resulting in a 1-block collision, a linearized variant of the hash function can be used. However, using algorithms from coding theory, we only find differential characteristics that maximize the overall probability and do not take the additional freedom we have in the choice of the constants in SHA-1 into account. Therefore, to minimize the overall attack complexity, we did not use these differential characteristics. Instead, we are interested in a differential characteristic such that the minimum of the three probabilities for Rounds 2, 3, and 4 is maximized. To find such a characteristic, we start with the best overall characteristic and modify it to suit our needs.

In previous attacks on SHA-1, the best differential characteristics for Rounds 2 to 4 have differences only at bit position 2 for some 16 consecutive state words A_i [Man11]. We assume that the best differential characteristic has the same property in our case. Hence, we only need to determine all 2^{16} possible differential characteristics with differences only at bit position 2 in 16 consecutive state words A_i and linearly expand them backward and forward. A similar approach has also been used to attack SHA-0 [CJ98] and SHA-1 [WYY05b; Man11] .

For each of these 2^{16} differential characteristics, we estimate the cost of finding a malicious single-block collision. These costs are roughly determined by the number of differences (disturbances) in A_i in each round. For details on the cost computations, we refer to the analysis of Pramstaller et al. [PRR05] and Mendel et al. [MPRR06b]. The estimated costs for the best differential characteristics suited for our attack are given in Table 7.20, and the corresponding message differences are given in Table 7.22 (on p. 220).

The high-probability differential characteristic with message difference $\Delta m^{(1)}$ is best suitable for our intended file formats (see Section 7.6.4), so we use it as the starting point to search for a low-probability differential

Table 7.20.: Differential probability of suitable characteristics.

Candidate	$r = 2$	$r = 3$	$r = 4$	Total
$\Delta m^{(1)}$	2^{-40}	2^{-40}	2^{-15}	2^{-95}
$\Delta m^{(2)}$	2^{-39}	2^{-42}	2^{-13}	2^{-94}
$\Delta m^{(3)}$	2^{-39}	2^{-42}	2^{-11}	2^{-92}

Table 7.21.: Characteristic for message difference $\Delta m^{(1)}$, with constraints for `jpeg`/`rar` polyglot collision.

i	A_i	W_i
0	1001111110001101100110001001010n	111111111101100011111111111000nu
1	00u1101001100--------001111u0n00	11u0001000000000-----0100unu0n00
2	n111n00---1------------uu000un00	u011n1-----------------000001000
3	0uuuu111--0---0---uu---0-0un11nn	nnu-n--------------------nn000u1
4	1n01u1110---u-n-------u0011001n0	uu1-u------------------00u0011uu
5	0011011n1n00--0-un0101-10n1u0n00	10u0u--------------------1101u11
6	n1n1n1n01000--1-100101-00n000011	1u111--------------------u-001u0
7	nu1nnnnnnnnnnnnnnnnnnnnnnnn000n1	0n10u00------------------n000nn1
8	101111-10011000000010000111nu0u1	n001u----------------------000u1
9	0-101010100000000000000001un001	10110100-----------------01u1u00
10	u1n00------------------------01u	1011n---------------------0-00n1
11	-00-0--------------------1100001	u0u----------------------n1n0n00
12	-0010-----------------------00-1	u01-n----------------------100n0
13	----1-----------------------1100	n0100----------------------11011
14	-------------------------------n	u1u-u---------------------0000nn
15	-1--------------------------0---	n1u-u-------------------0un10010
16	-0--------------------------1---	1110----------------------1000un
17	------------------------------u-	nn01---------------------10111u1
18	-------------------------------n	nn001--------------------n-010nu
19	--------------------------------	un1----------------------un111n1
20	--------------------------------	n11-----------------------0011nu
21	------------------------------n-	0u01---------------------01110n1
22	-------------------------------n	0u0----------------------u1100nu
23	--------------------------------	nn1----------------------un001n1
24	--------------------------------	n010-----------------------100un
25	------------------------------n-	1n1---------------------011001n0
26	-n-----------------------------u	011----------------------u-110un
27	--------------------------------	nn00---------------------nu0u0n0
28	------------------------------u-	nn101---------------------11001u
29	n-----------------------------n-	uu1n---------------------n0101n0
30	--------------------------------	1n1u---------------------u1u01u0
31	--------------------------------	uu0u---------------------11101u0
32	-------------------------------u	01n----------------------10110un
33	--------------------------------	01n--------------------10nu00001
34	u-------------------------------	10u----------------------10100nu
35	------------------------------n-	nu0--------------------1001n11n1
36	------------------------------n-	1n0----------------------u0111n0
37	--------------------------------	nun--------------------00u1100u0
38	------------------------------u-	n0u-----------------------110001
39	--------------------------------	10n0-------------------10n010111
40	n-------------------------------	u01----------------------10000n0
41	--------------------------------	101----------------------01u1001
42	u------------------------------u	u10-0---------------------0111un
43	--------------------------------	n0u--------------------01nun0010
44	--------------------------------	n1u----------------------10000nu
45	------------------------------n-	nu0----------------------00111n1
46	------------------------------u-	uuu----------------------u1111u1
47	--------------------------------	uun--------------------10n0000u0
48	------------------------------u-	n00---------------------10101100
49	--------------------------------	100--------------------11n010100
50	------------------------------u-	010--------------------1-0100010
51	--------------------------------	010--------------------01n100010
52	--------------------------------	u01----------------------11100n0
53	--------------------------------	101--------------------010110101
54	-------------------------------n	n11----------------------101100n
55	--------------------------------	u00--------------------110u11000
56	------------------------------u-	100----------------------00011uu
57	------------------------------u-	1u1--------------------11n1011u0
58	-------------------------------u	1u1----------------------n00101n
59	--------------------------------	nu0--------------------10nu001n1
60	--------------------------------	101---------------------110000nu
61	------------------------------n-	un1--------------------0010000n1
62	------------------------------n-	1n--1------------------10u0111n1
63	--------------------------------	uu1--------------------11u0111u0
64	--------------------------------	u10-1------------------0100000n0
65	--------------------------------	0--------------------11110001011
66	------------------------------n-	111-------------------0-000011n1
67	--------------------------------	u1-------------------0110u100100
68	------------------------------u-	--0--------------------011000100
69	--------------------------------	u1---------------------01n000010
70	------------------------------u-	n-1--------------------111011101
71	--------------------------------	-0-------------------1100n01100-
72	------------------------------u-	u----------------------000001110
73	--------------------------------	--1------------------0011n111011
74	------------------------------u-	n-----------------------101111--
75	--------------------------------	---------------------0000n011101
76	--------------------------------	u----------------------1110001u-
77	--------------------------------	-----------------------101000--1
78	--------------------------------	n---------------------000011101-
79	--------------------------------	n--------------------111000101--

characteristic for the first round of SHA-1. We use the same tool as in Section 7.5 to find the low-probability part of the characteristic. The resulting signed characteristic (with a few additional constraints from Section 7.6.4) is illustrated in Table 7.21. We show how to find a colliding message pair using malicious constants for this differential characteristic with roughly 2^{48} complexity in the following.

After the differential characteristic is fixed, we probabilistically search for a confirming message pair. We start with only the first constant K_1 fixed (e.g., to the standard value) and search for a message pair that confirms at least Round 1 (steps $0 \leq i < 20$) of the characteristic, and is compatible with any additional constraints in case of meaningful collisions. This is easier than finding a message pair that works for all four rounds (with fixed constants), since fewer constraints need to be satisfied. The complexity of this step is negligible.

Now, we can exhaustively search through all 2^{32} options for K_2 until we find one that confirms Round 2. If no such constant is found, we backtrack and modify the message words. Since the differential characteristic for message difference $\Delta m^{(1)}$ holds with probability 2^{-40} in Round 2 and we can test 2^{32} options for K_2, this step of the attack will succeed with probability 2^{-8}. Completing this step thus has a complexity of about 2^{40}.

Once we have found a candidate for K_2 such that the differential characteristic holds in Round 2, we proceed in the same way with K_3. Again, the differential characteristic will hold with only a probability of 2^{-40} in Round 3 and we can test only 2^{32} options for K_3. Therefore, we need to repeat the previous steps of the attack 2^8 times to find a solution. Including the expected 2^8 tries for the previous step to reach the current one, completing this step has an expected complexity of roughly 2^{48}.

Finally, we need to find K_4. Since the last round of the characteristic has a high probability, such a constant is very likely to exist and the additional cost of this step is negligible. In summary, we need about 2^{48} tries and freely choose 32-bit K_2, 32-bit K_3, and about 16 bits of K_4, or about 80 bits in total. On average, half of those 80 bits are different from the original constants.

With fixed constants, an attacker would have to backtrack in the case of a contradiction in the later steps. Eve as the designer, on the other hand, has a chance that choosing a different constant might repair any contradictions. This significantly improves the complexity of the differential attack compared to 2^{95} in the case of fixed constants.

7.6.4. Meaningful Collisions for Malicious SHA-1

To demonstrate the backdoor attack, we want to create meaningful collisions where both colliding messages m, m' are valid files in some file format. The format specification of file formats imposes additional constraints on both the characteristic and the confirming pairs. In particular, most file formats specify a fixed "magic signature" for the first few bytes of the file, which consequently needs to be free of differences. The signature is used to identify the type of binary, and often followed by various metadata such as format version or size information.

Constraints. Based on the attack strategy of Section 7.6.3, the two colliding files must each start with a 512-bit collision block with relatively fixed differences and little control over the contents, followed by an arbitrarily long common suffix with identical content. If the two files are to be semantically significantly different, these differences must either be encoded in or triggered from this first block. Due to the lack of freedom when choosing the first block, in our examples the bulk of the meaningful content of both files is contained in the later identical blocks. The differences in the first block only select the relevant parts using jumps or similar. Due to these limitations, for example, the method that was used to find colliding PostScript files with common prefixes for MD5 [DL05] cannot be applied here.

For the exact values of the first block, the attack allows a certain freedom and the attacker can fix the values of a few bits in advance. However, fixing too many bits will increase the attack complexity. Additionally, choosing the bits is constrained by the fixed message difference, which can be selected from a few candidate starting points listed in Table 7.22. In all our example files, we use message difference $\Delta m^{(1)}$, which offers a slightly better expected attack complexity than $\Delta m^{(2)}$ and $\Delta m^{(3)}$. All available message differences have a difference in both the first and the last byte.

$\Delta m^{(1)}$	00000003 20000074 88000000 e8000062 c8000043 28000004 40000042 48000046 88000002 00000014 08000002 a0000054 88000002 80000000 a8000003 a8000060
$\Delta m^{(2)}$	20000074 88000000 e8000062 c8000043 28000004 40000042 48000046 88000002 00000014 08000002 a0000054 88000002 80000000 a8000003 a8000060 00000003
$\Delta m^{(3)}$	88000000 e8000062 c8000043 28000004 40000042 48000046 88000002 00000014 08000002 a0000054 88000002 80000000 a8000003 a8000060 00000003 c0000002

Table 7.22.: List of message differences suitable for the attack

Script formats. A first simple example of an executable file collision is two Unix shell scripts that each execute an arbitrary program, and only differ in the first 64-byte block. The trick used is to introduce the colliding block as a comment in the script (that is, a line starting with a "`#`" character, and excluding new line characters `0a`). Although the block contains non-printable characters, it is interpreted as valid comment by the command-line interpreter, and thus ignored in the program execution. After the first block, we can append a newline character to end the comment and then arbitrary commands.

To achieve a different behavior for the two files, we include a command that reads the shell script file itself. Depending on whether the read value corresponds to the first or second file, we can execute arbitrary different commands. An example can be found in Figure 7.10.

A limitation of colliding shell scripts is that the malicious behavior is straightforward to detect, given the execution conditioned to the value of the first block. The actual malicious code executed by one of the two colliding scripts may be arbitrarily obfuscated, since the script can call any external binary file rather than executing as a sequence of shell commands. A similar trick can be used to produce colliding scripts in arbitrary languages: JavaScript, Python, Perl, etc.

```
> hd file0.sh
00000000  23 de eb 7c 8a 8a 34 ee  8d 60 3f 3e 38 9c 31 3c  |#..|..4..‘?>8.1<|
00000010  12 69 7f a2 e9 ad b8 d4  dc 8f 9f b2 b7 9d 1d c8  |.i..............|
00000020  d0 01 5f 70 87 cf e8 fa  99 68 3e ca 46 a1 cc d8  |.._p.....h>.F...|
00000030  1f a8 f0 2b f8 7e 06 1a  b5 82 07 45 c5 bc 7d 52  |...+.~.....E..}R|
00000040  0a 0a 69 66 20 5b 20 60  6f 64 20 2d 74 20 64 49  |..if [ ‘od -t dI|
00000050  20 2d 6a 33 20 2d 4e 31  20 2d 41 6e 20 22 24 7b  | -j3 -N1 -An "${|
00000060  30 7d 22 60 20 2d 65 71  20 22 31 32 34 22 20 5d  |0}"‘ -eq "124" ]|
00000070  3b 20 74 68 65 6e 20 0a  20 20 65 63 68 6f 20 22  |; then .  echo "|
00000080  67 6f 6f 64 2e 22 3b 0a  65 6c 73 65 0a 20 20 65  |good.";.else.  e|
00000090  63 68 6f 20 22 65 76 69  6c 2e 22 3b 0a 66 69 0a  |cho "evil.";.fi.|
000000a0

> hd file1.sh
00000000  23 de eb 7f aa 8a 34 9a  05 60 3f 3e d0 9c 31 5e  |#.....4..‘?>..1^|
00000010  da 69 7f e1 c1 ad b8 d0  9c 8f 9f f0 ff 9d 1d 8e  |.i..............|
00000020  58 01 5f 72 87 cf e8 ee  91 68 3e c8 e6 a1 cc 8c  |X._r.....h>.....|
00000030  97 a8 f0 29 78 7e 06 1a  1d 82 07 46 6d bc 7d 32  |...)x~.....Fm.}2|
(...)
000000a0
```

$K_{1\ldots4}$	5a827999 82b1c71a 5141963a a664c1d6
$h(m)$	3da8361d ae9ed92d ff98735b a4e95869 cbbc3945

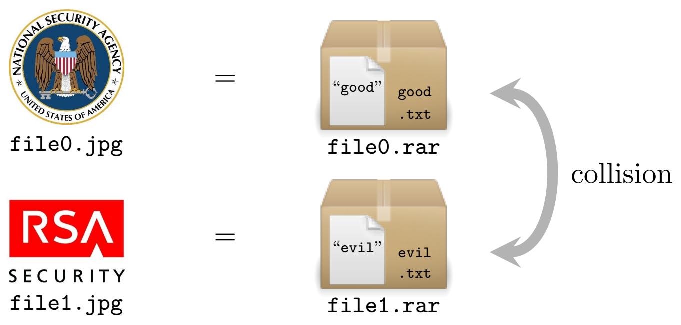
Figure 7.10.: Instance with colliding `sh`-script file pair (see Table 7.21).

Polyglots. Albertini investigated the suitability of a variety of file formats with these constraints [AAE+14; Alb17]. Most file formats start with 4-byte signatures and are thus not compatible with the message differences in Table 7.22. Due to the constraints of the attack, the most suitable formats are those without magic signatures, such as master boot records (`mbr`) and DOS executables (`com`), and those with variable starting offsets for the signature, such as archive formats (`rar`, `7z`). For headerless code formats, the initial differences of the characteristic can be used to encode jump instructions to different offsets. For archive formats that start at the first occurrence of the signature at an arbitrary position, the difference can be used to place this first signature at different offsets.

In particular, some of the file format constraints are not mutually exclusive; for example, a valid `rar` file can at the same time be a valid `jpg` file. This allows us to prepare initial 512-bit collision blocks that can later be extended to colliding files of different formats (or even one same file of several formats). It is worth noting that the latter – a single file with multiple meaningful interpretations – can be a security problem in itself, and can only be handled on an application layer, not with cryptographic approaches.

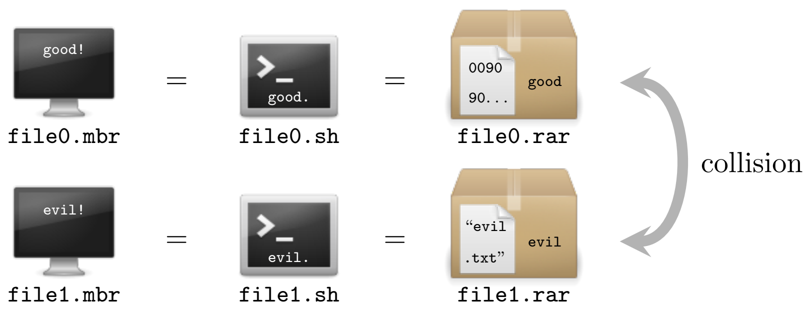
Using the attack strategy from Section 7.6.3, we constructed several Malicious SHA-1 instances each suitable for several file formats. Examples of colliding (and polyglot) proof-of-concept files created by Albertini are given in Figure 7.11.

For a polyglot collision that can later be filled with virtually arbitrary data in `jpeg` image format and `rar` archive format, we require a collision block that starts with `ffd8 ffe?` (for `jpeg`) and ends with `52` in one of the two files (for `rar`) $\Delta m^{(1)}$ is the only one of the message differences in Table 7.22 for this format. Using these constraints as a starting point, we can search for a differential characteristic. The result is given in Table 7.21. Note that at this point, the first round constant K_1 is fixed to an arbitrary value (we use the original constant), while K_2, K_3, K_4 are still free. They are determined together with the full first 64-byte block in the next phase. The result is the message pair already given in Figure 7.11a. The Malicious SHA-1 variant differs from the standardized SHA-1 in the values of 45 bits of the round constants. Now, we can append suitable followup blocks to create valid `jpeg` or `rar` file pairs, both with arbitrary content. As an example, both images in Figure 7.11a hash to the same digest.



$K_{1...4}$	5a827999 4eb9d7f7 bad18e2f d79e5877
m	ffd8ffe1 e2001250 b6cef608 34f4fe83 ffae884f afe56e6f fc50fae6 28c40f81
	1b1d3283 b48c11bc b1d4b511 a976cb20 a7a929f0 2327f9bb ecde01c0 7dc00852
m^*	ffd8ffe2 c2001224 3ecef608 dcf4fee1 37ae880c 87e56e6b bc50faa4 60c40fc7
	931d3281 b48c11a8 b9d4b513 0976cb74 2fa929f2 a327f9bb 44de01c3 d5c00832
Δm	00000003 20000074 88000000 e8000062 c8000043 28000004 40000042 48000046
	88000002 00000014 08000002 a0000054 88000002 80000000 a8000003 a8000060
$h(m)$	1896b202 394b0aae 54526cfa e72ec5f2 42b1837e

(a) Instance with colliding polyglot `jpeg`/`rar` file pair.



$K_{1...4}$	5a827999 82b1c71a 5141963a a664c1d6
m	23deeb7c 8a8a34ee 8d603f3e 389c313c 12697fa2 e9adb8d4 dc8f9fb2 b79d1dc8
	d0015f70 87cfe8fa 99683eca 46a1ccd8 1fa8f02b f87e061a b5820745 c5bc7d52
m^*	23deeb7f aa8a349a 05603f3e d09c315e da697fe1 c1adb8d0 9c8f9ff0 ff9d1d8e
	58015f72 87cfe8ee 91683ec8 e6a1cc8c 97a8f029 787e061a 1d820746 6dbc7d32
Δm	00000003 20000074 88000000 e8000062 c8000043 28000004 40000042 48000046
	88000002 00000014 08000002 a0000054 88000002 80000000 a8000003 a8000060
$h(m)$	0c426a87 6b6e8e42 f0558066 2ddd5a85 939db8b9

(b) Instance with colliding polyglot `mbr`/`sh`/`rar` file pair.

Figure 7.11.: Malicious SHA-1 instances and meaningful colliding pairs.

In a similar fashion, we were able to construct another example block pair for a different set of SHA-1 constants that is suitable for master boot records, shell scripts and `rar` archives, as illustrated in Figure 7.11b. The constants differ from the original values by 41 (of 128) bits. Both files can be and executed as a shell script, booted as a master boot record, (or emulated using `qemu-system-i386 -fda file0.mbr`), and extracted as a `rar` archive.

The characteristic already specifies the necessary format fragments required in Figure 7.11a: The first 28 bits of word W_0 are set to `ffd8ffe` to accommodate the JPEG format, and the last 8 bits of word W_{15} are fixed as `52` (in one message m, for the `rar` header) or `32` (in the other message m^*). Additionally, the first round constant is already fixed to the original value $K_1 =$ `5a827999`, while K_2, K_3, K_4 are still free to be chosen during message modification.

All example file pairs and code for verification can be found online at `http://malicioussha1.github.io/`.

The recently published practical attack on full SHA-1 [SBK+17] provides a pair of colliding `pdf` files based on similar considerations. Compared to Malicious SHA-1, the differential constraints are however different and less restrictive: the collision blocks can be placed arbitrarily within the file [Alb17]. On the other hand, the comparably low computational cost of Malicious SHA-1 leaves significantly more freedom and room for retries and experiments: Whereas new Malicious SHA-1 instances and collisions can be created in the equivalent of much less than 2^{48} SHA-1 computations (40 hours on 80 CPUs), the actual SHA-1 collisions later required over 2^{63} SHA-1 evaluations (6500 CPU years and 110 GPU years) [SBK+17].

8 Conclusions

Permutations and tweakable block ciphers have shown the potential to rival block ciphers in their role as the ideal primitive for efficient and elegant schemes. However, modes for these primitives require different state geometries than the classic 128-bit key, 128-bit state block ciphers. In addition, the known, chosen, or related round-key material of these primitives may in some cases further complicate their security analysis and make it more difficult to evaluate their security margin. This is particularly challenging for lightweight designs and otherwise constrained environments.

In this thesis, we analyzed the security of several different primitives using methods based on differential cryptanalysis, in a very broad sense. For two strongly aligned designs, we broke the designers' security claims: The tweakable block cipher MANTIS-5 in Chapter 3 and the permutation Simpira-4 v1 in Chapter 4. Properties we exploited in these attacks include differential clustering in a related-tweak setting due to the first-order and second-order differential properties of the S-box, the simple key schedule and FX construction, suitable initial structures for related-tweak or keyless settings, and properties of the generalized Feistel network. For two more, weakly aligned designs, we provided attacks on the round-reduced primitive: The block cipher LowMC in Chapter 5, by exploiting higher-order differential properties, and the compression function of the hash standard SHA-2 in Chapter 7, by improving an automatic search tool to obtain practical collision for a significantly higher number of rounds of SHA-512 and its truncated variants SHA-512/t than previous analysis. Finally, for two primitives, we showed results outside the standard security claim: A related-key attack on the authenticated cipher Prøst in Chapter 6, and a backdoor collision attack on the hash function SHA-1 with malicious round constants in Chapter 7.

Automated search tools play an increasingly important role in this context: In contrast to established block-cipher design approaches like the wide-trail design strategy, the analysis of these primitives often requires the help of a computer. In particular, characteristics for related-tweakey setups and for weakly aligned designs with large state sizes are often too complex for manual analysis. We observed that tools like SAT solvers or dedicated search tools can be surprisingly effective but still require significant work from the cryptanalyst for best results (Chapter 7). This includes judging the result of an automated search for bounds, characteristics, or other properties with caution (Chapter 3, Chapter 4).

The most important research questions arising from this thesis concern suitable design approaches to tackle these challenges for different primitives. The analysis we presented offers starting points to consider: For tweakable block ciphers, it is essential to factor the properties of the tweakey schedule into the analysis of any attack. For example, we suggest to extend the analysis of the differential behaviour to semi-truncated differences in order to detect differential clusters for fixed tweak differences (Chapter 3). In the case of unkeyed primitives, the adversary can profit from starting computations in the middle of the primitive and using techniques like inside-out computations and message modification to satisfy constraints in a suitable order (Chapter 4, Chapter 5, Chapter 7), whereas in regular use, the primitive is typically only evaluated in forward direction. This motivates the use of operations whose inverse is harder to compute. In the absence of randomizing round keys, the careful choice of the linear layer is particularly important to ensure indepence of differential conditions (Chapter 3, Chapter 4).

In conclusion, block-cipher alternatives offer interesting new directions in both design and analysis.

References

[AAE+14] A. Albertini, J.-P. Aumasson, M. Eichlseder, F. Mendel, and M. Schläffer. Malicious Hashing: Eve's Variant of SHA-1. In: Selected Areas in Cryptography – SAC 2014. Ed. by A. Joux and A. M. Youssef. Vol. 8781. LNCS. Springer, 2014, pp. 1–19. DOI: 10.1007/978-3-319-13051-4_1. IACR: 2014/694 (pp. 165, 169, 212, 213, 216, 222).

[ABB+14] E. Andreeva, B. Bilgin, A. Bogdanov, A. Luykx, B. Mennink, N. Mouha, and K. Yasuda. APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography. In: Fast Software Encryption – FSE 2014. Ed. by C. Cid and C. Rechberger. Vol. 8540. LNCS. Springer, 2014, pp. 168–186. DOI: 10.1007/978-3-662-46706-0_9 (p. 150).

[ABL+13] E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, E. Tischhauser, and K. Yasuda. Parallelizable and Authenticated Online Ciphers. In: Advances in Cryptology – ASIACRYPT 2013. Ed. by K. Sako and P. Sarkar. Vol. 8269. LNCS. Springer, 2013, pp. 424–443. DOI: 10.1007/978-3-642-42033-7_22 (p. 151).

[AC09] M. R. Albrecht and C. Cid. Algebraic Techniques in Differential Cryptanalysis. In: Fast Software Encryption – FSE 2009. Ed. by O. Dunkelman. Vol. 5665. LNCS. Springer, 2009, pp. 193–208. DOI: 10.1007/978-3-642-03317-9_12. IACR: 2008/177 (p. 51).

[Ada92] C. M. Adams. On Immunity Against Biham and Shamir's "Differential Cryptanalysis". In: Information Processing Letters 41.2 (1992), pp. 77–80. DOI: 10.1016/0020-0190(92)90258-W (p. 55).

[ADMV15] E. Andreeva, J. Daemen, B. Mennink, and G. Van Assche. Security of Keyed Sponge Constructions Using a Modular Proof Approach. In: Fast Software Encryption – FSE 2015. Ed. by G. Leander. Vol. 9054. LNCS. Springer, 2015, pp. 364–384. DOI: 10.1007/978-3-662-48116-5_18. URL: https://www.esat.kuleuven.be/cosic/publications/article-2502.pdf (p. 42).

[AG12] A. V. Adinetz and E. A. Grechnikov. Building a Collision for 75-Round Reduced SHA-1 Using GPU Clusters. In: Parallel Processing – Euro-Par 2012. Ed. by C. Kaklamanis, T. S. Papatheodorou, and P. G. Spirakis. Vol. 7484. LNCS. Springer, 2012, pp. 933–944. DOI: 10.1007/978-3-642-32820-6_91. IACR: 2011/641 (p. 215).

[AGM+09] K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang. Preimages for Step-Reduced SHA-2. In: Advances in Cryptology – ASIACRYPT 2009. Ed. by M. Matsui. Vol. 5912. LNCS. Springer, 2009, pp. 578–597. DOI: 10.1007/978-3-642-10366-7_34. IACR: 2009/479 (p. 173).

[AGR+16] M. R. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen. MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In: Advances in Cryptology – ASIACRYPT 2016. Ed. by J. H. Cheon and T. Takagi. Vol. 10031. LNCS. 2016, pp. 191–219. DOI: 10.1007/978-3-662-53887-6_7. IACR: 2016/492 (p. 129).

[AJN16] J.-P. Aumasson, P. Jovanovic, and S. Neves. NORX v3. Submission to CAESAR: Competition for Authenticated Encryption. Security, Applicability, and Robustness (Round 3). 2016. URL: http://competitions.cr.yp.to/round3/norxv30.pdf (p. 20).

[AKKM08] A. Atalay, O. Kara, F. Karakoç, and C. Manap. SHAMATA Hash Function Algorithm Specifications. Submission to NIST's SHA-3 Competition (Round 1). 2008. URL: http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/documents/SHAMATA.zip (p. 107).

[AL12] M. R. Albrecht and G. Leander. An All-In-One Approach to Differential Cryptanalysis for Small Block Ciphers. In: Selected Areas in Cryptography – SAC 2012. Ed. by L. R. Knudsen and H. Wu. Vol. 7707. LNCS. Springer, 2012, pp. 1–15. DOI: 10.1007/978-3-642-35999-6_1. IACR: 2012/401 (p. 62).

[Alb17] A. Albertini. Exploiting (identical prefix) hash collisions. BlackAlps 2017. 2017. URL: https://www.blackalps.ch/files/2017/talks/BlackAlps17-Albertini.pdf (pp. 222, 224).

[Alo14] B. Alomair. AVALANCHE v1. Submission to CAESAR: Competition for Authenticated Encryption. Security, Applicability, and Robustness (Round 1). 2014. URL: `http://competitions.cr.yp.to/round1/avalanchev1.pdf` (p. 149).

[AM09] J.-P. Aumasson and W. Meier. Zero-Sum Distinguishers for Reduced Keccak-f and for the core functions of Luffa and Hamsi. Presented at the rump session of Cryptographic Hardware and Embedded Systems – CHES 2009. 2009. URL: `https://131002.net/data/papers/AM09.pdf` (pp. 63, 129).

[AMG+16] M. Amy, O. D. Matteo, V. Gheorghiu, M. Mosca, A. Parent, and J. M. Schanck. Estimating the Cost of Generic Quantum Pre-image Attacks on SHA-2 and SHA-3. In: Selected Areas in Cryptography – SAC 2016. Ed. by R. Avanzi and H. M. Heys. Vol. 10532. LNCS. Springer, 2016, pp. 317–337. DOI: `10.1007/978-3-319-69453-5_18`. IACR: `2016/992` (p. 173).

[ARS+15] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In: Advances in Cryptology – EUROCRYPT 2015. Ed. by E. Oswald and M. Fischlin. Vol. 9056. LNCS. Springer, 2015, pp. 430–454. DOI: `10.1007/978-3-662-46800-5_17`. IACR: `2016/687` (pp. 3, 10, 127, 128, 130, 132, 133).

[ARS+16] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. IACR Cryptology ePrint Archive, Report 2016/687. 2016. IACR: `2016/687` (pp. 10, 127, 129, 146).

[AST+17] A. Abdelkhalek, Y. Sasaki, Y. Todo, M. Tolba, and A. M. Youssef. MILP Modeling for (Large) S-boxes to Optimize Probability of Differential Characteristics. In: IACR Transactions on Symmetric Cryptology 2017.4 (2017), pp. 99–129. DOI: `10.13154/tosc.v2017.i4.99-129` (p. 59).

[Aum11] J.-P. Aumasson. Eve's SHA3 candidate: malicious hashing. ECRYPT II Hash Workshop 2011. 2011. URL: `http://www.ecrypt.eu.org/hash2011/proceedings/hash2011_18.pdf` (pp. 212, 213).

[Ava16] R. Avanzi. A Salad of Block Ciphers. IACR Cryptology ePrint Archive, Report 2016/1171. 2016. URL: `2016/1171` (p. 17).

[Ava17] R. Avanzi. The QARMA Block Cipher Family – Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes. In: IACR Transactions on Symmetric Cryptology 2017.1 (2017), pp. 4–44. DOI: 10.13154/tosc.v2017.i1.4-44. IACR: 2016/444 (pp. 70, 98–100).

[AY15] R. AlTawy and A. M. Youssef. Watch your constants: malicious Streebog. In: IET Information Security 9.6 (2015), pp. 328–333. DOI: 10.1049/iet-ifs.2014.0540. IACR: 2014/879 (pp. 212, 213).

[BBI+15] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni. Midori: A Block Cipher for Low Energy. In: Advances in Cryptology – ASIACRYPT 2015. Ed. by T. Iwata and J. H. Cheon. Vol. 9453. LNCS. Springer, 2015, pp. 411–436. DOI: 10.1007/978-3-662-48800-3_17. IACR: 2015/1142 (pp. 71, 73, 99, 100).

[BBS05] E. Biham, A. Biryukov, and A. Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: Journal of Cryptology 18.4 (2005), pp. 291–311. DOI: 10.1007/s00145-005-0129-3 (p. 62).

[BBS99a] E. Biham, A. Biryukov, and A. Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: Advances in Cryptology – EUROCRYPT 1999. Ed. by J. Stern. Vol. 1592. LNCS. Springer, 1999, pp. 12–23. DOI: 10.1007/3-540-48910-X_2 (p. 62).

[BBS99b] E. Biham, A. Biryukov, and A. Shamir. Miss in the Middle Attacks on IDEA and Khufu. In: Fast Software Encryption – FSE 1999. Ed. by L. R. Knudsen. Vol. 1636. LNCS. Springer, 1999, pp. 124–138. DOI: 10.1007/3-540-48519-8_10 (p. 62).

[BC10] C. Boura and A. Canteaut. Zero-Sum Distinguishers for Iterated Permutations and Application to Keccak-f and Hamsi-256. In: Selected Areas in Cryptography – SAC 2010. Ed. by A. Biryukov, G. Gong, and D. R. Stinson. Vol. 6544. LNCS. Springer, 2010, pp. 1–17. DOI: 10.1007/978-3-642-19574-7_1 (p. 132).

[BCD11] C. Boura, A. Canteaut, and C. De Cannière. Higher-Order Differential Properties of Keccak and Luffa. In: Fast Software Encryption – FSE 2011. Ed. by A. Joux. Vol. 6733. LNCS. Springer, 2011, pp. 252–269. DOI: `10.1007/978-3-642-21702-9_15`. IACR: `2010/589` (p. 132).

[BCG+12] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçın. PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications. In: Advances in Cryptology – ASIACRYPT 2012. Ed. by X. Wang and K. Sako. Vol. 7658. LNCS. Springer, 2012, pp. 208–225. DOI: `10.1007/978-3-642-34961-4_14`. IACR: `2012/529` (pp. 71, 99, 100).

[BCKL17] C. Boura, A. Canteaut, L. R. Knudsen, and G. Leander. Reflection ciphers. In: Designs, Codes and Cryptography 82.1-2 (2017), pp. 3–25. DOI: `10.1007/s10623-015-0143-x` (p. 71).

[BD09] E. Biham and O. Dunkelman. The SHAvite-3 Hash Function. Submission to NIST's SHA-3 Competition (Round 2). 2009. URL: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/documents/SHAvite-3_Round2.zip` (p. 107).

[BDD+15] A. Bar-On, I. Dinur, O. Dunkelman, V. Lallemand, N. Keller, and B. Tsaban. Cryptanalysis of SP Networks with Partial Non-Linear Layers. In: Advances in Cryptology – EUROCRYPT 2015. Ed. by E. Oswald and M. Fischlin. Vol. 9056. LNCS. Springer, 2015, pp. 315–342. DOI: `10.1007/978-3-662-46800-5_13`. IACR: `2014/228` (p. 130).

[BDJR97] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption. In: Foundations of Computer Science – FOCS 1997. IEEE Computer Society, 1997, pp. 394–403. DOI: `10.1109/SFCS.1997.646128` (pp. 30–32).

[BDK02] E. Biham, O. Dunkelman, and N. Keller. Enhancing Differential-Linear Cryptanalysis. In: Advances in Cryptology – ASIACRYPT 2002. Ed. by Y. Zheng. Vol. 2501. LNCS. Springer, 2002, pp. 254–266. DOI: `10.1007/3-540-36178-2_16` (p. 63).

[BDMW10] K. A. Browning, J. F. Dillon, M. T. McQuistan, and A. J. Wolfe. An APN Permutation in Dimension Six. In: Finite Fields: Theory and Applications – Fq9. Ed. by G. McGuire, G. L. Mullen, D. Panario, and I. E. Shparlinski. Vol. 518. Contemporary Mathematics. American Mathematical Society, 2010, pp. 33–42. DOI: `10.1090/conm/518/10194` (p. 45).

[BDP+16a] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. Ketje v2. Submission to CAESAR: Competition for Authenticated Encryption. Security, Applicability, and Robustness (Round 3). 2016. URL: `http://competitions.cr.yp.to/round3/ketjev2.pdf` (p. 41).

[BDP+16b] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. Keyak v2.2. Submission to CAESAR: Competition for Authenticated Encryption. Security, Applicability, and Robustness (Round 3). 2016. URL: `http://competitions.cr.yp.to/round3/keyakv22.pdf` (p. 41).

[BDP15] A. Biryukov, P. Derbez, and L. Perrin. Differential Analysis and Meet-in-the-Middle Attack Against Round-Reduced TWINE. In: Fast Software Encryption – FSE 2015. Ed. by G. Leander. Vol. 9054. LNCS. Springer, 2015, pp. 3–27. DOI: `10.1007/978-3-662-48116-5_1`. IACR: `2015/240` (p. 86).

[BDPV07] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. Ecrypt Hash Workshop 2007. 2007. URL: `http://sponge.noekeon.org/SpongeFunctions.pdf` (pp. 5, 33, 37).

[BDPV08] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the Indifferentiability of the Sponge Construction. In: Advances in Cryptology – EUROCRYPT 2008. Ed. by N. P. Smart. Vol. 4965. LNCS. Springer, 2008, pp. 181–197. DOI: `10.1007/978-3-540-78967-3_11`. URL: `http://sponge.noekeon.org/SpongeIndifferentiability.pdf` (pp. 33, 37).

[BDPV10a] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Note on zero-sum distinguishers of Keccak-f. Public comment on the NIST Hash competition. 2010. URL: `http://keccak.noekeon.org/NoteZeroSum.pdf` (p. 63).

[BDPV10b] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge-Based Pseudo-Random Number Generators. In: Cryptographic Hardware and Embedded Systems – CHES

2010. Ed. by S. Mangard and F.-X. Standaert. Vol. 6225. LNCS. Springer, 2010, pp. 33–47. DOI: `10.1007/978-3-642-15031-9_3`. URL: `http://sponge.noekeon.org/SpongePRNG.pdf` (p. 42).

[BDPV11a] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Cryptographic sponge functions. 2011. URL: `http://sponge.noekeon.org/CSF-0.1.pdf` (p. 24).

[BDPV11b] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In: Selected Areas in Cryptography – SAC 2011. Ed. by A. Miri and S. Vaudenay. Vol. 7118. LNCS. Springer, 2011, pp. 320–337. DOI: `10.1007/978-3-642-28496-0_19`. IACR: `2011/499` (pp. 5, 33, 41, 42).

[BDPV11c] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the security of the keyed sponge construction. In: Symmetric Key Encryption Workshop – SKEW 2011. 2011. URL: `http://sponge.noekeon.org/SpongeKeyed.pdf` (p. 42).

[BDPV11d] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The Keccak reference. Submission to NIST's SHA-3 Competition (Round 3). 2011. URL: `http://keccak.noekeon.org/Keccak-reference-3.0.pdf` (pp. 37, 167, 185).

[BDPV12] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Permutation-based encryption, authentication and authenticated encryption. Workshop Records of DIAC 2012. 2012. URL: `http://www.hyperelliptic.org/djb/diac/record.pdf` (p. 42).

[Ber08] D. J. Bernstein. ChaCha, a variant of Salsa20. 2008. URL: `http://cr.yp.to/chacha/chacha-20080128.pdf` (p. 20).

[BG11] C. Blondeau and B. Gérard. Multiple Differential Cryptanalysis: Theory and Practice. In: Fast Software Encryption – FSE 2011. Ed. by A. Joux. Vol. 6733. LNCS. Springer, 2011, pp. 35–54. DOI: `10.1007/978-3-642-21702-9_3`. IACR: `2011/115` (p. 60).

[BGN12] C. Blondeau, B. Gérard, and K. Nyberg. Multiple Differential Cryptanalysis Using LLR and χ^2 Statistics. In: Security and Cryptography for Networks – SCN 2012. Ed. by I. Visconti and R. D. Prisco. Vol. 7485. LNCS. Springer, 2012, pp. 343–360. DOI: 10.1007/978-3-642-32928-9_19. IACR: 2012/360 (p. 60).

[BHH+15] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn. SPHINCS: Practical Stateless Hash-Based Signatures. In: Advances in Cryptology – EUROCRYPT 2015. Ed. by E. Oswald and M. Fischlin. Vol. 9056. LNCS. Springer, 2015, pp. 368–397. DOI: 10.1007/978-3-662-46800-5_15. IACR: 2014/795 (p. 110).

[Bih93] E. Biham. New Types of Cryptanalytic Attacks Using related Keys (Extended Abstract). In: Advances in Cryptology – EUROCRYPT 1993. Ed. by T. Helleseth. Vol. 765. LNCS. Springer, 1993, pp. 398–409. DOI: 10.1007/3-540-48285-7_34 (pp. 52, 149).

[Bih94a] E. Biham. New Types of Cryptanalytic Attacks Using Related Keys. In: Journal of Cryptology 7.4 (1994), pp. 229–246. DOI: 10.1007/BF00203965 (p. 52).

[Bih94b] E. Biham. On Matsui's Linear Cryptanalysis. In: Advances in Cryptology – EUROCRYPT 1994. Ed. by A. D. Santis. Vol. 950. LNCS. Springer, 1994, pp. 341–355. DOI: 10.1007/BFb0053449 (p. 63).

[BJK+16] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In: Advances in Cryptology – CRYPTO 2016. Ed. by M. Robshaw and J. Katz. Vol. 9815. LNCS. Springer, 2016, pp. 123–153. DOI: 10.1007/978-3-662-53008-5_5. IACR: 2016/660 (pp. 9, 69–71, 73, 75, 99, 100).

[BK03] M. Bellare and T. Kohno. A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications. In: Advances in Cryptology – EUROCRYPT 2003. Ed. by E. Biham. Vol. 2656. LNCS. Springer, 2003, pp. 491–506. DOI: 10.1007/3-540-39200-9_31 (pp. 149, 161).

[BK12] E. Barker and J. Kelsey. NIST SP 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised). National Institute of

Standards and Technology (NIST) Special Publication (SP). 2012. DOI: 10.6028/NIST.SP.800-90A (p. 213).

[BK93] M. Buro and H. Kleine Büning. Report on a SAT Competition. In: Bulletin of the EATCS 49 (1993), pp. 143–151. URL: https://skatgame.net/mburo/ps/satreport.pdf (p. 188).

[BKL+12] A. Bogdanov, L. R. Knudsen, G. Leander, F.-X. Standaert, J. P. Steinberger, and E. Tischhauser. Key-Alternating Ciphers in a Provable Setting: Encryption Using a Small Number of Public Permutations – (Extended Abstract). In: Advances in Cryptology – EUROCRYPT 2012. Ed. by D. Pointcheval and T. Johansson. Vol. 7237. LNCS. Springer, 2012, pp. 45–62. DOI: 10.1007/978-3-642-29011-4_5 (p. 147).

[BKN09] A. Biryukov, D. Khovratovich, and I. Nikolić. Distinguisher and Related-Key Attack on the Full AES-256. In: Advances in Cryptology – CRYPTO 2009. Ed. by S. Halevi. Vol. 5677. LNCS. Springer, 2009, pp. 231–249. DOI: 10.1007/978-3-642-03356-8_14 (p. 149).

[BKR94] M. Bellare, J. Kilian, and P. Rogaway. The Security of Cipher Block Chaining. In: Advances in Cryptology – CRYPTO 1994. Ed. by Y. Desmedt. Vol. 839. LNCS. Springer, 1994, pp. 341–358. DOI: 10.1007/3-540-48658-5_32. URL: https://cseweb.ucsd.edu/~mihir/papers/cbc.pdf (p. 38).

[BL11] D. J. Bernstein and T. Lange. eBASH: ECRYPT Benchmarking of All Submitted Hashes. 2011. URL: http://bench.cr.yp.to/ebash.html (p. 168).

[BL16] K. Bhargavan and G. Leurent. On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN. In: Computer and Communications Security – CCS 2016. Ed. by E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi. ACM, 2016, pp. 456–467. DOI: 10.1145/2976749.2978423. IACR: 2016/798 (p. 3).

[Bla06] J. Black. The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function. In: Fast Software Encryption – FSE 2006. Ed. by M. J. B. Robshaw. Vol. 4047. LNCS. Springer, 2006, pp. 328–340. DOI: 10.1007/11799313_21. IACR: 2005/210 (p. 23).

[BLMN11] A. Biryukov, M. Lamberger, F. Mendel, and I. Nikolić. Second-Order Differential Collisions for Reduced SHA-256. In: Advances in Cryptology – ASIACRYPT 2011. Ed. by D. H. Lee and X. Wang. Vol. 7073. LNCS. Springer, 2011, pp. 270–287. DOI: 10.1007/978-3-642-25385-0_15. IACR: 2011/037 (p. 174).

[BLN14] C. Blondeau, G. Leander, and K. Nyberg. Differential-Linear Cryptanalysis Revisited. In: Fast Software Encryption – FSE 2014. Ed. by C. Cid and C. Rechberger. Vol. 8540. LNCS. Springer, 2014, pp. 411–430. DOI: 10.1007/978-3-662-46706-0_21 (p. 63).

[BLN17] C. Blondeau, G. Leander, and K. Nyberg. Differential-Linear Cryptanalysis Revisited. In: Journal of Cryptology 30.3 (2017), pp. 859–888. DOI: 10.1007/s00145-016-9237-5 (p. 63).

[Blo17] C. Blondeau. Accurate Estimate of the Advantage of Impossible Differential Attacks. In: IACR Transactions on Symmetric Cryptology 2017.3 (2017), pp. 169–191. DOI: 10.13154/tosc.v2017.i3.169-191 (p. 62).

[BN00] M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In: Advances in Cryptology – ASIACRYPT 2000. Ed. by T. Okamoto. Vol. 1976. LNCS. Springer, 2000, pp. 531–545. DOI: 10.1007/3-540-44448-3_41. IACR: 2000/025 (pp. 39, 41).

[BN08] M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In: Journal of Cryptology 21.4 (2008), pp. 469–491. DOI: 10.1007/s00145-008-9026-x (p. 41).

[BN10] A. Biryukov and I. Nikolić. Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In: Advances in Cryptology – EUROCRYPT 2010. Ed. by H. Gilbert. Vol. 6110. LNCS. Springer, 2010, pp. 322–344. DOI: 10.1007/978-3-642-13190-5_17. IACR: 2010/248 (pp. 52, 59).

[BNS14] C. Boura, M. Naya-Plasencia, and V. Suder. Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon. In: Advances in Cryptology – ASIACRYPT 2014. Ed. by P. Sarkar and T. Iwata. Vol. 8873. LNCS. Springer, 2014, pp. 179–199. DOI: `10.1007/978-3-662-45611-8_10`. IACR: `2014/699` (p. 62).

[Boe88] B. den Boer. Cryptanalysis of F.E.A.L. In: Advances in Cryptology – EUROCRYPT '88. Ed. by C. G. Günther. Vol. 330. LNCS. Springer, 1988, pp. 293–299. DOI: `10.1007/3-540-45961-8_27` (p. 44).

[Bra16] D. Brash. ARMv8-A architecture – 2016 additions. ARM Community. 2016. URL: `https://community.arm.com/processors/b/blog/posts/armv8-a-architecture-2016-additions` (p. 98).

[Bre80] R. P. Brent. An improved Monte Carlo factorization algorithm. In: BIT Numerical Mathematics 20.2 (1980), pp. 176–184. DOI: `10.1007/BF01933190` (p. 53).

[BS90] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In: Advances in Cryptology – CRYPTO 1990. Ed. by A. Menezes and S. A. Vanstone. Vol. 537. LNCS. Springer, 1990, pp. 2–21. DOI: `10.1007/3-540-38424-3_1` (pp. 43–45, 47, 49–51, 53, 67).

[BS91] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In: Journal of Cryptology 4.1 (1991), pp. 3–72. DOI: `10.1007/BF00630563` (pp. 43, 51, 60, 67, 166).

[BS92] E. Biham and A. Shamir. Differential Cryptanalysis of the Full 16-Round DES. In: Advances in Cryptology – CRYPTO 1992. Ed. by E. F. Brickell. Vol. 740. LNCS. Springer, 1992, pp. 487–496. DOI: `10.1007/3-540-48071-4_34` (pp. 43, 51).

[BS93] E. Biham and A. Shamir. Differential Cryptanalysis of the Data Encryption Standard. Springer, 1993. ISBN: 978-1-4613-9316-0. DOI: `10.1007/978-1-4613-9314-6` (pp. 43, 51, 67).

[BS97] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In: Advances in Cryptology – CRYPTO 1997. Ed. by B. S. K. Jr. Vol. 1294. LNCS. Springer, 1997, pp. 513–525. DOI: `10.1007/BFb0052259` (p. 63).

[BU02] J. Black and H. Urtubia. Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption. In: USENIX Security Symposium 2002. Ed. by D. Boneh. USENIX, 2002, pp. 327–338. URL: http://www.usenix.org/publications/library/proceedings/sec02/black.html (p. 39).

[BV14] A. Biryukov and V. Velichkov. Automatic Search for Differential Trails in ARX Ciphers. In: Topics in Cryptology – CT-RSA 2014. Ed. by J. Benaloh. Vol. 8366. LNCS. Springer, 2014, pp. 227–250. DOI: 10.1007/978-3-319-04852-9_12. IACR: 2013/853 (p. 59).

[BVL16] A. Biryukov, V. Velichkov, and Y. Le Corre. Automatic Search for the Best Trails in ARX: Application to Block Cipher Speck. In: Fast Software Encryption – FSE 2016. Ed. by T. Peyrin. Vol. 9783. LNCS. Springer, 2016, pp. 289–310. DOI: 10.1007/978-3-662-52993-5_15. IACR: 2016/409 (p. 59).

[BW00] A. Biryukov and D. Wagner. Advanced Slide Attacks. In: Advances in Cryptology – EUROCRYPT 2000. Ed. by B. Preneel. Vol. 1807. LNCS. Springer, 2000, pp. 589–606. DOI: 10.1007/3-540-45539-6_41 (p. 147).

[CAE13] CAESAR Committee. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. Call for Submissions. 2013. URL: http://competitions.cr.yp.to/caesar-call.html (pp. 39, 148, 150).

[CAE16] CAESAR Committee. CAESAR submissions: Third-round candidates. 2016. URL: http://competitions.cr.yp.to/caesar-submissions.html (p. 41).

[CCF+16] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey. Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. In: Fast Software Encryption – FSE 2016. Ed. by T. Peyrin. Vol. 9783. LNCS. Springer, 2016, pp. 313–333. DOI: 10.1007/978-3-662-52993-5_16. IACR: 2015/113 (p. 129).

[CDMP05] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In: Advances in Cryptology – CRYPTO 2005. Ed. by V.

Shoup. Vol. 3621. LNCS. Springer, 2005, pp. 430–448. DOI: 10.1007/11535218_26 (pp. 36, 168, 173).

[CFG+14] A. Canteaut, T. Fuhr, H. Gilbert, M. Naya-Plasencia, and J.-R. Reinhard. Multiple Differential Cryptanalysis of Round-Reduced PRINCE. In: Fast Software Encryption – FSE 2014. Ed. by C. Cid and C. Rechberger. Vol. 8540. LNCS. Springer, 2014, pp. 591–610. DOI: 10.1007/978-3-662-46706-0_30. IACR: 2014/089 (p. 86).

[CGH04] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In: Journal of the ACM 51.4 (2004), pp. 557–594. DOI: 10.1145/1008731.1008734. IACR: 1998/11 (p. 35).

[CHP+17] C. Cid, T. Huang, T. Peyrin, Y. Sasaki, and L. Song. A Security Analysis of Deoxys and its Internal Tweakable Block Ciphers. In: IACR Transactions on Symmetric Cryptology 2017.3 (2017), pp. 73–107. DOI: 10.13154/tosc.v2017.i3.73-107. IACR: 2017/693 (p. 71).

[CJ98] F. Chabaud and A. Joux. Differential Collisions in SHA-0. In: Advances in Cryptology – CRYPTO 1998. Ed. by H. Krawczyk. Vol. 1462. LNCS. Springer, 1998, pp. 56–71. DOI: 10.1007/BFb0055720 (pp. 215, 217).

[Coc07] M. Cochran. Notes on the Wang et al. 2^{63} SHA-1 Differential Path. IACR Cryptology ePrint Archive, Report 2007/474. 2007. IACR: 2007/474 (p. 215).

[Coo71] S. A. Cook. The Complexity of Theorem-Proving Procedures. In: Symposium on Theory of Computing – STOC 1971. Ed. by M. A. Harrison, R. B. Banerji, and J. D. Ullman. ACM, 1971, pp. 151–158. DOI: 10.1145/800157.805047 (p. 57).

[Cop94] D. Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. In: IBM Journal of Research and Development 38.3 (1994), pp. 243–250. DOI: 10.1147/rd.383.0243 (p. 43).

[CS15] B. Cogliati and Y. Seurin. On the Provable Security of the Iterated Even-Mansour Cipher Against Related-Key and Chosen-Key Attacks. In: Advances in Cryptology – EUROCRYPT 2015. Ed. by E. Oswald and M. Fischlin.

Vol. 9056. LNCS. Springer, 2015, pp. 584–613. DOI: 10.1007/978-3-662-46800-5_23. IACR: 2015/069 (p. 149).

[CV94] F. Chabaud and S. Vaudenay. Links Between Differential and Linear Cryptanalysis. In: Advances in Cryptology – EUROCRYPT 1994. Ed. by A. D. Santis. Vol. 950. LNCS. Springer, 1994, pp. 356–365. DOI: 10.1007/BFb0053450 (p. 63).

[CW77] L. Carter and M. N. Wegman. Universal Classes of Hash Functions (Extended Abstract). In: Symposium on Theory of Computing – STOC 1977. Ed. by J. E. Hopcroft, E. P. Friedman, and M. A. Harrison. ACM, 1977, pp. 106–112. DOI: 10.1145/800105.803400 (p. 38).

[Dae91] J. Daemen. Limitations of the Even-Mansour Construction. In: Advances in Cryptology – ASIACRYPT 1991. Ed. by H. Imai, R. L. Rivest, and T. Matsumoto. Vol. 739. LNCS. Springer, 1991, pp. 495–498. DOI: 10.1007/3-540-57332-1_46 (pp. 28, 63, 147).

[Dae95] J. Daemen. Cipher and Hash Function Design. Strategies based on linear and differential cryptanalysis. PhD thesis. Katholieke Universiteit Leuven, 1995. URL: https://www.esat.kuleuven.be/cosic/publications/thesis-6.pdf (pp. 20, 56, 67, 181).

[Dam89] I. Damgård. A Design Principle for Hash Functions. In: Advances in Cryptology – CRYPTO 1989. Ed. by G. Brassard. Vol. 435. LNCS. Springer, 1989, pp. 416–427. DOI: 10.1007/0-387-34805-0_39 (pp. 35, 170, 173).

[Dan12] Q. H. Dang. NIST FIPS PUB 180-4: Secure Hash Standard (SHS). National Institute of Standards and Technology (NIST) Federal Information Processing Standards (FIPS) Publication. 2012. DOI: 10.6028/NIST.FIPS.180-4 (pp. 19, 20, 168, 170–172, 214).

[DB09] O. Dunkelman and E. Biham. The SHAvite-3 – A New Hash Function. In: Symmetric Cryptography. Ed. by H. Handschuh, S. Lucks, B. Preneel, and P. Rogaway. Vol. 09031. Dagstuhl Seminar Proceedings. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2009. URL: http://drops.dagstuhl.de/opus/volltexte/2009/1947/ (p. 107).

[DBN+01] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, and J. F. Dray Jr. NIST FIPS PUB 197: Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST) Federal Information Processing Standards (FIPS) Publication. 2001. DOI: `10.6028/NIST.FIPS.197`. URL: `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf` (p. 21).

[Dea99] R. D. Dean. Formal Aspects of Mobile Code Security. PhD thesis. Princeton University, 1999. URL: `http://sip.cs.princeton.edu/pub/ddean-thesis.pdf` (p. 36).

[DEKM17] C. Dobraunig, M. Eichlseder, D. Kales, and F. Mendel. Practical Key-Recovery Attack on MANTIS5. In: IACR Transactions on Symmetric Cryptology 2016.2 (2017), pp. 248–260. DOI: `10.13154/tosc.v2016.i2.248-260`. IACR: `2016/754` (pp. 69, 83, 84, 96, 97).

[DEM15a] C. Dobraunig, M. Eichlseder, and F. Mendel. Analysis of SHA-512/224 and SHA-512/256. In: Advances in Cryptology – ASIACRYPT 2015. Ed. by T. Iwata and J. H. Cheon. Vol. 9453. LNCS. Springer, 2015, pp. 612–630. DOI: `10.1007/978-3-662-48800-3_25`. IACR: `2016/374` (pp. 165, 169, 174, 194).

[DEM15b] C. Dobraunig, M. Eichlseder, and F. Mendel. Forgery Attacks on Round-Reduced ICEPOLE-128. In: Selected Areas in Cryptography – SAC 2015. Ed. by O. Dunkelman and L. Keliher. Vol. 9566. LNCS. Springer, 2015, pp. 479–492. DOI: `10.1007/978-3-319-31301-6_27`. IACR: `2015/392` (p. 167).

[DEM15c] C. Dobraunig, M. Eichlseder, and F. Mendel. Higher-Order Cryptanalysis of LowMC. In: Information Security and Cryptology – ICISC 2015. Ed. by S. Kwon and A. Yun. Vol. 9558. LNCS. Springer, 2015, pp. 87–101. DOI: `10.1007/978-3-319-30840-1_6`. IACR: `2015/407` (p. 127).

[DEM15d] C. Dobraunig, M. Eichlseder, and F. Mendel. Related-Key Forgeries for Prøst-OTR. In: Fast Software Encryption – FSE 2015. Ed. by G. Leander. Vol. 9054. LNCS. Springer, 2015, pp. 282–296. DOI: `10.1007/978-3-662-48116-5_14`. IACR: `2015/091` (p. 147).

[DEM15e] C. Dobraunig, M. Eichlseder, and F. Mendel. Security Evaluation of SHA-224, SHA-512/224, and SHA-512/256. Tech. Report CRYPTREC. 2015. URL: `http://www.cryptrec.go.jp/estimation/techrep_id2401.pdf` (pp. 165, 169).

[DEM16a] C. Dobraunig, M. Eichlseder, and F. Mendel. Cryptanalysis of Simpira v1. In: Selected Areas in Cryptography – SAC 2016. Ed. by R. Avanzi and H. M. Heys. Vol. 10532. LNCS. Springer, 2016, pp. 284–298. DOI: `10.1007/978-3-319-69453-5_16`. IACR: `2016/244` (p. 105).

[DEM16b] C. Dobraunig, M. Eichlseder, and F. Mendel. Square Attack on 7-Round Kiasu-BC. In: Applied Cryptography and Network Security – ACNS 2016. Ed. by M. Manulis, A.-R. Sadeghi, and S. Schneider. Vol. 9696. LNCS. Springer, 2016, pp. 500–517. DOI: `10.1007/978-3-319-39555-5_27`. IACR: `2016/326` (p. 70).

[DEMS15] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. Cryptanalysis of Ascon. In: Topics in Cryptology – CT-RSA 2015. Ed. by K. Nyberg. Vol. 9048. LNCS. Springer, 2015, pp. 371–387. DOI: `10.1007/978-3-319-16715-2_20`. IACR: `2015/030` (p. 167).

[Der16] P. Derbez. Note on Impossible Differential Attacks. In: Fast Software Encryption – FSE 2016. Ed. by T. Peyrin. Vol. 9783. LNCS. Springer, 2016, pp. 416–427. DOI: `10.1007/978-3-662-52993-5_21`. IACR: `2016/349` (p. 62).

[DGG+15] Y. Dodis, C. Ganesh, A. Golovnev, A. Juels, and T. Ristenpart. A Formal Treatment of Backdoored Pseudorandom Generators. In: Advances in Cryptology – EUROCRYPT 2015. Ed. by E. Oswald and M. Fischlin. Vol. 9056. LNCS. Springer, 2015, pp. 101–126. DOI: `10.1007/978-3-662-46800-5_5`. IACR: `2016/306` (p. 213).

[DGV94a] J. Daemen, R. Govaerts, and J. Vandewalle. Correlation Matrices. In: Fast Software Encryption – FSE 1994. Ed. by B. Preneel. Vol. 1008. LNCS. Springer, 1994, pp. 275–285. DOI: `10.1007/3-540-60590-8_21` (p. 63).

[DGV94b] J. Daemen, R. Govaerts, and J. Vandewalle. Invertible Shift-invariant Transformations on Binary Arrays. In: Journal of Applied Mathematics and Computation 62.2–3 (1994), pp. 259–277. DOI: `10.1016/0096-3003(94)90087-6` (p. 181).

[DH79] W. Diffie and M. E. Hellman. Privacy and authentication: An introduction to cryptography. In: Proceedings of the IEEE 67.3 (1979), pp. 397–427. DOI: 10.1109/PROC.1979.11256 (p. 32).

[DKR97] J. Daemen, L. R. Knudsen, and V. Rijmen. The Block Cipher Square. In: Fast Software Encryption – FSE 1997. Ed. by E. Biham. Vol. 1267. LNCS. Springer, 1997, pp. 149–165. DOI: 10.1007/BFb0052343 (p. 63).

[DKS12] O. Dunkelman, N. Keller, and A. Shamir. Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In: Advances in Cryptology – EUROCRYPT 2012. Ed. by D. Pointcheval and T. Johansson. Vol. 7237. LNCS. Springer, 2012, pp. 336–354. DOI: 10.1007/978-3-642-29011-4_21 (pp. 28, 147, 148, 151).

[DL05] M. Daum and S. Lucks. Hash Collisions (The Poisoned Message Attack). CRYPTO 2005 rump session. 2005. URL: http://th.informatik.uni-mannheim.de/people/lucks/HashCollisions/ (p. 220).

[DLL62] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. In: Commununications of the ACM 5.7 (1962), pp. 394–397. DOI: 10.1145/368273.368557. URL: http://doi.acm.org/10.1145/368273.368557 (pp. 57, 167, 187).

[DLMW15] I. Dinur, Y. Liu, W. Meier, and Q. Wang. Optimized Interpolation Attacks on LowMC. In: Advances in Cryptology – ASIACRYPT 2015. Ed. by T. Iwata and J. H. Cheon. Vol. 9453. LNCS. Springer, 2015, pp. 535–560. DOI: 10.1007/978-3-662-48800-3_22. IACR: 2015/418 (pp. 127, 129, 130, 145, 146).

[DLR16] S. Duval, V. Lallemand, and Y. Rotella. Cryptanalysis of the FLIP Family of Stream Ciphers. In: Advances in Cryptology – CRYPTO 2016. Ed. by M. Robshaw and J. Katz. Vol. 9814. LNCS. Springer, 2016, pp. 457–475. DOI: 10.1007/978-3-662-53018-4_17. IACR: 2016/271 (p. 129).

[DMR07] C. De Cannière, F. Mendel, and C. Rechberger. Collisions for 70-Step SHA-1: On the Full Cost of Collision Search. In: Selected Areas in Cryptography – SAC 2007. Ed. by C. M. Adams, A. Miri, and M. J. Wiener. Vol. 4876. LNCS.

Springer, 2007, pp. 56–73. DOI: 10.1007/978-3-540-77360-3_4 (p. 215).

[Dob96] H. Dobbertin. Cryptanalysis of MD4. In: Fast Software Encryption – FSE 1996. Ed. by D. Gollmann. Vol. 1039. LNCS. Springer, 1996, pp. 53–69. DOI: 10.1007/3-540-60865-6_43 (pp. 44, 61, 178).

[Dob98] H. Dobbertin. Cryptanalysis of MD4. In: Journal of Cryptology 11.4 (1998), pp. 253–271. DOI: 10.1007/s001459900047 (pp. 61, 178).

[DP07] J. P. Degabriele and K. G. Paterson. Attacking the IPsec Standards in Encryption-only Configurations. In: Security and Privacy – S&P 2007. IEEE Computer Society, 2007, pp. 335–349. DOI: 10.1109/SP.2007.8. IACR: 2007/125 (p. 39).

[DR01] J. Daemen and V. Rijmen. The Wide Trail Design Strategy. In: Cryptography and Coding – IMACC 2001. Ed. by B. Honary. Vol. 2260. LNCS. Springer, 2001, pp. 222–238. DOI: 10.1007/3-540-45325-3_20 (pp. 55, 56, 67).

[DR02] J. Daemen and V. Rijmen. The Design of Rijndael: AES – The Advanced Encryption Standard. Information Security and Cryptography. Springer, 2002. ISBN: 3-540-42580-2. DOI: 10.1007/978-3-662-04722-4 (pp. 17, 21).

[DR06] C. De Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In: Advances in Cryptology – ASIACRYPT 2006. Ed. by X. Lai and K. Chen. Vol. 4284. LNCS. Springer, 2006, pp. 1–20. DOI: 10.1007/11935230_1 (pp. 11, 59, 61, 167, 178–180, 215).

[DR07] J. Daemen and V. Rijmen. Probability distributions of correlation and differentials in block ciphers. In: Journal of Mathematical Cryptology 1.3 (2007), pp. 221–242. DOI: 10.1515/JMC.2007.011. IACR: 2005/212 (pp. 45, 47, 48).

[DR98] J. Daemen and V. Rijmen. The Block Cipher Rijndael. In: Smart Card Research and Applications – CARDIS 1998. Ed. by J.-J. Quisquater and B. Schneier. Vol. 1820. LNCS. Springer, 1998, pp. 277–284. DOI: 10.1007/10721064_26 (pp. 21, 56).

[DS08] H. Demirci and A. A. Selçuk. A Meet-in-the-Middle Attack on 8-Round AES. In: Fast Software Encryption – FSE 2008. Ed. by K. Nyberg. Vol. 5086. LNCS. Springer, 2008, pp. 116–126. DOI: 10.1007/978-3-540-71039-4_7 (p. 63).

[DS09] I. Dinur and A. Shamir. Cube Attacks on Tweakable Black Box Polynomials. In: Advances in Cryptology – EUROCRYPT 2009. Ed. by A. Joux. Vol. 5479. LNCS. Springer, 2009, pp. 278–299. DOI: 10.1007/978-3-642-01001-9_16. IACR: 2008/385 (p. 63).

[DV12] J. Daemen and G. Van Assche. Differential Propagation Analysis of Keccak. In: Fast Software Encryption – FSE 2012. Ed. by A. Canteaut. Vol. 7549. LNCS. Springer, 2012, pp. 422–441. DOI: 10.1007/978-3-642-34047-5_24. IACR: 2012/163 (p. 59).

[Dwo01] M. J. Dworkin. NIST SP 800-38A: Recommendation for Block Cipher Modes of Operation: Methods and Techniques. National Institute of Standards and Technology (NIST) Special Publication (SP). 2001. DOI: 10.6028/NIST.SP.800-38A (p. 32).

[Dwo04] M. J. Dworkin. NIST SP 800-38C: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. National Institute of Standards and Technology (NIST) Special Publication (SP). 2004. DOI: 10.6028/NIST.SP.800-38C (pp. 41, 160).

[Dwo07] M. J. Dworkin. NIST SP 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. National Institute of Standards and Technology (NIST) Special Publication (SP). 2007. DOI: 10.6028/NIST.SP.800-38D (pp. 38, 41).

[Dwo10] M. J. Dworkin. NIST SP 800-38A Addendum: Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode. National Institute of Standards and Technology (NIST) Special Publication (SP). 2010. DOI: 10.6028/NIST.SP.800-38A-Add (p. 32).

[Dwo15] M. J. Dworkin. NIST FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. National Institute of Standards and Technology (NIST) Federal Information Processing Standards (FIPS) Publication.

2015. DOI: https://dx.doi.org/10.6028/NIST.FIPS.202 (pp. 37, 167).

[EK17] M. Eichlseder and D. Kales. Clustering Related-Tweak Characteristics: Application to MANTIS-6. IACR Cryptology ePrint Archive, Report 2017/1136. 2017. IACR: 2017/1136 (pp. 69, 82–84, 86, 91, 97).

[EM91] S. Even and Y. Mansour. A Construction of a Cipher From a Single Pseudorandom Permutation. In: Advances in Cryptology – ASIACRYPT 1991. Ed. by H. Imai, R. L. Rivest, and T. Matsumoto. Vol. 739. LNCS. Springer, 1991, pp. 210–224. DOI: 10.1007/3-540-57332-1_17 (pp. 28, 147).

[EM97] S. Even and Y. Mansour. A Construction of a Cipher from a Single Pseudorandom Permutation. In: Journal of Cryptology 10.3 (1997), pp. 151–162. DOI: 10.1007/s001459900025 (pp. 28, 147).

[EMN+13] M. Eichlseder, F. Mendel, T. Nad, V. Rijmen, and M. Schläffer. Linear Propagation in Efficient Guess-and-Determine Attacks. In: International Workshop on Coding and Cryptography – WCC 2013, Preproceedings. Ed. by L. Budaghyan, T. Helleseth, and M. G. Parker. 2013, pp. 193–202. ISBN: 978-82-308-2269-2. URL: http://www.selmer.uib.no/WCC2013/ (pp. 165, 169).

[EMS14] M. Eichlseder, F. Mendel, and M. Schläffer. Branching Heuristics in Differential Collision Search with Applications to SHA-512. In: Fast Software Encryption – FSE 2014. Ed. by C. Cid and C. Rechberger. Vol. 8540. LNCS. Springer, 2014, pp. 473–488. DOI: 10.1007/978-3-662-46706-0_24. IACR: 2014/302 (pp. 165, 169, 174, 193–195, 200).

[EMST76] W. F. Ehrsam, C. H. W. Meyer, J. L. Smith, and W. L. Tuchman. Message verification and transmission error detection by block chaining. US Patent 4074066. 1976 (p. 31).

[ES03] N. Eén and N. Sörensson. An Extensible SAT-solver. In: Theory and Applications of Satisfiability Testing – SAT 2003. Ed. by E. Giunchiglia and A. Tacchella. Vol. 2919. LNCS. Springer, 2003, pp. 502–518. DOI: 10.1007/978-3-540-24605-3_37 (p. 188).

[Fei70] H. Feistel. Cryptographic Coding for Data-Bank Privacy. Tech. rep. IBM Research Report RC 2827. IBM Corp., 1970 (p. 26).

[Fei73] H. Feistel. Cryptography and Computer Privacy. In: Scientific American 228.5 (1973), pp. 15–23. DOI: `10.1038/scientificamerican0573-15` (pp. 20, 26).

[FLLW16] C. Forler, E. List, S. Lucks, and J. Wenzel. Efficient Beyond-Birthday-Bound-Secure Deterministic Authenticated Encryption with Minimal Stretch. In: Information Security and Privacy – ACISP 2016. Ed. by J. K. Liu and R. Steinfeld. Vol. 9723. LNCS. Springer, 2016, pp. 317–332. DOI: `10.1007/978-3-319-40367-0_20`. IACR: `2016/395` (p. 106).

[FNS75] H. Feistel, W. A. Notz, and J. L. Smith. Some Cryptographic Techniques for Machine-to-Machine Data Communications. In: Proceedings of the IEEE 63.11 (1975), pp. 1545–1554. DOI: `10.1109/PROC.1975.10005` (p. 26).

[FO89] P. Flajolet and A. M. Odlyzko. Random Mapping Statistics. In: Advances in Cryptology – EUROCRYPT '89. Ed. by J.-J. Quisquater and J. Vandewalle. Vol. 434. LNCS. Springer, 1989, pp. 329–354. DOI: `10.1007/3-540-46885-4_34` (p. 25).

[FP15] P. Farshim and G. Procter. The Related-Key Security of Iterated Even-Mansour Ciphers. In: Fast Software Encryption – FSE 2015. Ed. by G. Leander. Vol. 9054. LNCS. Springer, 2015, pp. 342–363. DOI: `10.1007/978-3-662-48116-5_17`. IACR: `2014/953` (p. 149).

[Fre95] J. W. Freeman. Improvements to Propositional Satisfiability Search Algorithms. PhD thesis. Philadelphia: Departement of computer and Information science, University of Pennsylvania, 1995. URL: `ftp://ftp.cis.upenn.edu/pub/freeman/thesis.ps.gz` (p. 188).

[FWG+16] K. Fu, M. Wang, Y. Guo, S. Sun, and L. Hu. MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck. In: Fast Software Encryption – FSE 2016. Ed. by T. Peyrin. Vol. 9783. LNCS. Springer, 2016, pp. 268–288. DOI: `10.1007/978-3-662-52993-5_14`. IACR: `2016/407` (p. 59).

[GB08] S. Goldwasser and M. Bellare. Lecture Notes on Cryptography. Lecture Notes. 2008. URL: `http://cseweb.ucsd.edu/~mihir/papers/gb.pdf` (p. 36).

[Gég27] J. J. Gégalkine. Sur le calcul des propositions dans la logique symbolique. In: Matematicheskii Sbornik 34.1 (1927), pp. 9–28. URL: `http://mi.mathnet.ru/msb7433` (p. 20).

[GGNS13] B. Gérard, V. Grosso, M. Naya-Plasencia, and F.-X. Standaert. Block Ciphers That Are Easier to Mask: How Far Can We Go? In: Cryptographic Hardware and Embedded Systems – CHES 2013. Ed. by G. Bertoni and J.-S. Coron. Vol. 8086. LNCS. Springer, 2013, pp. 383–399. DOI: `10.1007/978-3-642-40349-1_22`. IACR: `2013/369` (p. 129).

[GH03] H. Gilbert and H. Handschuh. Security Analysis of SHA-256 and Sisters. In: Selected Areas in Cryptography – SAC 2003. Ed. by M. Matsui and R. J. Zuccherato. Vol. 3006. LNCS. Springer, 2003, pp. 175–193. DOI: `10.1007/978-3-540-24654-1_13` (p. 173).

[GJN+16] J. Guo, J. Jean, I. Nikolić, K. Qiao, Y. Sasaki, and S. Sim. Invariant Subspace Attack Against Midori64 and The Resistance Criteria for S-box Designs. In: IACR Transactions on Symmetric Cryptology 2016.1 (2016), pp. 33–56. DOI: `10.13154/tosc.v2016.i1.33-56`. IACR: `2016/973` (p. 99).

[GJW11] S. Gueron, S. Johnson, and J. Walker. SHA-512/256. In: Information Technology: New Generations – ITNG 2011. Ed. by S. Latifi. IEEE Computer Society, 2011, pp. 354–358. DOI: `10.1109/ITNG.2011.69`. IACR: `2010/548` (p. 168).

[GLRW10] J. Guo, S. Ling, C. Rechberger, and H. Wang. Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In: Advances in Cryptology – ASIACRYPT 2010. Ed. by M. Abe. Vol. 6477. LNCS. Springer, 2010, pp. 56–75. DOI: `10.1007/978-3-642-17373-8_4`. IACR: `2010/16` (p. 173).

[GM16a] S. Gueron and N. Mouha. Simpira: A Family of Efficient Permutations Using the AES Round Function. IACR Cryptology ePrint Archive, Report 2016/122. 2016. IACR: `2016/122/20160214:005409` (pp. 9, 105, 106, 108, 111).

[GM16b] S. Gueron and N. Mouha. Simpira v2: A Family of Efficient Permutations Using the AES Round Function. In: Advances in Cryptology – ASIACRYPT 2016. Ed. by J. H. Cheon and T. Takagi. Vol. 10031. LNCS. Springer, 2016, pp. 95–125. DOI: `10.1007/978-3-662-53887-6_4`. IACR: `2016/122` (pp. 9, 105, 107, 124, 126).

[GM82] S. Goldwasser and S. Micali. Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information. In: Symposium on Theory of Computing – STOC 1982. Ed. by H. R. Lewis, B. B. Simons, W. A. Burkhard, and L. H. Landweber. ACM, 1982, pp. 365–377. DOI: `10.1145/800070.802212` (p. 30).

[GM84] S. Goldwasser and S. Micali. Probabilistic Encryption. In: Journal of Computer and System Sciences 28.2 (1984), pp. 270–299. DOI: `10.1016/0022-0000(84)90070-9` (pp. 30, 31).

[GN02] E. I. Goldberg and Y. Novikov. BerkMin: A Fast and Robust Sat-Solver. In: Design, Automation, and Test in Europe – DATE 2002. IEEE Computer Society, 2002, pp. 142–149. DOI: `10.1109/DATE.2002.998262` (p. 189).

[GR04] C. Gentry and Z. Ramzan. Eliminating Random Permutation Oracles in the Even-Mansour Cipher. In: Advances in Cryptology – ASIACRYPT 2004. Ed. by P. J. Lee. Vol. 3329. LNCS. Springer, 2004, pp. 32–47. DOI: `10.1007/978-3-540-30539-2_3` (p. 147).

[Gre10] E. A. Grechnikov. Collisions for 72-step and 73-step SHA-1: Improvements in the Method of Characteristics. IACR Cryptology ePrint Archive, Report 2010/413. 2010. IACR: `2010/413` (p. 215).

[GRR17] L. Grassi, C. Rechberger, and S. Rønjom. A New Structural-Differential Property of 5-Round AES. In: Advances in Cryptology – EUROCRYPT 2017. Ed. by J.-S. Coron and J. B. Nielsen. Vol. 10211. LNCS. 2017, pp. 289–317. DOI: `10.1007/978-3-319-56614-6_10`. IACR: `2017/118` (p. 3).

[HB03] M. Herbstritt and B. Becker. Conflict-Based Selection of Branching Rules. In: Theory and Applications of Satisfiability Testing – SAT 2003. Ed. by E. Giunchiglia and A.

Tacchella. Vol. 2919. LNCS. Springer, 2003, pp. 441–451. DOI: 10.1007/978-3-540-24605-3_33 (p. 188).

[HLL+00] S. Hong, S. Lee, J. Lim, J. Sung, D. H. Cheon, and I. Cho. Provable Security against Differential and Linear Cryptanalysis for the SPN Structure. In: Fast Software Encryption – FSE 2000. Ed. by B. Schneier. Vol. 1978. LNCS. Springer, 2000, pp. 273–283. DOI: 10.1007/3-540-44706-7_19 (p. 56).

[HM06] M. Heule and H. van Maaren. March_dl: Adding Adaptive Heuristics and a New Branching Strategy. In: JSAT 2.1-4 (2006), pp. 47–59. URL: http://jsat.ewi.tudelft.nl/content/volume2/JSAT2_3_Heule.pdf (p. 189).

[HM09] M. Heule and H. van Maaren. Look-Ahead Based SAT Solvers. In: Handbook of Satisfiability. Ed. by A. Biere, M. Heule, H. van Maaren, and T. Walsh. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 155–184. DOI: 10.3233/978-1-58603-929-5-155 (pp. 187, 189).

[HPR04] P. Hawkes, M. Paddon, and G. G. Rose. On Corrective Patterns for the SHA-2 Family. IACR Cryptology ePrint Archive, Report 2004/207. 2004. IACR: 2004/207 (p. 173).

[HT94] H. M. Heys and S. E. Tavares. The Design of Substitution-Permutation Networks Resistant to Differential and Linear Cryptanalysis. In: Computer Communications Security – CCS 1994. Ed. by D. E. Denning, R. Pyle, R. Ganesan, and R. S. Sandhu. ACM, 1994, pp. 148–155. DOI: 10.1145/191177.191206 (p. 55).

[HT96] H. M. Heys and S. E. Tavares. Substitution-Permutation Networks Resistant to Differential and Linear Cryptanalysis. In: Journal of Cryptology 9.1 (1996), pp. 1–19. DOI: 10.1007/BF02254789 (p. 55).

[IEE97] IEEE 802.11 Working Group. IEEE Std 802.11-1997: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Institute of Electrical and Electronics Engineers (IEEE) Standard for Information Technology. 1997. DOI: 10.1109/IEEESTD.1997.85951 (p. 149).

[IK04] T. Iwata and T. Kohno. New Security Proofs for the 3GPP Confidentiality and Integrity Algorithms. In: Fast Software Encryption – FSE 2004. Ed. by B. K. Roy and W. Meier. Vol. 3017. LNCS. Springer, 2004, pp. 427–445. DOI: `10.1007/978-3-540-25937-4_27`. IACR: `2004/019` (p. 149).

[IMPR08] S. Indesteege, F. Mendel, B. Preneel, and C. Rechberger. Collisions and Other Non-random Properties for Step-Reduced SHA-256. In: Selected Areas in Cryptography – SAC 2008. Ed. by R. M. Avanzi, L. Keliher, and F. Sica. Vol. 5381. LNCS. Springer, 2008, pp. 276–293. DOI: `10.1007/978-3-642-04159-4_18`. IACR: `2008/131` (pp. 168, 169, 174).

[IMPS09] S. Indesteege, F. Mendel, B. Preneel, and M. Schläffer. Practical Collisions for SHAMATA-256. In: Selected Areas in Cryptography – SAC 2009. Ed. by M. J. J. Jr., V. Rijmen, and R. Safavi-Naini. Vol. 5867. LNCS. Springer, 2009, pp. 1–15. DOI: `10.1007/978-3-642-05445-7_1` (p. 107).

[IS09] T. Isobe and K. Shibutani. Preimage Attacks on Reduced Tiger and SHA-2. In: Fast Software Encryption – FSE 2009. Ed. by O. Dunkelman. Vol. 5665. LNCS. Springer, 2009, pp. 139–155. DOI: `10.1007/978-3-642-03317-9_9` (p. 173).

[Jea16] J. Jean. Cryptanalysis of Haraka. In: IACR Transactions on Symmetric Cryptology 2016.1 (2016), pp. 1–12. DOI: `10.13154/tosc.v2016.i1.1-12`. IACR: `2016/396` (p. 107).

[JK97] T. Jakobsen and L. R. Knudsen. The Interpolation Attack on Block Ciphers. In: Fast Software Encryption – FSE 1997. Ed. by E. Biham. Vol. 1267. LNCS. Springer, 1997, pp. 28–40. DOI: `10.1007/BFb0052332` (pp. 55, 145).

[JLM14] P. Jovanovic, A. Luykx, and B. Mennink. Beyond $2^{c/2}$ Security in Sponge-Based Authenticated Encryption Modes. In: Advances in Cryptology – ASIACRYPT 2014. Ed. by P. Sarkar and T. Iwata. Vol. 8873. LNCS. Springer, 2014, pp. 85–104. DOI: `10.1007/978-3-662-45611-8_5`. IACR: `2014/373` (p. 42).

[JN16] J. Jean and I. Nikolic. Efficient Design Strategies Based on the AES Round Function. In: Fast Software Encryption – FSE 2016. Ed. by T. Peyrin. Vol. 9783. LNCS. Springer, 2016, pp. 334–353. DOI: `10.1007/978-3-662-52993-5_17`. IACR: `2016/299` (p. 106).

[JNP14a] J. Jean, I. Nikolić, and T. Peyrin. KIASU v1. Submission to CAESAR: Competition for Authenticated Encryption. Security, Applicability, and Robustness (Round 1). 2014. URL: `http://competitions.cr.yp.to/round1/kiasuv1.pdf` (pp. 3, 52).

[JNP14b] J. Jean, I. Nikolić, and T. Peyrin. Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In: Advances in Cryptology – ASIACRYPT 2014. Ed. by P. Sarkar and T. Iwata. Vol. 8874. LNCS. Springer, 2014, pp. 274–288. DOI: `10.1007/978-3-662-45608-8_15`. IACR: `2014/831` (pp. 24, 26, 52, 70).

[JNSW14] J. Jean, I. Nikolić, Y. Sasaki, and L. Wang. Practical Cryptanalysis of PAES. In: Selected Areas in Cryptography – SAC 2014. Ed. by A. Joux and A. M. Youssef. Vol. 8781. LNCS. Springer, 2014, pp. 228–242. DOI: `10.1007/978-3-319-13051-4_14` (p. 107).

[JNSW16] J. Jean, I. Nikolić, Y. Sasaki, and L. Wang. Practical Forgeries and Distinguishers against PAES. In: IEICE Transactions 99-A.1 (2016), pp. 39–48. URL: `http://search.ieice.org/bin/summary.php?id=e99-a_1_39` (p. 107).

[Joh09] T. R. Johnson. American Cryptology during the Cold War, 1945–1989. Book III: Retrenchment and Reform, 1972–1980. National Security Agency. Retrieved via Cryptome FOIA request. 2009. URL: `http://cryptome.org/0001/nsa-meyer.htm` (p. 4).

[Jou04] A. Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In: Advances in Cryptology – CRYPTO 2004. Ed. by M. K. Franklin. Vol. 3152. LNCS. Springer, 2004, pp. 306–316. DOI: `10.1007/978-3-540-28628-8_19` (pp. 36, 168, 173).

[JW90] R. G. Jeroslow and J. Wang. Solving Propositional Satisfiability Problems. In: Ann. Math. Artif. Intell. 1 (1990), pp. 167–187. DOI: `10.1007/BF01531077` (p. 188).

[Kar15] P. Karpman. From Distinguishers to Key Recovery: Improved Related-Key Attacks on Even-Mansour. In: Information Security – ISC 2015. Ed. by J. Lopez and C. J. Mitchell. Vol. 9290. LNCS. Springer, 2015, pp. 177–188.

DOI: 10.1007/978-3-319-23318-5_10. IACR: 2015/134 (pp. 148, 161).

[Kay07] R. F. Kayser. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. In: Federal Register Notice 72.212 (2007), pp. 62212–62220. URL: http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf (p. 34).

[KCP16] J. Kelsey, S.-j. Chang, and R. Perlner. NIST SP 800-185: SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash. National Institute of Standards and Technology (NIST) Special Publication (SP). 2016. DOI: 10.6028/NIST.SP.800-185 (p. 167).

[KD79] J. B. Kam and G. I. Davida. Structured Design of Substitution-Permutation Encryption Networks. In: IEEE Transactions on Computers 28.10 (1979), pp. 747–753. DOI: 10.1109/TC.1979.1675242 (p. 20).

[KK06] J. Kelsey and T. Kohno. Herding Hash Functions and the Nostradamus Attack. In: Advances in Cryptology – EUROCRYPT 2006. Ed. by S. Vaudenay. Vol. 4004. LNCS. Springer, 2006, pp. 183–200. DOI: 10.1007/11761679_12 (pp. 36, 168, 173).

[KLL+14] E. B. Kavun, M. M. Lauridsen, G. Leander, C. Rechberger, P. Schwabe, and T. Yalçın. Prøst v1. Submission to CAESAR: Competition for Authenticated Encryption. Security, Applicability, and Robustness (Round 1). 2014. URL: http://competitions.cr.yp.to/round1/proestv1.pdf (pp. 10, 147, 148, 150, 151).

[KLMR16a] S. Kölbl, M. M. Lauridsen, F. Mendel, and C. Rechberger. Haraka – Efficient Short-Input Hashing for Post-Quantum Applications. IACR Cryptology ePrint Archive, Report 2016/98. 2016. IACR: 2016/098 (p. 107).

[KLMR16b] S. Kölbl, M. M. Lauridsen, F. Mendel, and C. Rechberger. Haraka v2 - Efficient Short-Input Hashing for Post-Quantum Applications. In: IACR Transactions on Symmetric Cryptology 2016.2 (2016), pp. 1–29. DOI: 10.13154/tosc.v2016.i2.1-29. IACR: 2016/098 (p. 106).

[KMNS13] S. Kölbl, F. Mendel, T. Nad, and M. Schläffer. Differential Cryptanalysis of Keccak Variants. In: Cryptography and Coding – IMACC 2013. Ed. by M. Stam. Vol. 8308. LNCS. Springer, 2013, pp. 141–157. DOI: 10.1007/978-3-642-45239-0_9 (pp. 167, 181).

[Knu91] L. R. Knudsen. Cryptanalysis of LOKI. In: Advances in Cryptology – ASIACRYPT 1991. Ed. by H. Imai, R. L. Rivest, and T. Matsumoto. Vol. 739. LNCS. Springer, 1991, pp. 22–35. DOI: 10.1007/3-540-57332-1_2 (p. 149).

[Knu93] L. R. Knudsen. Practically Secure Feistel Ciphers. In: Fast Software Encryption – FSE 1993. Ed. by R. J. Anderson. Vol. 809. LNCS. Springer, 1993, pp. 211–221. DOI: 10.1007/3-540-58108-1_26 (p. 55).

[Knu94] L. R. Knudsen. Truncated and Higher Order Differentials. In: Fast Software Encryption – FSE 1994. Ed. by B. Preneel. Vol. 1008. LNCS. Springer, 1994, pp. 196–211. DOI: 10.1007/3-540-60590-8_16 (pp. 60, 62, 128).

[Knu98] L. Knudsen. DEAL – A 128-bit Block Cipher. Tech. rep. Technical Report 151. University of Bergen, Department of Informatics, 1998. URL: http://www2.mat.dtu.dk/people/Lars.R.Knudsen/papers/deal.pdf.gz (p. 62).

[Köl14] S. Kölbl. CryptoSMT: An easy to use tool for cryptanalysis of symmetric primitives. 2014. URL: https://github.com/kste/cryptosmt (p. 58).

[KPS15] P. Karpman, T. Peyrin, and M. Stevens. Practical Free-Start Collision Attacks on 76-step SHA-1. In: Advances in Cryptology – CRYPTO 2015. Ed. by R. Gennaro and M. Robshaw. Vol. 9215. LNCS. Springer, 2015, pp. 623–642. DOI: 10.1007/978-3-662-47989-6_30. IACR: 2015/530 (pp. 166, 215).

[KR01] J. Kilian and P. Rogaway. How to Protect DES Against Exhaustive Key Search (an Analysis of DESX). In: Journal of Cryptology 14.1 (2001), pp. 17–35. DOI: 10.1007/s001450010015 (pp. 28, 71).

[KR07] L. R. Knudsen and V. Rijmen. Known-Key Distinguishers for Some Block Ciphers. In: Advances in Cryptology – ASIACRYPT 2007. Ed. by K. Kurosawa. Vol. 4833. LNCS.

Springer, 2007, pp. 315–324. DOI: 10.1007/978-3-540-76900-2_19 (p. 129).

[KR11a] L. R. Knudsen and M. Robshaw. The Block Cipher Companion. Information Security and Cryptography. Springer, 2011. ISBN: 978-3-642-17341-7. DOI: 10.1007/978-3-642-17342-4 (p. 17).

[KR11b] T. Krovetz and P. Rogaway. The Software Performance of Authenticated-Encryption Modes. In: Fast Software Encryption – FSE 2011. Ed. by A. Joux. Vol. 6733. LNCS. Springer, 2011, pp. 306–327. DOI: 10.1007/978-3-642-21702-9_18 (p. 41).

[KR14] T. Krovetz and P. Rogaway. IETF RFC 7253: The OCB Authenticated-Encryption Algorithm. Internet Engineering Task Force (IETF) Request for Comments (RFC). 2014. DOI: 10.17487/RFC7253 (p. 160).

[KR96] J. Kilian and P. Rogaway. How to Protect DES Against Exhaustive Key Search. In: Advances in Cryptology – CRYPTO 1996. Ed. by N. Koblitz. Vol. 1109. LNCS. Springer, 1996, pp. 252–267. DOI: 10.1007/3-540-68697-5_20 (pp. 28, 71).

[Kra01] H. Krawczyk. The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?) In: Advances in Cryptology – CRYPTO 2001. Ed. by J. Kilian. Vol. 2139. LNCS. Springer, 2001, pp. 310–331. DOI: 10.1007/3-540-44647-8_19. IACR: 2001/045 (p. 41).

[KRS12] D. Khovratovich, C. Rechberger, and A. Savelieva. Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In: Fast Software Encryption – FSE 2012. Ed. by A. Canteaut. Vol. 7549. LNCS. Springer, 2012, pp. 244–263. DOI: 10.1007/978-3-642-34047-5_15 (p. 173).

[KS05] J. Kelsey and B. Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In: Advances in Cryptology – EUROCRYPT 2005. Ed. by R. Cramer. Vol. 3494. LNCS. Springer, 2005, pp. 474–490. DOI: 10.1007/11426639_28 (pp. 36, 168, 173).

[KS07] L. Keliher and J. Sui. Exact maximum expected differential and linear probability for two-round Advanced Encryption Standard. In: IET Information Security 1.2 (2007), pp. 53–57. DOI: `10.1049/iet-ifs:20060161`. IACR: `2005/321` (p. 115).

[KW02] L. R. Knudsen and D. Wagner. Integral Cryptanalysis. In: Fast Software Encryption – FSE 2002. Ed. by J. Daemen and V. Rijmen. Vol. 2365. LNCS. Springer, 2002, pp. 112–127. DOI: `10.1007/3-540-45661-9_9` (p. 63).

[KY00a] J. Katz and M. Yung. Complete characterization of security notions for probabilistic private-key encryption. In: Symposium on Theory of Computing – STOC 2000. Ed. by F. F. Yao and E. M. Luks. ACM, 2000, pp. 245–254. DOI: `10.1145/335305.335335` (p. 31).

[KY00b] J. Katz and M. Yung. Unforgeable Encryption and Chosen Ciphertext Secure Modes of Operation. In: Fast Software Encryption – FSE 2000. Ed. by B. Schneier. Vol. 1978. LNCS. Springer, 2000, pp. 284–299. DOI: `10.1007/3-540-44706-7_20` (pp. 39, 40).

[LA97] C. M. Li and Anbulagan. Heuristics Based on Unit Propagation for Satisfiability Problems. In: Artificial Intelligence – IJCAI 1997. Morgan Kaufmann, 1997, pp. 366–371 (pp. 187, 189).

[Laf13] F. Lafitte. CryptoSAT. 2013. URL: `https://qualsec.ulb.ac.be/people/frederic-lafitte/cryptosat/` (p. 58).

[Lai94] X. Lai. Higher Order Derivatives and Differential Cryptanalysis. In: Communications and Cryptography: Two Sides of One Tapestry. Ed. by R. E. Blahut, D. J. Costello Jr., U. Maurer, and T. Mittelholzer. Vol. 276. International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1994, pp. 227–233. DOI: `10.1007/978-1-4615-2694-0_23` (pp. 44, 62, 128, 132).

[Lam79] L. Lamport. Constructing digital signatures from a one-way function. Tech. rep. SRI-CSL-98. SRI International Computer Science Laboratory, 1979. URL: `http://research.microsoft.com/en-us/um/people/lamport/pubs/dig-sig.pdf` (p. 110).

[Leu12] G. Leurent. Analysis of Differential Attacks in ARX Constructions. In: Advances in Cryptology – ASIACRYPT 2012. Ed. by X. Wang and K. Sako. Vol. 7658. LNCS. Springer, 2012, pp. 226–243. DOI: 10.1007/978-3-642-34961-4_15 (pp. 59, 167, 178, 180).

[Leu13] G. Leurent. Construction of Differential Characteristics in ARX Designs Application to Skein. In: Advances in Cryptology – CRYPTO 2013. Ed. by R. Canetti and J. A. Garay. Vol. 8042. LNCS. Springer, 2013, pp. 241–258. DOI: 10.1007/978-3-642-40041-4_14 (pp. 59, 167, 178, 180).

[Leu15] G. Leurent. Differential Forgery Attack Against LAC. In: Selected Areas in Cryptography – SAC 2015. Ed. by O. Dunkelman and L. Keliher. Vol. 9566. LNCS. Springer, 2015, pp. 217–224. DOI: 10.1007/978-3-319-31301-6_13 (p. 86).

[LH94] S. K. Langford and M. E. Hellman. Differential-Linear Cryptanalysis. In: Advances in Cryptology – CRYPTO 1994. Ed. by Y. Desmedt. Vol. 839. LNCS. Springer, 1994, pp. 17–25. DOI: 10.1007/3-540-48658-5_3 (p. 63).

[Lib00] P. Liberatore. On the complexity of choosing the branching literal in DPLL. In: Artif. Intell. 116.1-2 (2000), pp. 315–326. DOI: 10.1016/S0004-3702(99)00097-1 (p. 187).

[LIS12] J. Li, T. Isobe, and K. Shibutani. Converting Meet-In-The-Middle Preimage Attack into Pseudo Collision Attack: Application to SHA-2. In: Fast Software Encryption – FSE 2012. Ed. by A. Canteaut. Vol. 7549. LNCS. Springer, 2012, pp. 264–286. DOI: 10.1007/978-3-642-34047-5_16 (pp. 173, 174).

[LJSH08] Y. Lee, K. Jeong, J. Sung, and S. Hong. Related-Key Chosen IV Attacks on Grain-v1 and Grain-128. In: Information Security and Privacy – ACISP 2008. Ed. by Y. Mu, W. Susilo, and J. Seberry. Vol. 5107. LNCS. Springer, 2008, pp. 321–335. DOI: 10.1007/978-3-540-70500-0_24 (p. 150).

[LM01] H. Lipmaa and S. Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In: Fast Software Encryption – FSE 2001. Ed. by M. Matsui. Vol. 2355. LNCS. Springer, 2001, pp. 336–350. DOI: 10.1007/3-540-45473-X_28. IACR: 2001/001 (p. 46).

[LM11] M. Lamberger and F. Mendel. Higher-Order Differential Attack on Reduced SHA-256. IACR Cryptology ePrint Archive, Report 2011/37. 2011. IACR: 2011/037 (p. 174).

[LM92] X. Lai and J. L. Massey. Hash Function Based on Block Ciphers. In: Advances in Cryptology – EUROCRYPT 1992. Ed. by R. A. Rueppel. Vol. 658. LNCS. Springer, 1992, pp. 55–70. DOI: 10.1007/3-540-47555-9_5 (pp. 36, 53).

[LMM91] X. Lai, J. L. Massey, and S. Murphy. Markov Ciphers and Differential Cryptanalysis. In: Advances in Cryptology – EUROCRYPT 1991. Ed. by D. W. Davies. Vol. 547. LNCS. Springer, 1991, pp. 17–38. DOI: 10.1007/3-540-46416-6_2. URL: http://www.isg.rhul.ac.uk/~sean/xuejia.pdf (pp. 44, 47, 48, 60).

[LMS+15] M. Lamberger, F. Mendel, M. Schläffer, C. Rechberger, and V. Rijmen. The Rebound Attack and Subspace Distinguishers: Application to Whirlpool. In: Journal of Cryptology 28.2 (2015), pp. 257–296. DOI: 10.1007/s00145-013-9166-5 (p. 54).

[LP13] F. Landelle and T. Peyrin. Cryptanalysis of Full RIPEMD-128. In: Advances in Cryptology – EUROCRYPT 2013. Ed. by T. Johansson and P. Q. Nguyen. Vol. 7881. LNCS. Springer, 2013, pp. 228–244. DOI: 10.1007/978-3-642-38348-9_14 (pp. 166, 178).

[LR88] M. Luby and C. Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. In: SIAM Journal on Computing 17.2 (1988), pp. 373–386. DOI: 10.1137/0217022 (p. 27).

[LRW02] M. Liskov, R. L. Rivest, and D. Wagner. Tweakable Block Ciphers. In: Advances in Cryptology – CRYPTO 2002. Ed. by M. Yung. Vol. 2442. LNCS. Springer, 2002, pp. 31–46. DOI: 10.1007/3-540-45708-9_3 (pp. 24, 41, 69).

[LRW11] M. Liskov, R. L. Rivest, and D. Wagner. Tweakable Block Ciphers. In: Journal of Cryptology 24.3 (2011), pp. 588–613. DOI: 10.1007/s00145-010-9073-y (pp. 24, 41).

[Luc05] S. Lucks. A Failure-Friendly Design Principle for Hash Functions. In: Advances in Cryptology – ASIACRYPT 2005. Ed. by B. K. Roy. Vol. 3788. Lecture Notes in Computer Science. Springer, 2005, pp. 474–494. DOI: 10.1007/11593447_26 (p. 36).

[LW17] A. K. Lenstra and B. Wesolowski. Trustworthy public randomness with sloth, unicorn, and trx. In: International Journal of Applied Cryptography (IJACT) 3.4 (2017), pp. 330–343. DOI: `10.1504/IJACT.2017.10010315`. IACR: `2015/366` (p. 213).

[Man11] S. Manuel. Classification and generation of disturbance vectors for collision attacks against SHA-1. In: Designs, Codes and Cryptography 59.1-3 (2011), pp. 247–263. DOI: `10.1007/s10623-010-9458-9`. IACR: `2008/469` (pp. 215, 217).

[Mas93] J. L. Massey. SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm. In: Fast Software Encryption – FSE 1993. Ed. by R. J. Anderson. Vol. 809. LNCS. Springer, 1993, pp. 1–17. DOI: `10.1007/3-540-58108-1_1` (p. 20).

[Mat93] M. Matsui. Linear Cryptanalysis Method for DES Cipher. In: Advances in Cryptology – EUROCRYPT 1993. Ed. by T. Helleseth. Vol. 765. LNCS. Springer, 1993, pp. 386–397. DOI: `10.1007/3-540-48285-7_33` (p. 166).

[Mat94] M. Matsui. On Correlation Between the Order of S-boxes and the Strength of DES. In: Advances in Cryptology – EUROCRYPT 1994. Ed. by A. D. Santis. Vol. 950. LNCS. Springer, 1994, pp. 366–375. DOI: `10.1007/BFb0053451` (pp. 59, 63).

[Mat96] M. Matsui. New Structure of Block Ciphers with Provable Security against Differential and Linear Cryptanalysis. In: Fast Software Encryption – FSE 1996. Ed. by D. Gollmann. Vol. 1039. LNCS. Springer, 1996, pp. 205–218. DOI: `10.1007/3-540-60865-6_54` (p. 55).

[Mat97] M. Matsui. New Block Encryption Algorithm MISTY. In: Fast Software Encryption – FSE 1997. Ed. by E. Biham. Vol. 1267. LNCS. Springer, 1997, pp. 54–68. DOI: `10.1007/BFb0052334` (p. 55).

[MDV17] S. Mella, J. Daemen, and G. Van Assche. New techniques for trail bounds and application to differential trails in Keccak. In: IACR Transactions on Symmetric Cryptology 2017.1 (2017), pp. 329–357. DOI: `10.13154/tosc.v2017.i1.329-357`. IACR: `2017/181` (p. 59).

[Men15] B. Mennink. Optimally Secure Tweakable Blockciphers. In: Fast Software Encryption – FSE 2015. Ed. by G. Leander. Vol. 9054. LNCS. Springer, 2015, pp. 428–448. DOI: 10.1007/978-3-662-48116-5_21. IACR: 2015/363 (p. 41).

[Men16] B. Mennink. XPX: Generalized Tweakable Even-Mansour with Improved Security Guarantees. In: Advances in Cryptology – CRYPTO 2016. Ed. by M. Robshaw and J. Katz. Vol. 9814. LNCS. Springer, 2016, pp. 64–94. DOI: 10.1007/978-3-662-53018-4_3. IACR: 2015/476 (p. 160).

[Mer79] R. C. Merkle. Secrecy, Authentication, and Public Key Systems. PhD thesis. California: Departement of electrical engineering, Stanford University, 1979. URL: http://www.merkle.com/papers/Thesis1979.pdf (pp. 34, 35).

[Mer89] R. C. Merkle. One Way Hash Functions and DES. In: Advances in Cryptology – CRYPTO 1989. Ed. by G. Brassard. Vol. 435. LNCS. Springer, 1989, pp. 428–446. DOI: 10.1007/0-387-34805-0_40 (pp. 35, 53, 170, 173).

[Min14] K. Minematsu. Parallelizable Rate-1 Authenticated Encryption from Pseudorandom Functions. In: Advances in Cryptology – EUROCRYPT 2014. Ed. by P. Q. Nguyen and E. Oswald. Vol. 8441. LNCS. Springer, 2014, pp. 275–292. DOI: 10.1007/978-3-642-55220-5_16 (pp. 148, 151).

[MJSC16] P. Méaux, A. Journault, F.-X. Standaert, and C. Carlet. Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts. In: Advances in Cryptology – EUROCRYPT 2016. Ed. by M. Fischlin and J.-S. Coron. Vol. 9665. LNCS. Springer, 2016, pp. 311–343. DOI: 10.1007/978-3-662-49890-3_13. IACR: 2016/254 (p. 129).

[MMZ+01] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In: Design Automation Conference – DAC 2001. ACM, 2001, pp. 530–535. DOI: 10.1145/378239.379017. URL: http://doi.acm.org/10.1145/378239.379017 (p. 188).

[MNS11a] F. Mendel, T. Nad, and M. Schläffer. Cryptanalysis of Round-Reduced HAS-160. In: Information Security and Cryptology – ICISC 2011. Ed. by H. Kim. Vol. 7259. LNCS. Springer, 2011, pp. 33–47. DOI: 10.1007/978-3-642-31912-9_3 (pp. 167, 180).

[MNS11b] F. Mendel, T. Nad, and M. Schläffer. Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In: Advances in Cryptology – ASIACRYPT 2011. Ed. by D. H. Lee and X. Wang. Vol. 7073. LNCS. Springer, 2011, pp. 288–307. DOI: 10.1007/978-3-642-25385-0_16 (pp. 11, 59, 61, 165, 167, 168, 171, 172, 174–180, 188, 189, 191, 194, 201).

[MNS12] F. Mendel, T. Nad, and M. Schläffer. Collision Attacks on the Reduced Dual-Stream Hash Function RIPEMD-128. In: Fast Software Encryption – FSE 2012. Ed. by A. Canteaut. Vol. 7549. LNCS. Springer, 2012, pp. 226–243. DOI: 10.1007/978-3-642-34047-5_14 (p. 167).

[MNS13a] F. Mendel, T. Nad, and M. Schläffer. Finding Collisions for Round-Reduced SM3. In: Topics in Cryptology – CT-RSA 2013. Ed. by E. Dawson. Vol. 7779. LNCS. Springer, 2013, pp. 174–188. DOI: 10.1007/978-3-642-36095-4_12 (pp. 167, 178).

[MNS13b] F. Mendel, T. Nad, and M. Schläffer. Improving Local Collisions: New Attacks on Reduced SHA-256. In: Advances in Cryptology – EUROCRYPT 2013. Ed. by T. Johansson and P. Q. Nguyen. Vol. 7881. LNCS. Springer, 2013, pp. 262–278. DOI: 10.1007/978-3-642-38348-9_16 (pp. 11, 59, 165, 167, 168, 174–178, 192, 194, 197, 200, 201).

[MNSS12] F. Mendel, T. Nad, S. Scherz, and M. Schläffer. Differential Attacks on Reduced RIPEMD-160. In: Information Security – ISC 2012. Ed. by D. Gollmann and F. C. Freiling. Vol. 7483. LNCS. Springer, 2012, pp. 23–38. DOI: 10.1007/978-3-642-33383-5_2 (pp. 166, 167, 178).

[Mor15] P. Morawiecki. Malicious Keccak. In: IACR Cryptology ePrint Archive, Report 2015/1085 (2015). IACR: 2015/1085 (p. 213).

[MP13] N. Mouha and B. Preneel. Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20. IACR Cryptology ePrint Archive, Report 2013/328. 2013. IACR: 2013/328 (p. 58).

[MPRR06a] F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. Analysis of Step-Reduced SHA-256. In: Fast Software Encryption – FSE 2006. Ed. by M. J. B. Robshaw. Vol. 4047.

LNCS. Springer, 2006, pp. 126–143. DOI: 10.1007/11799313_9. IACR: 2008/130 (p. 173).

[MPRR06b] F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. The Impact of Carries on the Complexity of Collision Attacks on SHA-1. In: Fast Software Encryption – FSE 2006. Ed. by M. J. B. Robshaw. Vol. 4047. LNCS. Springer, 2006, pp. 278–292. DOI: 10.1007/11799313_18 (p. 217).

[MPS+13] F. Mendel, T. Peyrin, M. Schläffer, L. Wang, and S. Wu. Improved Cryptanalysis of Reduced RIPEMD-160. In: Advances in Cryptology – ASIACRYPT 2013. Ed. by K. Sako and P. Sarkar. Vol. 8270. LNCS. Springer, 2013, pp. 484–503. DOI: 10.1007/978-3-642-42045-0_25. IACR: 2013/600 (p. 167).

[MRH04] U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In: Theory of Cryptography – TCC 2004. Ed. by M. Naor. Vol. 2951. LNCS. Springer, 2004, pp. 21–39. DOI: 10.1007/978-3-540-24638-1_2. IACR: 2003/161 (p. 35).

[MRST09] F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Fast Software Encryption – FSE 2009. Ed. by O. Dunkelman. Vol. 5665. LNCS. Springer, 2009, pp. 260–276. DOI: 10.1007/978-3-642-03317-9_16 (p. 54).

[MRV15] B. Mennink, R. Reyhanitabar, and D. Vizár. Security of Full-State Keyed Sponge and Duplex: Applications to Authenticated Encryption. In: Advances in Cryptology – ASIACRYPT 2015. Ed. by T. Iwata and J. H. Cheon. Vol. 9453. LNCS. Springer, 2015, pp. 465–489. DOI: 10.1007/978-3-662-48800-3_19. IACR: 2015/541 (p. 42).

[Mur90] S. Murphy. The Cryptanalysis of FEAL-4 with 20 Chosen Plaintexts. In: Journal of Cryptology 2.3 (1990), pp. 145–154. DOI: 10.1007/BF00190801. URL: http://www.isg.rhul.ac.uk/~sean/feal.pdf (p. 44).

[MV04] D. A. McGrew and J. Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In: Progress in Cryptology – INDOCRYPT 2004. Ed. by A. Canteaut and K. Viswanathan. Vol. 3348. LNCS. Springer, 2004, pp. 343–355. DOI: 10.1007/978-3-540-30556-9_27. IACR: 2004/193 (pp. 38, 41).

[MWGP11] N. Mouha, Q. Wang, D. Gu, and B. Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In: Information Security and Cryptology – Inscrypt 2011. Ed. by C. Wu, M. Yung, and D. Lin. Vol. 7537. LNCS. Springer, 2011, pp. 57–76. DOI: 10.1007/978-3-642-34704-7_5 (pp. 58, 111).

[MY92] M. Matsui and A. Yamagishi. A New Method for Known Plaintext Attack of FEAL Cipher. In: Advances in Cryptology – EUROCRYPT 1992. Ed. by R. A. Rueppel. Vol. 658. LNCS. Springer, 1992, pp. 81–91. DOI: 10.1007/3-540-47555-9_7 (p. 166).

[Nat85] National Institute of Standards and Technology. NIST FIPS PUB 113: Computer Data Authentication. National Institute of Standards and Technology (NIST) Federal Information Processing Standards (FIPS) Publication. 1985. URL: http://csrc.nist.gov/publications/fips/fips113/fips113.html (p. 38).

[NB08] I. Nikolić and A. Biryukov. Collisions for Step-Reduced SHA-256. In: Fast Software Encryption – FSE 2008. Ed. by K. Nyberg. Vol. 5086. LNCS. Springer, 2008, pp. 1–15. DOI: 10.1007/978-3-540-71039-4_1 (p. 174).

[Nik15] I. Nikolić. Tiaoxin v2. Submission to CAESAR: Competition for Authenticated Encryption. Security, Applicability, and Robustness (Round 2). 2015. URL: http://competitions.cr.yp.to/round2/tiaoxinv2.pdf (p. 106).

[NK92] K. Nyberg and L. R. Knudsen. Provable Security Against Differential Cryptanalysis. In: Advances in Cryptology – CRYPTO 1992. Ed. by E. F. Brickell. Vol. 740. LNCS. Springer, 1992, pp. 566–574. DOI: 10.1007/3-540-48071-4_41 (pp. 45, 55).

[NK95] K. Nyberg and L. R. Knudsen. Provable Security Against a Differential Attack. In: Journal of Cryptology 8.1 (1995), pp. 27–37. DOI: 10.1007/BF00204800 (pp. 45, 55).

[NLV11] M. Naehrig, K. E. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In: Cloud Computing Security – CCSW 2011. Ed. by C. Cachin and T. Ristenpart. ACM, 2011, pp. 113–124. DOI: 10.1145/2046660.2046682. IACR: 2011/405 (p. 128).

[NRS13] C. Namprempre, P. Rogaway, and T. Shrimpton. AE5 Security Notions: Definitions Implicit in the CAESAR Call. IACR Cryptology ePrint Archive, Report 2013/242. 2013. IACR: 2013/242 (p. 39).

[NRS14] C. Namprempre, P. Rogaway, and T. Shrimpton. Reconsidering Generic Composition. In: Advances in Cryptology – EUROCRYPT 2014. Ed. by P. Q. Nguyen and E. Oswald. Vol. 8441. LNCS. Springer, 2014, pp. 257–274. DOI: 10.1007/978-3-642-55220-5_15. IACR: 2014/206 (p. 41).

[Nyb93] K. Nyberg. Differentially Uniform Mappings for Cryptography. In: Advances in Cryptology – EUROCRYPT 1993. Ed. by T. Helleseth. Vol. 765. LNCS. Springer, 1993, pp. 55–64. DOI: 10.1007/3-540-48285-7_6 (p. 45).

[Ouy98] M. Ouyang. How Good Are Branching Rules in DPLL? In: Discrete Applied Mathematics 89.1-3 (1998), pp. 281–286. DOI: 10.1016/S0166-218X(98)00045-6 (p. 187).

[Pey09] T. Peyrin. Chosen-salt, chosen-counter, pseudo-collision for the compression function of SHAvite-3. NIST mailing list. 2009. URL: http://ehash.iaik.tugraz.at/uploads/e/ea/Peyrin-SHAvite-3.txt (p. 107).

[PGV93a] B. Preneel, R. Govaerts, and J. Vandewalle. Differential Cryptanalysis of Hash Functions Based on Block Ciphers. In: Computer and Communications Security – CCS 1993. Ed. by D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby. ACM, 1993, pp. 183–188. DOI: 10.1145/168588.168611 (p. 54).

[PGV93b] B. Preneel, R. Govaerts, and J. Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In: Advances in Cryptology – CRYPTO 1993. Ed. by D. R. Stinson. Vol. 773. LNCS. Springer, 1993, pp. 368–378. DOI: 10.1007/3-540-48329-2_31. URL: https://www.esat.kuleuven.be/cosic/publications/article-48.pdf (p. 28).

[PRR05] N. Pramstaller, C. Rechberger, and V. Rijmen. Exploiting Coding Theory for Collision Attacks on SHA-1. In: Cryptography and Coding – IMA 2005. Ed. by N. P. Smart. Vol. 3796. LNCS. Springer, 2005, pp. 78–95. DOI: 10.1007/11586821_7 (pp. 215, 217).

[PS16] T. Peyrin and Y. Seurin. Counter-in-Tweak: Authenticated Encryption Modes for Tweakable Block Ciphers. In: Advances in Cryptology – CRYPTO 2016. Ed. by M. Robshaw and J. Katz. Vol. 9814. LNCS. Springer, 2016, pp. 33–63. DOI: 10.1007/978-3-662-53018-4_2. IACR: 2015/1049 (p. 41).

[PW13] E. Pasalic and Y. Wei. Generic related-key and induced chosen IV attacks using the method of key differentiation. IACR Cryptology ePrint Archive, Report 2013/586. 2013. IACR: 2013/586 (p. 150).

[Rab78] M. O. Rabin. Digitalized Signatures. In: Foundations of Secure Computation. Ed. by R. A. DeMillo, R. J. Lipton, D. P. Dobkin, and A. K. Jones. Academic Press, 1978, pp. 155–168. ISBN: 0122103505. URL: https://smartech.gatech.edu/handle/1853/40598 (p. 34).

[RBBK01] P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In: Computer and Communications Security – CCS 2001. Ed. by M. K. Reiter and P. Samarati. ACM, 2001, pp. 196–205. DOI: 10.1145/501983.502011. IACR: 2001/026. URL: http://web.cs.ucdavis.edu/~rogaway/ocb/ocb-full.pdf (pp. 31, 39).

[RDP+96] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. D. Win. The Cipher SHARK. In: Fast Software Encryption – FSE 1996. Ed. by D. Gollmann. Vol. 1039. LNCS. Springer, 1996, pp. 99–111. DOI: 10.1007/3-540-60865-6_47 (p. 56).

[RO05] V. Rijmen and E. Oswald. Update on SHA-1. In: Topics in Cryptology – CT-RSA 2005. Ed. by A. Menezes. Vol. 3376. LNCS. Springer, 2005, pp. 58–71. DOI: 10.1007/978-3-540-30574-3_6. IACR: 2005/10 (p. 215).

[Rog02] P. Rogaway. Authenticated-encryption with associated-data. In: Computer and Communications Security – CCS 2002. Ed. by V. Atluri. ACM, 2002, pp. 98–107. DOI: 10.1145/

586110.586125. URL: http://web.cs.ucdavis.edu/~rogaway/papers/ad.pdf (p. 39).

[Rog04a] P. Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In: Advances in Cryptology – ASIACRYPT 2004. Ed. by P. J. Lee. Vol. 3329. LNCS. Springer, 2004, pp. 16–31. DOI: 10.1007/978-3-540-30539-2_2. URL: http://web.cs.ucdavis.edu/~rogaway/papers/offsets.pdf (pp. 5, 26, 28, 41).

[Rog04b] P. Rogaway. Nonce-Based Symmetric Encryption. In: Fast Software Encryption – FSE 2004. Ed. by B. K. Roy and W. Meier. Vol. 3017. LNCS. Springer, 2004, pp. 348–359. DOI: 10.1007/978-3-540-25937-4_22 (pp. 30, 39, 150).

[Rog11] P. Rogaway. Evaluation of Some Blockcipher Modes of Operation. Tech. Report CRYPTREC. 2011. URL: https://www.cryptrec.go.jp/estimation/techrep_id2012_2.pdf (pp. 31, 39).

[Røn16] S. Rønjom. Invariant subspaces in Simpira. IACR Cryptology ePrint Archive, Report 2016/248. 2016. IACR: 2016/248 (pp. 107, 124, 126).

[RS04] P. Rogaway and T. Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In: Fast Software Encryption – FSE 2004. Ed. by B. K. Roy and W. Meier. Vol. 3017. LNCS. Springer, 2004, pp. 371–388. DOI: 10.1007/978-3-540-25937-4_24. IACR: 2004/35 (p. 34).

[RS06] P. Rogaway and T. Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In: Advances in Cryptology – EUROCRYPT 2006. Ed. by S. Vaudenay. Vol. 4004. LNCS. Springer, 2006, pp. 373–390. DOI: 10.1007/11761679_23. IACR: 2006/221 (p. 40).

[RWZ12] P. Rogaway, M. Wooding, and H. Zhang. The Security of Ciphertext Stealing. In: Fast Software Encryption – FSE 2012. Ed. by A. Canteaut. Vol. 7549. LNCS. Springer, 2012, pp. 180–195. DOI: 10.1007/978-3-642-34047-5_11 (p. 32).

[SB02] A. A. Selçuk and A. Biçak. On Probability of Success in Linear and Differential Cryptanalysis. In: Security in Communication Networks – SCN 2002. Ed. by S. Cimato, C. Galdi, and G. Persiano. Vol. 2576. LNCS. Springer, 2002, pp. 174–185. DOI: `10.1007/3-540-36413-7_13` (pp. 51, 96).

[SBK+17] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov. The First Collision for Full SHA-1. In: Advances in Cryptology – CRYPTO 2017. Ed. by J. Katz and H. Shacham. Vol. 10401. LNCS. Springer, 2017, pp. 570–596. DOI: `10.1007/978-3-319-63688-7_19`. IACR: `2017/190`. URL: `https://shattered.io` (pp. 59, 166, 215, 224).

[Sel08] A. A. Selçuk. On Probability of Success in Linear and Differential Cryptanalysis. In: Journal of Cryptology 21.1 (2008), pp. 131–147. DOI: `10.1007/s00145-007-9013-7` (p. 51).

[SFKR] B. Schneier, M. Fredrikson, T. Kohno, and T. Ristenpart. Surreptitiously Weakening Cryptographic Systems. IACR: `2015/097` (p. 213).

[Sha49] C. E. Shannon. Communication Theory of Secrecy Systems. In: Bell System Technical Journal 28.4 (1949), pp. 656–715. DOI: `10.1002/j.1538-7305.1949.tb00928.x` (pp. 18, 23, 26).

[SHW+14a] S. Sun, L. Hu, M. Wang, P. Wang, K. Qiao, X. Ma, D. Shi, L. Song, and K. Fu. Towards Finding the Best Characteristics of Some Bit-oriented Block Ciphers and Automatic Enumeration of (Related-key) Differential and Linear Characteristics with Predefined Properties. IACR Cryptology ePrint Archive, Report 2014/747. 2014. IACR: `2014/747` (pp. 59, 77).

[SHW+14b] S. Sun, L. Hu, P. Wang, K. Qiao, X. Ma, and L. Song. Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers. In: Advances in Cryptology – ASIACRYPT 2014. Ed. by P. Sarkar and T. Iwata. Vol. 8873. LNCS. Springer, 2014, pp. 158–178. DOI: `10.1007/978-3-662-45611-8_9` (pp. 59, 77).

[Sil99] J. P. M. Silva. The Impact of Branching Heuristics in Propositional Satisfiability Algorithms. In: Progress in Artificial Intelligence – EPIA 1999. Ed. by P. Barahona and J. J. Alferes. Vol. 1695. LNCS. Springer, 1999, pp. 62–74. DOI: 10.1007/3-540-48159-1_5 (p. 188).

[SKP16] M. Stevens, P. Karpman, and T. Peyrin. Freestart Collision for Full SHA-1. In: Advances in Cryptology – EUROCRYPT 2016. Ed. by M. Fischlin and J.-S. Coron. Vol. 9665. LNCS. Springer, 2016, pp. 459–483. DOI: 10.1007/978-3-662-49890-3_18. IACR: 2015/967 (pp. 166, 215).

[SLW07] M. Stevens, A. K. Lenstra, and B. de Weger. Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In: Advances in Cryptology – EUROCRYPT 2007. Ed. by M. Naor. Vol. 4515. LNCS. Springer, 2007, pp. 1–22. DOI: 10.1007/978-3-540-72540-4_1 (p. 59).

[SO06] M. Schläffer and E. Oswald. Searching for Differential Paths in MD4. In: Fast Software Encryption – FSE 2006. Ed. by M. J. B. Robshaw. Vol. 4047. LNCS. Springer, 2006, pp. 242–261. DOI: 10.1007/11799313_16 (pp. 59, 167, 178).

[Soo16] M. Soos. CryptoMiniSat 5: An advanced SAT solver. 2016. URL: https://github.com/msoos/cryptominisat (p. 58).

[SS07] S. K. Sanadhya and P. Sarkar. New Local Collisions for the SHA-2 Hash Family. In: Information Security and Cryptology – ICISC 2007. Ed. by K.-H. Nam and G. Rhee. Vol. 4817. LNCS. Springer, 2007, pp. 193–205. DOI: 10.1007/978-3-540-76788-6_16. IACR: 2007/352 (p. 173).

[SS08] S. K. Sanadhya and P. Sarkar. New Collision Attacks against Up to 24-Step SHA-2. In: Progress in Cryptology – INDOCRYPT 2008. Ed. by D. R. Chowdhury, V. Rijmen, and A. Das. Vol. 5365. LNCS. Springer, 2008, pp. 91–103. DOI: 10.1007/978-3-540-89754-5_8. IACR: 2008/270 (pp. 168, 169, 174).

[SS09] S. K. Sanadhya and P. Sarkar. A combinatorial analysis of recent attacks on step reduced SHA-2 family. In: Cryptography and Communications 1.2 (2009), pp. 135–173. DOI: 10.1007/s12095-009-0011-5. IACR: 2008/271 (p. 174).

[SS96] J. P. M. Silva and K. A. Sakallah. GRASP – a new search algorithm for satisfiability. In: ICCAD. 1996, pp. 220–227. DOI: 10.1109/ICCAD.1996.569607 (pp. 57, 188).

[SSA+07] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata. The 128-Bit Blockcipher CLEFIA (Extended Abstract). In: Fast Software Encryption – FSE 2007. Ed. by A. Biryukov. Vol. 4593. LNCS. Springer, 2007, pp. 181–195. DOI: 10.1007/978-3-540-74619-5_12 (p. 125).

[Ste13] M. Stevens. New Collision Attacks on SHA-1 Based on Optimal Joint Local-Collision Analysis. In: Advances in Cryptology – EUROCRYPT 2013. Ed. by T. Johansson and P. Q. Nguyen. Vol. 7881. LNCS. Springer, 2013, pp. 245–261. DOI: 10.1007/978-3-642-38348-9_15 (pp. 166, 215, 216).

[Sti06] D. R. Stinson. Some Observations on the Theory of Cryptographic Hash Functions. In: Designs, Codes and Cryptography 38.2 (2006), pp. 259–277. DOI: 10.1007/s10623-005-6344-y. IACR: 2001/20 (p. 34).

[SY15] Y. Sasaki and K. Yasuda. How to Incorporate Associated Data in Sponge-Based Authenticated Encryption. In: Topics in Cryptology – CT-RSA 2015. Ed. by K. Nyberg. Vol. 9048. LNCS. Springer, 2015, pp. 353–370. DOI: 10.1007/978-3-319-16715-2_19 (p. 42).

[Tay14] C. Taylor. Calico v8. Submission to CAESAR: Competition for Authenticated Encryption. Security, Applicability, and Robustness (Round 1). 2014. URL: http://competitions.cr.yp.to/round1/calicov8.pdf (p. 149).

[TG91] A. Tardy-Corfdir and H. Gilbert. A Known Plaintext Attack of FEAL-4 and FEAL-6. In: Advances in Cryptology – CRYPTO 1991. Ed. by J. Feigenbaum. Vol. 576. LNCS. Springer, 1991, pp. 172–181. DOI: 10.1007/3-540-46766-1_12 (p. 166).

[Tie16] T. Tiessen. Polytopic Cryptanalysis. In: Advances in Cryptology – EUROCRYPT 2016. Ed. by M. Fischlin and J.-S. Coron. Vol. 9665. LNCS. Springer, 2016, pp. 214–239. DOI: 10.1007/978-3-662-49890-3_9. IACR: 2016/160 (p. 63).

[US 77] US National Bureau of Standards. NIST FIPS PUB 46: Data Encryption Standard (DES). National Institute of Standards and Technology (NIST) Federal Information Processing Standards (FIPS) Publication. 1977 (p. 27).

[US 80] US National Bureau of Standards. NIST FIPS PUB 81: DES Modes of Operation. National Institute of Standards and Technology (NIST) Federal Information Processing Standards (FIPS) Publication. 1980 (p. 31).

[Vau02] S. Vaudenay. Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS ... In: Advances in Cryptology – EUROCRYPT 2002. Ed. by L. R. Knudsen. Vol. 2332. LNCS. Springer, 2002, pp. 534–546. DOI: `10.1007/3-540-46035-7_35` (p. 39).

[Vau94] S. Vaudenay. On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER. In: Fast Software Encryption – FSE 1994. Ed. by B. Preneel. Vol. 1008. LNCS. Springer, 1994, pp. 286–297. DOI: `10.1007/3-540-60590-8_22` (pp. 20, 56).

[Vau98] S. Vaudenay. Provable Security for Block Ciphers by Decorrelation. In: Theoretical Aspects of Computer Science – STACS 1998. Ed. by M. Morvan, C. Meinel, and D. Krob. Vol. 1373. LNCS. Springer, 1998, pp. 249–275. DOI: `10.1007/BFb0028566` (pp. 55, 63).

[Vie07] M. Vielhaber. Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack. IACR Cryptology ePrint Archive, Report 2007/413. 2007. IACR: `2007/413` (p. 63).

[vW92] J. H. van Lint and R. M. Wilson. A course in combinatorics. Cambridge University Press, 1992. ISBN: 978-0-521-41057-1 (p. 137).

[Wag99] D. A. Wagner. The Boomerang Attack. In: Fast Software Encryption – FSE 1999. Ed. by L. R. Knudsen. Vol. 1636. LNCS. Springer, 1999, pp. 156–170. DOI: `10.1007/3-540-48519-8_12` (p. 63).

[WC81] M. N. Wegman and L. Carter. New Hash Functions and Their Use in Authentication and Set Equality. In: Journal of Computer and System Sciences 22.3 (1981), pp. 265–279. DOI: `10.1016/0022-0000(81)90033-7` (p. 38).

[WGZ+16] L. Wang, J. Guo, G. Zhang, J. Zhao, and D. Gu. How to Build Fully Secure Tweakable Blockciphers from Classical Blockciphers. In: Advances in Cryptology – ASIACRYPT 2016. Ed. by J. H. Cheon and T. Takagi. Vol. 10031. LNCS. 2016, pp. 455–483. DOI: `10.1007/978-3-662-53887-6_17`. IACR: `2016/876` (p. 41).

[WHF03] D. Whiting, R. Housley, and N. Ferguson. IETF RFC 3610: Counter with CBC-MAC (CCM). Internet Engineering Task Force (IETF) Request for Comments (RFC). 2003. DOI: `10.17487/RFC3610` (pp. 41, 160).

[Win84] R. S. Winternitz. A Secure One-Way Hash Function Built from DES. In: IEEE Symposium on Security and Privacy 1984. IEEE Computer Society, 1984, pp. 88–90. DOI: `10.1109/SP.1984.10027` (p. 28).

[WLF+05] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In: Advances in Cryptology – EUROCRYPT 2005. Ed. by R. Cramer. Vol. 3494. LNCS. Springer, 2005, pp. 1–18. DOI: `10.1007/11426639_1` (pp. 54, 61, 166, 175, 177, 178).

[WP14] H. Wu and B. Preneel. AEGIS v1. Submission to CAESAR: Competition for Authenticated Encryption. Security, Applicability, and Robustness (Round 2). 2014. URL: `http://competitions.cr.yp.to/round1/aegisv1.pdf` (p. 106).

[WSMP11] M. Wang, Y. Sun, N. Mouha, and B. Preneel. Algebraic Techniques in Differential Cryptanalysis Revisited. In: Information Security and Privacy – ACISP 2011. Ed. by U. Parampalli and P. Hawkes. Vol. 6812. LNCS. Springer, 2011, pp. 120–141. DOI: `10.1007/978-3-642-22497-3_9` (p. 51).

[WT85] A. F. Webster and S. E. Tavares. On the Design of S-Boxes. In: Advances in Cryptology – CRYPTO 1985. Ed. by H. C. Williams. Vol. 218. LNCS. Springer, 1985, pp. 523–534. DOI: `10.1007/3-540-39799-X_41` (p. 20).

[WW11] S. Wu and M. Wang. Security Evaluation against Differential Cryptanalysis for Block Cipher Structures. IACR Cryptology ePrint Archive, Report 2011/551. 2011. IACR: `2011/551` (p. 58).

[WWGY14] Y. Wang, W. Wu, Z. Guo, and X. Yu. Differential Cryptanalysis and Linear Distinguisher of Full-Round Zorro. In: Applied Cryptography and Network Security – ACNS 2014. Ed. by I. Boureanu, P. Owesarski, and S. Vaudenay. Vol. 8479. LNCS. Springer, 2014, pp. 308–323. DOI: `10.1007/978-3-319-07536-5_19`. IACR: `2013/775` (p. 130).

[WY05] X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In: Advances in Cryptology – EUROCRYPT 2005. Ed. by R. Cramer. Vol. 3494. LNCS. Springer, 2005, pp. 19–35. DOI: `10.1007/11426639_2` (pp. 54, 61, 166, 175, 177, 178).

[WYY05a] X. Wang, A. C. Yao, and F. Yao. Cryptanalysis on SHA-1. CRYPTO 2005 rump session and NIST – First Cryptographic Hash Workshop 2005. 2005. URL: `http://csrc.nist.gov/groups/ST/hash/documents/Wang_SHA1-New-Result.pdf` (p. 215).

[WYY05b] X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In: Advances in Cryptology – CRYPTO 2005. Ed. by V. Shoup. Vol. 3621. LNCS. Springer, 2005, pp. 17–36. DOI: `10.1007/11535218_2` (pp. 166, 175, 177, 215–217).

[YB14] H. Yu and D. Bai. Boomerang Attack on Step-Reduced SHA-512. In: Information Security and Cryptology – Inscrypt 2014. Ed. by D. Lin, M. Yung, and J. Zhou. Vol. 8957. LNCS. Springer, 2014, pp. 329–342. DOI: `10.1007/978-3-319-16745-9_18`. IACR: `2014/945` (p. 174).

[YHB16] H. Yu, Y. Hao, and D. Bai. Evaluate the security margins of SHA-512, SHA-256 and DHA-256 against the boomerang attack. In: SCIENCE CHINA Information Sciences 59.5 (2016), 052110:1–052110:14. DOI: `10.1007/s11432-015-5389-4` (p. 174).

[YI14] S. Yanagihara and T. Iwata. Type 1.x Generalized Feistel Structures. In: IEICE Transactions 97-A.4 (2014), pp. 952–963. URL: `http://search.ieice.org/bin/summary.php?id=e97-a_4_952` (pp. 107, 111).

[Yuv79] G. Yuval. How to swindle Rabin. In: Cryptologia 3.3 (1979), pp. 187–191. DOI: `10.1080/0161-117991854025` (pp. 25, 53).

[YW09] H. Yu and X. Wang. Distinguishing Attack on the Secret-Prefix MAC Based on the 39-Step SHA-256. In: Information Security and Privacy – ACISP 2009. Ed. by C. Boyd and J. M. G. Nieto. Vol. 5594. LNCS. Springer, 2009, pp. 185–201. DOI: `10.1007/978-3-642-02620-1_13` (p. 173).

[YWH+14] D. Ye, P. Wang, L. Hu, L. Wang, Y. Xie, S. Sun, and P. Wang. PAES v1. Submission to CAESAR: Competition for Authenticated Encryption. Security, Applicability, and Robustness (Round 1). 2014. URL: `http://competitions.cr.yp.to/round1/paesv1.pdf` (p. 107).

[ZD16] R. Zong and X. Dong. Meet-in-the-Middle Attack on QARMA Block Cipher. IACR Cryptology ePrint Archive, Report 2016/1160. 2016. URL: `2016/1160` (p. 98).

[ZMI89] Y. Zheng, T. Matsumoto, and H. Imai. On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses. In: Advances in Cryptology – CRYPTO 1989. Ed. by G. Brassard. Vol. 435. LNCS. Springer, 1989, pp. 461–480. DOI: `10.1007/0-387-34805-0_42` (p. 125).

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.