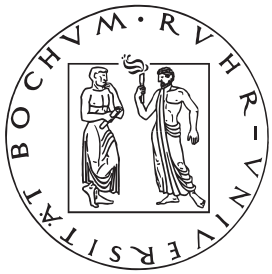
Analysis and Design of Block Cipher Constructions



Dissertation
zur
Erlangung des Grades eines
Doktor-Ingenieurs
der
Fakultät für Elektrotechnik und Informationstechnik
an der Ruhr-Universität Bochum

von

Andrey Bogdanov

Bochum, 2009

Thesis Advisor: **Prof. Dr.-Ing. Christof Paar** Ruhr University Bochum, Germany
External Referee: **Prof. Dr. Vincent Rijmen** K.U.Leuven, Belgium and TU Graz, Austria

Author contact information: `and.bogdanov@gmail.com`

Abstract

This thesis is dedicated to symmetric cryptographic algorithms. The major focus of the work is on block ciphers themselves as well as on hash functions and message authentication codes based on block ciphers. Three main approaches to the cryptanalysis of symmetric cryptographic algorithms are pursued. First, several block cipher constructions are analyzed mathematically using statistical cryptanalysis. Second, practical attacks on real-world symmetric cryptosystems are considered. Finally, novel cryptanalytic techniques using side-channel leakage are studied with applications to block ciphers and message authentication codes.

Differential and linear cryptanalyses are well-known statistical attacks on block ciphers. This thesis studies the security of unbalanced Feistel networks with contracting MDS diffusion with respect to differential and linear cryptanalysis. Upper bounds on the differential trail probabilities and linear probabilities of linear trails in such constructions are proven. It is shown that such unbalanced Feistel networks can be highly efficient and are comparable to many known balanced Feistel network constructions with respect to differential and linear cryptanalysis. Ultra-lightweight substitution-permutation networks with diffusion layers based on the co-design of S-boxes and bit permutations are proposed. This results in lightweight block ciphers and block cipher based compression functions for hash functions designed and analyzed. These constructions have very small footprint and can be efficiently implemented on the majority of RFID tags

This work also studies practical attacks on real-world symmetric cryptographic systems. Attacks are proposed on the KeeLoq block cipher and authentication systems widely used for automotive access control and component identification. Cryptanalysis of the A5/2 stream cipher used for protecting GSM connections worldwide is performed. Linear slide attacks on KeeLoq are proposed resulting in the fastest known attack on the KeeLoq block cipher working for all keys. Severe weaknesses of the KeeLoq key management are identified. The KeeLoq real-world authentication protocols for access control and component identification are also analyzed. A special-purpose hardware architecture for attacking A5/2 is developed that allows for real-time key recovery within one second for different GSM channels. This engine is based on an optimized hardware algorithm for fast Gaussian elimination over binary finite fields.

Finally, this thesis deals with methods of cryptanalysis using side-channel leakage such as power or electromagnetic traces obtained from the attacked implementation of a cryptographic algorithm. Unlike simple and differential side-channel analysis, side-channel collision attacks possess the distinctive feature that they substantially rely on the cryptanalytic properties of the attacked algorithm. Additionally to applying basic side-channel collision attacks to AES-based message authentication codes, this thesis proposes numerous ways of optimizing side-channel collision attacks, including generalized collisions, linear and algebraic collision-based key recovery as well as statistical multiple-differential collision detection methods. In case of AES, these techniques provide considerable improvements and can make side-channel collision attacks more efficient than such state-of-the-art side-channel attacks as stochastic side-channel analysis and template attacks.

Kurzdarstellung

Die vorliegende Dissertation beschäftigt sich mit symmetrischen kryptographischen Algorithmen. Den Schwerpunkt bilden Blockchiffren sowie Hashfunktionen und Message Authentication Codes (MAC), welche auf Blockchiffren basieren. Diese Arbeit umfasst drei Themenkomplexe — verschiedenen Ansätzen der symmetirscher Kryptoanalyse entsprechend: Erstens werden mehrere Blockchiffrenkonstruktionen in Bezug auf statistische Kryptoanalyse evaluiert. Zweitens werden praktische Angriffe auf weit eingesetzte symmetrische Kryptosysteme vorgeschlagen, teilweise mit Beschleunigung in Hardware. Drittens werden neuartige kryptoanalytische seitenkanalbasierte Techniken entwickelt und auf Blockchiffrenkonstruktionen angewendet.

Differentielle und lineare Kryptoanalysen sind bekannte statistische Angriffe auf Blockchiffren. In dieser Dissertation wird die Sicherheit von Unbalanced Feistel Networks (UFN) mit komprimierender MDS-Diffusion in Bezug auf differentielle und lineare Kryptoanalyse untersucht. Obere Schranken für die Wahrscheinlichkeiten der differentiellen Charakteristiken und die linearen Wahrscheinlichkeiten der linearen Charakterisktiken solcher Konstruktionen werden bewiesen. Es wird demonstriert, dass die Effizienz solcher UFNs mit der von Balanced Feistel Networks vergleichbar sein kann. Des Weiteren werden einige kryptographische Lightweight-Konstruktionen designt, welche Substitution-Permutation Networks mit Bitpermutationen verwenden. Als Ergebnis werden Lightweight-Blockchiffren und blockchiffrenbasierte Kompressionsfunktionen für Lightweight-Hashfunktionen entworfen und analysiert. Diese Konstruktionen erfordern nur geringe Chipfläche und lassen sich auf den meisten RFID-Tags effizient implementieren.

In dieser Arbeit werden auch praktische Angriffe auf symmetrische Kryptosysteme betrachtet, am Beispiel der KeeLoq-Blockchiffre und KeeLoq-Authentifizierungssysteme, welche bei der Zutrittskontorolle im Automotive-Bereich weiten Einsatz findet, sowie am Beispiel der A5/2-Stromchiffre, welche weltweit zum Schutz von GSM-Verbindungen benutzt wird. Lineare Slide-Angriffe auf KeeLoq werden vorgeschlagen, was zum schnellsten bekannten für alle Schlüssel funktionierenden Angriff führt. Ernsthafte Schwächen der KeeLoq-Schlüsselverwaltung werden indentifiziert. Die Authentifizierungsprotokolle von KeeLoq sind analysiert. Eine dedizierte Hardwarearchitektur zum Angreifen von A5/2 wird entwickelt, welche Schlüsselbestimmung innerhalb einer Sekunde in Echtzeit für verschiedene GSM-Kommunikationskanäle

erlaubt. Dieser Hardware-Baustein basiert auf einem optimierten Hardwarealgorithmus zur schnellen Gauss-Elimination über binären endlichen Körpern.

Der letzte Themenkomplex dieser Dissertation ist die Seitenkanalanalyse von Blockchiffrenkonstruktionen. Wie einfache und differentielle Seitenkanalanalysen, benutzen die Kollisionsangriffe ebenfalls Seitenkanalinformationen, z.B. Messkurven des Stromverbrauchs oder der elektromagnetischen Abstrahlung der Implementierungen anzugreifender kryptographischer Algorithmen. Bezeichnend für Kollisionsangriffe ist allerdings, dass diese die kryptoanalytischen Eigenschaften der Kryptoalgorithmen wesentlich benutzen. Zusätzlich zur Anwendung von Kollisionsangriffen auf AES-basierte MACs, werden in dieser Arbeit auch zahlreiche Methoden zum Optimieren von Kollisionsangriffen vorgeschlagen. Die Verbesserungen schließen den Begriff der verallgemeinerten Kollisionen, lineare und algebraische kollisionsbasierte Schlüsselbestimmung sowie neue statistische Methoden zur Kollisionsdetektierung ein. Im Falle von AES bringen diese Techniken wesentliche Vorteile und machen Kollisionsangriffe auf vielen Plattformen effizierter als solche etablierte Angriffsmethoden wie stochastische Angriffe und Template-Angriffe.

Acknowledgements

During the 2.5 years of working on this thesis, I got help and support from many people. Without some of them this thesis would not have been possible.

First of all, I would like to thank my advisor Christof Paar for his guidance, useful hints and support throughout my work. He accepted me as a PhD student and allowed me to pursue my own line of research which was not always in the major research track of our group. I express my deep appreciation to Vincent Rijmen, who kindly agreed to be the external referee of the thesis.

I had the chance of working and co-authoring papers with such top-class symmetric cryptologists as Alex Biryukov, Lars Knudsen and Matt Robshaw. I would like to thank them all. Many special thanks go to Gregor Leander: I learned a lot from him while we were working on lightweight cryptographic constructions. I am grateful to other external scientists with whom I have collaborated throughout my research: Dmitry Khovratovich, Ilya Kizhvatov, Andrey Pyshkin, and Yannick Seurin as well as to my colleagues and co-authors at the Ruhr University Bochum: Thomas Eisenbarth, Timo Kasper, Marius Mertens, Jan Pelzl, Axel Poschmann, Andy Rupp, and Christopher Wolf. I am also deeply grateful to our team assistant Irmgard Kühn for her kind way of approaching problems and constant support in all organizational issues. I am thankful to Emmanuel Prouff, Francesco Regazzoni and Carsten Rolfes for discussing some questions related to the topics of the thesis.

I have been employed at escrypt GmbH - Embedded Security all the time during the work on this thesis, which provided me with the unique possibility of gaining lots of experience with applied cryptography in the industry in parallel. I would like to thank my colleagues at escrypt GmbH for that.

Last, but by no means the least, I express my deepest gratitude to my parents and to my wife Oxana for their constant encouragement and support.

Contents

Chapter 1

Introduction

1.1 Motivation

Cryptology comprises two interacting counterparts: cryptography and cryptanalysis. While cryptography deals with the *design* of mechanisms providing certain security goals, cryptanalysis studies *attacks* on such mechanisms, aiming to violate the security goals. The set of security goals is application-specific and can include confidentiality, integrity, authenticity, anonymity, non-repudiation, and freshness, to call only a small fraction of all security goals considered in modern cryptology.

Cryptology has immense influence on today's computer systems. Communicating via virtual private networks and sending encrypted emails in the office, purchasing goods in online shops or filling up electronic tax returns at home - all these and many other everyday actions are unthinkable without sound cryptographic algorithms. Though numerous cryptographic algorithms currently considered practically secure are known, alternative cryptographic algorithms may turn out more efficient, leading to the higher performance and lower costs of cryptographic solutions. Thus, the problem of *evaluating new approaches to construct efficient cryptographic algorithms* remains topical. The problem of *studying novel cryptanalytic techniques* to attack known algorithms and their implementations is probably even more important and is far from being resolved.

As opposed to these rather generic research problems, some areas require individual approach. With the spread of RFID tags in such fields as transportation, logistics, human implants or sensor networks, pervasive computing is becoming a reality. As a matter of fact, these innovative technologies give rise to numerous security concerns such as tag authenticity, privacy and anonymity. According to these new demands, a lot of cryptographic protocols have been developed. Often the designers assume that suitable cryptographic algorithms (typically, ciphers or hash functions) will be deployed on the tags. However, there are only few cryptographic mechanisms capable of being implemented on RFID tags, first of all due to severe area limitations. Therefore, one faces the problem of *designing efficient lightweight ciphers and hash functions*.

Mobile communication (e.g. DECT, UMTS, CDMA2000, GSM) is probably the most massive application field of cryptography. Here the phone and the base station share a secret key to enable the encryption of the communication between them, providing data confidentiality. The adversary may want to violate this security goal by eavesdropping on the communication. Another wide-spread application of cryptographic algorithms is access control (e.g. HITAG, Mifare, KeeLoq), where a fob authenticates itself to the main system using a pre-shared secret. An attack scenario could be to get unauthorized access to the object protected by the access control system. Hence, the problem of the *practical cryptanalysis* of such cryptosystems arises.

1.2 Symmetric Cryptology

This thesis mainly deals with the research problems formulated above. Symmetric cryptographic constructions in general and, more specifically, block ciphers and block cipher based cryptographic constructions form the major object of our research. Figure 1.1 gives a classification of cryptographic algorithms. Figure 1.2 provides a dichotomy of cryptanalytic techniques, first of all as applied to symmetric cryptographic constructions. Constructions and attacks relevant to the topics of this thesis are marked grey in Figures 1.1 and 1.2.

1.2.1 Symmetric Cryptographic Algorithms

A *symmetric cryptographic algorithm* is a transformation that accepts a minimum of two inputs (data and a secret key), and maps them to one output. Such algorithms are called symmetric, since they require the knowledge of the secret key by all communication parties involved. An initialization vector can be an additional input to a symmetric cryptographic algorithm. It is a public value such that that the same input mapped to a different output for different values of the initialization vector with a high probability (encryption diversity).

We distinguish between three basic types of symmetric cryptographic algorithms: block ciphers, stream ciphers, and message authentication codes. Cryptographic hash functions, strictly speaking, do not belong to the symmetric cryptographic algorithms, as they usually do not accept any secret key. However, hash functions can be easily built from block ciphers. Moreover, sound message authentication codes can be built from hash functions. Thus, the properties of hash functions are tightly connected with those of symmetric cryptographic algorithms and are also considered in this thesis. Note that there are also other unkeyed cryptographic algorithms (such as entropy extractors) which are not treated here.

Only a subset of all possible security goals can be attained by symmetric cryptographic algorithms directly (only data confidentiality and data integrity are mentioned below). A much larger set of security goals can be provided by cryptographic protocols based on symmetric cryptographic algorithms. However, there are settings, e.g. server-less key establishment without pre-shared secrets, where symmetric cryptography cannot always achieve security goals required.

Block ciphers and stream ciphers are symmetric cryptographic algorithms providing data confidentiality by encrypting a *plaintext* (input data to the symmetric cryptographic algorithm) to a *ciphertext* (output of the symmetric cryptographic algorithm). For practically secure block and stream ciphers, this means that it should not be practically feasible to recover the plaintext without the knowledge of the secret key. If the key and the initialization vector are fixed, block ciphers and stream ciphers become invertible mappings. Thus, both block and stream ciphers can be used for data encryption, where decryption is a necessary requirement imposed on

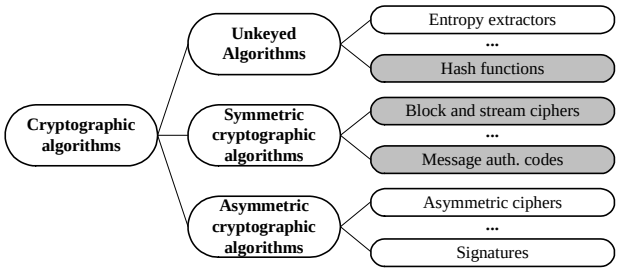


Figure 1.1: Cryptographic algorithms

the transformation.

In a first approximation, a *block cipher* can be thought of as a class of memoryless permutations on blocks of bits. By fixing a key, a concrete permutation is selected out of this class. By applying this permutation to the plaintext, the encryption is performed, giving ciphertext. Applying the inverse permutation to the ciphertext, one obtains the decryption routine, thus, getting the plaintext again.

Stream ciphers represent another type of encryption transformations. Informally speaking, a *stream cipher* is a finite state automaton with internal memory, accepting plaintext and outputting ciphertext symbol by symbol. In other words, plaintext symbols are encrypted one at a time, where the transformation of successive symbols varies during the encryption, e.g. bit by bit or byte by byte.

Message authentication codes are symmetric cryptographic algorithms providing data integrity. The sender computes an *authentication tag* (output of the symmetric cryptographic algorithm) for some message (input of the symmetric cryptographic algorithm) which is then sent to the receiver together with the message. The receiver then computes the authentication tag corresponding to the message he has received, using the same secret key. If the authentication tag calculated by the receiver coincides with the one obtained from the communication channel, the message is considered as integer. Message authentication codes do not have to be invertible, since the sender and receiver perform exactly the same operation of computing the authentication tag.

Basically, hash functions are akin to message authentication codes and they can also be used to provide data integrity. A *hash function* is an algorithm which maps a message of length up to a certain finite limit to a *hash value* of a fixed length. Its major difference to a message authentication code is that it is unkeyed and everyone can compute the hash value for a given message. Hash functions compute a digest for a message. The desired properties of hash functions can include its one-wayness, second preimage resistance or collision resistance. One straightforward way of using hash functions for data integrity protection is to compute the hash value for some data and store the hash value in a secure environment. Then the data does not have to be stored securely. After loading the data back into the secure context, the

hash value can be recomputed and verified against the securely maintained hash value. If the two values coincide, the data is considered as integer. More usually, hash functions are used in conjunction with asymmetric digital signatures as an efficient method of extending the domain of asymmetric algorithms. Merkle signatures are asymmetric digital signatures whose security is based on the security of the underlying hash functions.

1.2.2 Approaches to Symmetric Cryptanalysis

In the *conventional attack* scenario for a symmetric cryptographic algorithm, the adversary typically has reading access to outputs and inputs of the algorithm and can (adaptively) choose different types of inputs to the algorithm. Sometimes the adversary can be additionally allowed to choose relations between keys input to the algorithm. In case of encryption algorithms, the goal of the adversary can be to obtain information about the secret key, to encrypt/decrypt some texts or to distinguish the behavior of the concrete algorithm from the behavior of an ideal algorithm of the same type. In case of message authentication codes, the adversary might additionally wish to generate a valid authentication code for a manipulated message without any knowledge of the key. In case of hash functions, the adversary may want to find two different inputs leading to the same hash value (a collision) or a (second) pre-image for a given output. Other attack goals can be also formulated.

As applied to symmetric cryptographic algorithms, numerous cryptanalytic techniques exist. *Black-box cryptanalysis* aims to use the generic representation of algorithms to construct attacks, based on the size of the internal state, if any, as well as on the lengths of inputs and outputs (time-memory trade-off attacks, time-memory-data trade-off attacks, etc.). *Structural cryptanalysis* includes attacks based on the generic internal structure of algorithms, in other words, on the properties of the high-level building blocks of the algorithms (e.g. slide attacks and reflection attacks are based on the black-box properties of the key-schedule subalgorithm). *Statistical cryptanalysis* uses the individual properties of the low-level building blocks of the algorithm, that is, the properties of the Boolean mappings describing the algorithm transformation (e.g. linear cryptanalysis is based on how well the algorithm transformation can be approximated by a linear mapping; differential cryptanalysis is based on how well differences in the input propagate to output differences). *Algebraic cryptanalysis* constructs and solves linear and nonlinear equations on input, output and key variables using algebraic representations of the algorithm transformation.

Once a theoretical attack flow for a cryptographic algorithm has been identified, the question of efficiently implementing the attack in practice arises. Attack efficiency is a complex notion that can be interpreted in terms of time needed for the attack to succeed, financial costs of the attack, etc. Depending on the concrete tuning of the path identified, these parameters can greatly vary. Hardware-assisted attacks can considerably reduce both the time complexity and the financial effort required for

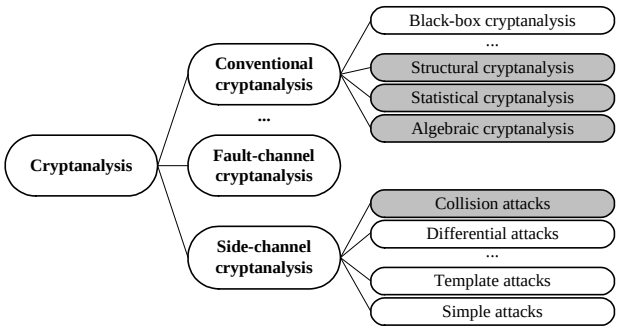


Figure 1.2: Cryptanalytic methods

the attack.

Any cryptographic algorithm has to be implemented in order to be applied. Thus, one can use the features of this physical implementation to cryptanalyze the algorithm. There are two main types of physical cryptanalysis: side-channel cryptanalysis and fault-channel cryptanalysis. As opposed to the conventional cryptanalysis, where the adversary accesses the communication channel to the implementation of a cryptographic algorithm, the *side-channel cryptanalysis* additionally allows the adversary to observe physical parameters of the implementation during algorithm execution. Such physically observable variables can include execution time, electromagnetic radiation, power consumption, optical image or sound field of the implementation. *Fault-channel cryptanalysis* additionally allows the adversary to induce errors to the execution of a cryptographic algorithm. Thus, while side-channel cryptanalysis is passive (monitoring physical observables of the implementation), fault-channel cryptanalysis also requires active physical interaction with the implementation (injecting faults into the implementation by physical means).

Though side-channel cryptanalysis is a rather new area of research, many types of cryptanalysis based on side-channel leakage are already known. *Simple side-channel attacks* try to recognize dependency of a single side-channel trace directly on the internal algorithm variables. *Differential side-channel attacks* use multiple side-channel traces corresponding to different inputs to the algorithm with the same key. The adversary guesses an unknown internal variable and verifies this hypothesis based on the traces, if there is some statistical dependency between points of the traces and the internal variables processed. In *side-channel template attacks*, the adversary has full access to the attacked implementation prior to the attack and can profile it. Based on the profiling data, the adversary classifies the traces acquired from the implementation and determines the value of internal variables. In a *side-channel collision attack*, the adversary detects equal internal variables (collisions) and sets up linear and nonlinear equalities which he then tries to resolve with respect to unknown internal variables of the algorithm.

1.3 Research Contributions and Outline

This thesis is mainly dedicated to the analysis and design of block ciphers. Message authentication codes and hash functions based on block ciphers are also considered in this work. Special-purpose stream ciphers are treated in the context of practical key recovery.

- In Chapter 2, we more formally introduce block ciphers and such block cipher based constructions as message authentication codes and hash functions and provide an introduction to the classical analysis of symmetric constructions, including differential cryptanalysis, linear cryptanalysis and algebraic cryptanalysis. We also discuss the attack models of side-channel cryptanalysis in more detail and outline the basic principles and methods behind simple side-channel analysis, differential side-channel analysis, template attacks and basic collision attacks.
- In Chapter 3, we analyse unbalanced Feistel networks with contracting optimal diffusion with respect to differential and linear cryptanalysis. We prove a lower bound on the number of differentially active S-boxes for some cryptographically interesting constructions of this kind. We also prove an upper bound on the differential trail probability for these constructions, which is not an obvious consequence of the number of active S-boxes, as several distinct single-round differential trails can contribute to one single-round differential. Upper bounds on the linear probability of linear trails of such constructions are also proven by counting the minimal number of linearly active S-boxes. Regarding the number of S-box computations and linear operations required for a given security level against differential and linear cryptanalysis, we demonstrate that Feistel networks with contracting optimal diffusion are comparable to balanced Feistel networks and substitution-permutation constructions. The results of Chapter 3 are published in [40] and [41].
- In Chapter 4, the problem of designing an efficient lightweight block cipher is discussed. As a result, the block cipher PRESENT is proposed providing very small footprint and relatively high throughput in hardware. PRESENT is a substitution-permutation network with diffusion layer being a bit permutation. We prove upper bounds on the probability of a differential trail and on the bias of a linear trail for a class of PRESENT-like SPNs as regards linear and differential cryptanalysis, respectively, based on the co-design of S-boxes and bit permutations. Moreover, we discuss ways of building lightweight hash functions from PRESENT-type substitution-permutation networks. This chapter is based on papers [44] and [45].
- In Chapter 5, attacks on real-world cryptosystems are proposed. First, linear slide attacks on the KeeLoq block cipher widely used for automotive access

control and component identification are described resulting in the fastest known key recovery attack on KeeLoq working for all keys. This is followed by a security analysis of KeeLoq key management and authentication protocols used in KeeLoq access control systems. The chapter is based on papers [36], [38] and [48].

- Chapter 6 presents a hardware-assisted variant of the attack on A5/2 by Barkan, Biham and Keller, resulting in a vast reduction of the time complexity of the attack. Moreover, we extend the attack to the GSM speech traffic channel instead of a specific control channel attacked before. This chapter is based on papers [42] and [46].

- Chapter 7 starts with an extension of the basic side-channel collision attacks to the AES-based message authentication code Alpha-MAC under the assumption that all keyed rounds are perfectly masked. Our collision attack recovers the internal state of Alpha-MAC. It is also demonstrated how to efficiently perform message forgery attacks when the internal state of unkeyed rounds is known. However, the main contributions of Chapter 7 are the improvement of the basic side-channel collision attacks by introducing the idea of generalized internal collisions, the techniques of linear and algebraic key recovery as well as multiple-differential collision detection. Binary and ternary votings are proposed to deal with strong collision detection errors. We discuss the effect of profiling measurements which further decrease the measurement complexity of side-channel collision attacks, making them comparable to template-based stochastic methods. The results of Chapter 7 are published in [26], [37], [39] and [43].

Chapter 2

Basic Concepts

2.1 Block Ciphers

As already mentioned in the previous chapter, block ciphers belong to the class of symmetric encryption algorithms that aim to provide data confidentiality by sharing a secret between communication parties and transforming plaintexts to ciphertexts using this secret in a way that the adversary (possessing no knowledge of the secret) is not able to obtain the plaintext.

Cryptologists recognized the need for the formalization of secrecy in communication systems quite early. Already in 1949, Shannon [238] defined[1] perfect secrecy based on the stochastic notion of mutual information. A cipher provides perfect secrecy if the ciphertext does not give the adversary any additional information about the plaintext. Shannon proved that the entropy of the key in a cipher possessing perfect secrecy has to be at least as high as the entropy of the plaintext. This result implies that such ciphers require a key which has to have at least the length of the plaintext and cannot be reused for different plaintexts. This makes ciphers[2] with perfect secrecy impractical in most settings, where large amounts of data need to be encrypted. Thus, one needs other designs of encryption algorithms and other (probably not that strict) approaches to the definition of security notions. Block ciphers represent one of such more efficient ways to construct encryption algorithms.

2.1.1 Definition

A block cipher can be thought of as a keyed permutation. The key chooses a certain permutation from a class of permutations. For a fixed key, the mapping becomes bijective. More formally, one can give the following definition:

Definition 1 (Block cipher)**.** *A mapping $E : \mathbb{F}_2^b \times \mathbb{F}_2^k \to \mathbb{F}_2^b$ is called a block cipher with block size b bits and key size k bits, if the mapping $E(K, \cdot)$ is a bijection for each $K \in \mathbb{F}_2^k$, that is, if the inverse mapping $E^{-1}(K, \cdot)$ exists with $E^{-1}(K, E(K, x)) = x$ for each $K \in \mathbb{F}_2^k$ and $x \in \mathbb{F}_2^b$.*

The input and output of $E(K, \cdot)$ are called plaintext and ciphertext, respectively. K is referred to as the encryption key.

2.1.2 Some Notions of Security and Adversary Models

A major question that arises in cryptology with respect to different constructions is how secure a specific design is. The answer to this question is often given in terms

[1] These results by Shannon appeared in a classified report in 1946.

[2] The Vernam cipher (also known as one-time pad) fulfils this definition of perfect secrecy. Its design is very simple. If one has to encrypt an l-bit plaintext, an l-bit key is needed. The encryption (decryption) is defined as the bitwise XOR of the plaintext (ciphertext) and the key. If some key bits are reused at least once, no perfect secrecy is guaranteed any more.

of the expected time and memory complexities of concrete attacks and depends on two factors:

- The attack outcome, i.e., what goal the adversary can achieve with a certain attack.
- The adversary model assumed, i.e., what the adversary is allowed to undertake to perform a certain attack.

Attack outcomes. Possible attack outcomes can differ greatly. In a somewhat simplified model, following Knudsen [143], a classification of attack outcomes for block ciphers can be given:

- *Key recovery:* The adversary can determine the encryption key K. This is the most powerful attack outcome.
- *Global deduction:* The adversary can compute $E(K, \cdot)$ or $E^{-1}(K, \cdot)$ without knowing K.
- *Instance deduction:* The adversary can compute $E(K, x)$ or $E^{-1}(K, y)$ without knowing K for some plaintext x or ciphertext y.
- *Information deduction:* The adversary is able to obtain some information (in the information-theoretical sense) about K or about unknown inputs to $E(K, \cdot)$ or $E^{-1}(K, \cdot)$.
- *Distinguishing:* The adversary can distinguish $E(K, \cdot)$ from a randomly drawn permutation.

Note that the attack outcomes mentioned assume that the block cipher is used for encryption only. In practice block ciphers are often applied as building blocks for other cryptographic constructions such as hash functions, where the adversary frequently knows the key or even has substantial control over the key. In this context one should speak about distinguishing the 2^k permutations $E(K, \cdot)$ building the block cipher E as a whole from a set of 2^k permutations randomly drawn from all $2^b!$ possible permutations as the most generic attack outcome.

Adversary models. There also exist various adversary models that are classified with respect to the operations on the inputs and outputs of the cipher (i.e. plaintext, ciphertext and key) the adversary is allowed to perform. Normally, one distinguishes between such operations as reading access (known values), writing access (chosen values) and adaptive writing access (adaptively chosen values). So the main adversary models of attacks on block ciphers are the following:

- *Ciphertext-only attacks:* The adversary only knows ciphertexts, without any access to plaintexts. If a block cipher is vulnerable to this type of attacks by assuming any distribution of the inputs, it is considered especially weak, as it means that the permutation provided by the cipher is extremely non-random.
- *Known plaintext attacks:* The adversary knows the plaintexts processed by the cipher and has reading access to the corresponding ciphertexts. Linear cryptanalysis [177] represents an example of such an attack.
- *Chosen plaintext attacks:* The adversary can choose plaintexts to be encrypted by the cipher prior to the attack and has reading access to the corresponding ciphertexts during the attack. The most known attack of this kind is differential cryptanalysis [23].
- *Chosen ciphertext attacks:* The adversary can choose ciphertexts to be decrypted by the cipher prior to the attack and has reading access to the resulting plaintexts during the attack.
- *Adaptive chosen plaintext attacks:* The adversary can choose plaintexts during the attack (based on the intermediate results of the attack) and has reading access to the corresponding ciphertexts.
- *Adaptive chosen ciphertext attacks:* The adversary can choose ciphertexts during the attack (based on the intermediate results of the attack) and has reading access to the corresponding plaintexts.
- *Related key attacks:* The adversary can encrypt plaintexts/decrypt ciphertexts with the attacked key and with keys related to it [15], e.g. by choosing bit differences between the original key and the related keys.

Attacks from these classes can be combined with each other, often resulting in more efficient approaches. For example, boomerang attacks [251] combine chosen plaintext and adaptive chosen ciphertext scenarios.

Note that other adversary models can be considered for block ciphers, e.g. known and chosen key attacks [148]. These are however mainly relevant when the block cipher is used to construct a hash function where the adversary can have knowledge of the key or even full control over the key.

2.1.3 Ideal Block Cipher and Black-Box Attacks

By definition, a block cipher is a set of 2^k permutations on b-bit words. There are exactly $2^b!$ distinct permutations of b-bit words. Therefore, it seems natural to require an ideal block cipher to be a set of 2^k permutations randomly drawn out of these $2^b!$ permutations:

Definition 2 (Ideal block cipher)**.** *A block cipher E is called ideal, if E is defined by assigning a random element of the symmetric group $S_{\mathbb{F}_2^b}$ to each of the 2^k permutations $E(K, \cdot)$.*

Even ideal block ciphers do exhibit some properties that might be undesirable for a perfect encryption scheme (cf. the Vernam cipher), being susceptible to black-box attacks, that is, to attacks that only base on the generic parameters of the block cipher (block length and key length) and do not exploit any internal structure of the cipher. Here we briefly mention some of black-box attacks on block ciphers:

- *Brute-force attack:* In such attacks, the adversary tries all possible keys and tests them based on a number of plaintext-ciphertext pairs. The number of pairs needed can be derived from the unicity distance [179]. The computational complexity of this attack is about 2^k operations.

- *Ciphertext-collision attack:* In the first stage, the adversary obtains $2^{b/2}$ random plaintext-ciphertext pairs for the attacked key (known-plaintext scenario) and tabulates them. In the second stage, he observes $2^{b/2}$ random ciphertexts (ciphertext-only scenario). Due to the birthday paradox, one of these ciphertexts coincides with one of the pre-tabulated ciphertexts with a high probability. Then the adversary immediately gets the corresponding plaintext as it is tabulated. In a version of this attack, the adversary can tabulate all 2^b plaintext-ciphertext pairs. Then he is able to read all plaintexts without any knowledge of the key.

- *Time-memory trade-off attack [112], [201]:* It is probably the most efficient type of black-box attacks on block ciphers known today. It is a known-plaintext attack which trades memory against attack time. Hellman's time-memory trade-off attack finds a key in $2^{2k/3}$ operations with $2^{2k/3}$ words of memory by using 2^k precomputations to generate the memory entries. These can however be reused to attack multiple keys.

- *Key-collision attacks [14]:* In this attack proposed by Biham, the adversary uses the birthday paradox to determine the key. It is a chosen-plaintext attack. The idea is to encrypt a set of plaintexts (the exact required number can again be computed using unicity distance) with $2^{k/2}$ random known keys first. These keys and the resulting ciphertexts are stored. Then the same plaintext set is encrypted with $2^{k/2}$ random unknown keys. With a high probability, one of these keys will coincide with one of the known keys, for which ciphertexts have been precomputed, resulting in the same ciphertext values.

- *Related-key cube-based attacks [56]:* In the related-key adversary model all block ciphers are susceptible to a variant of the cube attack [88] requiring $\frac{2}{k}2^k$ operations, only $\log k$ memory words and some negligible precomputations for key recovery.

2.1.4 Secure Block Ciphers

From the practical point of view, an ideal cipher as defined above is extremely difficult to implement for most practically relevant block sizes: For a given key, one randomly drawn permutation has to be stored. For the usual block size of $b = 128$ bit, the storage of such a permutation is practically infeasible.

Thus, more structured (and, therefore, less random) assigning of permutations to key values has to be used. As a rule, to build a block cipher, a set of permutation generators is taken and parameterized by the key. This leads to the necessity of a more loose definition of a secure block cipher:

Definition 3 (Secure block cipher)**.** *A block cipher E is called secure, if no attacks on E exist with both expected computational and data complexities lower than the corresponding complexities of any black-box attack.*

Note also that some attacks have considerable memory complexities by requiring large storage volumes and lots of memory access operations. The definition above ignores this additional complexity. Moreover, the question of to what extent the attack is parallelizable is not covered by the definition. However, the definition emphasizes the importance of computational complexity for evaluating attacks.

2.1.5 Iterative Block Ciphers

Today, the most efficient block ciphers are of iterative nature. The idea of an iterative block cipher is to apply several (possibly distinct) keyed permutations one by one to build the block cipher. The superposition of these simplier permutations called *rounds* forms a more complex permutation, the resulting block cipher. As a rule the round transformations are similar in order to favour efficient implementations. Figure 2.1 shows a graphical representation of iterative block ciphers.

Definition 4 (Iterative block cipher)**.** *A block cipher E is called an iterative block cipher with r rounds if for each key it can be represented as a composition of keyed round permutations, that is, if for each $K \in \mathbb{F}_2^k$:*

$$E(K, \cdot) = f_r(K_r, \cdot) \circ f_{r-1}(K_{r-1}, \cdot) \circ \cdots \circ f_2(K_2, \cdot) \circ f_1(K_1, \cdot),$$

where $\circ$ denotes the superposition of permutations, $f_i(K_i, \cdot) : \mathbb{F}_2^b \to \mathbb{F}_2^b$ are key-dependent round permutations, K_i are round subkeys derived from the cipher key K using a key schedule algorithm

$$g : K \mapsto (K_1, K_2, \ldots, K_{r-1}, K_r).$$

An iterative block cipher with all equal round transformations $f_i = f$ is called an *iterated block cipher*. One often speaks about an iterated block cipher, even if the

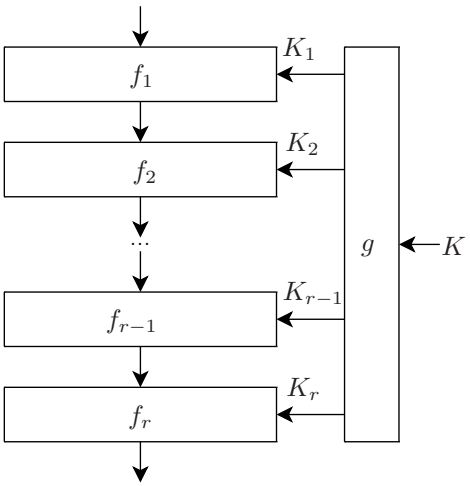


Figure 2.1: Iterative block cipher

round transformations in the first and/or the last round are not identical with the other ones.

A *key-alternating block cipher* [76] is an iterative block cipher with special-type round function f_i:

$$f_i(K_i, x) = f_i'(K_i \oplus x),$$

where $f_i'(\cdot)$ is a permutation not dependent on the round key. Thus, the round transformations of key-alternating block ciphers have very simple dependency on the round key that is merely added to the input modulo 2.

Iterative ciphers can be built in many different ways. Probably the two most wide-spread approaches are balanced Feistel networks (BFNs) and substitution-permutation networks (SPNs). While the former U.S. encryption standard (DES – Data Encryption Standard) [82] is a BFN, the current one (AES – Advanced Encryption Standard) [1] is an SPN architecture. DES is an iterated non-key-alternating block cipher. AES is a key-alternating block cipher.

There are also other approaches to the design of iterative block ciphers. The Lai-Massey scheme is followed by IDEA [157]. Generalized Feistel schemes have also been used for block cipher design [241]. Rather surprisingly, unbalanced Feistel schemes have not been that popular though exhibiting good security properties. Chapter 3 of this thesis studies unbalanced Feistel schemes with contracting MDS diffusion.

2.1.6 Block-Cipher Based Hash Functions

In this thesis, hash functions are considered in the context of block cipher security only. Moreover, only hash functions based on block ciphers are treated. So we briefly

recall the main ideas behind hash functions and mention some basic ways to build a hash function from a block cipher.

Definition 5 (Unkeyed hash function)**.** *A hash function χ with output length m bit and message set M is a mapping from a message set M to $\mathbb{F}_2^m$:*

$$\chi : M \to \mathbb{F}_2^m.$$

M can be the set $\mathbb{F}_2^*$ of all bit strings of a finite length or its subset.

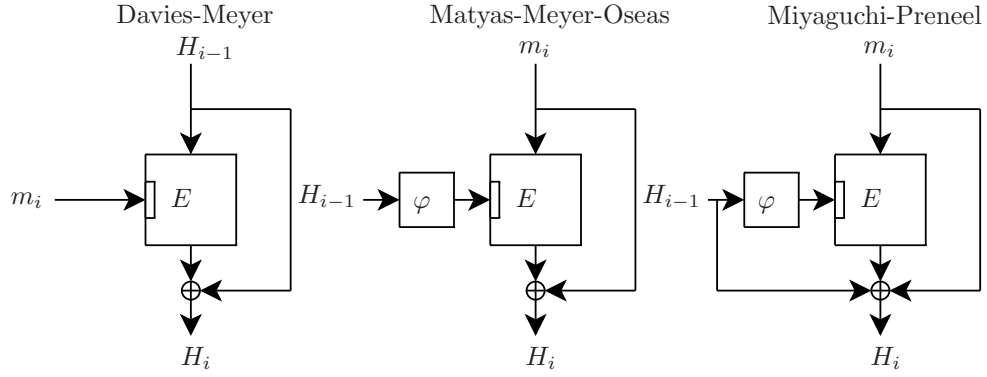
Definition 6 (Ideal hash function)**.** *A hash function χ with output length m bit and message set M is called ideal if it is defined by choosing a random mapping from M to $\mathbb{F}_2^m$.*

Note that this definition of an ideal hash function allows obviously insecure functions to be an ideal hash function (e.g. a function with a trivial set of images). To cope with this, one can either declare that this event is extremely improbable or consider a sort of Kolmogorov complexity [168] for discrete functions.

If a hash function is considered to be secure, it is expected to have preimage resistance (one-way property), second preimage resistance (weak collision resistance) and collision resistance (strong collision resistance). Compromising these properties for a secure hash function should require at least 2^m, 2^m and $2^{m/2}$ operations, respectively.

There are numerous ways to build an unkeyed hash function from a block cipher. The works [218] and [35] describe several straightforward possibilities where the maximum length of the hash value is equal to the block length of the block cipher used. The three most known constructions of this type (namely, Davies-Meyer, Matyas-Meyer-Oseas and Miyaguchi-Preneel constructions) are presented in Figure 2.2. For these constructions there are security proofs under the assumption that the underlying block cipher is ideal.

In double block length hash functions, a block cipher with an n-bit block is used to compute a hash value of length $m = 2n$. Several constructions of this type are known. MDC-2 [68] was filed by IBM in 1987 and uses two parallel invocations of the Matyas-Meyer-Oseas hash function. See [147] for some cryptanalysis. ABREAST-DM [158] and Parallel-DM [122] are alternatives to MDC-2 from early 1990s exhibiting some distinctive features. More recent double block length hash functions were proposed by Nandi et al [190] in 2005 and Hirose [120] in 2006. See Figure 2.3 for the details of Hirose's construction. The work [214] proposes several further ways of building double block length hash functions. The idea can be generalized to a multiple block length hash function (see e.g. [214] for the framework and [45] for an example of triple block length hash function), though such generalized constructions quickly become less efficient. That is, other design approaches are needed for long-output hash functions. Chapter 4 of this thesis provides a more detailed study of constructing lightweight hash functions with long outputs.

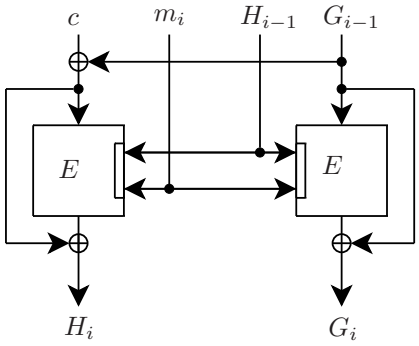


$H_0 = IV$
$H_i = E(m_i, H_{i-1}) \oplus H_{i-1}$
$H = \tau(H_t)$

$H_0 = IV$
$H_i = E(\varphi(H_{i-1}), m_i) \oplus m_i$
$H = \tau(H_t)$

$H_0 = IV$
$H_i = E(\varphi(H_{i-1}), m_i) \oplus m_i \oplus H_{i-1}$
$H = \tau(H_t)$

Figure 2.2: Hash functions from block ciphers (single block length hash functions). IV is an n-bit initialization vector. φ denotes a proper mapping from $\mathbb{F}_2^k$ to $\mathbb{F}_2^n$ to adjust the intermediate hash value to the key size. τ is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ to obtain the m-bit hash value H. $m_1, \ldots, m_t$ is the input message which has been appropriately padded. Note that $m_i \in \mathbb{F}_2^k$ for Davies-Meyer and $m_i \in \mathbb{F}_2^n$ for Matyas-Meyer-Oseas and Miyaguchi-Preneel



$H_0 = IV_H, G_0 = IV_G$
$G_i = E(H_{i-1}||m_i, G_{i-1}) \oplus G_{i-1}$
$H_i = E(H_{i-1}||m_i, G_{i-1} \oplus c) \oplus G_{i-1} \oplus c$
$H = \tau(H_t||G_t)$

Figure 2.3: Double block length hash function due to Hirose [120]. This construction requires $k = n + b$, where b is the bit size of each input message chunk $m_i \in \mathbb{F}_2^b$. $m_1, \ldots, m_t$ is the input message which has been appropriately padded. IV_H and IV_G are some n-bit initialization vectors. c denotes a non-zero n-bit constant. τ is a mapping from $\mathbb{F}_2^{2n}$ to $\mathbb{F}_2^m$ to obtain the m-bit hash value H

2.1.7 Block-Cipher Based MACs

Message authentication codes (MACs) can be viewed as a generalization of hash functions: A MAC can be thought of as a family of hash functions. Here we provide a short introduction to the main ideas behind MACs and recall some block-cipher based MACs.

Definition 7 (Message authentication code, MAC)**.** *A message authentication code μ with output length m bit, key length k bit and message set M is a mapping from the key space $\mathbb{F}_2^k$ and the message set M to $\mathbb{F}_2^m$, that is:*

$$\mu : \mathbb{F}_2^k \times M \rightarrow \mathbb{F}_2^m.$$

M can be the set $\mathbb{F}_2^*$ of all bit strings of finite lengths or its subset.

Definition 8 (Ideal MAC)**.** *A message authentication code μ with output length m bit, key length k bit and message set M is called ideal if it is defined by assigning a random mapping from M to $\mathbb{F}_2^m$ to $\mu(K, \cdot)$ for each key value $K \in \mathbb{F}_2^k$.*

If a MAC is considered secure, it is expected that there is no key recovery attacks with complexity less than 2^k and no forgery attack (generation of a message-output pair which has not been produced by someone knowing the key) with complexity significantly less than $min(2^k, 2^m)$ [179].

There are several known ways to build a MAC from a block cipher. The traditional method to compute a MAC value has been the CBC mode (Cipher Block Chaining) which is often used for data encryption as well. The CBC mode was invented by IBM in 1976 [91]. CBC-MAC is proven secure for messages of a fixed length under the assumption that the underlying block cipher is ideal [128]. For messages of variable lengths, CBC-MAC becomes significantly less secure due to the XOR-attack and its extensions. To cope with this problem, EMAC (Encrypted CBC-MAC) was proposed by Petrank and Rackoff in 2000 [212] (see Figure 2.4). It is basically CBC-MAC that uses an additional call to the block cipher with an additional key at the end. In 2000 Black and Rogaway proved XCBC [34], a MAC construction without any additional cipher calls using three different keys, to be secure in the ideal block cipher model. In 2003, based on XCBC, Iwata and Kurosawa proposed a two-key MAC called TMAC [156] and a one-key MAC called OMAC [129]. TMAC and OMAC are also proven secure in the ideal cipher model, though they only use two keys or one key, respectively. A derivation of OMAC is the only NIST recommended authentication mode of operation for block ciphers under the name of CMAC [197]. OMAC1, one of the two variants of OMAC, is presented in Figure 2.5. Many other block-cipher based MAC schemes are known. RMAC [130] is a randomized MAC provably secure beyond the birthday barrier. PMAC [33] is a MAC scheme which is well parallelizable and proven secure in the ideal cipher model.

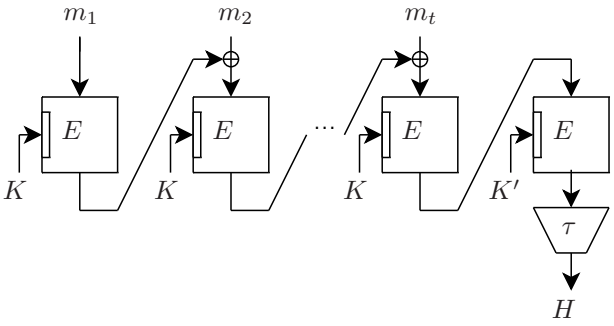


Figure 2.4: EMAC [212]. τ is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ to obtain the m-bit MAC value H. $m_1, \ldots, m_t$ is the appropriately padded input message, $m_i \in \mathbb{F}_2^n$. K and K' are two independent secret keys

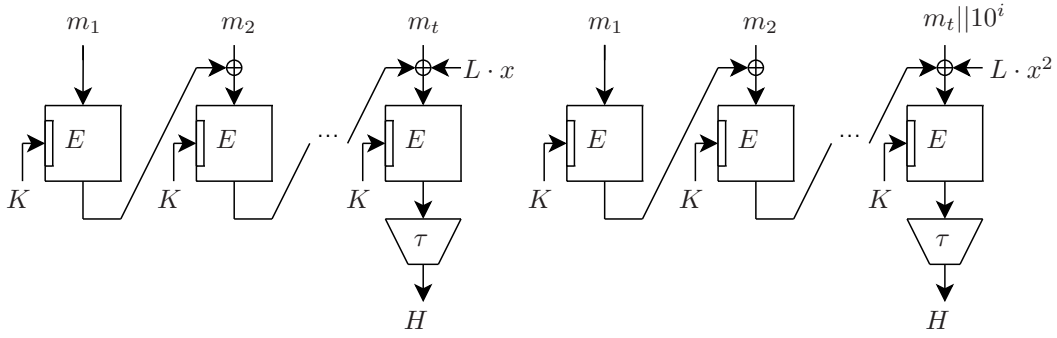


Figure 2.5: OMAC1 [129]. τ is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ to obtain the m-bit MAC value H. $m_1, \ldots, m_t$ is the *non-padded* input message, $m_i \in \mathbb{F}_2^n$ for $i = 1, \cdots, t-1$. If m_t is n-bit, no padding is required and the MAC value is computed according to the left-hand procedure. If m_t is shorter than n-bit, it is padded with 1 and i 0's and the MAC value to computed according to the right-hand procedure. L is the encrypted zero plaintext block, $L = E(K, 0)$. $L \cdot x$ and $L \cdot x^2$ are the multiplications of L by x and x^2, respectively, in $\mathbb{F}_{2^n} \simeq \mathbb{F}_2[x]/g(x)$, where $g(x)$ is an irreducible polynomial over $\mathbb{F}_2$

Another approach to the design of block-cipher based MACs is to directly build upon the round transformations of a block cipher rather than to use the block cipher as a whole. This approach was followed by Daemen and Rijmen in [78] and resulted in highly efficient constructions Alpha-MAC and Pelican-MAC. However, a message forgery attack against Alpha-MAC has been recently shown by Yuan et al [261] which indicates that this design approach still requires some research to be invested. In Chapter 7 of this thesis, a collision attack on Alpha-MAC is provided using side-channel leakage.

2.2 Attacks on Block Ciphers

2.2.1 Differential Cryptanalysis

Differential cryptanalysis was published in the scientific literature in 1990 by Biham and Shamir with applications to DES [23]. However, it seems to have been known to the designers of DES at IBM in early 1970s [67]. As the name suggests, the main idea of differential cryptanalysis is to exploit correlations between differences in the inputs and outputs of a non-ideal block cipher to recover the key. It is a chosen-plaintext attack.

In the differential cryptanalysis of iterative block ciphers, the attacker looks at the input difference $\Delta x_1 = x_1 \oplus x'_1$ for some input pair x_1 and x'_1 and at the difference $\Delta x_i = x_i \oplus x'_i$ after the first $i-1$ rounds[3] $f_{i-1} \circ \cdots \circ f_1$ for some pair of intermediate states x_i and x'_i. Let Δx_1 propagate to Δx_i for different input pairs with probability p. If $f_{i-1} \circ \cdots \circ f_1$ were an ideal cipher, p would be about 2^{1-n}. That is, if p is significantly higher[4] than 2^{1-n}, one can distinguish $f_{i-1} \circ \cdots \circ f_1$ from an ideal cipher by encrypting a number of plaintext pairs with input difference Δx_1 and checking how often the output difference Δx_i occurs. The pair of differences $(\Delta x_1, \Delta x_i)$ is called an $(i-1)$-round differential. p is called the differential probability for $(\Delta x_1, \Delta x_i)$.

In order to turn this differential distinguisher for $f_{i-1} \circ \cdots \circ f_1$ into a key-recovery attack on the full cipher $f_r \circ \cdots \circ f_1$, one can guess (the relevant parts of) the round subkeys $K_r, \ldots, K_i$ of rounds r down to i and decrypt the ciphertexts x_{r+1} and x'_{r+1} corresponding to the inputs x_1 and x'_1 with these subkeys. If the guess is correct, this partial decryption yields the correct intermediate values x_i and x'_i and the differential $(\Delta x_1, \Delta x_i)$ appears with the differential probability p. Otherwise, if the guess is incorrect, it is expected that the differential $(\Delta x_1, \Delta x_i)$ is followed with some other probability p_r. If p is higher than p_r, the correct guess regarding $K_r, \ldots, K_i$ can be identified against the background of incorrect key hypotheses by counting

[3] As f_j are permutations, we denote their subsequent application in the reverse order.

[4] If p is significantly lower than 2^{1-n}, an impossible differential attack can be possible, see the extensions of the differential cryptanalysis below.

the number of input pairs following the differential $(\Delta x_1, \Delta x_i)$ after $i-1$ rounds for each key hypothesis. The right key hypothesis is then expected to correspond to a counter having one of the highest values.

The quality of a differential can be measured using the notion of the signal-to-noise ratio $S_N = p/p_r$. The success of the differential attack is frequently evaluated with respect to the advantage provided by the attack: If m key bits are guessed in the differential attack and the correct value is ranked among the highest t of 2^m hypotheses, the attack is said to obtain an $(m-\log_2 t)$-bit advantage over exhaustive search [234] (complexity reduction by a factor of $2^{m-\log_2 t}$). The work [234] also contains the following results on the success probability and data complexity of differential cryptanalysis. Under the assumption that the key counters used for key recovery are independent and that they are identically distributed for all wrong key hypotheses, one has the following success probability P_S for a differential attack with an a-bit advantage and N pairs of plaintexts and ciphertexts:

$$P_S = \Phi\left(\frac{\sqrt{pNS_N} - \Phi^{-1}(1-2^{-m})}{\sqrt{S_N+1}}\right),$$

where Φ is the cumulative distribution function of the standard normal distribution. This result can be reformulated in terms of data complexity for a differential attack to succeed with probability P_S:

$$N = \frac{\sqrt{S_N+1}\Phi^{-1}(P_S) + \Phi^{-1}(1-2^{-m})^2}{S_N} p^{-1}.$$

That is, the data complexity of differential cryptanalysis is proportional to $1/p$. As the assumption that the counters are independent is rather unrealistic, these results provide good approximations for high values of P_S and can only be used as a rough estimation for small values of P_S [234].

For many ciphers it is not easy to find a differential $(\Delta x_1, \Delta x_i)$ with high differential probability p directly. So one uses the notion of a differential trail to identify a good differential. An $(i-1)$-round differential trail is a sequence of i differences $(\Delta x_1, \Delta x_2, \ldots, \Delta x_{i-1}, \Delta x_i)$. That is, unlike differential, the differential trail cares about how the input difference propagates from round to round. The difference propagation through linear components (e.g. diffusion layers) of a block cipher is of deterministic nature: A given input difference propagates to a certain output difference with probability 1. At the same time, the difference propagation through nonlinear components (e.g. S-boxes) is of probabilistic nature: A given input difference results in a certain output differences with some probability.

Generally speaking, the probability of a given differential can depend on the value of the key as well as on the concrete input and intermediate values. However, for practical cryptanalysis, it is often assumed that these types of dependency are negligible: Markov cipher assumption, hypothesis of stochastic equivalence and hypothesis of independent round keys [159]. For some ciphers these assumptions lead

to plausible results. Nonetheless, there are actual ciphers where these assumptions are either not necessary for some results to hold (e.g. the hypothesis of independent round keys with respect to key alternating ciphers such as AES-type constructions) or are not valid (e.g. the hypothesis of stochastic equivalence with respect to IDEA [157]) [79].

The differential $(\Delta x_1, \Delta x_i)$ can be seen as the set of all possible differential trails $(\Delta x_1, \ldots, \Delta x_i)$ with the same input and output differences. The probability of a differential for a given key can be shown to be the sum of the probabilities of all these individual differential trails for this key. The probability of a differential trail for a given key is often assumed to be approximated by $\prod_{j=1}^{i-1} p_j$, where p_j is the probability of this differential trail to hold in the individual round j for this key. Expected probabilities of differential trails and differentials averaged over all encryption keys or all round keys are also often considered.

Several techniques extending the differential cryptanalysis are known. Truncated differential cryptanalysis by Knudsen [144] works with truncated differentials that are differentials where only parts of input and input differences are taken care of. Impossible differential cryptanalysis introduced in [51] attempts to make use of impossible differentials (which are differentials having a very low or even a zero probability [16]) rather than highly probable differentials. Higher-order differential cryptanalysis by Knudsen [144] treats multiple differences simultaneously by arranging them in several levels of recursion. Chapter 3 of this thesis applies differential cryptanalysis to some unbalanced Feistel networks. In Chapter 4 we study differential trails of special-type SPNs.

2.2.2 Linear Cryptanalysis

Linear cryptanalysis as applied to DES was proposed by Matsui in 1993 [177]. However, similar ideas were published by Shamir [236] in 1985 as well as Tardy-Corfdir and Gilbert [249] in 1991. Linear cryptanalysis uses linear approximations of block ciphers to perform key recovery. It is a known-plaintext attack.

The main idea behind linear cryptanalysis is to search for a *linear approximation* to the first $i-1$ rounds $f_{i-1} \circ \cdots \circ f_1$ of the cipher holding with probability p:

$$\Gamma_1 \diamond x_1 \oplus \Gamma_i \diamond x_i = 0,$$

where $\diamond$ denotes the scalar product[5] over $\mathbb{F}_2$. Γ_1 and Γ_i are called input and output *linear masks*, respectively. If $f_{i-1} \circ \cdots \circ f_1$ were an ideal block cipher, this relation would hold with probability about $1/2$. If p deviates from $1/2$ significantly, this linear approximation can be used to distinguish $f_{i-1} \circ \cdots \circ f_1$ from an ideal cipher by encrypting random plaintexts x_1 to outputs x_i and checking how often the linear relation between them holds. The quality of a linear approximation can

[5]For $a = (a_n, \ldots, a_1) \in \mathbb{F}_2^n$ and $b = (b_n, \ldots, b_1) \in \mathbb{F}_2^n$ with $a_i, b_i \in \mathbb{F}_2$, $a \diamond b = \bigoplus_{i=1}^{n} a_i b_i$

be measured in terms of the *bias* $\epsilon = |p - 1/2|$ of this linear approximation. The bias characterizes the distance of the cipher from an ideal cipher with respect to the linear approximation.

Again, this distinguisher can be turned into a key recovery attack on the full cipher $f_r \circ \cdots \circ f_1$. To do this, the attacker guesses the round subkey bits needed to determine the bits of x_i given by the output mask Γ_i from the ciphertext x_{r+1}. For each of these key hypotheses, the bias is estimated. If enough plaintext-ciphertext pairs are available, the correct key hypothesis is likely to correspond to one of the highest biases estimated.

The work [234] evaluates the success probability of linear cryptanalysis with respect to the advantage over exhaustive search. More precisely, the probability P_S that a m-bit or higher advantage is attained using linear cryptanalysis with N plaintext-ciphertext pairs can be estimated as follows. Under the assumption that the probability for a linear approximation to hold is independent for each key hypothesis and is equal to $1/2$ for wrong key hypotheses, for sufficiently large m and N one has:

$$P_S = \Phi\left(2\sqrt{N}\epsilon - \Phi^{-1}(1 - 2^{-m-1})\right).$$

This translates to the following estimation on the number of data needed to attain success probability P_S:

$$N = \left(\frac{\Phi^{-1}(P_S) + \Phi^{-1}(1 - 2^{-m-1})}{2}\right)^2 \epsilon^{-2}.$$

That is, the data complexity is proportional to $1/\epsilon^2$. The experimental verification [234] shows that these estimations for linear cryptanalysis are quite precise and indicates that the assumptions made seem realistic enough.

In order to find efficient linear approximations, the concept of a linear trail is used. An $(i-1)$-round *linear trail* is a sequence of i linear masks $(\Gamma_1, \Gamma_2, \ldots, \Gamma_{i-1}, \Gamma_i)$. Each of the round linear approximations

$$\Gamma_j \diamond x_j \oplus \Gamma_{j+1} \diamond x_{j+1} = 0$$

with bias ϵ_i has to hold in order to the full linear trail to hold. Summing up all $i - 1$

linear relations, one obtains the following linear approximation over $i-1$ rounds:

$$\begin{array}{lcll}
\Gamma_1 \diamond x_1 & \oplus & \Gamma_2 \diamond x_2 & \oplus \\
\Gamma_2 \diamond x_2 & \oplus & \Gamma_3 \diamond x_3 & \oplus \\
\Gamma_3 \diamond x_3 & \oplus & \Gamma_4 \diamond x_4 & \oplus \\
\ldots & & & \\
\Gamma_{i-3} \diamond x_{i-3} & \oplus & \Gamma_{i-2} \diamond x_{i-2} & \oplus \\
\Gamma_{i-2} \diamond x_{i-2} & \oplus & \Gamma_{i-1} \diamond x_{i-1} & \oplus \\
\Gamma_{i-1} \diamond x_{i-1} & \oplus & \Gamma_i \diamond x_i & \\
= & & & \\
\Gamma_1 \diamond x_1 & \oplus & \Gamma_i \diamond x_i & \\
= 0 & & &
\end{array}$$

The pair of linear masks (Γ_1, Γ_i) specifies the so-called *linear hull* consisting of all linear trails $(\Gamma_1, \ldots, \Gamma_i)$ with the same input and output linear masks. The probabilities of individual linear approximations to hold are again deterministic for linear and probabilistic for nonlinear cipher components.

The *linear probability* of a linear approximation with bias ϵ is defined as $4\epsilon^2$. If ϵ_j is the bias of the round linear approximation in round $j = 1, \ldots, i-1$, its linear probability is $q_j = 4\epsilon_j^2$ that is in general dependent on the key value. For a given key, the linear probability of the full linear trail can be computed as $q = \prod_{j=1}^{i-1} q_j$ (*piling-up lemma* [177]). It can be demonstrated for many ciphers that the expected linear probability of a linear approximation averaged over all round keys is the sum of the expected linear probabilities of all relevant linear trails averaged over all round keys [200]. For key-alternating ciphers, even a stronger claim can be proved: The expected linear probability of a linear approximation averaged over all round keys is the sum of the linear probabilities of all relevant trails which are then independent of the key [79].

There are several extensions of linear cryptanalysis. Chosen plaintexts can be used to improve the complexity of linear cryptanalysis [142]. Linear cryptanalysis with multiple linear approximations attempts to obtain more information by considering several linear approximations simultaneously [134]. Nonlinear approximations can be used to generalize the linear cryptanalysis and can potentially lead to the recovery of additional information [149]. Chapter 3 of this thesis provides a lower bound on the number of linearly active S-boxes in an unbalanced Feistel network construction. Chapter 4 proves some bounds on the linear trails for special-type SPNs. Chapter 5 uses linear cryptanalysis in combination with slide attacks to cryptanalyze the KeeLoq block cipher.

2.2.3 Algebraic Cryptanalysis

The main idea of algebraic cryptanalysis in general is as follows: Given input and output of a block cipher, it is often possible to construct a system of multivariate

nonlinear equations with respect to the key bits that can be then recovered by solving this system. The efficiency of algebraic cryptanalysis with respect to a certain block cipher is defined by the structure of this system of nonlinear equations and, therefore, by the complexity of solving this system. The problem of solving generic systems of multivariate nonlinear equations over finite fields is NP-hard [103]. However, there are obviously instances where it is not the case. This discrepancy combined with the fundamental complexity of rigourous analysis sometimes leads to certain controversy regarding the validity of some proposed algebraic attacks.

To put the principles of algebraic cryptanalysis somewhat more formal, consider a finite field $\mathbb{F} = \mathbb{F}_q$ of characteristic 2 and a multivariate polynomial ring $\mathbb{F}[x_1, \ldots, x_l]$. Then the key might be recovered by solving an equation system

$$\begin{cases} \phi_1(x_1, \ldots, x_l) = 0 \\ \ldots \\ \phi_t(x_1, \ldots, x_l) = 0, \end{cases}$$

where $\phi_1, \ldots, \phi_t$ is a set of polynomials from $\mathbb{F}[x_1, \ldots, x_l]$ that generates an ideal $I = \langle \phi_1, \ldots, \phi_t \rangle \lhd \mathbb{F}[x_1, \ldots, x_l]$. If this equation system has a solution over $\mathbb{F}$, it is contained in the affine variety V_I defined by I. V_I may also contain some solutions in the algebraic closure $\overline{\mathbb{F}}$ of $\mathbb{F}$. In order to eliminate such solutions, one adds field equations to the system with respect to each variable. Equation systems arising in the algebraic cryptanalysis of block ciphers are often overdefined (there are more equations than variables).

Linearisation. As each polynomial can be seen as a linear combination of monomials, probably the most natural method to solve systems of nonlinear equations is to perform the *linearisation* of the system by assigning a new independent variable to each nonlinear monomial. If $\mathbb{F} = \mathbb{F}_2$ and the degree of polynomials in a generic system of nonlinear equations is bounded by d, there will be about $N = l^d$ distinct monomials. Generally, for $\mathbb{F} = \mathbb{F}_q$ there will be about $N = \binom{l+d}{d}$ distinct monomials. Thus, one obtains a system of linear equations with N variables and t equations. In order to solve this system directly, one has to require $t \geq N - 1$. In other words, the original system of nonlinear equations has to be highly overdefined. For solving the resulting system of linear equations, one can use Gaussian elimination with complexity $\mathcal{O}(N^3)$, Konovaltsev's algorithm [155] with complexity $\mathcal{O}\left(\frac{N^3}{\log_q N}\right)$, or Strassen's algorithm [248] with complexity $\mathcal{O}\left(N^{log_2 7}\right)$ field operations.

For the majority of modern block ciphers, the arising systems either have a high maximal algebraic degree d or are not sufficiently overdefined, both leading to a considerable growth of solution complexity. Moreover, if many polynomials are linearly dependent, linearisation methods can fail. To cope with the latter problem, several methods have been proposed. One of them, the *XL algorithm* (extended linearisation) [141], multiplies the original polynomials by all monomials up to a

certain degree $D - d$. Then the algorithm looks for univariate polynomials using linearisation. The complexity of the XL algorithm is defined by the minimal required degree D_{min} which was shown to be $D_{min} \geq \frac{l}{\sqrt{t-l-1}+1}$ for $t \geq l + 1$ [84]. The number of distinct monomials can be then estimated as at least $N = \sum_{j=0}^{D_{min}} \binom{l}{j}$ [84]. Applying one of the linearisation methods mentioned above yields the approximate overall complexity of the XL algorithm.

The algebraic structure of an iterated block cipher can usually be represented as a sparse iterated system of nonlinear equations. The *XSL algorithm* (extended sparse linearisation) [74] is a variant of the XL algorithm which takes this fact into account. In the XSL algorithm, the polynomials are only multiplied by the product of monomials that were originally present in the system of equations which should decrease the number N of distinct monomials before linearisation (see complexity estimations above).

Gröbner basis methods. The computation of a reduced *Gröbner basis* of the ideal I represents an alternative [55] to the class of linearization methods. Generally speaking, one obtains different Gröbner bases G for different orderings of monomials. The *lex* monomial ordering [64] is useful for obtaining the variety V_I, being an *elimination monomial ordering*. Such orderings allow for the efficient computation of bases for associated ideals on a part of the variables only. In other words, a Gröbner basis for I with respect to an elimination monomial ordering can be used to eliminate variables from the system of equations leading to the computation of V_I. If a Gröbner basis of a zero-dimensional ideal I has been found with respect to some other monomial ordering, it can be converted to an elimination ordering using the *FGLM algorithm* [97]. If the ideal I is not zero-dimensional, the *Gröbner Walk* [75] can be used for such a conversion. Moreover, equation systems resulting from cryptographic algorithms often possess a unique solution $(\alpha_1, \ldots, \alpha_l)$ lying in the base field $\mathbb{F}$. In this case the reduced Gröbner basis of I is expected to be $\{x_1 + \alpha_1, \ldots, x_l + \alpha_l\}$.

The classical way of computing a Gröbner basis is *Buchberger's algorithm* proposed by Buchberger in 1965 [55] which can be considered as a generalization of the Euclidean algorithm and the Gaussian reduction. An alternative to Buchberger's algorithm proposed by Faugere in 1999 [96] is the *F4 algorithm*. It is potentially more efficient as it simultaneously executes operations that are performed by Buchberger's algorithm sequentially. The F4 algorithm was improved by Faugere in 2002 by eliminating some unnecessary operations which resulted in the *F5 algorithm* [95]. Today, the F4 and F5 algorithms are the fastest known Gröbner basis finding algorithms for generic systems of equations.

It is not trivial to estimate the efficiency of these algorithms with respect to a special system of equations. However, some cases were considered in [9] for generic systems of quadratic equations over $\mathbb{F} = \mathbb{F}_2$. If t grows linearly with l, the asymptotic

complexity of F5 is exponential in l. If $l \ll t \ll l^2$ holds, the asymptotic complexity of F5 is subexponential in l. If t grows linearly with l^2, the F5 asymptotic complexity is polynomial in l.

There have been many attempts to apply algebraic cryptanalysis to established block ciphers, the AES algorithm being the most frequent object of these attacks. However, so far there has not been enough evidence that any of the proposed algorithms, generic or specialized, can really recover the AES key faster than exhaustive search. On the positive side, F4 and F5 work well for some small-scale versions of AES [65]. Moreover, recently the techniques of algebraic cryptanalysis were demonstrated to allow for attacking more rounds using differential cryptanalysis [2]. In Chapter 7 of this thesis it is shown that algebraic cryptanalysis can be very useful in the side-channel cryptanalysis of AES.

2.2.4 Some Other Attacks

A lot of other attacks on block ciphers are also known, often building upon linear and differential cryptanalyses. Only some of them are mentioned here.

Other statistical attacks. The *differential-linear attack* was proposed in 1994 by Hellman and Langford [161] and is a mix of linear and differential cryptanalyses. The block cipher is first partitioned into two parts. Then a differential with differential probability p over the first part is used to make properties propagate. The second part is covered by a suitable linear approximation ϵ. Under some assumptions, the bias of the resulting linear approximation can be shown to be $\epsilon' = 2p\epsilon^2$.

The *boomerang attack* proposed by Wagner [251] in 1999 uses two disjunct differentials, each covering only a half of the rounds. It relies on both encryption and decryption oracles (combined chosen plaintext and adaptive chosen ciphertext scenario). A boomerang can be observed by the attacker if the differential for the first half of the cipher holds twice and the differential for the second cipher half holds twice. If p and q are the probabilities of differentials over the first and the second parts of the cipher, respectively, the probability to observe the boomerang can be estimated as $p' = p^2q^2$. Amplified boomerang attack [139] and rectangle attack [21] are two improvements of the boomerang attack.

Related-key attacks were proposed independently by Biham [15]. They are based on the assumptions that the attacker knows or even can choose differences of encryption keys that are used for encryption/decryption, without knowing the original key itself. They are often related to various types of differential cryptanalysis that can incorporate the additional differences resulting from related keys in a natural way. Currently this is probably the most powerful technique available to cryptanalyze block ciphers. It can also be very useful in the cryptanalysis of block-cipher based hash functions. Chapter 4 performs some related-key cryptanalysis of certain special-type SPN-based compression functions for hashing.

Structural attacks. As opposed to the statistical cryptanalysis that first of all relies on low-level statistical behaviour of round functions, structural attacks use high-level properties of block ciphers. *Non-surjective attacks* proposed by Rijmen and Preneel in 1995 [223] represent an example of such techniques. The attack can apply for Feistel networks if the round function F is non-surjective or is surjective but unbalanced. This leaks information about round keys and can be used to rank the most probable key candidates. If a block cipher is mainly word-oriented rather than bit-oriented, *multiset attacks* tend to be useful. A multiset is a set where each element can appear several times (each multiset element is said to have its own multiplicity). Multiset attacks track the propagation of multisets and their properties through the cipher. For highly structured cipher, some properties of multisets can be detected after a considerable number of rounds. AES is probably the most prominent object of multiset attacks due to the fact that all transformations used in AES are byte-oriented. Examples of multiset attacks on AES-type constructions include square attack [77], integral cryptanalysis [151], saturation attack [172], structural cryptanalysis [29].

Slide attacks [32], [31] proposed by Biryukov and Wagner in 1999 and *reflection attacks* [136], [137] first introduced by Kara and Manap in 2007 use self-similarity properties of block ciphers. Slide attacks rely on self-similarities of round functions in the encryption routine to obtain plaintext-ciphertext pairs for a reduced number of rounds. Reflection attacks make use of self-similarities between encryption and decryption routines and analyze the propagation of biased fixed-point distributions through the cipher. Frequently such properties can result from highly structured key schedule algorithms. Self-similarity attacks are often more efficient for Feistel networks. Involutions contained in the ciphers can favour reflection attacks. The idea of slide attacks is used in Chapter 5 to cryptanalyze the KeeLoq block cipher.

2.2.5 Attacks Using Side-Channel Leakage

The major distinctive feature of side-channel cryptanalysis, as compared to the types of conventional cryptanalysis described in the previous subsections, is the different attack model: Additionally to knowing/influencing (parts of) plaintexts/ciphertext/keys of the cryptographic algorithm, the attacker is allowed to observe physical variables connected with the implementation of the attacked algorithm. This assumption strengthens the possibilities of the attacker by a considerable margin. As a result, even attacks requiring knowledge of neither plaintexts nor ciphertexts can be possible in the side-channel scenario.

The physical observables may include any physical parameters correlated with the values processed by the implementation of the attacked cryptographic algorithm. Probably the most important physical leakage channels are timing as well as power consumption and electromagnetic radiation of the implementation which will be discussed in this section.

All side-channel attacks are implementation dependent and, therefore, have a number of additional technical parameters. To call only the most important ones, we mention knowledge of implementation details, type of physical access to the implementation, laboratory equipment, attack stages, number of algorithm executions, and number of cryptographic devices needed.

Attack stages. A side-channel attack has usually up to three stages: Profiling stage, online stage and offline stage. In the (optional) *profiling stage*, the attacker has access to the implementation. Here he can study it by obtaining detailed information on correlations between values of internal variables of the algorithm in this implementation and side-channel parameters being leaked by the implementation. The attacker may be allowed to know/choose inputs and keys of the algorithm in the profiling stage. In the *online stage*, the attacker triggers the implementation to execute the algorithm with some inputs and unknown keys and measures the physical side-channel parameters. If the attacker is allowed to observe or choose inputs/outputs to the algorithm, he can also record the data corresponding to these executions. In the *offline stage*, the attacker tries to recover the key information from the recorded side-channel parameters and inputs/outputs based on his knowledge of the implementation gained in the profiling stage, if he has been allowed to perform profiling.

Timing analysis. Though side-channel leakage seems to have been extensively used by state security agencies for many decades to obtain secret information [152], the idea of applying side-channel attacks to implementations of cryptographic algorithms appeared in the open scientific literature quite recently. The first side-channel attack published was *timing analysis* proposed by Kocher [152] in 1996 and based on observing the execution time of such public-key cryptographic methods as Diffie-Hellman [85], RSA [226] and DSS [101]. The basic idea is as follows. The selection of operations performed by the underlying cryptographic algorithms depends on the key information. The execution times of different operations are often different. Therefore, the overall execution time of the whole algorithm can vary depending on the key, thus, leaking some key information which can be used to recover the key. The timing analysis by Kocher requires a profiling stage.

As applied to block ciphers, the usefulness of timing analysis was also pointed out in [152], where Kocher noticed that table look-ups in software can take different times for different inputs due to RAM cache hits. When these inputs are key-dependent, this timing information frequently turns out sufficient to recover the key. DES was attacked using timing analysis in [209] by Page. AES was first analyzed with timing analysis by Bernstein in [13]. Many other cache-timing attacks on block ciphers are known.

Leakage models. The question of what information is leaked by a device implementing cryptographic algorithms arised quite early in the research of side-channel analysis. For some side-channel attacks it is assumed that the leakage is due to some model. The two basic leakage models that proved to be very realistic are Hamming weight and Hamming distance. This is true for both power consumption and electromagnetic emanation of many cryptographic devices.

In the *Hamming weight model*, the device processing the value x leaks the Hamming weight of x, that is, the number of ones in the binary representation of x. From the physical perspective, this behaviour can be explained by charging internal busses and setting internal flip-flops of the device from 0 to x. Thus, the number of bus lines/flip-flops that has to be charged is exactly the number of ones in x, leading to the power consumption/electromagnetic radiation proportional to the Hamming weight of x.

In the *Hamming distance model*, it is also important what value was previously processed by the device. So, if x is currently processed by the device and x' was the the previous value processed, the side-channel leakage reveals the Hamming weight of $x \oplus x'$ (also known as the Hamming distance between x and x'). The physical explanation for this is that in some devices the values are updated directly by recharging single bus lines/flip-flops corresponding to the bits that differ in x and x': The number of bus lines/flip-flops that need to be recharged is the number of ones in $x \oplus x'$, being proportional to the power consumption/electromagnetic radiation of the device.

Other leakage models can also be useful. Some side-channel attacks use the fact that the implementation actually leaks more information than only Hamming weight or Hamming distance. Others do not explicitly rely on any leakage model.

Signal and noise. Side-channel parameters such as power consumption or electromagnetic radiation are usually measured by a digital oscilloscope that translates the continuous instantaneous physical observables in a given time interval into a sequence of digitized values. An oscilloscope is characterized by several parameters including sampling rate (how often the observable is sampled) and resolution (number of levels for the discretization of real-valued physical observables).

The measured side-channel parameter corresponding to a relevant cryptographic operation is called a *side-channel trace* and is denoted by $\tau = (\tau_1, \ldots, \tau_n) \in \mathbb{R}^n$. τ can be modelled statistically as follows. For fixed inputs, the device produces some deterministic signal $s = (s_1, \ldots, s_n) \in \mathbb{R}^n$. Through a large number of external factors, this signal vector s is randomly noised. This can be modelled by adding a vector of random variables $r = (r_1, \ldots, r_n) \in \mathbb{R}^n$ to s (additive noise):

$$\tau = s + r,$$

where s and r are added as vectors in the linear space $\mathbb{R}^n$. In many cases, the noise r is well described by a multivariate normal distribution with zero vector of means.

In practice, it often turns out that most non-diagonal elements of the corresponding $n \times n$ covariation matrix are close to zero. That is, r can be sometimes approximated by n independent univariate normal distributions[6], $r_i \sim \mathcal{N}(0, \sigma_i^2)$ and $\tau_i \sim \mathcal{N}(s_i, \sigma_i^2)$. Note that by averaging a trace t times one achieves a reduction of σ_i by factor $\sqrt{t}$.

Simple side-channel analysis. *Simple side-channel analysis* published by Kocher, Jaffe and Jun in 1999 [153] uses the instantaneous power consumption of the device implementing public-key cryptographic algorithms. The attack is referred to as *simple power analysis* (SPA) in [153]. The idea is the following. The order of underlying arithmetic operations in such algorithms (e.g. the sequence of multiplications and squarings in the exponentiation) often depends on the key value deterministically. At the same time, the side-channel trace of the device corresponding to different arithmetic operations is often according to different patterns. So one can sometimes distinguish between these arithmetic operations (e.g. multiplication and squaring in exponentiation can have different execution times and different forms of trace curves), thus, obtaining key-dependent information. The original simple side-channel cryptanalysis [153] does not make use of any profiling and uses power consumption as side-channel leakage. Simple side-channel analysis can also use the electromagnetic emanations of the attacked device [220].

As applied to block ciphers, the simple side-channel analysis without profiling does not directly lead to key recovery in most cases. This is due to the fact that it is usually much more difficult to localize the dependency of the side-channel leakage on key chunks for block ciphers. Here each key bit has only limited influence on the measured side-channel parameters of the implementation because of the internal structure of most block ciphers as opposed to such algorithms as modular exponentiation (e.g. in RSA) or scalar point multiplication (e.g. in elliptic curve cryptosystems). However, if the Hamming weight of internal cipher states can be extracted from the trace τ (e.g. after multiple averaging), the key schedule of DES [25] and AES [174] can be successfully attacked. If the round subkeys have been pre-computed and are not calculated online, the data path of some ciphers including AES can be attacked using SAT-solvers to exploit the algebraic structure of AES (algebraic side-channel analysis), if the Hamming weight of intermediate variables is detectable from the traces [221].

Differential side-channel analysis. *Differential side-channel analysis* was also first published by Kocher, Jaffe and Jun in the same work [153] under the name of differential power analysis (DPA). However, similar analysis can also be performed using electromagnetic traces [220].

[6]Here and below, $\mathcal{N}(a, b)$ denotes a univariate normal distribution with mean a and variance b.

Differential side-channel analysis is a statistical hypothesis test. It works as follows. First, some part of the key is guessed by the attacker. Then, for this guess and each known input/output of the algorithm, a part of the internal state of the algorithm is predicted by the attacker (usually it can be directly calculated). After this, the corresponding traces are classified with respect to some leakage model for each hypothesis. For instance, if the Hamming weight leakage model is assumed, the traces are partitioned into the classes depending on the Hamming weight value of the predicted part of the internal state. In the next step, the key hypothesis is checked: If it has been correct, the classification is also correct and it is expected that the side-channel parameter is correlated with the Hamming weight, which is checked then using a statistical test like the T-test [167].

Usually, the side-channel traces exhibit correlation to the leakage model parameter only in a small proportion of all time instants. The knowledge of time instants leaking this information can make the differential side-channel analysis more efficient. However, an important advantage of differential side-channel analysis is that the attack finds the instants of high correlation by itself: The key hypothesis with one of the highest peaks corresponding to some time instant of traces will be correct with high probability, if enough traces are available for the given noise level.

There are numerous generalizations of the differential side-channel attack, in both profiling-based and online-only scenarios. Here we only mention attacks based on stochastic models [167] and mutual information analysis [104].

Template attacks. Template attacks were proposed by Chari, Rao and Rohatgi [60] in 2002 and enhanced in [5], [245]. As a rule template attacks require a profiling stage, where an internal variable of the algorithm is usually targeted which corresponds to a subset of side-channel trace instances. As a result of the profiling, the attacker obtains a set of trace templates corresponding to all possible values of the targeted internal variable.

Usually, one distinguishes between two types of templates: Gaussian templates and reduced templates. To build the Gaussian template, the measured noised signal corresponding to each value of the targeted internal variable is interpreted as a sample of a Gaussian multivariate distribution with mean vector μ_j and covariance matrix C_j. In the profiling stage, the attacker uses the implementation to get estimations μ_j^* and C_j^* of μ_j and C_j for each possible value of the internal variable. The reduced template does not include C_j^* and deals with the mean vector μ_j^* only. Thus, the Gaussian template is a set of pairs μ_j^* and C_j^* for all values of the targeted internal variable. The reduced template is the set of the corresponding mean vectors μ_j^*.

In the online stage the attacker triggers the implementation to obtain a trace sample (several trace samples) corresponding to an unknown value of the internal variable. Then one has to decide which template the unknown internal value corre-

sponds to. If the Gaussian template is used, the measured trace is used to perform a test based on the maximum-likelihood principle. If the reduced template is used, one can compute the Euclidean distance of the measured trace to all μ_j^* and choose the one with μ_j^* closest to the trace with respect to the Euclidean distance between real-valued vectors. If the quality of the estimation in the profiling stage and the number of online measurements were sufficient at a given noise level, a high probability of the correct decision can be attained. The result is a set of exact internal variable values for an unknown key. The determined values can be then used to recover this key. Note that template attacks are the most efficient side-channel attacks in an information-theoretical sense [60].

Basic collisions attacks. Side-channel attacks based on internal collisions were proposed in 2003 with applications to DES [233] and were extended to AES in [232]. An internal collision occurs if a function ϕ within a cryptographic algorithm processes different input arguments, but returns an equal output. This event can be detected using side-channel leakage. Having detected the collisions, the attacker is often able to recover the key by solving a system of equations, as each collision delivers an equation. Indeed, if a key-dependent function ϕ causes a collision for two inputs a and b, $a \neq b$, then one obtains the equation $\phi(a, k) = \phi(b, k)$. If a and b are known, the attacker gets an equation on key bits. In [233] and [232], the equation systems are basically solved basically by brute-force.

The problem of collision detection is akin to the template-based classification of traces in template attacks: One has two side-channel traces and wants to decide if they both correspond to the same internal value processed. The difference to template attacks is that one does not have to decide which value exactly it is. In [233] and [232], the Euclidean distance between the two traces is used for this purpose: If the Euclidean distance is lower than a certain decision threshold, this pair is considered as a collision.

Chapter 7 of this thesis deals with side-channel collision attacks in much more detail by applying them to other cryptographic construction, considering new types of collisions and solving the arising systems of equations using methods of algebraic cryptanalysis.

Chapter 3

Cryptanalysis of Unbalanced Feistel Networks with Contracting Functions

Though unbalanced Feistel networks (UFN) are widely considered as an alternative to balanced Feistel networks (BFN) and substitution-permutation networks (SPN) in symmetric cryptography, little has been known yet about their resistance against differential and linear cryptanalysis.

In this chapter, we tackle the problem at the example of d-branch SP-type UFNs with contracting MDS diffusion (dCUFN-SP). Under some restrictions on the contracting MDS matrices over multiple rounds, we prove lower bounds on the number of differentially active S-boxes for dCUFN-SP with $d \in \{3, 4\}$ and on the number of linearly active S-boxes for dCUFN-SP with $d \geq 3$.

As opposed to SPNs and BFNs, the number of differentially active S-boxes for such constructions does not directly translate to an upper bound on the probability of differential trails. So we provide a thorough analysis of single-round differentials that yields an upper bound on the probability of a differential trail. It is also shown that the efficiency level of dCUFN-SP is comparable to that of BFNs and SPNs with respect to differential and linear cryptanalysis.

This chapter is based on the independent work of the author published in [40] and [41].

3.1 Introduction

Many sound cryptographic constructions such as stream ciphers and hash functions can be built from block ciphers. There are several traditional ways of designing block ciphers including balanced Feistel networks (BFN) and substitution-permutation networks (SPN), to call the two most important ones. The approach of BFNs [98] is followed by the former U.S. encryption standard DES [82] and many other block ciphers. The latter approach is adopted by Rijndael [79], a subset of which became the U.S. encryption standard AES [1]. In this chapter we study unbalanced Feistel networks (UFN) [231], [133], [171], [191], which represent a well-known alternative to BFNs and SPNs.

SP-type balanced Feistel networks. A block cipher encrypts a b-bit plaintext block with a k-bit key to obtain a b-bit ciphertext block. As a rule, the block cipher encryption is the iterative application of r similar invertible transformations (called *rounds* or *round transformations*) to the plaintext. All round transformations are usually key-dependent and the transformation of round i obtains its own subkey k_i which is derived from the cipher key K using a key-schedule algorithm.

In case of BFNs, one deals with block ciphers having a special-type round transformation that works as follows. Assume that (x_{i-1}, x_i) is the input to round i consisting of two $b/2$-bit halves (*branches*). Then (x_i, x_{i+1}) is the output of round i, if

$$x_{i+1} = F_i(x_i, k_i) \oplus x_{i-1},$$

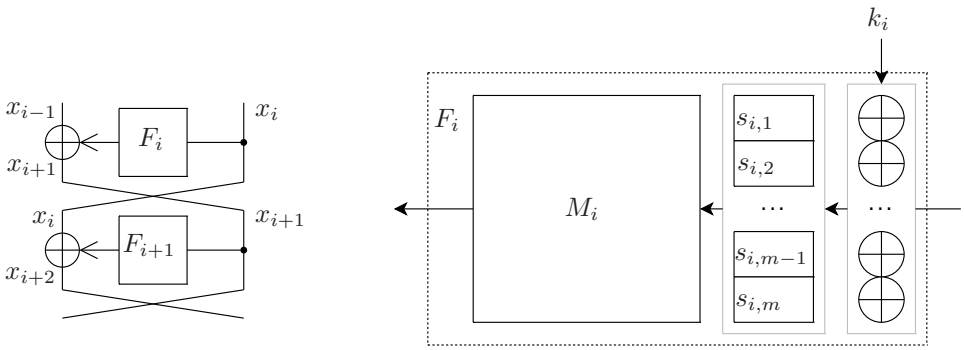


Figure 3.1: Two rounds of BFN (on the left) and SP-type F-function F_i of BFN-SP (on the right)

where F_i is the F-function in round i (see Figure 3.1).

Typically, the round F-functions F_i are chosen to be highly nonlinear key-dependent transformations with good diffusion. A standard way to provide these properties for F_i is to use the *substitution-permutation* (SP) structure consisting of three layers (see Figure 3.1): In the first layer the subkey k_i is added to x_i, which provides key dependency. In the second layer, nonlinear functions (*S-boxes*, substitution) acting on parts of the state are applied in parallel. If each S-box operates on n out of $b/2$ bits, there are $m = \frac{b}{2n}$ n-bit S-boxes working in parallel. In the third layer, these m parts are diffused using a linear mapping (*diffusion mapping*, permutation). In other words, F_i is an SP-type F-function, if

$$F_i(x_i, k_i) = \theta_i(\gamma_i(x_i \oplus k_i)),$$

where $\gamma_i : \mathbb{F}_2^{b/2} \to \mathbb{F}_2^{b/2}$ is the bricklayer function generated by concatenating $m = \frac{b}{2n}$ n-bit S-boxes $s_{i,j} : \mathbb{F}_2^n \to \mathbb{F}_2^n$, $1 \leq j \leq m$, and $\theta_i : \mathbb{F}_2^{b/2} \to \mathbb{F}_2^{b/2}$ is an $\mathbb{F}_2^n$-linear Boolean mapping. Each θ_i is represented by an $m \times m$ matrix M_i over $\mathbb{F}_2^n$. Note that the length of k_i is $b/2$ bit. We refer to BFN with SP-type F-functions as *BFN-SP*.

The cryptographic properties of BFN-SP differ greatly depending on the choice of the S-boxes and diffusion mappings. Two major attacks to be thwarted by a block cipher are differential cryptanalysis [23] and linear cryptanalysis [177], based on the propagation of differential and linear relations through the cipher, respectively. Sequences of such differential and linear relations over several rounds constitute differential and linear *trails* (see e.g. [79] for details). To make the propagation of these pattern types in SP-based constructions hard, one chooses such S-boxes and diffusion mappings that:

- $s_{i,j}$ is highly nonlinear with low maximum differential probability [23] and

low maximum linear bias [177]. This guarantees a good reduction of the trail quality if it has to go through an S-box (i.e. if this is an *active S-box*).

- M_i provides good diffusion properties maximizing the number of active S-boxes, that is, the number of S-boxes involved into the propagation of differential/linear patterns following a trail.

Branch number and optimal diffusion. In BFN-SP, M_i can lead to a high number of differentially and linearly active S-boxes, if for a nonzero input it keeps the number of nonzero elements at its inputs and outputs high. This property translates to the distance of the linear code induced by M_i and can be dealt with in more detail as follows.

We begin with the notion of a branch number [79]. Let $\tau \in \mathbb{F}_2^{nc}$ be represented by a bundle $\tau = (\tau_1, \ldots, \tau_c)$ of c elements in $\mathbb{F}_2^n$, $\tau_i \in \mathbb{F}_2^n$. Let $w(\tau)$ denote the *bundle weight* of τ, that is, the number of nonzero $\mathbb{F}_2^n$-components in τ, $w(\tau) = \#\{\tau_i : \tau_i \neq 0, 1 \leq i \leq c\}$. The *branch number*[1] of a linear mapping $\theta : \mathbb{F}_2^{nc} \to \mathbb{F}_2^{nm}$, $c \geq m$, is then defined as $\mathcal{B}(\theta) = \min_{\tau \neq 0}\{w(\tau) + w(\theta(\tau))\}$.

To attain a better diffusion in BFN-SP, one can increase the branch numbers of its linear diffusion mappings M_i which leads to a higher number of nonzero n-bit elements at inputs and outputs of F_i. The linear mapping θ is called a linear *optimal diffusion mapping* (linear ODM), if $\mathcal{B}(\theta)$ is maximal, namely, if $\mathcal{B}(\theta) = m+1$. Linear ODMs can be constructed using maximum distance separable (MDS) codes [169].

Single- and multiple-round diffusion in BFN-SP. In the BFN-SP construction, one can either apply a single diffusion matrix in all rounds $M = M_i$ (resulting in BFN-SP with single-round diffusion, *BFN-SP-SR*) or use a set of several distinct matrices M_i (leading to BFN-SP with multiple-round diffusion, *BFN-SP-MR*). These constructions will behave differently with respect to the number of active S-boxes guaranteed within several rounds.

The number of active S-boxes in BFN-SP-SR was studied in [135], where Kanda proved

Theorem 1 (BFN-SP-SR [135])**.** *In BFN-SP-SR, if $M = M_i$ is an ODM, every $4R$ $(R \geq 1)$ consecutive rounds provide at least $(m+1)R + \lfloor R/2 \rfloor$ active S-boxes.*

Shirai and Shibutani demonstrated in [240] that a much better bound on the number of active S-boxes can be attained for the class of BFN-SP constructions. This is due to the fact that the differential trails with minimal numbers of active S-boxes in BFN-SP-SR suffer from the *difference cancellation* effect that occurs at the addition of $F_i(x_i, k_i)$ with x_{i-1}: Even though the diffusion properties of F_i are

[1] Note that this branch number of a mapping has nothing to do with the number of branches in a Feistel cipher which is the number of subblocks.

optimal and there are many nonzero differences at the output of F_i, x_{i-1} can be chosen in a way cancelling these differences, thus, leading to a lower number of differentially active S-boxes in the next rounds. In [239] Shirai and Preneel treated this more rigorously and proved that essentially the same number of active S-boxes in BFN-SP-MR is provided by significantly less rounds:

Theorem 2 (BFN-SP-MR [239])**.** *In BFN-SP-MR, if* $[M_i|M_{i+2}|M_{i+4}]$ *is an ODM for each* $1 \leq i \leq r-4$*, every* $3R$ *(*$R \geq 2$*) consecutive rounds provide at least* $(m+1)R$ *active S-boxes.*

Unbalanced Feistel networks with contracting functions. As opposed to BFNs considered above, UFNs use a round transformation that treats the b-bit input as a set of d equally large branches with $d > 2$ instead of $d = 2$. d-branch UFNs with contracting F-functions (dCUFN) represent the main research object of this chapter and can be described as follows.

Let the number of branches d divide the block size b. Then we can partition the input to the round transformation in round i into d variables:

$$(x_{i-1}, x_i, \ldots, x_{i+d-2}) \in (\mathbb{F}_2^{b/d})^d.$$

The round transformation defines a d-branch UFN with contracting F-functions, if its output is

$$(x_i, x_{i+1}, \ldots, x_{i+d-1}) \in (\mathbb{F}_2^{b/d})^d$$
$$\text{with}$$
$$x_{i+d-1} = F_i(x_i, \ldots, x_{i+d-2}, k_i) \oplus x_{i-1},$$

where F_i is the contracting round F-function and k_i is the round subkey in round i. For $d \in \{3, 4\}$ this construction is illustrated in Figure 3.2.

Recently structural analysis of UFNs with contracting and expanding functions has been conducted [210], [211]. Due to some structural properties, at least $2d$ rounds of dCUFN for $d \geq 4$ and at least 7 rounds of 3CUFN are required to avoid all known generic attacks on this construction. However, the analysis of dCUFN with respect to differential [23] and linear [177] properties has received little attention so far.

At the same time, dCUFNs with SP-type functions are closely related to a classification of generalized Feistel networks with SP-type functions which is still lacking after many years of studies in the area of Feistel constructions. Moreover, dCUFNs with SP-type functions (especially the versions with higher values of d) present an option for building ciphers for hardware, being inherently more repetitive and providing smaller footprint.

Contributions and organization of the chapter. This chapter tackles the problem of the more detailed cryptanalysis of UFNs at the example of d-branch SP-

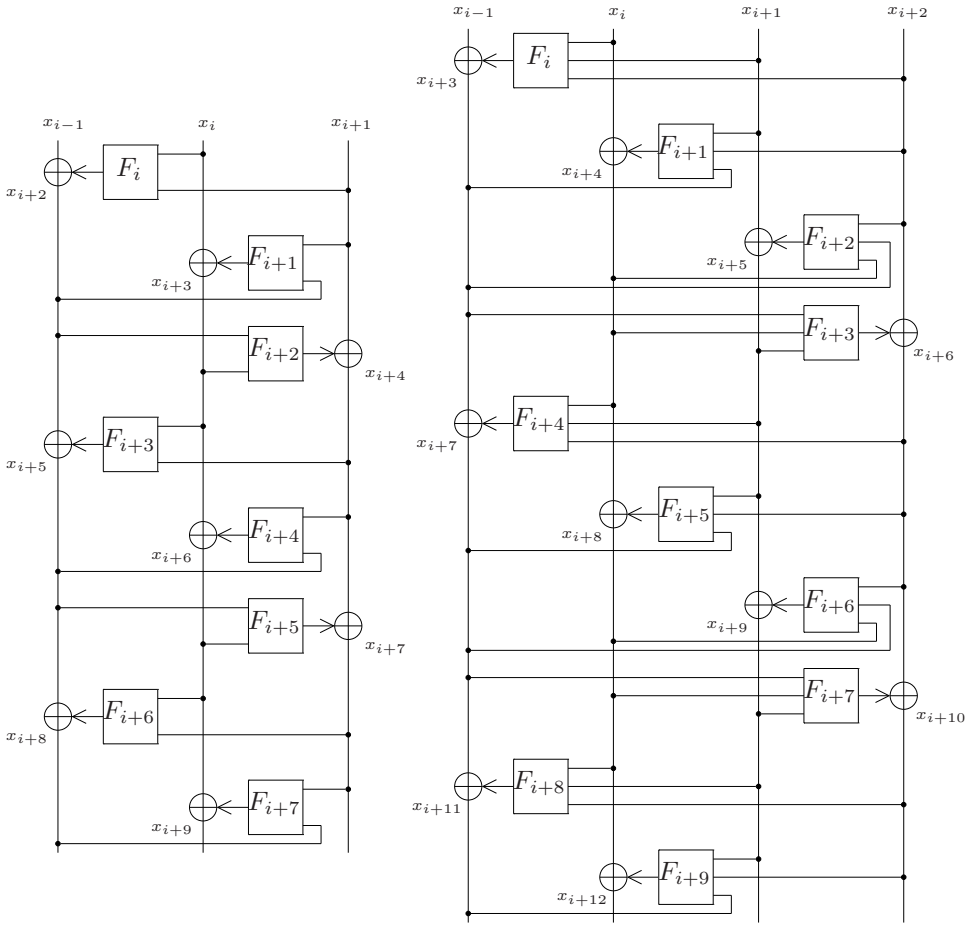


Figure 3.2: 3- and 4-branch UFNs with contracting F-functions

type UFNs with linear contracting optimal diffusion mappings over $\mathbb{F}_2^n$ with respect to differential and linear properties.

First, an SP-type F-function F_i is defined for dCUFNs (Section 3.2), similarly to BFN-SP. UFNs with such F-functions are referred to as dCUFN-SP throughout the work. We impose some restrictions on the linear diffusion matrices over multiple rounds in a way similar to Theorem 2 to avoid much of the difference cancellation. Under these conditions, a tight lower bound on the number of differentially active S-boxes over $2(d+1)R$ consecutive rounds is proven for $d \in \{3, 4\}$ (Section 3.4).

In contrast to BFN-SP, an upper bound on the differential trail probability for such constructions does not follow directly from the number of differentially active S-boxes. This is due to the linear contracting diffusion, where several single-round differential trails contribute to the same single-round differential. Therefore, we prove

an upper bound on the single-round differential probability for a given number of differentially active S-boxes for $d > 2$ (Section 3.3).

A differential trail over many rounds is a sequence of single-round differentials. So we consider worst-case distributions of active S-boxes over several single-round differentials and prove an upper bound on the differential trail probability over many rounds (Section 3.5) using both the proven minimum number of differentially active S-boxes from Section 3.4 and the proven maximum single-round differential probability from Section 3.3.

Section 3.6 proves a minimum number of linearly active S-boxes over $(d+1)R$ rounds of dCUFN-SP by imposing some further restrictions on the linear layer within one round (Theorem 4). This result holds for $d > 2$.

Section 3.7 shows that the efficiency of dCUFN-SP with the conditions on the linear layer mentioned above is comparable to that of many BFNs and SPNs, discusses the results and formulates some open problems. We conjecture that Proposition 4 and Theorem 3 also hold for $d > 4$ and stress that all other results of the chapter (including Theorem 4) are valid for $d > 2$.

3.2 UFN with SP-Type Contracting Functions

3.2.1 Definition of dCUFN-SP

Definition 9 (dCUFN with SP-type F-functions, dCUFN-SP)**.** *A dCUFN is called a d-branch unbalanced Feistel network with SP-type F-functions and contracting diffusion (dCUFN-SP), if $d > 2$ and F_i is defined as*

$$F_i(x_i, \ldots, x_{i+d-2}, k_i) = \theta_i(\gamma_i((x_i, \ldots, x_{i+d-2}) \oplus k_i)),$$

where $\gamma_i : \mathbb{F}_2^{(d-1)b/d} \to \mathbb{F}_2^{(d-1)b/d}$ is the bricklayer function generated by concatenating $(d-1)m$, $m = \frac{b}{nd}$, bijective n-bit S-boxes $s_{i,j} : \mathbb{F}_2^n \to \mathbb{F}_2^n$, $1 \leq j \leq (d-1)m$, and $\theta_i : \mathbb{F}_2^{(d-1)b/d} \to \mathbb{F}_2^{b/d}$ is a contracting $\mathbb{F}_2^n$-linear mapping. Each θ_i is represented by an $m \times (d-1)m$ matrix M_i over $\mathbb{F}_2^n$. k_i has length $(d-1)b/d$ bit. Let $n > 2$ and $m \geq 2$.

Figure 3.3 shows the internal structure of contracting SP-type F-functions for dCUFN-SP with $d \in \{3, 4\}$. We represent the diffusion matrix M_i by $d-1$ square $m \times m$ matrices

$$L_i^{(1)}, \ldots, L_i^{(d-1)}$$

over $\mathbb{F}_2^n$, $M_i = [L_i^{(1)} | \ldots | L_i^{(d-1)}]$, which are applied to the output of the S-box layer in parallel. That is, each F-function consists of $d-1$ parallel rounds of smaller SPNs, whose outputs are added together.

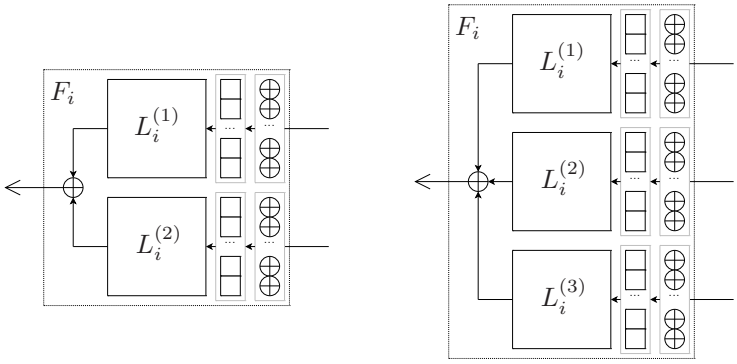


Figure 3.3: Contracting F-functions of 3CUFN-SP and 4CUFN-SP

A Special Case of dCUFN-SP. The construction of dCUFN-SP allows for an equivalent representation. The round function F_i can be seen as consisting of $d-1$ rounds of domain-preserving SPNs switched in parallel (see Figure 3.3). Each value x_i is used in $d-1$ consecutive rounds by $d-1$ of these domain-preserving SPNs, $\psi_1, \ldots, \psi_{d-1}$, which are parts of the corresponding round functions $F_{i-d+2}, \ldots, F_i$.

Assume now that $\psi_1, \ldots, \psi_{d-1}$ accepting x_i as argument share the same key, the same S-box layer as well as the same linear diffusion matrix L_i. In other words, assume that they are equal as functions: $\psi_1 = \cdots = \psi_{d-1} = \psi$. The clear advantage of this special-case construction is that ψ has to be computed only once instead of $d-1$ computations in the generic case. This makes the construction more efficient by a factor of $d-1$ than the generic dCUFN-SP.

For this construction, one has to thoroughly investigate the consequences of the fact that the round functions cannot be seen as independent anymore. For instance, this circumstance might lead to more efficient dedicated attacks (discarding the internal structure of the domain-preserving building blocks) similar to those of [210] or favour other types of cryptanalysis. We have chosen the generic dCUFN-SP construction rather than the special-case one as our principle object of study. Note that a vast subset of the results presented in this chapter are also applicable to this special-case construction.

3.2.2 On Motivation behind dCUFN-SP

Apart from the natural interest in the differential and linear properties of new-type constructions (design-space exploration), we had several further lines of motivation in mind when initiating our study of the UFN constructions with contracting functions, of both theoretical and practical nature.

We consider the results of this chapter as a first step towards a proper classification of generalized Feistel networks (GFNs) with SP-type functions. Several

different GFNs have been proposed [263], [199], [241], all of them sharing the property of being based on domain-preserving functions. Actually, many more GFNs with domain-preserving functions exist and dCUFN-SP can be seen as such a GFN (as mentioned above in this section). dCUFN-SP will play a central role in such a classification. This is due to the fact that the absolute majority of conceivable GFNs with SP-type functions in such a classification will have contracting linear diffusion mappings.

Another line of motivation we had is related to the design of lightweight symmetric constructions. Most compact block ciphers such as PRESENT [44] or KATAN and KTANTAN [57] are among those allowing for high serialization, which is inherently the case for dCUFN. A round function updates only a part of the cipher state at a time and higher values of d correspond to higher degrees of serialization. This makes dCUFN an interesting object of study from the perspective of designing hardware-efficient ciphers. A major question in this context is how the security of dCUFN behaves with the increase of d.

3.3 Differential Properties in One Round

3.3.1 Differential Trails vs Differentials

For BFNs, the number of S-boxes active in one round directly translates to the probability of a single-round differential in this round, as a single-round differential consists of only one single-round differential trail: If the input difference of a BFN F-function leads to t active S-boxes and the maximum differential probability of a single S-box is p, then the probability of this single-round differential can be upper-bounded by p^t.

Generally speaking, this one-to-one correspondence is not valid any more for a round of dCUFN-SP, as often many different single-round differential trails contribute to the same single-round differential because of the linear contracting diffusion. Thus, the probability of a single-round differential cannot be directly derived from the number of active S-boxes: For a fixed input difference before F_i, several output differences of the S-box layer γ_i can lead to the same output difference of F_i.

3.3.2 Notations

Let Δx be the $(d-1)mn$-bit input difference of some dCUFN-SP F-function F and Δy be the mn-bit output difference of F. For the purposes of this section, we will disregard the number of round i in all notations. We will consider the $(d-1)mn$-bit intermediate differences Δz after the S-box layer of F for some fixed differential $(\Delta x, \Delta y)$.

3.3.3 Upper Bound on the Differential Probability

Proposition 1 (Maximum differential probability in one round)**.** *In dCUFN-SP, let each M_i be an ODM and let t be the number of active S-boxes given by the input difference to the round F-function. Let p be the maximum differential probability of a single S-box. Then the following upper bound Q on the probability P of a single-round differential holds:*

- $P \leq Q = p^t$*, if* $t \leq m$ *and*
- $P \leq Q = p^m$*, if* $t > m$*.*

Proof. If $t \leq m$, the MDS diffusion matrix M establishes a one-to-one correspondence between Δy and Δz. This means that for a fixed differential $(\Delta x, \Delta y)$, there is exactly one intermediate difference Δz leading to the output difference Δy. Therefore, there is only one single-round differential trail $(\Delta x, \Delta z, \Delta y)$ corresponding to the differential. If there are at most t active S-boxes in a differential trail and the maximum S-box differential probability is p, we obtain that the differential probability P for one round is upper-bounded by p^t, $P \leq p^t$.

If $t > m$, there may be many differential trails $(\Delta x, \Delta z, \Delta y)$ contributing to the same differential $(\Delta x, \Delta y)$. In other words, there can be several Δz compatible with the fixed Δx and Δy. The probability of a differential can be computed as the sum of the probabilities of all differential trails constituting the differential. As a difference propagates through the linear map θ with probability one, we have

$$P = \Pr\{F : \Delta x \to \Delta y\} = \sum_{\Delta z : \Delta y = \theta(\Delta z)} \Pr\{\gamma : \Delta x \to \Delta z\}. \tag{3.1}$$

In the remainder of the proof we will aim to upper-bound the sum on the right-hand side of (3.1).

The propagation of Δx to Δz occurs through $(d-1)m$ S-boxes s_j switched in parallel. As only t S-boxes are active, accepting a nonzero input difference, and all S-boxes are bijective, the propagation of zero input differences through all the other $(d-1)m-t$ non-active S-boxes is completely deterministic: Each non-active S-boxes maps a zero input difference to a zero output difference with probability one. So only the active S-boxes are relevant. In order to differentiate active and non-active S-boxes, we consider a subset U of the set of all $(d-1)m$ indices j such that $j \in U$ iff s_j is an active S-box as defined by Δx. That is, U is the set of indices corresponding to the t active S-boxes, $|U| = t$. Summarizing these reasonings, we have:

$$\Pr\{\gamma : \Delta x \to \Delta z\} = \prod_{1 \leq j \leq (d-1)m} \Pr\{s_j : \Delta x_j \to \Delta z_j\} = \prod_{j \in U} \Pr\{s_j : \Delta x_j \to \Delta z_j\}.$$

The diffusion matrix M connects $(d-1)m$ input components with m output components, that is, dm components in total. This can be seen as an underdetermined system of linear equations with respect to these dm components. As M possesses the MDS property, fixing any $(d-1)m$ of the dm variables unambiguously determines the remaining m variables.

Now we fix some $t-m$ values of Δz_j with $j \in U$ to some values. Let $U' \subset U$ be the subset indices corresponding to the components fixed, $|U'| = t-m$. From the $(d-1)m$ input components Δz_j, $(d-1)m-t$ are zero and t are nonzero. Therefore each such fixation determines $((d-1)m-t)+(t-m) = (d-2)m$ input components. Together with m output components Δy which are fully specified for a given differential, this results in $(d-1)m$ components of Δz and Δy being fixed. Due to the observation above, the remaining components of Δz will be unambiguously determined. Thus, the values Δz_j with $j \in U'$ determine the values Δz_j with $j \in U \backslash U'$, $|U \backslash U'| = m$. As each S-box differential probability is upper-bounded by p, we obtain:

$$\begin{array}{rcl} & \Pr\{\gamma : \Delta x \to \Delta z\} & \\ = & \prod_{j \in U} \Pr\{s_j : \Delta x_j \to \Delta z_j\} & \\ = & \prod_{j \in U'} \Pr\{s_j : \Delta x_j \to \Delta z_j\} \prod_{j \in U \backslash U'} \Pr\{s_j : \Delta x_j \to \Delta z_j\} & \\ \leq & p^m \prod_{j \in U'} \Pr\{s_j : \Delta x_j \to \Delta z_j\}. & \end{array} \tag{3.2}$$

Combining (3.1) and (3.2), one has:

$$\begin{array}{rcl} P & \leq & \sum_{\Delta z : \Delta y = \theta(\Delta z)} p^m \prod_{j \in U'} \Pr\{s_j : \Delta x_j \to \Delta z_j\} \\ & \leq & p^m \sum_{\Delta z : \Delta y = \theta(\Delta z)} \prod_{j \in U'} \Pr\{s_j : \Delta x_j \to \Delta z_j\}. \end{array}$$

The sum is computed over all sets $\{\Delta z_j\}_{j \in U'}$ compatible with Δy. As the sum of products does not exceed the product of sums, we can change the order of summation and multiplication:

$$\begin{array}{rcl} P & \leq & p^m \sum_{\{\Delta z_j\}_{j \in U'} : \Delta y = \theta(\Delta z)} \prod_{j \in U'} \Pr\{s_j : \Delta x_j \to \Delta z_j\} \\ & \leq & p^m \prod_{j \in U'} \sum_{\{\Delta z_j\}_{j \in U'} : \Delta y = \theta(\Delta z)} \Pr\{s_j : \Delta x_j \to \Delta z_j\}. \end{array}$$

For a fixed input difference Δx_j, the sum $\sum_{\{\Delta z_j\}_{j \in U'} : \Delta y = \theta(\Delta z)} \Pr\{s_j : \Delta x_j \to \Delta z_j\}$ cannot exceed 1. The product of such sums will be upper-bounded by 1 as well. So one has $P \leq p^m$ which proves the claim of the proposition for $t > m$. □

Proposition 1 is illustrated in Figure 3.4. For small numbers of active S-boxes ($t < m$), Q decreases exponentially with the linear increase of the number of differentially active S-boxes, as it is the case for BFNs and SPNs. After the point of saturation ($t = m$) is passed accounting for the minimal $Q = p^m$, it stops decreasing and higher numbers of active S-boxes give the same Q. This behaviour is distinctive for dCUFN-SP and does not inhere in BFNs or SPNs.

Note that the upper bound proven holds a.o. for two interesting classes of S-boxes: Rijndael-type S-boxes with $p = 2^{2-n}$ [79] and n even as well as almost perfect nonlinear (APN) S-boxes with $p = 2^{1-n}$ [198]. Though there were numerous instances of APN-permutations for n odd, no APN-permutations for n even were known till recently. In [86], Dillon demonstrates that APN-permutations for $n = 6$ do exist and provides an example. At the same time, it is known that there are no APN-permutations in dimension 4 [125].

3.4 Differentially Active S-boxes over Many Rounds

3.4.1 Notations

For the treatment of differential properties over many rounds we have to introduce some further notations. $\Delta x_i \in \mathbb{F}_2^{nm}$ denotes the XOR difference of two values x_i and x_i' of the intermediate state in round i, $\Delta x_i = x_i \oplus x_i'$. The difference weight of Δx_i is denoted by $D_i = w(\Delta x_i)$ and is the number of nonzero $\mathbb{F}_2^n$-components of Δx_i. The truncated difference weight of Δx_i is denoted by δ_i and is defined as $\delta_i = \delta(D_i)$, where $\delta(a)$ is 0, if $a = 0$, and 1, otherwise. The input of θ_i (and the output of γ_i) in F_i is denoted by $z_i^{(i)}, \ldots, z_{i+d-2}^{(i)} \in \mathbb{F}_2^{nm}$. Differences after γ_i are correspondingly denoted by $\Delta z_i^{(i)}, \ldots, \Delta z_{i+d-2}^{(i)}$. Generally speaking, $\Delta z_i^{(j_1)} \neq \Delta z_i^{(j_2)}$ for $j_1 \neq j_2$, however, $w(\Delta z_i^{(j_1)}) = w(\Delta z_i^{(j_2)}) = w(\Delta x_i) = D_i$.

3.4.2 Difference Weight Relations over Many Rounds

The internal structure of dCUFN-SP imposed by Definition 9 provides some important inequalities on the difference weights over multiple rounds.

Proposition 2 (Single-matrix relation)**.** *In dCUFN-SP, if M_{i+1} is an ODM and $\delta_{i+1}, \ldots, \delta_{i+d-1}$ are not all zero, the following inequality holds:*

$$\sum_{0 \le c \le d} D_{i+c} \ge m + 1.$$

Proof. Consider F_{i+1}. The input difference $\Delta x_{i+1}, \ldots, \Delta x_{i+d-1}$ propagates to

$$\Delta z_{i+1}^{(i+1)}, \ldots, \Delta z_{i+d-1}^{(i+1)}$$

after the S-box layer S_{i+1}. Then the output difference of F_{i+1} can be computed as

$$M_{i+1}[\Delta z_{i+1}^{(i+1)}, \ldots, \Delta z_{i+d-1}^{(i+1)}]^T.$$

On the other hand, the output difference of F_{i+1} is $\Delta x_i \oplus \Delta x_{i+d}$. As at least one of $\Delta z_{i+1}^{(i+1)}, \ldots, \Delta z_{i+d-1}^{(i+1)}$ is nonzero and M_{i+1} is an ODM, we have:

$$w(\Delta z_{i+1}^{(i+1)}) + \cdots + w(\Delta z_{i+d-1}^{(i+1)}) + w(\Delta x_i \oplus \Delta x_{i+d}) \geq m + 1.$$

As $w(a) + w(b) \geq w(a \oplus b)$ and $w(\Delta z_j^{(i+1)}) = w(\Delta x_j)$ for each j:

$$w(\Delta x_{i+1}) + \cdots + w(\Delta x_{i+d-1}) + w(\Delta x_i) + w(\Delta x_{i+d}) \geq m + 1.$$

□

Proposition 3 (Double-matrix relation)**.** *In dCUFN-SP, if* $[M_{i+1}|M_{i+d+1}]$ *is an ODM and* $\delta_{i+1}, \ldots, \delta_{i+d-1}, \delta_{i+d+1}, \ldots, \delta_{i+2d-1}$ *are not all zero, the following inequality holds:*

$$\sum_{0 \leq c \leq 2d} D_{i+c} - D_{i+d} \geq m + 1.$$

Proof. Similarly to the proof of Proposition 2, consider F_{i+1} and F_{i+d+1}. The XOR difference of the outputs of F_{i+1} and F_{i+d+1} can be again represented in two different ways leading to:

$$\sum_{1 \leq c \leq i+d-1} w(\Delta z_{i+c}^{(i+1)}) + \sum_{i+d+1 \leq c \leq 2d-1} w(\Delta z_{i+c}^{(i+d+1)}) + w(\Delta x_i \oplus \Delta x_{i+2d}) \geq m + 1.$$

Thus,

$$\sum_{1 \leq c \leq i+d-1} w(\Delta x_{i+c}) + \sum_{i+d+1 \leq c \leq 2d-1} w(\Delta x_{i+c}) + w(\Delta x_i) + w(\Delta x_{i+2d}) \geq m + 1.$$

□

3.4.3 Number of Differentially Active S-Boxes in $2(d+1)$ Rounds

We want to prove that every $2(d + 1)$ rounds of dCUFN-SP add $2(d - 1)(m + 1)$ differentially active S-boxes for[2] $d = 3$ and $d = 4$, if $[M_i|M_{i+d}]$ is an ODM:

Proposition 4 ($2(d+1)$ rounds of dCUFN-SP)**.** *For* $d = 3$ *and* $d = 4$*, if* $[M_i|M_{i+d}]$ *is an ODM for each* i*, every* $2(d+1)$ *consecutive rounds in the dCUFN-SP construction provide at least* $2(d - 1)(m + 1)$ *differentially active S-boxes.*

[2]We conjecture that a similar proposition holds for all $d > 2$ even in a stronger form: Every $d + 1$ rounds of dCUFN-SP add $(d - 1)(m + 1)$ differentially active S-boxes. Though this was indicated by our experiments, we were not able to find a compact proof for that.

We first reformulate the claim of Proposition 4. The number $A_{2(d+1)}$ of differentially active S-boxes within $2(d+1)$ rounds is the sum of all numbers of differentially active S-boxes in each F-function (there are exactly $3d$ relevant difference weights D_i within $2(d+1)$ rounds, see Figure 3.2):

$$A_{2(d+1)} = \sum_{c=1}^{d-2} c(D_{i+c-1} + D_{i+3d-c}) + \sum_{j=d-2}^{2d+1} (d-1)D_{i+j}.$$

To prove the proposition, one has to show that the inequality

$$A_{2(d+1)} \geq 2(d-1)(m+1)$$

holds for all possible nonzero input differences for $d = 3$ and $d = 4$. We will prove this inequality by treating all truncated difference distributions over $2(d+1)$ rounds, based on Propositions 2 and 3:

- If $\delta_{i+j+1}, \ldots, \delta_{i+j+d-1}$ are not all zero, Proposition 2 holds and the corresponding single-matrix relation for M_{i+j+1} can be used (if $[M_{i+j+1}|M_{i+j+d+1}]$ is an MDS matrix, each of its two submatrices M_{i+j+1} and $M_{i+j+d+1}$ is also an MDS matrix):

$$\tilde{s}_j = \sum_{0 \leq c \leq d} D_{i+j+c} \geq m+1, \ j \in \{0, \ldots, 2d-1\}.$$

- If $\delta_{i+j+1}, \ldots, \delta_{i+j+d-1}, \delta_{i+j+d+1}, \ldots, \delta_{i+j+2d-1}$ are not all zero, Proposition 3 holds and the corresponding double-matrix relation is applicable (as the matrix $[M_{i+j+1}|M_{i+j+d+1}]$ is an ODM):

$$\tilde{d}_j = \sum_{0 \leq c \leq 2d} D_{i+j+c} - D_{i+j+d} \geq m+1, \ j \in \{0, \ldots, d-1\}.$$

We treat $\tilde{s}_j$ and $\tilde{d}_j$ as polynomials in difference weights over the integers. Now the claim of the proposition reduces to the question, if there is at least one positive integer solution $v_0, \ldots, v_{2d-1}$, $u_0, \ldots, u_{d-1}$ to the polynomial equation

$$\sum_{j=0}^{2d-1} v_j \tilde{s}_j + \sum_{j=0}^{d-1} u_j \tilde{d}_j = A_{2(d+1)} \tag{3.3}$$

with

$$\sum_{j=0}^{2d-1} v_j + \sum_{j=0}^{d-1} u_j \geq 2(d-1)$$

for each possible distribution of truncated difference weights $\delta_i, ..., \delta_{i+3d-1}$. In doing so only the set of active single-matrix ($\tilde{s}_j$) and double-matrix ($\tilde{d}_j$) relations corresponding to a given distribution of truncated weights is to be used. The integers v_j and u_j define how many times the inequalities on s_j and d_j are used, respectively.

Not all possible distributions of truncated difference weights have to be considered. It can be proven that several of those truncated difference weight distributions are impossible (*impossible trails*). Moreover, there are large classes of related distributions (*trails with isolated zeros* and *symmetric trails*) for which we can reduce the problem of solving (3.3) to one representative. Furthermore, in some cases difference weights can be *transposed* while searching for a solution. These properties are formulated and proven in the next subsections as Lemmata 1 to 4. The proof of Proposition 4 is substantially based on these lemmata and can be sketched as follows.

Proof of Proposition 4. For $d = 3$ (respectively, for $d = 4$), consider all 2^9 (all 2^{12}) distributions of truncated difference weights $\delta_i, \ldots, \delta_{i+8}$ ($\delta_i, \ldots, \delta_{i+11}$). Using Lemmata 1 to 3, the number of cases to treat reduces to 20 (to 43). All other cases are either impossible (Lemma 1) or can be reduced to one of these cases (Lemma 2 and Lemma 3). In 4 (in 6) out of these cases, Lemma 4 has to be additionally used to find a solution. These distributions of truncated difference weights and the corresponding solutions are given in Table 3.1. □

From Table 3.1 one can observe that the two-matrix relations are needed only for a subset of all truncated difference weight distributions. In 8 out of 20 cases for $d = 3$ (respectively, in 21 out of 43 cases for $d = 4$), the two-matrix ODM property is indeed required, the single-matrix ODM property being sufficient for approximately the half of the cases.

3.4.4 Impossible and Equivalent Difference Weight Distributions

Now we prove the following lemmata which are used in the proof of Proposition 4 to identify relevant distributions of truncated difference weights and to find a solution to (3.3).

Lemma 1 (Impossible differential trails)**.** *If the input difference is nonzero and M_i is an ODM for each i, differential trails with the following difference weight distributions:*

- $(D_i, \ldots, D_{i+d-1})$ *with* $D_i = \cdots = D_{i+d-1} = 0$ *(d consecutive difference weights are all zero),*
- $(D_i, \ldots, D_{i+d})$ *with* $D_i = D_{i+d} = 0$ *and exactly one of* $D_{i+1}, D_{i+2}, \ldots, D_{i+d-1}$ *nonzero*

Table 3.1: Relevant distributions of truncated difference weights δ_{i+j} for $2(d+1)$ rounds of dCUFN-SP together with the corresponding non-negative integer solutions v_j and u_j to polynomial equation (3.3) for $d = 3$ and $d = 4$

$d = 3$

	δ_{i+j}									v_j						u_j		
j	0	1	2	3	4	5	6	7	8	1	2	3	4	5	6	1	2	3
	0	0	1	1	0	0	1	1	0	2	0	0	0	2	0	0	0	0
	0	0	1	1	0	0	1	1	1	2	0	0	0	1	1	0	0	0
	0	0	1	1	1	0	0	1	1	0	0	0	1	0	1	1	1	0
	0	0	1	1	1	1	0	0	1	1	0	0	0	2	0	0	1	0
	0	0	1	1	1	1	1	0	0	2	0	0	0	2	0	0	0	0
	0	0	1	1	1	1	1	1	0	2	0	0	0	2	0	0	0	0
	0	0	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	0
	0	1	1	0	0	1	1	1	0	2	0	0	0	2	0	0	0	0
	0	1	1	0	0	1	1	1	1	2	0	0	0	1	1	0	0	0
	0	1	1	1	0	0	1	1	1	2	0	0	0	1	1	0	0	0
	0	1	1	1	1	0	0	1	1	0	0	0	1	0	1	1	1	0
	0	1	1	1	1	1	0	0	1	1	0	0	0	2	0	0	1	0
	0	1	1	1	1	1	1	1	0	2	0	0	0	2	0	0	0	0
	0	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	0
	1	0	0	1	1	0	0	1	1	0	1	0	0	0	1	1	1	0
	1	0	0	1	1	1	0	0	1	0	1	0	0	1	0	0	2	0
	1	0	0	1	1	1	1	1	1	0	2	0	0	0	1	0	1	0
	1	1	0	0	1	1	1	1	1	1	0	1	0	0	0	0	1	1
	1	1	1	0	0	1	1	1	1	1	1	0	0	1	1	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	0

$d = 4$

	δ_{i+j}												v_j								u_j			
j	0	1	2	3	4	5	6	7	8	9	10	11	1	2	3	4	5	6	7	8	1	2	3	4
	0	0	0	1	1	0	0	0	1	1	0	0	3	0	0	0	0	3	0	0	0	0	0	0
	0	0	0	1	1	0	0	0	1	1	1	0	3	0	0	0	0	1	2	0	0	0	0	0
	0	0	0	1	1	0	0	0	1	1	1	1	3	0	0	0	0	1	1	1	0	0	0	0
	0	0	0	1	1	1	0	0	0	1	1	0	0	1	0	0	1	0	2	0	1	1	0	0
	0	0	0	1	1	1	0	0	0	1	1	1	0	1	0	0	1	0	1	1	1	1	0	0
	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0	0	0	2	0	1	0	1	1	0
	0	0	0	1	1	1	1	1	0	0	0	0	2	0	0	0	0	2	1	0	0	0	1	0
	0	0	0	1	1	1	1	1	1	0	0	0	3	0	0	0	0	3	0	0	0	0	0	0
	0	0	0	1	1	1	1	1	1	1	0	0	3	0	0	0	0	3	0	0	0	0	0	0
	0	0	0	1	1	1	1	1	1	1	1	0	1	0	2	0	0	1	0	2	0	0	0	0
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	0	0
	0	0	1	1	0	0	0	1	1	1	0	0	3	0	0	0	0	3	0	0	0	0	0	0
	0	0	1	1	0	0	0	1	1	1	1	0	3	0	0	0	0	1	2	0	0	0	0	0
	0	0	1	1	0	0	0	1	1	1	1	1	3	0	0	0	0	1	1	1	0	0	0	0
	0	0	1	1	1	0	0	0	1	1	1	0	3	0	0	0	0	1	2	0	0	0	0	0
	0	0	1	1	1	0	0	0	1	1	1	1	3	0	0	0	0	1	1	1	0	0	0	0
	0	0	1	1	1	1	0	0	0	1	1	0	0	1	0	0	1	0	2	0	1	1	0	0
	0	0	1	1	1	1	0	0	0	1	1	1	0	1	0	0	1	0	1	1	1	1	0	0
	0	0	1	1	1	1	1	0	0	0	1	1	1	0	0	0	0	2	0	1	0	1	1	0
	0	0	1	1	1	1	1	1	0	0	0	1	2	0	0	0	0	2	1	0	0	0	1	0
	0	0	1	1	1	1	1	1	1	0	0	0	3	0	0	0	0	3	0	0	0	0	0	0
	0	0	1	1	1	1	1	1	1	1	1	0	1	0	2	0	0	1	0	2	0	0	0	0
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	0	0
	0	1	1	0	0	0	1	1	0	0	0	1	2	0	0	2	0	0	0	0	0	0	1	1
	0	1	1	0	0	0	1	1	1	1	1	0	2	0	0	1	0	1	0	0	0	0	1	1
	0	1	1	0	0	0	1	1	1	1	1	1	2	0	0	1	0	1	0	0	0	0	1	1
	0	1	1	1	0	0	0	1	1	1	1	0	2	0	1	0	0	1	2	0	0	0	0	0
	0	1	1	1	0	0	0	1	1	1	1	1	2	0	1	0	0	1	1	1	0	0	0	0
	0	1	1	1	1	0	0	0	1	1	1	1	2	0	1	0	0	1	1	1	0	0	0	0
	0	1	1	1	1	1	0	0	0	1	1	1	0	0	1	0	1	0	1	1	1	1	0	0
	0	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	0	2	0	1	0	1	1	0
	0	1	1	1	1	1	1	1	0	0	0	1	0	2	0	0	0	0	3	0	0	0	1	0
	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	1	1	1	0	0	0	0
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	0	0
	1	0	0	0	1	1	0	0	0	1	1	1	0	2	0	0	0	0	1	1	1	1	0	0
	1	0	0	0	1	1	1	0	0	0	1	1	0	1	1	0	0	1	0	1	0	2	0	0
	1	0	0	0	1	1	1	1	0	0	0	1	0	2	0	0	0	0	1	1	0	1	0	0
	1	0	0	0	1	1	1	1	1	1	1	1	0	2	1	0	0	0	1	1	0	1	0	0
	1	1	0	0	0	1	1	0	0	0	1	1	1	0	2	0	0	0	0	1	0	1	1	0
	1	1	0	0	0	1	1	1	1	1	1	1	1	0	2	0	0	0	0	1	0	1	1	0
	1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	1	0	1	0	0	0	0	1	1
	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	0	0

cannot occur in dCUFN-SP.

Proof. The first claim of the lemma follows directly from the bijectivity of dCUFN-SP, since it is a keyed permutation (a nonzero input difference leads to a nonzero output difference).

To prove the second claim, we consider F_{i+1}. Since $D_i = D_{i+d} = 0$, one has:

$$D_i + D_{i+d} = 0. \tag{3.4}$$

At the same time, exactly one, say D_{i+j} for some $1 \leq j \leq d-1$, of $d-1$ input differences of F_{i+1} is nonzero. The maximum value of D_{i+j} is m. As M_{i+j} is an ODM, its branch number is $m+1$ and the minimum weight of the output difference for F_{i+1} is 1:

$$D_i + D_{i+d} \geq (m+1) - m = 1. \tag{3.5}$$

The contradiction between (3.4) and (3.5) proves the claim. □

Thus, a differential trail with any of these d truncated difference weight distributions is impossible and can be discarded in the further consideration.

Definition 10 (Isolated zeros)**.** *Let the difference weight distribution of a differential trail over r rounds of dCUFN-SP*

$$\mathcal{D} = (D_i, \ldots, D_{i+j-1}, D_{i+j}, \ldots, D_{i+j+c-1}, D_{i+j+c}, \ldots, D_{j+r})$$

be such that $D_{i+j-1} \neq 0$, $D_{i+j+c} \neq 0$ and $D_{i+j} = \cdots = D_{i+j+c-1} = 0$ for some $1 \leq c \leq r-2$. Then $\mathcal{D}$ is called a difference weight distribution with c consecutive isolated zeros.

Lemma 2 (Difference weight distributions with isolated zeros)**.** *Let $\mathcal{D}$ be a difference weight distribution for dCUFN-SP with up to $d-2$ consecutive isolated zeros. Let $\mathcal{D}'$ be $\mathcal{D}$ with these consecutive isolated zeros replaced by nonzero difference weights. Then, if there is a solution to (3.3) for $\mathcal{D}'$, there will be also a solution to (3.3) for $\mathcal{D}$. Moreover, each solution for $\mathcal{D}'$ will be a solution for $\mathcal{D}$.*

Proof. Let $D_{i+j-1} \neq 0, D_{i+j} \neq 0, \ldots, D_{i+j+d-3} \neq 0, D_{i+j+d-2} \neq 0$ be all nonzero in $\mathcal{D}'$. Setting them to zero $D_{i+j} = \cdots = D_{i+j+d-3} = 0$ in $\mathcal{D}$ does not change the set of active single- and double-matrix relations. That is, all relations used for $\mathcal{D}'$ can also be used for $\mathcal{D}$. This is due to the fact that setting these difference weights to zero does not open up additional runs of more than $d-2$ consecutive zeros (as D_{i+j-1} and $D_{i+j+d-2}$ are still nonzero).

Thus, the coefficients on the left-hand side of (3.3) remain the same. As both sides of (3.3) are equal as polynomials over the integers, eliminating some variables on both sides simultaneously does not change the equality. □

Lemma 3 (Symmetric differential trails)**.** *Let $\mathcal{D}$ be the difference weight distribution of a differential trail for dCUFN-SP. Let $\widetilde{\mathcal{D}}$ be $\mathcal{D}$ taken in the reverse order. Then, if there is a solution to (3.3) for $\mathcal{D}$, there is also a solution to (3.3) for $\widetilde{\mathcal{D}}$.*

Proof. The proposition follows directly from the symmetry of unbalanced Feistel networks, as the encryption and decryption procedures are equivalent up to the order of subkeys and diffusion matrices. □

These two lemmata say that it is not necessary to solve (3.3) for certain types of differential trails. So, (3.3) has to be solved for one of each two relevant symmetric difference weight distributions only. Moreover, it is not necessary to consider differential trails with up to $d-2$ isolated zero differences.

Lemma 4 (Difference weight transposition)**.** *Let $\mathcal{D}$ be a difference weight distribution for dCUFN-SP. If $\mathcal{D}$ has $d-1$ consecutive zero weights $D_{i+1} = D_{i+2} = \cdots = D_{i+d-1} = 0$, the weights of the two neighbouring differences are equal: $D_i = D_{i+d}$.*

Proof. Consider F_{i+1}. Since $D_{i+1} = D_{i+2} = \cdots = D_{i+d-1} = 0$, its input difference is zero. Then the output difference of F_{i+1} is also zero. Hence, $\Delta x_i \oplus \Delta x_{i+d} = 0$ and $D_i = D_{i+d}$. □

This lemma says that $\mathcal{D}$ can be modified by transposing difference weights over $d-1$ consecutive zero differences while looking for solutions to (3.3).

3.5 Differential Trail Probability over Many Rounds

A differential trail over r rounds is a sequence of r single-round differentials with compatible input and output differences. Consider r rounds of dCUFN-SP. Let A_r be the number of differentially active S-boxes in these r rounds. To prove a level of resistance against differential cryptanalysis provided by r rounds, we will first treat the probability of a differential trail for a given value of A_r. Let P_i be the probability of the single-round differential in round $i = 1, \ldots, r$. The probability of an r-round differential trail is commonly computed as the product of all r single-round differential probabilities: $\pi_r = \prod_{i=1}^{r} P_i$. As it has been shown in Section 3.3, the value of P_i is strongly dependent on the number t_i of differentially active S-boxes in round i. We will obtain an upper bound ν_r on π_r from a lower bound on A_r using the fact that ν_r as a function of A_r is monotonously decreasing.

First we consider the distribution of exactly A_r active S-boxes over r round functions in a way maximizing ν_r. The maximum of ν_r is attained when the number of active S-boxes in each round having active S-boxes is maximal. This happens if:

- Exactly $\alpha = \left\lfloor \frac{A_r}{m(d-1)} \right\rfloor$ round functions have all their $m(d-1)$ S-boxes active (see Figure 3.4 for $l = d-1$), which involves $\alpha m(d-1)$ active S-boxes, and
- All the remaining $\beta = A_r - \alpha m(d-1)$ active S-boxes occur in the same further round (see Figure 3.4 for $l < d-1$).

When combined with Proposition 1, this reasoning provides $\nu_r = p^{\alpha m+\beta}$, if $\beta \leq m$, and $\nu_r = p^{(\alpha+1)m}$, if $\beta > m$. Now recall the claim of Proposition 4 which says that there are at least $2(d-1)(m+1)$ active S-boxes in every $2(d+1)$ rounds, that is, $A_{2(d+1)} \geq 2(d-1)(m+1)$. If $m = 2$, we have $\alpha = 3$ and $\beta = 0$. If $m > 2$, we have $\alpha = 2$ and $\beta = 2(d-1)$. Using the kind of reasoning illustrated in Figure 3.4, one obtains

Theorem 3 (Differential trail probability over $2(d+1)R$ rounds)**.** *For dCUFN-SP with $d \in \{3, 4\}$, let $[M_i|M_{i+d}]$ be an ODM for each i. Let p be the maximum differential probability of a single S-box. Then there exist no differential trails over $2(d+1)R$ consecutive rounds with probability exceeding*

- $p^{2(m+d-1)R}$*, if* $2(d-1) \leq m$*, and*
- p^{3mR}*, if* $2(d-1) > m$*.*

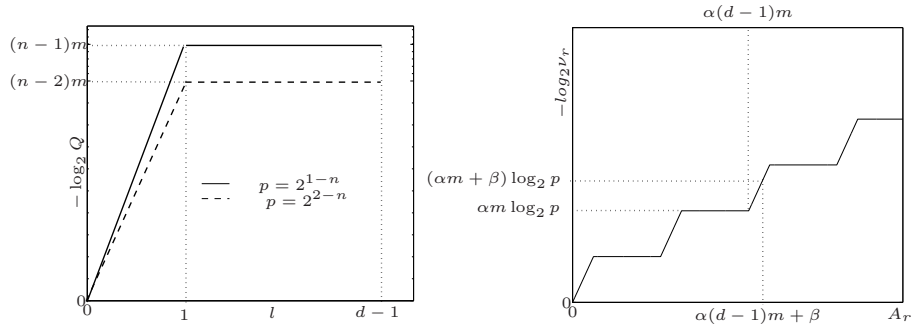


Figure 3.4: On the left: Upper bound Q on the single-round differential probability P as a function of $l = t/m$ (Proposition 1). On the right: Upper bound ν_r on the probability π_r of a differential trail with A_r active S-boxes in dCUFN-SP (Theorem 3)

3.6 Linear Trails of dCUFN-SP

In this section, we prove a lower bound on the number of linearly active S-boxes over $(d-1)R$ rounds of dCUFN-SP. To be able to do that, we first consider restrictions

imposed on the selection patterns by the SP-type structure of the round function F_i. It turns out that the linearity of compression makes it possible to prove strong properties of the weights of selection patterns. These are formulated as two lemmata. Finally, we combine the four lemmata to obtain the lower bound.

After that, we consider the limitations on the values of selection patterns which follow from the dCUFN construction without assuming anything about the internal structure of the round function. The results are also summarized as two lemmata.

3.6.1 Single-Round Linear Trails

In this subsection, single-round trails of dCUFN are considered. As mentioned above, the same value of x_i is used up to $d-1$ times in a dCUFN construction. The corresponding $d-1$ selection patterns $\Gamma x_i^{(1)}, \ldots, \Gamma x_i^{(d-1)}$ can be different (see Figure 3.5). The value of $F_i(x_i, \ldots, x_{i+d-2}, k_i)$ is denoted by y_i with the corresponding selection pattern Γy_i.

We first consider the propagation of selection patterns through the linear diffusion layer M_i of F_i. It compresses the intermediate values $z_i^{(i)}, \ldots, z_{i+d-2}^{(i)}$ to the output value y_i:

$$y_i^T = M_i \cdot [z_i^{(i)}, \ldots, z_{i+d-2}^{(i)}]^T. \tag{3.6}$$

For the input Γy_i and output $\Gamma z_{i+c}^{(i)}$ selection patterns of M_i, we have:

$$\Gamma y_i \cdot y_i^T \oplus \sum_{c=0}^{d-2} \Gamma z_{i+c}^{(i)} \cdot (z_{i+c}^{(i)})^T = 0.$$

Using (3.6), y_i can be replaced with expressions using intermediate values $z_{i+c}^{(i)}$:

$$\begin{aligned}
&\Gamma y_i \cdot \left(M_i \cdot [z_i^{(i)}, \ldots, z_{i+d-2}^{(i)}]^T \right) \oplus \textstyle\sum_{c=0}^{d-2} \Gamma z_{i+c}^{(i)} \cdot (z_{i+c}^{(i)})^T = \\
&\Gamma y_i \cdot M_i \cdot [z_i^{(i)}, \ldots, z_{i+d-2}^{(i)}]^T \oplus [\Gamma z_i^{(i)}, \ldots, \Gamma z_{i+d-2}^{(i)}] \cdot [z_i^{(i)}, \ldots, z_{i+d-2}^{(i)}]^T = \\
&\left(\Gamma y_i \cdot M_i \oplus [\Gamma z_i^{(i)}, \ldots, \Gamma z_{i+d-2}^{(i)}] \right) \cdot [z_i^{(i)}, \ldots, z_{i+d-2}^{(i)}]^T = 0.
\end{aligned}$$

The latter equality is attained for random values $z_i^{(i)}, \ldots, z_{i+d-2}^{(i)}$ iff the selection patterns on the left-hand side cancel out each other:

$$\Gamma y_i \cdot M_i = [\Gamma z_i^{(i)}, \ldots, \Gamma z_{i+d-2}^{(i)}].$$

In other words, expressing M_i through its square submatrices, we directly obtain:

$$\Gamma y_i \cdot [L_i^{(1)} | \ldots | L_i^{(d-1)}] = [\Gamma z_i^{(i)}, \ldots, \Gamma z_{i+d-2}^{(i)}].$$

This matrix equation is equivalent to the following $d-1$ equations holding simultaneously if all matrices $L_i^{(c)}$ with $c = 1, \ldots, d-1$ are invertible:

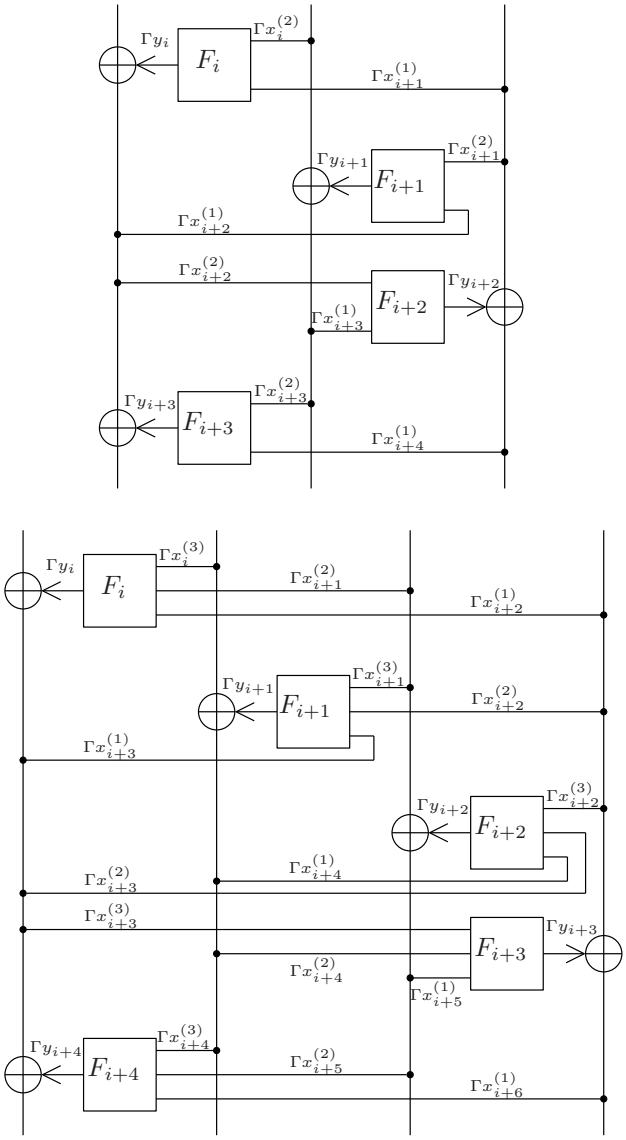


Figure 3.5: Linear selection patterns in dCUFN for $d = 3$ and $d = 4$

$$\begin{array}{l} \Gamma y_i \cdot L_i^{(1)} = \Gamma z_i^{(i)} \\ \Gamma y_i \cdot L_i^{(2)} = \Gamma z_{i+1}^{(i)} \\ \cdots \\ \Gamma y_i \cdot L_i^{(d-1)} = \Gamma z_{i+d-2}^{(i)} \end{array} \quad \Rightarrow \quad \begin{array}{l} \Gamma y_i = \Gamma z_i^{(i)} \cdot (L_i^{(1)})^{-1} \\ \Gamma y_i = \Gamma z_{i+1}^{(i)} \cdot (L_i^{(2)})^{-1} \\ \cdots \\ \Gamma y_i = \Gamma z_{i+d-2}^{(i)} \cdot (L_i^{(d-1)})^{-1}. \end{array}$$

Now we want to switch from selection patterns for the intermediate values of F_i to selection patterns for the inputs of F_i. We do that by proceeding to a weight-oriented consideration in the formulae above. As the weights of input and output selection patterns for the S-box layer of F_i coincide and $w(\Gamma z_{i+c-1}^{(i)}) = w(\Gamma x_{i+c-1}^{(d-c)})$ for $c = 1, \ldots, d-1$, the following two lemmata can be proven:

Lemma 5 (Zero and nonzero round selection patterns)**.** *If all matrices $L_i^{(c)}$ with $c = 1, \ldots, d-1$ are invertible, the following d linear selection patterns are either all zero or all nonzero:*

$$\Gamma y_i \text{ and } \Gamma x_{i+c}^{(d-1-c)} \text{ with } c = 0, \ldots, d-2.$$

Lemma 6 (Weights of round selection patterns)**.** *If $L_i^{(c_1)}(L_i^{(c_2)})^{-1}$ is an MDS matrix with $c_1, c_2 \in \{1, \ldots, d-1\}$, $c_1 \neq c_2$, and if $\Gamma x_{i+c_1-1}^{(d-c_1)}$ and $\Gamma x_{i+c_2-1}^{(d-c_2)}$ are nonzero, then*

$$w(\Gamma x_{i+c_1-1}^{(d-c_1)}) + w(\Gamma x_{i+c_2-1}^{(d-c_2)}) \geq m + 1.$$

3.6.2 Linear Trails over Many Rounds

Here we consider $d+1$ rounds of dCUFN with respect to linear cryptanalysis. Note that all linear relations in this subsection are valid for general dCUFNs without imposing any specific structure of the round functions. Our procedure in this subsection will be to linearly approximate the round functions of dCUFN and to combine these approximations in a way corresponding to the dCUFN architecture. This combination then yields a linear approximation for many rounds of dCUFN. Having obtained such an approximation, we make several useful observations concerning the limitations which are imposed on selection patterns by the dCUFN structure.

First, we linearly approximate each of the $d+1$ round functions $F_i, F_{i+1}, \ldots, F_{i+d}$ separately. For each round function, the approximation involves the corresponding output selection pattern as well as $d-1$ input selection patterns:

$$\begin{array}{l} \Gamma y_i \cdot y_i^T \oplus \Gamma x_i^{(d-1)} \cdot x_i^T \oplus \cdots \oplus \Gamma x_{i+d-2}^{(1)} \cdot x_{i+d-2}^T = 0 \\ \Gamma y_{i+1} \cdot y_{i+1}^T \oplus \Gamma x_{i+1}^{(d-1)} \cdot x_{i+1}^T \oplus \cdots \oplus \Gamma x_{i+d-1}^{(1)} \cdot x_{i+d-1}^T = 0 \\ \cdots \\ \Gamma y_{i+d} \cdot y_{i+d}^T \oplus \Gamma x_{i+d}^{(d-1)} \cdot x_{i+d}^T \oplus \cdots \oplus \Gamma x_{i+2d-2}^{(1)} \cdot x_{i+2d-2}^T = 0. \end{array} \tag{3.7}$$

Second, we consider the XOR-operations between branches in each of the $d+1$ rounds in dCUFN. Each XOR involves three variables which should have the same selection pattern:

$$\begin{array}{l} \Gamma y_i \cdot y_i^T \oplus \Gamma y_i \cdot x_{i-1}^T \oplus \Gamma y_i \cdot x_{i+d-1}^T = 0 \\ \Gamma y_{i+1} \cdot y_{i+1}^T \oplus \Gamma y_{i+1} \cdot x_i^T \oplus \Gamma y_{i+1} \cdot x_{i+d}^T = 0 \\ \ldots \\ \Gamma y_{i+d} \cdot y_{i+d}^T \oplus \Gamma y_{i+d} \cdot x_{i+d-1}^T \oplus \Gamma y_{i+d} \cdot x_{i+2d-1}^T = 0. \end{array} \tag{3.8}$$

Summing up all individual equations of (3.7) and (3.8) as well as eliminating

$$\Gamma y_i \cdot y_i^T, \ldots, \Gamma y_{i+d} \cdot y_{i+d}^T,$$

we obtain a linear approximation of $d+1$ rounds

$$\begin{array}{l} \Gamma_{i-1} \cdot x_{i-1}^T \oplus \Gamma_i \cdot x_i^T \oplus \cdots \oplus \Gamma_{i+d-2} \cdot x_{i+d-2}^T \oplus \\ \Gamma_{i+d} \cdot x_{i+d}^T \oplus \Gamma_{i+d+1} \cdot x_{i+d+1}^T \oplus \cdots \oplus \Gamma_{i+2d-1} \cdot x_{i+2d-1}^T \oplus \\ (\Gamma y_i \oplus \Gamma y_{i+d} \oplus \sum_{j=1}^{d-1} \Gamma x_{i+d-1}^{(j)}) \cdot x_{i+d-1}^T = 0 \end{array}$$

involving d selection patterns for inputs

$$\begin{array}{lcl} \Gamma_{i-1} & = & \Gamma y_i \\ \Gamma_i & = & \Gamma y_{i+1} \oplus \Gamma x_i^{(d-1)} \\ \ldots & & \\ \Gamma_{i+d-2} & = & \Gamma y_{i+d-1} \oplus \sum_{j=1}^{d-1} \Gamma x_{i+d-2}^{(j)}, \end{array} \tag{3.9}$$

d selection patterns for outputs

$$\begin{array}{lcl} \Gamma_{i+d} & = & \Gamma y_{i+d+1} \oplus \sum_{j=1}^{d-1} \Gamma x_{i+d}^{(j)} \\ \Gamma_{i+d+1} & = & \Gamma y_{i+d+2} \oplus \sum_{j=2}^{d-1} \Gamma x_{i+d+1}^{(j)} \\ \ldots & & \\ \Gamma_{i+2d-1} & = & \Gamma y_{i+2d} \end{array}$$

as well as $d+1$ intermediate linear selection patterns which have to sum up to zero:

$$\Gamma y_i \oplus \Gamma y_{i+d} \oplus \sum_{j=1}^{d-1} \Gamma x_{i+d-1}^{(j)} = 0. \tag{3.10}$$

Equation (3.10) immediately yields

Lemma 7 (Intermediate selection patterns)**.** *For dCUFN constructions, the following holds in a linear trail:*

$$\Gamma y_i \oplus \Gamma y_{i+d} \oplus \sum_{j=1}^{d-1} \Gamma x_{i+d-1}^{(j)} = 0.$$

As at least some of the input selection patterns are nonzero in a nontrivial linear approximation, from (3.9) one can directly derive the following

Lemma 8 (Input selection patterns)**.** *For a non-trivial linear trail of dCUFN, the following d values cannot be simultaneously zero:*

$$\Gamma y_i \text{ and } \Gamma y_{i+c} \oplus \sum_{j=d-c}^{d-1} \Gamma x_{i+c}^{(j)} \text{ with } c = 1, \ldots, d-1.$$

3.6.3 Linearly Active S-Boxes over $d+1$ Rounds

Here we combine the limitations on selection patterns related to SP-type compressing round functions with those related to the overall dCUFN construction. This allows us to prove a central property of dCUFN-SP regarding linear cryptanalysis which is mainly based on Lemmata 5,7 and 8. Loosely speaking, this property states that, if the linear diffusion matrices are properly chosen, there will be at least two linearly active rounds among every $d+1$ rounds:

Proposition 5 (Rounds with nonzero selection patterns)**.** *In a non-trivial linear trail of dCUFN-SP, there are at least two rounds with all linear selection patterns nonzero among $d+1$ consecutive rounds, if all matrices $L_i^{(c)}$ with $c = 1, \ldots, d-1$ are invertible.*

Proof. In a non-trivial linear trail for dCUFN-SP, the input selection pattern is nonzero (otherwise, it is impossible to get a linear approximation with a nonzero correlation value in a key-alternating cipher). According to Lemma 8, the selection patterns Γy_i, Γy_{i+c}, $\Gamma x_{i+c}^{(j)}$ for $j = d-c, \ldots, d-1$ and $c = 1, \ldots, d-1$ cannot be simultaneously zero for a non-trivial linear trail. Therefore, at least one of the selection patterns is nonzero. All these selection patterns belong to rounds $i, i+1, \ldots, i+d-1$. That is, according to Lemma 5, there is at least one round with all selection patterns being nonzero.

The $d+1$ selection patterns involved into the linear relation of Lemma 7 belong to different rounds i, $i+1$, $\ldots$, $i+d$. Lemma 7 says that either all $d+1$ values are zero or at least two of them are nonzero. Otherwise, if only one of the involved values is nonzero, the sum is nonzero, which contradicts to Lemma 7. Thus, at least two rounds have nonlinear linear selection patterns in a non-trivial linear trail of dCUFN-SP. □

Now using Proposition 5 and Lemma 6 we can prove an upper bound on the linear probability of a linear trail in dCUFN-SP which is formulated as

Theorem 4 (Linear probability over $(d+1)R$ rounds)**.** *For dCUFN-SP, let q be the maximum linear probability of a single S-box. Then there exist no linear trails over $(d+1)R$ consecutive rounds with linear probability exceeding $q^{(d-1)(m+1)R}$*

- *for d odd, if all $L_i^{(j)}(L_i^{(j+1)})^{-1}$ are ODM for $j \in \{1, 3, \ldots, d-2\}$ in each round i,*
- *for d even, if all $L_i^{(j)}(L_i^{(j+1)})^{-1}$ are ODM for $j \in \{1, 2, \ldots, d-2\}$ and if the matrix $L_i^{(1)}(L_i^{(d+1)})^{-1}$ is ODM in each round i.*

Proof. The theorem is proven by lower-bounding the number of linearly active S-boxes in $d+1$ consecutive rounds. In order to count the number of linearly active S-boxes in rounds i, $i+1$,...,$i+d$, one has to compute the sum of the weights of selection patterns on inputs to these rounds:

$$\sum_{j\in\{i,i+1,\ldots,i+d\}} \left(\sum_{c=1}^{d-1} w(\Gamma x_{j+c-1}^{(d-c)}) \right).$$

Proposition 5 states that there are at least two active rounds with nonzero linear selection patterns (say, rounds j_1 and j_2, $j_1 \neq j_2$).

If d is odd (and, therefore, $d-1$ is even), one can lower-bound the number of linearly active S-boxes in each of these two rounds j_1 and j_2, using $\frac{d-1}{2}$ inequalities according to Lemma 6:

$$\sum_{c\in\{1,3,\ldots,d-2\}} \left(w(\Gamma x_{j+c-1}^{(d-c)}) + w(\Gamma x_{j+c}^{(d-c+1)}) \right) \geq \frac{d-1}{2}(m+1).$$

If d is even (and, therefore, $d-1$ is odd), we use $d-1$ inequalities of Lemma 6 to obtain:

$$w(\Gamma x_j^{(d-1)}) + w(\Gamma x_{j+d-2}^{(1)}) + \sum_{c\in\{1,2,\ldots,d-2\}} \left(w(\Gamma x_{j+t-1}^{(d-c)}) + w(\Gamma x_{j+c}^{(d-c+1)}) \right) \geq (d-1)(m+1)$$

and

$$2\sum_{c=1}^{d-1} w(\Gamma x_{j+c-1}^{(d-c)}) \geq (d-1)(m+1)$$

with $j \in \{j_1, j_2\}$. By summing up these inequalities for both $j = j_1$ and $j = j_2$, we have:

$$2\sum_{c=1}^{d-1} w(\Gamma x_{j_1+c-1}^{(d-c)}) + 2\sum_{c=1}^{d-1} w(\Gamma x_{j_2+c-1}^{(d-c)}) \geq 2(d-1)(m+1)$$

The claims of the theorem follow. □

Table 3.2: Parameters of different block cipher constructions: dCUFN-SP, balanced Feistel networks BFN-SP-SR and BFN-SP-MR as well as SHARK-type and Rijndael-type SPNs

	b	S_r	L_r
BFN-SP-SR[135]	$2mn$	rm	rm^{1+a}
BFN-SP-MR[239]	$2mn$	rm	rm^{1+a}
SHARK-type SPN[222]	mn	rm	rm^{1+a}
Rijndael-type SPN[79]	dmn	rdm	rdm^{1+a}
dCUFN-SP	dmn	$r(d-1)m$	$r(d-1)m^{1+a}$

3.7 Discussion and Open Problems

On the efficiency of dCUFN-SP. To judge on the efficiency of different constructions, one has to define a metric in which to compare. There does not appear to be one realistic metric dealing with all implementation types and details. Therefore, any comparison not related to a concrete implementation is of illustrative nature.

However, there seem to be metrics that capture at least some important efficiency features. Inspired by [240] and [239], where the portion of active S-boxes in all S-boxes is applied for this purpose, we define the following efficiency metrics with respect to differential and linear cryptanalysis:

$$E_r^d = \frac{-\log_2 \pi_r}{S_r + \lambda L_r} \text{ and } E_r^l = \frac{-\log_2 \xi_r}{S_r + \lambda L_r},$$

where π_r is the maximum differential trail probability and ξ_r is the maximum linear probability over r rounds, S_r is the number of S-box computations within r rounds, L_r is the number of $\mathbb{F}_2^n$-multiplications by a constant within r rounds, and λ is the relative complexity of one $\mathbb{F}_2^n$-multiplication by a constant with respect to the complexity of one n-bit S-box computation. Note that typically $\lambda \leq 1$.

A linear increase of the size m of diffusion matrices often leads to a non-linear (subquadratic or quadratic) increase in the number L_r of $\mathbb{F}_2^n$-multiplications by constants. Thus, one matrix-vector multiplication requires m^{1+a} multiplications in $\mathbb{F}_2^n$ by constants with $0 \leq a \leq 1$, depending on the concrete matrix and implementation.

We derived the metric values for dCUFN-SP using Theorem 3 and Theorem 4 as well as corresponding results for several standard block cipher constructions. See Tables 3.3 for these results, where $\lim_{r\to\infty} E_r^d$ and $\lim_{r\to\infty} E_r^l$ practically characterize the efficiency of constructions over many rounds. This indicates that the efficiency

of dCUFN-SP for $d \in \{3, 4\}$ can be comparable to that of standard Feistel and SPN constructions. However, it is still an open problem to perform a more detailed study of the dCUFN-SP efficiency properties compared to other constructions.

Table 3.3: Differential and linear efficiency of dCUFN-SP in comparison with balanced Feistel networks BFN-SP-SR and BFN-SP-MR as well as SHARK-type and Rijndael-type SPNs. Note that constructions with similar block sizes b are comparable only (Table 3.2). μ_r, ν_r, η_r, ζ_r and ω_r are functions of r and m, vanishing for $r \to \infty$

	$-\log_2 \pi_r$	$\lim_{r\to\infty} E_r^d$
BFN-SP-SR[135], Th. 1	$\left((m+1)\left\lfloor\frac{r}{4}\right\rfloor + \left\lfloor\frac{r}{8}\right\rfloor + \mu_r\right)\log_2 p$	$\frac{-(m+1.5)\log_2 p}{4m(1+\lambda m^a)}$
BFN-SP-MR[239], Th. 2	$\left((m+1)\left\lfloor\frac{r}{3}\right\rfloor + \nu_r\right)\log_2 p$	$\frac{-(m+1)\log_2 p}{3m(1+\lambda m^a)}$
SHARK-type SPN[222]	$\left((m+1)\left\lfloor\frac{r}{2}\right\rfloor + \eta_r\right)\log_2 p$	$\frac{-(m+1)\log_2 p}{2m(1+\lambda m^a)}$
Rijndael-type SPN[79]	$\left((m+1)^2\left\lfloor\frac{r}{4}\right\rfloor + \zeta_r\right)\log_2 p$	$\frac{-(m+1)^2\log_2 p}{4dm(1+\lambda m^a)}$
dCUFN-SP, $d \in \{3,4\}$, Th. 3	$\left(1.5m\left\lfloor\frac{r}{d+1}\right\rfloor + \omega_r\right)\log_2 p$	$\frac{-1.5m\log_2 p}{(d^2-1)m(1+\lambda m^a)}$
	$-\log_2 \xi_r$	$\lim_{r\to\infty} E_r^l$
BFN-SP-SR[135], Th. 1	$\left((m+1)\left\lfloor\frac{r}{4}\right\rfloor + \left\lfloor\frac{r}{8}\right\rfloor + \mu_r\right)\log_2 q$	$\frac{-(m+1.5)\log_2 q}{4m(1+\lambda m^a)}$
BFN-SP-MR[239], Th. 2	$\left((m+1)\left\lfloor\frac{r}{3}\right\rfloor + \nu_r\right)\log_2 q$	$\frac{-(m+1)\log_2 q}{3m(1+\lambda m^a)}$
SHARK-type SPN[222]	$\left((m+1)\left\lfloor\frac{r}{2}\right\rfloor + \eta_r\right)\log_2 q$	$\frac{-(m+1)\log_2 q}{2m(1+\lambda m^a)}$
Rijndael-type SPN[79]	$\left((m+1)^2\left\lfloor\frac{r}{4}\right\rfloor + \zeta_r\right)\log_2 q$	$\frac{-(m+1)^2\log_2 q}{4dm(1+\lambda m^a)}$
dCUFN-SP, $d > 2$, Th. 4	$\left((d-1)(m+1)\left\lfloor\frac{r}{d+1}\right\rfloor + \omega_r\right)\log_2 q$	$\frac{-(m+1)\log_2 q}{(d+1)m(1+\lambda m^a)}$

On the extension of the results. There are examples of differential trails leading to exactly $2(d-1)(m+1)$ differentially active S-boxes over $2(d+1)$ rounds. That is, the upper bound on the number of differentially active S-boxes imposed by Proposition 4 is actually tight. At the same time we conjecture that the upper bound of Theorem 3 on the differential trail probability is not tight and can be lower in reality. This is mainly due to the reason that a pessimistic worst-case distribution of A_r differentially active S-boxes over r rounds is assumed in the proof. Thus, one open problem is to find and prove tighter bounds on the differential trail probability given the tight bound on $A_{2(d+1)}$.

Other attacks such as impossible differential cryptanalysis, truncated differential cryptanalysis, and multiset-type cryptanalysis are doubtlessly worth investigating with respect to the generic dCUFN-SP as well as to the special-case construction outlined in Subsection 3.2.1.

As mentioned in Section 3.2, our consideration opens up the possibility of a proper classification of generalized Feistel networks with SP-type domain-preserving round functions with respect to their security properties. We think that this would substantially advance our understanding of generalized Feistel networks.

Chapter 4

Design and Analysis of Lightweight Cryptographic Algorithms

With the establishment of the AES the need for new block ciphers has been greatly diminished; for almost all block cipher applications the AES is an excellent and preferred choice. However, despite recent implementation advances, the AES is not suitable for extremely constrained environments such as RFID tags and sensor networks. In this chapter we describe a lightweight block cipher, PRESENT. Both security and hardware efficiency have been equally important during the design of the cipher. The hardware requirements for PRESENT are competitive with today's leading compact stream ciphers: Depending on implementation as few as 1000 GE may be needed.

The security challenges posed by RFID-tag deployments are well-known. In response there is a rich literature on new cryptographic protocols and an on-tag hash function is often assumed by protocol designers. Yet cheap tags pose severe implementation challenges and it is far from clear that a suitable hash function even exists. In this chapter we consider the options available, including constructions based around compact block ciphers such as PRESENT. Several concrete hash function constructions are considered.

This chapter is based on the joint work of the author with Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matt J.B. Robshaw, Yannick Seurin and Charlotte Vikkelsoe published in [44] and [45].

4.1 Introduction

One defining trend of this century's IT landscape will be the extensive deployment of tiny computing devices. Not only will these devices feature routinely in consumer items, but they will form an integral part of a pervasive communication infrastructure. It is already recognized that such deployments bring a range of very particular security risks. Yet at the same time the cryptographic solutions, and particularly the cryptographic primitives, we have at hand are unsatisfactory for extremely resource-constrained environments.

In this chapter we propose a new hardware-optimized block cipher that has been carefully designed with area and power constraints uppermost in our mind. Yet, at the same time, we have tried to avoid a compromise in security. In achieving this we have looked back at the pioneering work embodied in the DES [195] and complemented this with features from the AES finalist candidate Serpent [22] which demonstrated excellent performance in hardware.

At this point it would be reasonable to ask why we might want to design a new block cipher. After all, it has become an "accepted" fact that stream ciphers are, potentially, more compact. Indeed, renewed efforts to understand the design of compact stream ciphers were made in the eSTREAM [90] project, where several promising proposals offer appealing performance profiles. But we note a couple of reasons why we might want to consider a compact block cipher. First, a block cipher

is a versatile primitive and by running a block cipher in *counter mode* (say) we get a stream cipher. But second, and perhaps more importantly, the art of block cipher design seems to be a little better understood than that of stream ciphers. For instance, while there is a rich theory under-pinning the use of linear feedback shift registers [169] it is not easy to combine these building blocks to give a secure proposal. We suspect that a carefully designed block cipher could be a less risky undertaking than a newly designed stream cipher. Thus, we feel that a block cipher that requires similar hardware resources as a compact stream cipher could be of considerable interest.

It is important to realise that in developing a new block cipher, particularly one with aggressive performance characteristics, we are not just looking for innovative implementation. Rather, the design and implementation of the cipher go hand-in-hand and this has revealed several fundamental limits and inherent contradictions. For instance, a given security level places lower bounds on the block length and key length. Just processing a 64-bit state with an 80-bit key places fundamental lower limits on the amount of space we require. We also observe that hardware implementation — particularly compact hardware implementation — favours repetition. Even minor variations can have an unfortunate effect on the space required for an implementation. Yet, at the same time, the cryptanalyst also favours repetition and seeks mathematical structures that propagate easily across many rounds. How much simple, repetitive structure can we include without compromising its security?

A variety of new protocols have been proposed and in many of them, particularly ones intended to preserve user privacy and to anonymize tag interactions, it is assumed that a cryptographic hash function will be used on the tag.

However which hash function might be used in practice is rarely identified. Looking at dedicated hash functions from the last 20 years, we have become used to their impressive hashing speed (though this is a view that we might have to change in the future). This fast throughput might lead some designers to believe that hash functions are "efficient" in other regards and that they can be routinely used in low-cost environments. This is a mistake, a point that was convincingly made in a paper by Feldhofer and Rechberger [100]. Generally speaking, current hash functions are not at all suitable for constrained environments. They require significant amounts of state and the operations in current dedicated designs are not hardware friendly. This is not surprising since modern hash functions were designed with 32-bit processors in mind, but it means that very few RFID-oriented protocols appealing to a hash function could ever be used on a modestly-capable tag.

After a brief survey of the existing literature, the rest of the chapter is organised as follows. PRESENT is described in Section 4.3 with the design decisions described in Section 4.4. The security analysis follows in Section 4.5 along with a detailed performance analysis in Section 4.6. In this chapter we also consider RFID tag-enabled applications and the use of hash functions in RFID protocols. We then turn

our attention to the design of hash functions in Section 4.7 and we explore whether a block cipher makes an appropriate starting point for a compact hash function instead of a dedicated design. In Section 4.8 we instantiate lightweight hash functions using literature-based constructions and the compact block cipher PRESENT. This allows us to implement a range of representative constructions that, for their given parameter sets, are the most compact hash functions available today. In Section 4.9 we then look at some challenging problems in designing hash functions with greater hash output lengths. While the chapter reveals positive results, our work also serves to highlight the difficult issue of compact hash functions; we therefore close the chapter with problems for future research.

4.2 Cryptography and RFID Tags

When considering applications based around the deployment of RFID tags we are working with very specific applications with rather unique requirements. The difficulty of implementing cryptography in such environments has spurred considerable research, and this can be roughly divided into two approaches:

1. Devise new algorithms and protocols for tag-based applications. Such new protocols might use new cryptographic problems or might be based almost exclusively on very lightweight operations, *e.g.* bitwise exclusive-or and vector inner-products.

2. Optimise existing algorithms and protocols so that they become suitable for RFID tag-based applications. This approach might not give the compact results that some of the more exotic proposals do, but the security foundations may be more stable.

The work in this chapter is more in-line with the second approach—seeing what we can do with what we have—though we hope it will be helpful to protocol designers.

4.2.1 Lightweight Ciphers

While there is a growing body of work on low-cost cryptography, the number of papers dealing with ultra-lightweight ciphers is surprisingly limited. Since our focus is on algorithm design we won't refer to work on low-cost communication and authentication protocols. Some of the most extensive work on compact implementation took place within the eSTREAM project. As part of that initiative, new stream ciphers suitable for efficient hardware implementation were proposed. Some promising candidates were proposed [58, 111]. While the trade-offs are complex, implementation papers [108] suggest that around 1300-2600 *gate equivalents* (GE) would be required for the more compact eSTREAM ciphers.

With regards to block ciphers it is well-known that DES was designed with hardware efficiency in mind. Given the very limited state of semiconductor circuits in the early 1970s, it is not surprising that DES possesses very competitive implementation properties. Work on DES reveals an implementaton of around 3000 GE [250] while a serialized implementation can be realized with around 2300 GE [162]. The key length of DES limits its usefulness in many applications and makes proposals such as DESXL (2168 GE) of some considerable interest [162].

For modern block ciphers, the landmark paper of [99] gives a very thorough analysis of a low-cost implementation of the AES [193]. However, the resources required for this cipher are around 3600 GE, which is an indirect consequence of the fact that Rijndael was designed for software efficiency on 8- and 32-bit processors. Implementation requirements for the *Tiny Encryption Algorithm* TEA [255, 256] are not known, but a crude estimate is that TEA needs at least 2100 GE and XTEA needs[1] at least 2000 GE. Four dedicated proposals for low-cost implementation are mCrypton [170], HIGHT [124], SEA [246], and CGEN [227], though the latter is not primarily intended as a block cipher. mCrypton has a precise hardware assessment and requires 2949 GE, HIGHT requires around 3000 GE while SEA with parameters comparable to PRESENT requires around 2280 GE.

4.2.2 Lightweight Hash Functions

Informally, a cryptographic hash function H takes an input of variable size and returns a hash value of fixed length while satisfying the properties of preimage resistance, second preimage resistance, and collision resistance [179]. For a hash function with n-bit output, compromising these should require 2^n, 2^n, and $2^{n/2}$ operations respectively. These properties make hash functions very appealing in a range of protocols. For tag-based applications, the protocols in question are often focused on authentication or on providing some form of anonymity and/or privacy [3, 6, 87, 102, 114, 166, 203]. However some estimates suggest that no more than 2 000 GE are available for security in low-cost RFID tags [132] and the hash functions available are unsuitable in practice. When we consider what we need from a hash function in an RFID tag-based application the following issues can be identified:

1. In tag-based applications we are unlikely to hash large amounts of data. Most tag protocols require that the hash function process a challenge, an identifier, and/or perhaps a counter. The typical input is usually much less than 256 bits.

[1] These figures and others in Section 4.2.1 are "back-of-an-envelope" where we assume the following requirements: 32-bit XOR = 80 GE, 32-bit arithmetic ADD = 148 GE, 192-bit FF = 1344 GE, SHIFT = 0 GE. All estimated figures lack any control logic which might significantly increase the required area.

2. In many tag-based applications we do not need the property of *collision resistance*. Most often the security of the protocol depends on the *one-way* property. In certain situations, therefore, it is safe to use hash functions with smaller hash outputs.

3. Applications will (typically) only require moderate security levels. Consequently 80-bit security, or even less, may be adequate. This was also the position taken in the eSTREAM project [90]. An algorithm should be chosen according to the relevant security level and in deployment, where success depends on every dollar and cent spent, there is no point using extra space to get a 256-bit security level if 64-bit security is all that is required.

4. While the physical space for an implementation is often the primary consideration, the peak and average power consumption are also important. The time for a computation will matter if we consider how a tag interacts with higher-level communication and anti-collision protocols.

5. Some protocols use a hash function to build a *message authentication code* (MAC), often by appealing to the HMAC construction [194]. When used as a MAC a number of interesting issues arise such as choosing an appropriate key length and understanding whether keys will be changed, something that will almost certainly be impossible in most tag-enabled applications. There might also be the possibility of side-channel analysis on the MAC. However such attacks will rarely be worthwhile for cheap tag-enabled applications and we do not consider this issue further.

Taking account of these considerations allows us to make some pragmatic choices. There will be applications that just require one-wayness and the application may only require 80-bit or 64-bit security. Note that this is the view adopted by Shamir in the proposal SQUASH for use in RFID tags [237]. For other applications we might like to see 80-bit security against collision attacks.

Since current hash functions of dedicated design are either too big or broken, we first consider hash functions that are built around block ciphers. In particular we use the compact block cipher PRESENT as a building block and we consider the implementation of a range of hash functions offering 64-bit and 128-bit outputs using established techniques. We also consider hash functions that offer larger outputs and we highlight some design directions along with their potential hardware footprint.

4.3 The Block Cipher PRESENT

PRESENT is an example of an SP-network [179] and consists of 31 rounds. The block length is 64 bits and two key lengths of 80 and 128 bits are supported. Given the applications we have in mind, we recommend the version with 80-bit keys. This

generateRoundKeys()
for $i = 1$ to 31 **do**

  addRoundKey(STATE,K_i)
  sBoxLayer(STATE)
  pLayer(STATE)
end for

addRoundKey(STATE,K_{32})

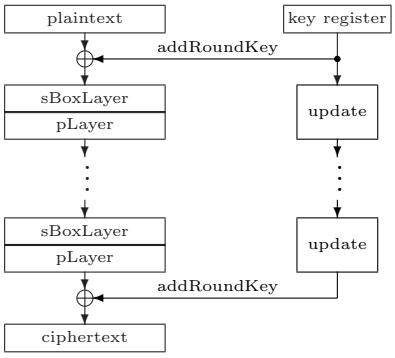


Figure 4.1: A top-level algorithmic description of PRESENT

is more than adequate security for the low-security applications typically required in tag-based deployments, but just as importantly, this matches the design goals of hardware-oriented stream ciphers in the eSTREAM project and allows us to make a fairer comparison. The security claims and performance attributes of the 128-bit version are provided in an appendix.

Each of the 31 rounds consists of an XOR operation to introduce a round key K_i for $1 \leq i \leq 32$, where K_{32} is used for post-whitening, a linear bitwise permutation and a non-linear substitution layer. The non-linear layer uses a single 4-bit S-box S which is applied 16 times in parallel in each round. The cipher is described in pseudo-code in Figure 4.1, and each stage is now specified in turn. The design rationale are given in Section 4.4 and throughout we number bits from zero with bit zero on the right of a block or word.

addRoundKey. Given round key $K_i = \kappa^i_{63} \dots \kappa^i_0$ for $1 \leq i \leq 32$ and current STATE $b_{63} \dots b_0$, addRoundKey consists of the operation for $0 \leq j \leq 63$,

$$b_j \rightarrow b_j \oplus \kappa^i_j.$$

sBoxLayer. The S-box used in PRESENT is a 4-bit to 4-bit S-box $S : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$. The action of this box in hexadecimal notation is given by the following table.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

For sBoxLayer the current cipher STATE $b_{63} \dots b_0$ is considered as sixteen 4-bit words $w_{15} \dots w_0$ where $w_i = b_{4*i+3}||b_{4*i+2}||b_{4*i+1}||b_{4*i}$ for $0 \leq i \leq 15$ and the output nibble $S[w_i]$ provides the updated state values in the obvious way.

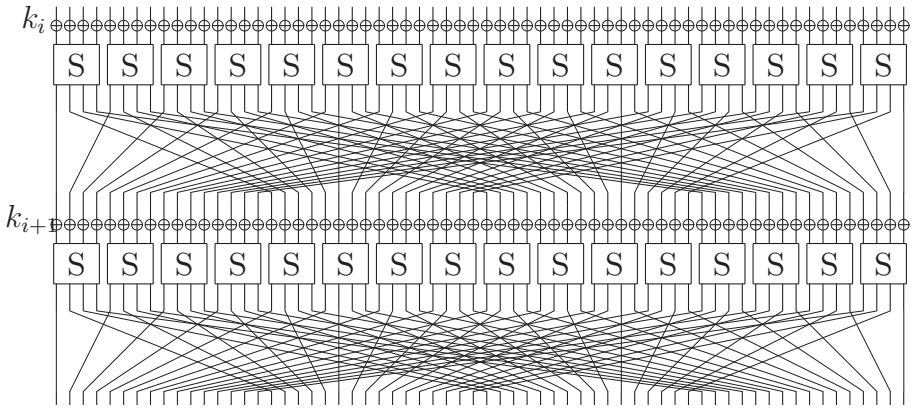


Figure 4.2: The SP-network for PRESENT

pLayer. The bit permutation used in PRESENT is given by the following table. Bit i of STATE is moved to bit position $P(i)$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

4.3.1 The Key Schedule

PRESENT can take keys of either 80 or 128 bits. However we focus on the version with 80-bit keys. The user-supplied key is stored in a key register K and represented as $k_{79}k_{78}\ldots k_0$. At round i the 64-bit round key $K_i = \kappa_{63}\kappa_{62}\ldots\kappa_0$ consists of the 64 leftmost bits of the current contents of register K. Thus at round i we have that:

$$K_i = \kappa_{63}\kappa_{62}\ldots\kappa_0 = k_{79}k_{78}\ldots k_{16}.$$

After extracting the round key K_i, the key register $K = k_{79}k_{78}\ldots k_0$ is updated as follows.

1. $[\mathtt{k}_{79}\mathtt{k}_{78}\ldots\mathtt{k}_1\mathtt{k}_0] = [\mathtt{k}_{18}\mathtt{k}_{17}\ldots\mathtt{k}_{20}\mathtt{k}_{19}]$
2. $[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}] = S[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}]$
3. $[\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] = [\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] \oplus \mathtt{round\_counter}$

Thus, the key register is rotated by 61 bit positions to the left, the left-most four bits are passed through the PRESENT S-box, and the `round_counter` value i is exclusive-ored with bits $\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}$ of K with the least significant bit of `round_counter` on the right.

4.3.2 Optional Key Schedule for 128-Bit Keys

Here we describe a key schedule for a version of PRESENT that takes 128-bit keys. The user-supplied key is stored in a key register K and represented as $k_{127}k_{126}\ldots k_0$. At round i the 64-bit round key $K_i = \kappa_{63}\kappa_{62}\ldots\kappa_0$ consists of the 64 leftmost bits of the current contents of register K. Thus at round i we have that:

$$K_i = \kappa_{63}\kappa_{62}\ldots\kappa_0 = k_{127}k_{126}\ldots k_{64}.$$

After extracting the round key K_i, the key register $K = k_{127}k_{126}\ldots k_0$ is updated as follows.

1. $[\mathtt{k}_{127}\mathtt{k}_{126}\ldots\mathtt{k}_1\mathtt{k}_0] = [\mathtt{k}_{66}\mathtt{k}_{65}\ldots\mathtt{k}_{68}\mathtt{k}_{67}]$
2. $[\mathtt{k}_{127}\mathtt{k}_{126}\mathtt{k}_{125}\mathtt{k}_{124}] = S[\mathtt{k}_{127}\mathtt{k}_{126}\mathtt{k}_{125}\mathtt{k}_{124}]$
3. $[\mathtt{k}_{123}\mathtt{k}_{122}\mathtt{k}_{121}\mathtt{k}_{120}] = S[\mathtt{k}_{123}\mathtt{k}_{122}\mathtt{k}_{121}\mathtt{k}_{120}]$
4. $[\mathtt{k}_{66}\mathtt{k}_{65}\mathtt{k}_{64}\mathtt{k}_{63}\mathtt{k}_{62}] = [\mathtt{k}_{66}\mathtt{k}_{65}\mathtt{k}_{64}\mathtt{k}_{63}\mathtt{k}_{62}] \oplus \mathtt{round\_counter}$

Thus, the key register is rotated by 61 bit positions to the left, the left-most eight bits are passed through two PRESENT S-boxes, and the `round_counter` value i is exclusive-ored with bits $\mathtt{k}_{66}\mathtt{k}_{65}\mathtt{k}_{64}\mathtt{k}_{63}\mathtt{k}_{62}$ of K with the least significant bit of `round_counter` on the right.

4.4 Design Issues for PRESENT

Besides security and efficient implementation, the main goal when designing PRESENT was simplicity. It is therefore not surprising that similar designs have been considered in other contexts [117] and can even be used as a tutorial for students [116]. In this section we justify the decisions we took during the design of PRESENT.

The permutation layer. When choosing the mixing layer, our focus on hardware efficiency demands a linear layer that can be implemented with a minimum number of processing elements, *i.e.* transistors. This leads us directly to bit permutations. Given our focus on simplicity, we have chosen a regular bit-permutation and this helps to make a clear security analysis (see Section 4.5).

The S-box. We use a single 4-bit to 4-bit S-box $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ in PRESENT. This is a direct consequence of our pursuit of hardware efficiency, with the implementation of such an S-box typically being much more compact than that of an 8-bit S-box. Since we use a bit permutation for the linear diffusion layer, AES-like diffusion techniques [79] are not an option for PRESENT. Therefore we place some additional conditions on the S-boxes to improve the so-called *avalanche of change*. More precisely, the S-box for PRESENT fullfils the following conditions, where we denote the

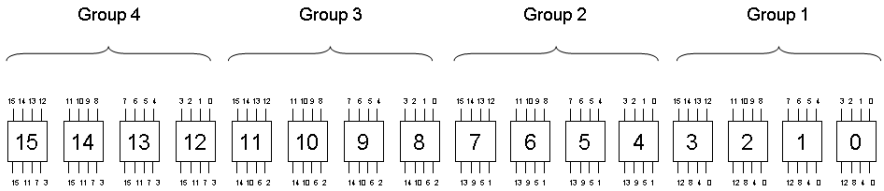


Figure 4.3: The grouping of S-boxes in PRESENT for the purposes of cryptanalysis. The input numbers indicate the S-box origin from the preceeding round and the output numbers indicate the destination S-box in the following round

Fourier coefficient of S by

$$S_b^W(a) = \sum_{x \in \mathbb{F}_2^4} (-1)^{\langle b, S(x) \rangle + \langle a, x \rangle}.$$

1. For any fixed non-zero input difference $\Delta_I \in \mathbb{F}_2^4$ and any fixed non-zero output difference $\Delta_O \in \mathbb{F}_2^4$ we require

$$\#\{x \in \mathbb{F}_2^4 \,|S(x) + S(x + \Delta_I) = \Delta_O\} \leq 4.$$

2. For any fixed non-zero input difference $\Delta_I \in \mathbb{F}_2^4$ and any fixed output difference $\Delta_O \in \mathbb{F}_2^4$ such that $\mathrm{wt}(\Delta_I) = \mathrm{wt}(\Delta_O) = 1$ we have

$$\{x \in \mathbb{F}_2^4 \,|S(x) + S(x + \Delta_I) = \Delta_O\} = \emptyset.$$

3. For all non-zero $a \in \mathbb{F}_2^4$ and all non-zero $b \in \mathbb{F}_4$ it holds that $|S_b^W(a)| \leq 8$.

4. For all $a \in \mathbb{F}_2^4$ and all non-zero $b \in \mathbb{F}_4$ such that $\mathrm{wt}(a) = \mathrm{wt}(b) = 1$ it holds that $S_b^W(a) = \pm 4$.

As will become clear in Section 4.5, these conditions will ensure that PRESENT is resistant to differential and linear attacks. Using a classification of all 4-bit S-boxes that fulfill the above conditions [163] we chose an S-box that is particularly well-suited to efficient hardware implementation.

4.5 Security Analysis of PRESENT

We now present the results of a security analysis of PRESENT.

4.5.1 Differential and Linear Cryptanalysis

Differential [24] and linear [176] cryptanalysis are among the most powerful techniques available to the cryptanalyst. In order to gauge the resistance of PRESENT to differential and linear cryptanalysis we provide a lower bound to the number of so-called *active* S-boxes involved in a differential (or linear) characteristic.

Differential cryptanalysis. The case of differential cryptanalysis is captured by the following theorem.

Theorem 5. *Any five-round differential characteristic of* PRESENT *has a minimum of 10 active S-boxes.*

We first make the following observations. We divide the 16 S-boxes into four groups (see Figure 4.3) and by examining the permutation layer one can then establish the following.

1. The input bits to an S-box come from 4 distinct S-boxes of the same group.
2. The input bits to a group of four S-boxes come from 16 different S-boxes.
3. The four output bits from a particular S-box enter four distinct S-boxes, each of which belongs to a distinct group of S-boxes in the subsequent round.
4. The output bits of S-boxes in distinct groups go to distinct S-boxes.

Proof. Recalling that the rounds are indexed from 1 to 31, consider five consecutive rounds of PRESENT ranging from $i-2$ to $i+2$ for $i \in [3 \dots 29]$. Let D_j be the number of active S-boxes in round j. If $D_j \geq 2$, for $i-2 \leq j \leq i+2$, then the theorem trivially holds. So let us suppose that one of the D_j is equal to one. We can distinguish several cases:

Case $D_i = 1$. The S-box of PRESENT is such that a difference in a single input bit causes a difference in at least two output bits (*cf.* the second design criterion). Thus $D_{i-1} + D_{i+1} \geq 3$. Using observation 1 above, all active S-boxes of round $i-1$ belong to the same group, and each of these active S-boxes have only a single bit difference in their output. So according to observation 2 we have that $D_{i-2} \geq 2D_{i-1}$. Conversely, according to observation 3, all active S-boxes in round $i+1$ belong to distinct groups and have only a single bit difference in their input. So according to observation 4 we have that $D_{i+2} \geq 2D_{i+1}$. Together this gives $\sum_{j=i-2}^{i+2} D_j \geq 1 + 3 + 2 \times 3 = 10$.

Case $D_{i-1} = 1$. If $D_i = 1$ we can refer to the first case, so let us suppose that $D_i \geq 2$. According to observation 3 above, all active S-boxes of round i belong to distinct groups and have only a single bit difference in their input.

Thus, according to observation 4, $D_{i+1} \geq 2D_i \geq 4$. Further, all active S-boxes in round $i+1$ have only a single bit difference in their input and they are distributed so that at least two groups of S-boxes contain at least one active S-box. This means that $D_{i+2} \geq 4$ and we can conclude that $\sum_{j=i-2}^{i+2} D_j \geq 1+1+2+4+4 = 12$.

Case $D_{i+1} = 1$. If $D_i = 1$ we can refer to the first case. So let us suppose that $D_i \geq 2$. According to observation 1 above, all active S-boxes of round i belong to the same group and each of these active S-boxes has only a single bit difference in their output. Thus, according to observation 2, $D_{i-1} \geq 2D_i \geq 4$. Further, all active S-boxes of round $i-1$ have only a single bit difference in their output, and they are distributed so that at least two groups contain at least two active S-boxes. Thus, we have that $D_{i-2} \geq 4$ and therefore that $\sum_{j=i-2}^{i+2} D_j \geq 4+4+2+1+1 = 12$.

Cases $D_{i+2} = 1$ or $D_{i-2} = 1$. The reasoning for these cases is similar to those for the second and third cases.

The theorem follows. □

By using Theorem 5 any differential characteristic over 25 rounds of PRESENT must have at least $5 \times 10 = 50$ active S-boxes. The maximum differential probability of a PRESENT S-box is 2^{-2} and so the probability of a single 25-round differential characteristic is bounded by 2^{-100}. Advanced techniques allow the cryptanalyst to remove the outer rounds from a cipher to exploit a shorter characteristic. However even if we allow an attacker to remove six rounds from the cipher, a situation without precedent, then the data required to exploit the remaining 25-round differential characteristic exceeds the amount available. Thus, the security bounds are more than we require. However, we have practically confirmed that the bound on the number of active S-boxes in Theorem 5 is tight.

Practical confirmation. We can identify characteristics that involve ten S-boxes over five rounds. The following two-round iterative characteristic involves two S-boxes per round and holds with probability 2^{-25} over five rounds.

$$\begin{aligned} \Delta &= 0000000000000011 \\ &\rightarrow 0000000000030003 \\ &\rightarrow 0000000000000011 = \Delta. \end{aligned}$$

A more complicated characteristic holds with probability 2^{-21} over five rounds.

$$\begin{aligned}
\Delta &= 0000000000007070 \\
&\rightarrow 000000000000000A \\
&\rightarrow 0001000000000000 \\
&\rightarrow 0000000010001000 \\
&\rightarrow 0000000008800088 \\
&\rightarrow 0033000000330033.
\end{aligned}$$

While the probability of this second characteristic is very close to the bound of 2^{-20}, it is non-iterative and of little practical value. Instead we have experimentally confirmed the probability of the two-round iterative differential characteristic. In experiments over 100 independent sub-keys using 2^{23} chosen plaintext pairs, the observed probability was as predicted. This seems to suggest that for this particular characteristic there is no accompanying significant differential. However, determining the extent of any differential effect is a complex and time-consuming task even though our preliminary analysis has been encouraging.

Linear cryptanalysis. The case of the linear cryptanalysis of PRESENT is handled by the following theorem where we analyse the best linear approximation to four rounds of PRESENT.

Theorem 6. *Let $\epsilon_{4\mathrm{R}}$ be the maximal bias of a linear approximation of four rounds of* PRESENT. *Then $\epsilon_{4\mathrm{R}} \leq \frac{1}{2^7}$.*

Proof. Recall that Matsui's piling-up lemma [176] estimates the bias of a linear approximation involving n S-boxes to be

$$2^{n-1} \prod_{i=1}^{n} \epsilon_i,$$

where the values ϵ_i are the individual bias of each (independent) S-box. According to the design principles of PRESENT, the bias of all linear approximations is less than 2^{-2} while the bias of any single-bit approximation is less than 2^{-3}. Let $\epsilon_{4\mathrm{R}}^{(j)}$ denote the bias of a linear approximation over 4 rounds involving j active S-boxes. Now consider the following three cases.

1. Suppose that each round of a four-round linear approximation has exactly one active S-box. Then the bias of each of the two S-boxes in the middle rounds is at most 1/8 and the overall bias for a four round approximation can be bounded as follows:

$$\epsilon_{4\mathrm{R}}^{(4)} \leq 2^3 \times (2^{-3})^2 \times (2^{-2})^2 = 2^{-7}.$$

2. Suppose, instead, that there are exactly five active S-boxes over four rounds. Then by the grouping of S-boxes in Figure 4.3, the active S-boxes over three consecutive rounds cannot form the pattern 1-2-1. For this to happen, the two active S-boxes in the middle round are activated by the same S-box and must therefore belong to two different groups of S-boxes. But if this is the case they couldn't activate only one S-box in the following round. Consequently the number of active S-boxes is either 2-1-1-1 or 1-1-1-2, so that

$$\epsilon_{4\mathrm{R}}^{(5)} \leq 2^4 \times (2^{-3}) \times (2^{-2})^4 = 2^{-7}.$$

3. Finally, suppose that there are more than five active S-boxes. Thus

$$\epsilon_{4\mathrm{R}}^{(j)} \leq 2^{j-1} \times (2^{-2})^j = 2^{-j-1} \leq 2^{-7} \text{ for } j > 5.$$

   The equality is theoretically attainable for $j = 6$. This is a strict inequality for all other j's.

The theorem follows.

□

We can use this theorem directly to bound the maximal bias of a 28-round linear approximation by

$$2^6 \times \epsilon_{4\mathrm{R}}^7 = 2^6 \times (2^{-7})^7 = 2^{-43}.$$

Therefore under the assumption that a cryptanalyst need only approximate 28 of the 31 rounds in PRESENT to mount a key recovery attack, linear cryptanalysis of the cipher would require of the order of 2^{84} known plaintext/ciphertexts. Such data requirements exceed the available text.

Some other attacks. The structure of PRESENT allows us to consider some dedicated forms of attacks. However none have yielded an attack that requires less text than the lower bound on text requirements for linear cryptanalysis. Among the dedicated attacks we considered was one using palindromic differences, since symmetrical differences are preserved with probability one over the diffusion layer, and some advanced variants of differential-linear attacks [113]. While the attacks seemed promising over a few rounds, they very quickly lost their practical value and are unlikely to be useful in the cryptanalysis of PRESENT. We also established that truncated differential cryptanalysis [145, 150] was likely to have limited value. Even when used to reduce the length of the differential characteristics already identified, the data requirements still remain excessive. As to the algebraic attacks, the PRESENT S-box is described by 21 quadratic equations in the eight input/output-bit variables over $\mathbb{F}_2$. This is not surprising since it is well-known that any four bit S-box can be described by at least 21 such equations. The entire cipher can then described

by $e = n \times 21$ quadratic equations in $v = n \times 8$ variables, where n is the number of S-boxes in the encryption algorithm and the key schedule. For PRESENT we have $n = (31 \times 16) + 31$ thus the entire system consists of $11,067$ quadratic equations in $4,216$ variables. Simulations on small-scale versions of PRESENT showed that for all but the very smallest ones one quickly encounters difficulties in both time and memory complexity.

4.5.2 Key Schedule Attacks

Since there are no established guidelines to the design of key schedules, there is both a wide variety of designs and a wide variety of schedule-specific attacks. The most effective attacks come under the general heading of related-key attacks [15] and slide attacks [31], and both rely on the build-up of identifiable relationships between different sets of subkeys. To counter this threat, we use a round-dependent counter so that subkey sets cannot easily be "slid", and we use a non-linear operation to mix the contents of the key register K. In particular,

- all bits in the key register are a non-linear function of the 80-bit user-supplied key by round 21,
- that each bit in the key register after round 21 depends on at least four of the user-supplied key bits, and
- by the time we arrive at deriving K_{32}, six bits are degree two expressions of the 80 user-supplied key bits, 24 bits are of degree three, while the remaining bits are degree six or degree nine function of the user-supplied key bits.

We believe these properties to be sufficient to resist key schedule-based attacks.

4.5.3 Further Cryptanalysis

Since its publication, PRESENT has received attention from the research community. Here we briefly outline and discuss most interesting cryptanalysis of reduced-round PRESENT known hitherto.

The bit-pattern based integral attack was introduced in [262]. This is a variant of multiset cryptanalysis based on the propagation of bit patterns rather than word patterns. This attack recovers the key of 6-round PRESENT-80, requiring $2^{22.4}$ chosen plaintexts, $2^{41.7}$ operations and negligible memory. With respect to PRESENT-128, the attack allows key recovery for 7 rounds, requiring $2^{24.3}$ chosen plaintexts, $2^{100.1}$ operations, and 2^{77} bytes memory. Probably the most interesting result of the paper as applied to PRESENT is that for attacking 5 rounds only 80 chosen plaintexts are required. At the same time, the standard differential cryptanalysis would require about 2^{20} chosen plaintexts. Thus, this attack does significantly decrease the

data complexity of cryptanalysis for round-reduced versions of PRESENT but quickly becomes less efficient than brute-force as the number of rounds grows.

The paper [252] analyzes PRESENT using differential cryptanalysis by finding a 14-round differential trail holding with probability 2^{-62}. This allows the authors to mount a key-recovery attack on 16 rounds of PRESENT requiring all 2^{64} plaintext-ciphertext pairs, 2^{32} counters in memory, and 2^{64} memory access. Note that the 14-round differential trail used does not attain the upper bound on the differential trail probability of Theorem 5 though this bound over 5 rounds is tight.

The work [2] proposes a method to extend key recovery using differential cryptanalysis. Basically, instead of guessing parts of subkeys at key recovery, the adversary tries to solve a system of nonlinear equations over several rounds involving the differences from differential cryptanalysis. The authors of [2] use differential trails found in [252]. Experiments show that in case of PRESENT-128, this technique can enable a key-recovery attack on 19 rounds on PRESENT-128 requiring about 2^{113} CPU cycles. The algebraic methods of [2] do not seem to provide any significant advantage over differential cryptanalysis for PRESENT-80. Note that the complexity of the algebraic part of this attack grows significantly with a small increase in the number of rounds attacked.

A statistical saturation attack on PRESENT which can be viewed as a multiset-type attacks that tracks the propagation of a certain distribution of inputs through the cipher was proposed in [66]. Based on its simple diffusion layer, the practical attack on 15 rounds of PRESENT requires $2^{35.6}$ plaintext-ciphertext pairs. However, it seems to be an open question how this attack applies to PRESENT with larger numbers of rounds. Under some non-standard assumptions, the authors of [66] provide some heuristic arguments why the attack could be able to break up to 24 rounds. Neither more rigorous evidence nor experimental has been provided on that yet. Thus, it certainly remains an interesting open research problem to find such evidence or to show that the attack complexity grows more rapidly than conjectured. However, we think the saturation approach is most promising in the cryptanalysis of PRESENT so far.

In the related-key scenario, PRESENT has been analyzed in [208]. The boomerang related-key attack proposed there can break up to 17 rounds of PRESENT-128, requiring 2^{63} chosen plaintexts with time complexity 2^{104} memory accesses and 2^{53} bytes memory. Among other things, several 5-round related-key differential trails with only 3 active S-boxes were found there. However, even assumed that these differential trails are iterative (which is even not the case for less probable related-key differential trails used in [208] having more active S-boxes), this would result in data complexity close to the upper bound on data complexity 2^{64}.

Table 4.1: Comparison of hardware implementations for some lightweight block and stream ciphers

	Key size	Block size	Cycles per block	Throughput at 100KHz (Kbps)	Logic process	Area GE	Area rel.
Block ciphers							
PRESENT-80 [44]	80	64	32	200	0.18μm	1 570	1
PRESENT-80 [229]	80	64	563	11.4	0.18μm	1 075	0.68
AES-128 [99]	128	128	1032	12.4	0.35μm	3400	2.17
HIGHT [124]	128	64	1	6400	0.25μm	3048	1.65
mCrypton [170]	96	64	13	492.3	0.13μm	2681	1.71
Camellia [4]	128	128	20	640	0.35μm	11350	7.23
DES [162]	56	64	144	44.4	0.18μm	2309	1.47
DESXL [162]	184	64	144	44.4	0.18μm	2168	1.38
Stream ciphers							
Trivium [108]	80	1	1	100	0.13μm	2599	1.66
Grain [108]	80	1	1	100	0.13μm	1294	0.82

4.6 Implementation Issues for PRESENT

Basically, there are several approaches to the hardware implementation of block ciphers which differ in the parallelization degree: When many operations are performed in parallel, one obtains larger area and higher throughput. If as many as possible operations are executed serially, this usually results in a more compact implementation.

The most natural way of serializing a block cipher implementation is to use its iterative nature at the round level by implementing a 64-bit datapath: One round is then performed in a clock cycle. A round-serialized implementation of PRESENT-80 was performed [44] in VHDL and synthesized for the Virtual Silicon (VST) standard cell library based on the UMC L180 0.18μ 1P6M Logic process. This implementation requires 32 clock cycles to encrypt a 64-bit plaintext with an 80-bit key and occupies 1570 GE.

As a single 4-bit S-box is used in the design of PRESENT, its implementation can be also serialized at the S-box level by updating only 4-bits in a clock cycle. This type of implementation approach for PRESENT-80 is pursued in [229] using for a similar hardware process. The resulting implementation requires 563 clock cycles to encrypt a 64-bit plaintext with an 80-bit key and occupies 1075 GE.

Note that most area consumption in both implementations is due to the state consisting of flip-flops for the intermediate datapath variable and the current round key. A comparison of these PRESENT-80 implementations with other ciphers can be found in Table 4.1.

4.7 Hash Function Constructions

Hash functions in use today are built around the use of a *compression function* and appeal to the theoretical foundations laid down by Merkle and Damgård [81, 180].

The compression function h has a fixed-length input, consisting of a *chaining variable* and a message extract, and gives a fixed-length output. A variety of results [83, 131, 140] have helped provide a greater understanding of this construction and while there are some limitations there are some countermeasures [18]. Since our goal is to obtain *representative* performance estimates, we will not go into the details of hash function designs. Instead we will assume that our hash function uses a compression function in an appropriate way and that the compression function takes as input some words of chaining variable, represented by H_i, and some words of (formatted) message extract, represented by M_i. We then restrict our focus to the cost of implementing the compression function.

In the hash function literature it is common to distinguish between two popular ways of building a compression function. The first is to use a compression function of a dedicated design and the second is to use an established, and trusted, block cipher.

4.7.1 Dedicated Constructions

The separation of dedicated constructions from block cipher-based constructions tends to disguise the fact that even dedicated hash functions like SHA-1 [192] and MD5 [225] are themselves built around a block cipher. Remove the feed-forward from compression functions in the MD-family and we are left with a reversible component that can be used as a block cipher (such as SHACAL [109] in the case of SHA-1). However the underlying block cipher we are left with is rather strange and has a much larger-than-normal block and key size combination. The problem with dedicated hash functions is that recent attacks [253, 254] have shown that there is much to learn in designing block ciphers with such strange parameter sizes. There is therefore some value in considering approaches that use a more "classical" block cipher as the basis for a compression function.

4.7.2 Block Cipher Based Constructions

The use of a block cipher as a building block in hash function design [68] is as old as DES [195]. The topic has been recently revisited and Black *et al.* [35] have built on the work of Preneel [216] to present a range of secure $2n$- to n-bit compression functions built around an n-bit block cipher that takes an n-bit key. Among these are the well-known Davies-Meyer, Matyas-Meyer-Oseas, and Miyaguchi-Preneel constructions.

A hash function with an output of n bits can only offer a security level of 2^n operations for pre-image and second pre-image attacks and $2^{n/2}$ operations against finding collisions. While a security level of 128 bits is typical for mainstream applications, 80-bit security is often a reasonable target for RFID tag-based applications. Either way, there is a problem since the hash functions we need cannot always be immediately constructed out of the block ciphers we have to hand. This

is not a new problem. But it is not an easy one to solve either, and there has been mixed success in constructing $2n$-bit hash functions from an n-bit block cipher [54, 68, 146, 158, 160, 217, 219]. While limitations have been identified in many constructions, work by Hirose [119, 120] has identified a family of double-block-length hash functions that possess a proof of security. These use block ciphers with a key length that is twice the block length. Such a property is shared by AES-256 [193] and PRESENT-128 and so in Section 4.8.2 we consider the performance of an Hirose-style construction instantiated using PRESENT-128.

When it comes to providing a replacement for SHA-1, the parameter sizes involved provide a difficult challenge. If we are to use a 64-bit block cipher like PRESENT-128, then in arriving at a hash function with an output of at least 160 bits we need a construction that delivers an output three times the block size (thereby achieving a 192-bit hash function). There are no "classical" constructions for this and so Section 4.9.1 illustrate two possible design directions. These give representative constructions and we use them to gauge the hardware requirements of different design approaches. We hope that this will be of interest to future hash function designers.

4.8 Compact 64- and 128-Bit Hashing

In this section we will consider a variety of approaches to compact hashing when we use the block cipher PRESENT as a building block. PRESENT is a 64-bit SPN block cipher which can be used with either an 80-bit or a 128-bit key. These will be referred to as PRESENT-80 and PRESENT-128. We denote encrypting a message M under the key K with PRESENT-80 or PRESENT-128 to obtain the ciphertext C as $C = E(M, K)$ and $A\|B$ denotes the concatenation of A and B.

4.8.1 64-Bit Designs: DM-PRESENT-80 and -128

There are a variety of choices for building a 64-bit hash function from a 64-bit block cipher. We will illustrate these with the *Davies-Meyer* mode where a single 64-bit chaining variable H_i is updated using a message extract M_i according to the computation $H'_i = E(H_i, M) \oplus H_i$.

In our case E denotes encryption with either PRESENT-80 or PRESENT-128, see Figure 4.4. Such hash functions will only be of use in applications that require the one-way property and 64-bit security.[2] At each iteration of the compression function 64 bits of chaining variable and 80 bits (*resp.* 128 bits) of message-related input are compressed. Therefore the two proposals DM-PRESENT-80 and DM-PRESENT-128

[2]These properties are identical to those offered by the proposal SQUASH [237].

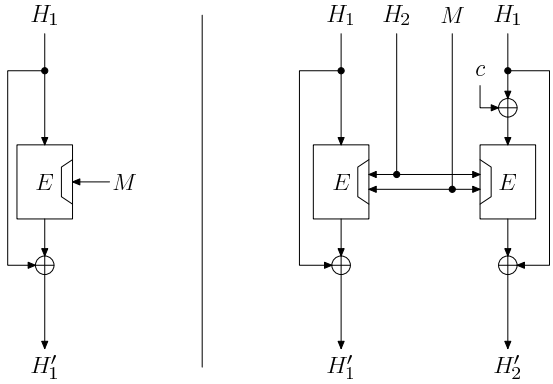


Figure 4.4: Compression functions for the 64-bit and 128-bit hash functions: DM-PRESENT-80 and DM-PRESENT-128 (on the left) as well as H-PRESENT-128 (on the right)

provide a simple trade-off between space and throughput. We also provide figures for a serial and parallel implementation of PRESENT, see Table 4.3.

While we have focused on using Davies-Meyer, it is important to note that these figures are a good indication of the cost for *any* single block-length hash function construction. If one prefers to implement *Matyas-Meyer-Oseas* or *Miyaguchi-Preneel* based on PRESENT (instead of *Davies-Meyer*) then the cost of DM-PRESENT-80 will be a reasonable guide. Moving away from PRESENT to a different block cipher will almost certainly cause an increase to the space required for an implementation.

4.8.2 A Compact 128-Bit Design: H-PRESENT-128

When designing a 128-bit hash function from the 64-bit output block cipher PRESENT, we have to appeal to so-called double-block-length hash function constructions. Natural candidates are MDC-2 [68] and Hirose's constructions [119, 120]. These schemes possess security proofs in the ideal cipher model, where the underlying block cipher is modeled as a family of random permutations, one permutation being chosen independently for each key. However MDC-2 is not an ideal construction [247] and so we base our 128-bit hash function H-PRESENT-128 on the construction studied in [120].

The scheme H-PRESENT-128 is illustrated in Figure 4.4. The compression function takes as input two 64-bit chaining variables and a 64-bit message extract, denoted by the triple (H_1, H_2, M), and outputs the pair of updated chaining variables (H'_1, H'_2) according to the computation

$$H'_1 = E(H_1, H_2 \| M) \oplus H_1 \quad \text{and} \quad H'_2 = E(H_1 \oplus c, H_2 \| M) \oplus H_1$$

where E denotes PRESENT-128 and c is a non-zero constant that needs to be fixed

[59]. Thus the chaining variable $H_1 \| H_2$ is 128 bits long and 64 bits of message-related input are hashed per iteration.

Hirose showed that, in the ideal cipher model, an adversary has to make at least 2^n queries to the cipher in order to find a collision with non-negligible advantage, where n is the block size of the cipher. It is possible to make the same kind of analysis for preimage resistance (see proof of Theorem 4 in [118]) and to show that any adversary has to make at least 2^{2n} queries to the cipher to find a preimage. As for Section 4.8.1 our implementation results are presented for both a parallel and serial implementation of PRESENT-128, see Table 4.3. These results should be viewed as indicative of the cost of a double-block-length construction using PRESENT. Since only one key schedule needs to be computed per iteration of the compression function, the Hirose construction is probably one of the most efficient constructions of this type, e.g. in the case of PRESENT around 1 000 GE can be saved in this way.

4.9 Compact 160-Bit Hashing

It is possible that some tag-enabled applications might need collision-resistance at a security level of 2^{80} operations. For this we need a hash output of 160 bits or greater. If one continues the approach used so far and considers building a hash function with a hash output greater than 160 bits from PRESENT *as is* using established results in the literature, this is unlikely to be successful. Instead we move towards a dedicated hash function though we keep elements of PRESENT in our constructions. Our dedicated proposals are deliberately simple and obvious, and in this way we aim to provide some first results on the impact different design choices might have in moving towards a new, compact, hash function.

Hash function design is notoriously difficult and so an interesting first step is to identify some general approaches and to understand their security and performance trade-offs. In this section we describe the results of some prototyping which tests a range of approaches and provides good background to our ongoing work. Our basic premise is to stay close to the design elements of PRESENT and to modify the design so as to give a block cipher with a much larger block size. We then adapt the key schedule in a way similar to to Whirlpool [10] and give implementation results for this approach.

4.9.1 Dedicated Design Blocks Inspired By PRESENT

Our construction will continue to be based on the Davies-Meyer (DM) scheme $H_{i+1} = E(H_i, M_i) \oplus H_i$ though the form of our encryption function E will now change. In general, the encryption function E can be described as:

$$E : \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2^n \ ,$$
$$E : \text{PLAINTEXT} \times \text{KEY} \mapsto \text{CIPHERTEXT}$$

At a top-level we can write the r-round encryption of the plaintext STATE as:

for $i = 1$ to r **do**
  STATE $\leftarrow$ STATE $\oplus$ eLayer(KEY, i)
  STATE $\leftarrow$ sBoxLayer(STATE)
  STATE $\leftarrow$ pLayer(STATE)
  KEY $\leftarrow$ genLayer(KEY, i)
end for
STATE $\leftarrow$ STATE $\oplus$ eLayer(KEY, $r + 1$),

where eLayer describes how a subkey is combined with a cipher STATE, sBoxLayer and pLayer describe how the STATE evolves, and genLayer is used to describe the generation of the next subkey.

When used in the DM mode we recast the plaintext and ciphertext as hash function STATE and use the (formatted) message extract as the key. For ease of design we will choose the parameters k and n so that $k|n$ and $4|n$, and our proposal will have the following (unmodified) structure:

for $i = 1$ to r **do**
  STATE $\leftarrow$ STATE $\oplus$ eLayer(MESSAGE, i)
  STATE $\leftarrow$ sBoxLayer(STATE)
  STATE $\leftarrow$ pLayer(STATE)
  MESSAGE $\leftarrow$ genLayer(MESSAGE, i)
end for
STATE $\leftarrow$ STATE $\oplus$ eLayer(MESSAGE, $r + 1$)

The following building blocks are unchanged between the two proposals and are merely generalizations of the PRESENT structure to larger 160-bit block sizes.

1. sBoxLayer: This denotes use of the PRESENT 4-bit to 4-bit S-box S and it is applied $n/4$ times in parallel.

2. pLayer: This is an extension of the PRESENT bit-permutation and moves bit i of STATE to bit position $P(i)$, where

$$P(i) = \begin{cases} i \cdot n/4 \mod n - 1, & \text{if } i \in \{0, \ldots, n-2\} \\ n - 1, & \text{if } i = n - 1. \end{cases}$$

It is in the specification of genLayer, which transforms the message of length k from round-to-round, that exhibits more difference to PRESENT.

4.9.2 PROP: A Proposal for Compression Function

We consider a structure that has some similarity to the Whirlpool hash function. Our parameter set is $n = 160$ and $k = 160$ which allows us to use a longer message extract

at each iteration of the compression function. For prototyping and implementation estimates we set $r = 80$. The building blocks eLayer and genLayer are specified as:

1. eLayer(MESSAGE, i) = MESSAGE
2. genLayer(MESSAGE, i) = pLayer(sBoxLayer(MESSAGE $\oplus$ i)), being just a copy of the data path with round constant addition.

In words, we imagine that our message extract is a 160-bit key and we process the key in a key-schedule that is identical to the encryption process.

We estimated the hardware figures for different architectures when implementing PROP. Our implementation estimates range from a 4-bit width data path (highly serialized) up to a 160-bit width data path which offers one round of processing in one cycle. Since PROP uses a very similar key schedule (*i.e.* message path) and encryption routine, we can give a further two different implementation options: one with a shared sBoxLayer between the data path and the message path and one with an individual sBoxLayer. The results are summarized below with the efficiency *eff.* being measured in bps/GE.

Understanding the best trade-offs for the different approaches is not easy. As one can see, all implementations scale nicely. Much would also depend on a thorough security analysis of any final proposal and while some initial analysis below suggests the possibility of optimizations to an approach like PROP, this is something to explore in future work during the design of an explicit proposal.

4.9.3 Elements of Cryptanalysis for PROP

Our proposed design elements are not intended to be specifications. Nevertheless, some preliminary analysis follows from the simple structures proposed. In particular, for a fixed message block and two different chaining values we can apply Theorem 4 directly. This states that at least 10 active S-boxes are involved in any 5-round differential characteristic. However, for the more important case of two different message blocks, the analysis has to be slightly modified. The following result on the differential behaviour of the proposal can be viewed as a first step towards a deeper analysis:

Theorem 7. *Let* $Q_{(\Delta_1,\Delta_2)\mapsto(\Delta_1',\Delta_2')}$ *be the probability that a differential trail* $\Delta_2 \neq 0$ *propagates through 80 rounds of PROP from the input variable to the variables right before the final XOR of* STATE *and* MESSAGE *branches. Then* $Q_{(\Delta_1,\Delta_2)\mapsto(\Delta_1',\Delta_2')} \leq 2^{-400}$.

Proof. The compression function consists of a data path and a message path, 80 rounds each. The message path acts independently. It is a bijection accepting a non-zero input difference $\Delta_1 \neq 0$. Accoring to a version of Theorem 4, each 5 rounds

of the message path provide at least 10 differentially active S-boxes in this case. In other words, each 5 rounds contribute the factor 2^{-20} to the upper bound on the differential trail probability. Thus, 80 rounds of the message path contribute the factor $2^{-20 \cdot \frac{80}{5}} = 2^{-320}$ to the upper bound on the differential trail probability over 80 rounds. However, there will be also some differentially active S-boxes in the data path of the construction.

To count the minimum number of differentially active S-boxes in the data path over 80 rounds, we consider some two consecutive rounds first. Without loss of generality, we will consider two first rounds which are referred to as round 1 and round 2. There are two cases minimizing the number of differentially active S-boxes over two rounds:

Case $\Delta_1 = 0$. In round 1, a non-zero difference from the message path is added with the zero difference in the data path . The resulting non-zero difference is then processed by the S-box layer followed by the bit permutation layer. In round 2, a non-zero difference from the message path is added with the non-zero difference in the data path. In the best case for the adversary, these differences are equal and cancel out. Thus, the S-box layer of round 2 accepts a zero difference. After round 2, one has a non-zero difference in the message path and a zero difference in the data path again.

Case $\Delta_1 \neq 0$. In round 1, a non-zero difference from the message path is added with the non-zero difference in the data path. In the best case for the adversary, these differences are equal and cancel out. Thus, the S-box layer of round 1 accepts a zero difference and the output difference of round 1 is zero. In round 2, a non-zero difference from the message path is added with the zero difference in the data path, resulting in a non-zero difference which is input to the S-box layer of round 2. After round 2, one has a non-zero difference in the message path and a non-zero difference in the data path again.

That is, among each two consecutive rounds, at least one accepts a non-zero input difference. Under the assumption that the difference weight is minimal, one obtains one differentially active S-box. So, there is at least one differentially active S-box provided by each two consecutive rounds. The 80 rounds of PROP give 40 active S-boxes in the data path and contribute the factor $2^{-2 \cdot 40} = 2^{-80}$ to the upper bound on the differential trail probability over 80 rounds.

Combining the contributions of the message path and data path yields the claim of the theorem. □

However, for the security of the full compression function the probability $P_{(\Delta_1,\Delta_2)\mapsto\Delta}$ of a differential $(\Delta_1, \Delta_2) \mapsto \Delta = \Delta'_1 \oplus \Delta'_2$ is much more relevant. Now under the assumption that the differential trail probability $Q_{(\Delta_1,\Delta_2)\mapsto(\Delta'_1,\Delta'_2)}$ of Theorem 7 is comparable to that of the corresponding differential (which was indicated by

Table 4.2: Estimations for different-type hardware implementations of PROP

data path width	PROP (shared)			PROP (ind.)		
	area (GE)	cycles	*eff.*	area (GE)	cycles	*eff.*
4	3 010	6 481	0.82	3 020	3 281	1.62
16	3 310	1 621	2.92	3 380	821	5.77
32	3 730	811	5.11	3 860	411	10.09
80	4 960	325	9.29	5 300	165	18.3
160	5 730	163	15.29	6 420	83	30.03

our experiments for PRESENT), one gets the following estimation on the differential probability $P_{(\Delta_1,\Delta_2)\mapsto\Delta}$:

$$P_{(\Delta_1,\Delta_2)\mapsto\Delta} \approx \sum_{\Delta=\Delta'_1\oplus\Delta'_2} Q_{(\Delta_1,\Delta_2)\mapsto(\Delta'_1,\Delta'_2)}.$$

For a given Δ, there are 2^{160} combinations of Δ'_1 and Δ'_2 leading to Δ. Using Theorem 7, we obtain a rough approximation of the upper bound on the differential probability $P_{(\Delta_1,\Delta_2)\mapsto\Delta}$:

$$P_{(\Delta_1,\Delta_2)\mapsto\Delta} \leq 2^{160} \cdot 2^{-400} = 2^{-240}.$$

Note that, as any estimation based on differential trails, this approximate upper bound ignores the effect of several differential trails constituting the same differential $(\Delta_1, \Delta_2) \mapsto \Delta$.

All this indicates in a first approximation that the differential properties might be strong enough to thwart pre-image, second pre-image and collision attacks for the proposal. The most appropriate trade-off between the number of rounds r and the security level remains an area of research.

4.10 Implementation Issues for Hashing

To consider the efficiency of the standard constructions, two different architectures (round-based and serial) were implemented in VHDL and simulated using *Mentor Graphics Modelsim SE PLUS 6.3a* in [45] (synthesis using *Virtual Silicon* (VST) standard cell library *UMCL18G212T3* based on the *UMC L180 0.18µm 1P6M* logic process and having a typical voltage of 1.8 Volt). A clock frequency of 100 KHz was used there, which is a typical operating frequency for RFID applications.

Table 4.3 summarizes these results and compares them to other hashing functions and AES-based schemes. When the hash output length is 128 bits or lower, a construction based around PRESENT seems to have potential. Certainly they are far more competitive than current hash functions, the primary reason being that there exist efficient block cipher-based constructions for this size of hash output. Even a larger block cipher such as AES makes for a more compact hash function than current dedicated designs at this security level, though the throughput suffers.

Table 4.3: The performance of different hash functions based on the direct application of PRESENT. For comparison with our hash functions with 128-bit output we include estimates for the AES-based 128-bit hash function in Davies-Meyer mode. For comparison with MAME we include estimates for the 256-bit hash function built from the AES in Hirose's construction

	Hash output size	Data path size	Cycles per block	Throughput at 100KHz (Kbps)	Efficiency (bps/GE)	Logic process	Area GE
MD4 [100]	128	32	456	112.28	15.3	0.13μm	7 350
MD5 [100]	128	32	612	83.66	10	0.13μm	8 400
SHA-1 [100]	160	32	1 274	40.19	4.9	0.35μm	8 120
SHA-256 [100]	256	32	1 128	45.39	4.2	0.35μm	10 868
MAME [259]	256	256	96	266.67	32.9	0.18μm	8 100
DM-PRESENT-80	64	64	33	242.42	109.5	0.18μm	2 213
DM-PRESENT-80	64	4	547	14.63	9.1	0.18μm	1 600
DM-PRESENT-128	64	128	33	387.88	153.3	0.18μm	2 530
DM-PRESENT-128	64	4	559	22.9	12.1	0.18μm	1 886
H-PRESENT-128	128	128	32	200	47	0.18μm	4 256
H-PRESENT-128	128	8	559	11.45	4.9	0.18μm	2 330
AES-based *DM scheme*	128	8	$> 1\,032$	< 12.4	< 2.8	*estimate*	$> 4\,400$
AES-based *Hirose scheme*	256	8	$> 1\,032$	< 12.4	< 1.3	*estimate*	$> 9\,800$

4.11 Conclusions

In this chapter we have described the new block cipher PRESENT. Our goal has been an ultra-lightweight cipher that offers a level of security commensurate with a 64-bit block size and an 80-bit key. Intriguingly PRESENT has implementation requirements similar to many compact stream ciphers. As such, we believe it to be of both theoretical and practical interest.

While compact hash functions are often proposed in protocols for RFID tags, there are currently no sufficiently compact candidates to hand. Here we have explored the possibility of building a hash function out of a block cipher such as PRESENT. We have described hash functions that offer 64- and 128-bit outputs based on current design strategies. For their parameter sets these are the most compact hash function candidates available today. In particular, H-PRESENT-128 requires around 4 000 GE, which is similar to the best known AES implementation and about 50% smaller than the best reported MD5 implementation. At the same time, H-PRESENT-128 requires between 20–30 *times* fewer clock cycles than compact AES and MD5 implementations, giving it a major time-area advantage.

Obviously 128-bit hash functions are relevant for applications where a security-performance trade-off is warranted. To obtain larger hash outputs there are severe complications and we suspect that dedicated designs could be more appropriate. Clearly there are many areas of open research, not least the design of very compact hash functions. In parallel, it might also be worth revisiting tag-based protocols that use hash functions to see if the same goals can be achieved in a different way.

Chapter 5

Cryptanalysis of KeeLoq

KeeLoq is a block cipher used in numerous wide-spread remote keyless entry systems as well as in various component identification applications. The KeeLoq algorithm has a 64-bit key and operates on 32-bit blocks. It is based on an NLFSR with a nonlinear feedback function of 5 variables.

In this chapter key recovery attacks on KeeLoq are proposed. The first one has a complexity of about $2^{50.6}$ KeeLoq encryptions. The second attack finds the key in 2^{40} encryptions. Both attacks work for the whole key space. In our attacks we use the techniques of guess-and-determine, slide, and linear attacks as well as cycle structure analysis. Both attacks need 2^{32} known plaintext-ciphertext pairs.

We also analyze the KeeLoq key management and authentication protocols applied in rolling-code and IFF access systems widely used in real-world applications. We demonstrate several practical vulnerabilities.

This chapter is based on the independent research of the author published in [36] and [38] as well as on his joint work with Christof Paar published in [48].

5.1 Introduction

KeeLoq is a block cipher based on an NLFSR with a nonlinear Boolean feedback function of 5 variables. The algorithm uses a 64-bit key and operates on 32-bit blocks. Its architecture consists of two registers (a 32-bit text register and a 64-bit key register), which are rotated by one position in each of 528 encryption cycles, and of a nonlinear function (NLF) providing nonlinear feedback. One bit of the key is added to the output of the NLF modulo 2 in each cycle.

The light-weight architecture of the KeeLoq cipher allows for an extremely low-cost and efficient hardware implementation (we estimate it to require about 700 GE and 528 clock cycles per block). This contributed to the popularity of the KeeLoq cipher among designers of remote keyless entry systems, automotive and burglar alarm systems, automotive immobilizers, gate and garage door openers, identity tokens, component identification systems. For instance, the KeeLoq block cipher is used by some well-known automotive OEMs [258] and in the HomeLink wireless control systems to secure communication with garage door openers [123]. The KeeLoq technology supplied by Microchip Technology Inc. includes the KeeLoq cipher and a number of authentication protocols as well as key management schemes. Our description of KeeLoq is based on the newly published article [258], [186] and a number of the manufacturer's documents [182], [183], [189], [187].

Our contribution. The contribution of this chapter is three-fold. First, two key-recovery attacks on KeeLoq are proposed. A direct attack recovers the key in $2^{50.6}$ KeeLoq operations. A cycle-based technique allows us to improve the direct attack and achieve a reduced complexity of 2^{40} KeeLoq operations, working for the whole

key space. Second, severe vulnerabilities of the KeeLoq key management systems are demonstrated. Finally, we also analyze the KeeLoq IFF authentication protocol.

Our cryptanalysis of the KeeLoq algorithm is based on the following weaknesses of the KeeLoq structure: The key schedule is self-similar, which allows us to mount a slide-type attack [32], [31], [20]. It is supported by the existence of an efficient linear approximation of the NLF used to recover a part of the key. Then the remainder of the key bits is obtained using other linear relations within KeeLoq.

The key recovery complexity of our first attack is $2^{50.6}$. The attack requires 2^{32} plaintext-ciphertext pairs and a memory of 2^{32} 32-bit words. Several computing devices can share the memory during the attack. All computations are perfectly parallelizable. The property inherited from the slide attacks [32], [31] is that the complexity of our attack is independent of the number of encryption cycles, which is as a rule not the case for linear or differential cryptanalysis, where the complexity often grows exponentially with the number of iterations.

The second attack is an extension of our first attack by using the cycle structure analysis introduced in [70]. This attack finds the key in 2^{40} steps. It also requires 2^{32} known plaintext-ciphertext pairs and a memory to hold 2^{32} 32-bit words. Additionally, 2^{32} bits of memory are needed for exploring the cycle structure of the KeeLoq permutation. Our techniques work for all keys, unlike most other attacks on KeeLoq working for a fraction of all keys. Though requiring the entire codebook, our second attack is the fastest known attack on the KeeLoq block cipher working for the whole key space.

Generic attacks on the KeeLoq IFF authentication protocol are also discussed here. We notice that only about 2^{17} sessions are required to obtain a valid authentication with high probability, if no additional organizational countermeasures are taken.

We also show how the cryptanalytic attacks on the KeeLoq block cipher apply to the KeeLoq hopping codes and IFF (Identify Friend or Foe) systems supplied by Microchip which are based on this algorithm. It is demonstrated that the attacks pose a real threat for a number of applied key management schemes: After attacking one instance of KeeLoq using attacks presented in this chapter, one can reduce the effective key length of all other KeeLoq systems of the same series to 32, 48, or 60 bits depending on the key management scheme applied. In some attack models, the attacker is even able to retrieve the individual encryption key instantly.

The chapter is organized as follows. Section 5.2 describes the KeeLoq algorithm. In Section 5.3 our basic linear slide key recovery attack on KeeLoq and its extension with a reduced complexity are presented. In Section 5.4 we discuss the impact of attacks on the KeeLoq algorithm with respect to the standard KeeLoq real-word applications supplied by Microchip. Section 5.5 considers some generic attacks on the KeeLoq IFF authentication protocol. We conclude in Section 5.6.

5.2 Description of the KeeLoq Elements

5.2.1 KeeLoq Algorithm

KeeLoq is a block cipher with a 64-bit key which operates on 32-bit words [258], [186]. Its design is based on a nonlinear feedback shift register (NLFSR) of length 32 bits with a nonlinear feedback function of 5 variables. The feedback depends linearly on two other register bits and on the next key bit taken from the rotated key register of length 64 bits.

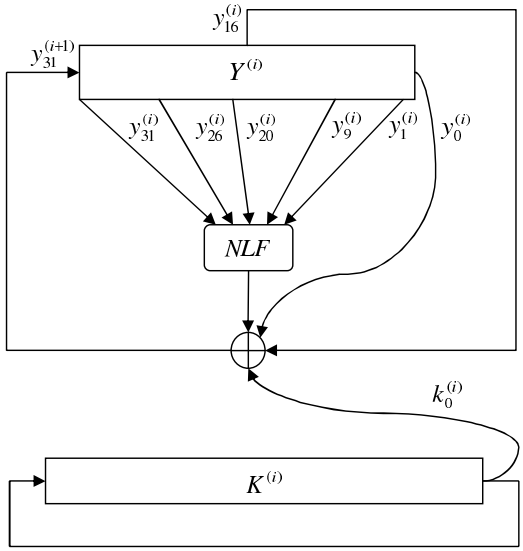


Figure 5.1: The i-th KeeLoq encryption cycle

Let $\mathbb{F}_2^n$ be the set of all n-bit words and $Y^{(i)} = (y_{31}^{(i)}, \ldots, y_0^{(i)}) \in \mathbb{F}_2^{32}$, $y_j^{(i)} \in \mathbb{F}_2$, describe the state of the text register in cycle i for $j = 0, \ldots, 31$ and $i = 0, 1, \ldots$ Let also $K^{(i)} = (k_{63}^{(i)}, \ldots, k_0^{(i)}) \in \mathbb{F}_2^{64}$, $k_j^{(i)} \in \mathbb{F}_2$, denote the state of the key register in cycle i for $j = 0, \ldots, 63$ and $i = 0, 1, \ldots$ Then each cycle of encryption can be described using the following algorithm (see Figure 5.1):

Compute the feedback bit: $\varphi = NLF(y_{31}^{(i)}, y_{26}^{(i)}, y_{20}^{(i)}, y_9^{(i)}, y_1^{(i)}) \oplus y_{16}^{(i)} \oplus y_0^{(i)} \oplus k_0^{(i)}$

Rotate text and insert feedback: $R^{(i+1)} = (\varphi, y_{31}^{(i)}, \ldots, y_1^{(i)})$

Rotate key: $K^{(i+1)} = (k_0^{(i)}, k_{63}^{(i)}, \ldots, k_1^{(i)})$.

For encryption the key register is filled with the 64 key bits $K = (k_{63}, \ldots k_0) \in \mathbb{F}_2^{64}$, $k_j \in \mathbb{F}_2$, $j = 0, \ldots, 63$, in the straightforward way: $K^{(0)} = K$. If $X = (x_{31}, \ldots, x_0) \in \mathbb{F}_2^{32}$, $x_j \in \mathbb{F}_2$, $j = 0, \ldots, 31$, is a block of plaintext, the initial state of the text

register is $Y^{(0)} = (x_{31}, \ldots, x_0)$. The output of the algorithm is the ciphertext $Z = (z_{31}, \ldots, z_0) = Y^{(528)} \in \mathbb{F}_2^{32}$, $z_j \in \mathbb{F}_2$, $j = 0, \ldots, 31$.

For decryption the key register is filled in the same way:

$$K^{(0)} = K = (k_{63}, \ldots k_0) \in \mathbb{F}_2^{64}.$$

But the decryption procedure complements the encryption. One decryption cycle can be defined by the following sequence of operations:

Compute the feedback bit: $\varphi = NLF(y_{30}^{(i)}, y_{25}^{(i)}, y_{19}^{(i)}, y_8^{(i)}, y_0^{(i)}) \oplus y_{15}^{(i)} \oplus y_{31}^{(i)} \oplus k_{15}^{(i)}$

Rotate text and insert feedback: $R^{(i+1)} = (y_{30}^{(i)}, \ldots, y_0^{(i)}, \varphi)$

Rotate key: $K^{(i+1)} = (k_{62}^{(i)}, \ldots, k_0^{(i)}, k_{63}^{(i)})$.

The ciphertext and plaintext are input/output in a similar way: The ciphertext is input into the text register before decryption, $Y^{(0)} = Z$, and the plaintext can be read out after 528 decryption cycles, $Y^{(528)} = X$.

The NLF is a Boolean function of 5 variables and is of algebraic degree 3. In the specification [186] the NLF is assigned using a table. This corresponds to the following ANF:

$$\begin{aligned} NLF(x_4, x_3, x_2, x_1, x_0) \quad = \quad & x_0 \oplus x_1 \oplus \\ & x_0x_1 \oplus x_1x_2 \oplus x_2x_3 \oplus x_0x_4 \oplus x_0x_3 \oplus x_2x_4 \oplus \\ & x_0x_1x_4 \oplus x_0x_2x_4 \oplus x_1x_3x_4 \oplus x_2x_3x_4. \end{aligned} \tag{5.1}$$

The NLF is balanced and its correlation immunity order is 1, $\mathrm{cor}(NLF) = 1$ [242], [243]. This means that the NLF is 1-resilient [62], which is the maximum for a function of 5 variables with $\deg(NLF) = 3$ due to Siegenthaler's inequality [242]:

$$\deg(NLF) + \mathrm{cor}(NLF) \leq 4.$$

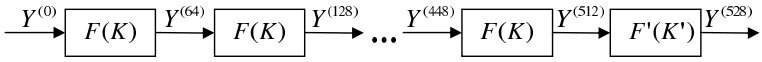


Figure 5.2: Round structure of KeeLoq encryption

The KeeLoq algorithm has the following round structure. We define a KeeLoq round as the permutation $F(K) : \mathbb{F}_2^{32} \to \mathbb{F}_2^{32}$ depending on the key $K \in \mathbb{F}_2^{64}$ and consisting of 64 encryption cycles. A KeeLoq quarter round is defined as the permutation $F'(K') : \mathbb{F}_2^{32} \to \mathbb{F}_2^{32}$ depending on the subkey $K' = (k_{15}, \ldots, k_0) \in \mathbb{F}_2^{16}$ and consisting of 16 encryption cycles. Then the whole KeeLoq encryption mapping consists of successively computing 8 full round permutations $F(K)$ and consequently applying the last quarter round permutation $F'(K')$, see Figure 5.2. Note that the

first 8 full rounds are identical. The decryption can be represented in a similar way using inverse permutations $F'(K')^{-1}$ and $F(K)^{-1}$.

The algorithm allows for an extremely simple hardware implementation comprising a 32-bit shift register with taps on fixed positions, a 64-bit shift resister with a single tap and a 32-bit (5×1) look-up table (LUT) for the NLF. The LUT can be replaced with the corresponding logical elements according to (5.1). Then a compact hardware implementation of KeeLoq is estimated to require about 700 GE and 528 clock cycles per block, which is to be compared to about 1000-2000 GE needed for modern light-weight hardware-oriented stream ciphers such as Grain or Trivium or block ciphers such as PRESENT.

5.2.2 KeeLoq Protocols

Here the major types of security protocols are outlined in which the KeeLoq block cipher is involved. There are two main groups of authentication protocols used for component identification and secure vehicle access today: rolling codes (also known as hopping codes) and challenge-response protocols.

In *rolling/hopping codes*, the authentication of a component (also called transponder) to the main system (also called decoder) occurs through sending a dynamically changed bit combination from the transmitter to the main system. Rolling codes do not require the component to be capable of receiving any information. The main principle consists in maintaining a synchronized counter between the component and the main system. To authenticate itself to the main system, the component takes the current value of the counter, encrypts it using a block cipher, and sends it to the main system. After this, the counter is incremented on both communication sides. If the counter is kept synchronized and the parameters are well chosen, such protocols can be considered as secure. The most wide-spread product of this kind is KeeLoq supplied by Microchip, also extensively used in automotive applications. However, the necessity of synchronization and the unidirectional nature of communication may make this type of protocols difficult to implement in real-world applications in a secure way. For instance, if the attacker has access to the legal component for some time, he can trigger it the needed number of times to obtain the next hopping code values.

In *challenge-response protocols* the main system sends a random challenge to the component which is supposed to reply with the encrypted value of the challenge using the corresponding encryption key. The main system verifies this by decrypting the received value or by encrypting the challenge. If the parameters are well chosen, such protocol can guarantee an arbitrarily high security level. However, severe running time restrictions persist in typical real-world authentication scenarios. The typical real-time requirement is that a complete authentication session should take at most 50 ms. For instance, in the automotive area this can be connected with such applications as passive entry, where the human user triggers the vehicle to start

an authentication protocol that has to be finished in real time. This circumstance and the usually very limited effective bandwidth of 3.5 Kbps for LF transmission make the designers to meet half-way between security and performance, which is the case among others for Microchip KeeLoq [186]. Moreover, the number of bits transmitted often plays an important role in numerous applications with limited power consumption.

KeeLoq hopping codes. These are also known as rolling codes and provide authentication of an transponder to the decoder (the main system) by sending an encrypted counter value (unilateral communication) [182]. The transponder and decoder share a 64-bit symmetric key and a 16-bit synchronized counter value. To authenticate itself the transponder encrypts the next counter value and sends it to the decoder which decrypts the message and verifies whether the received counter value is within the acceptance window of length 16. A resynchronization mechanism exists to repair communication in case the counter value received exceeds the bounds of the acceptance window. See also [183], [184].

KeeLoq IFF. The IFF (Identify Friend or Foe) systems provide authentication of a component (transponder) to the main systems (decoder) using a simple challenge-response protocol (bilateral communication), see [189]. The transponder and decoder share a 64-bit symmetric key K. The transponder sends its 28-bit identifier N to the main system. To require authentication the decoder sends a 32-bit random challenge R to the transponder that replies with the corresponding KeeLoq-encrypted challenge using K. The decoder encrypts the genuine challenge using K corresponding to N and compares the message received as a reply with this value. If they coincide, the authentication is accepted. See also [185]. The protocol can be specified as follows[1] [185]:

$$\begin{array}{lll} \mathcal{V} \to \mathcal{T}: & R & (32) \\ \mathcal{T} \to \mathcal{V}: & \textsc{KeeLoq}_K(R) & (32), \end{array}$$

where $\textsc{KeeLoq}_K : \mathbb{F}_2^{32} \to \mathbb{F}_2^{32}$ is the KeeLoq block cipher.

5.2.3 KeeLoq Key Management Schemes

The KeeLoq systems use different key management mechanisms depending on the concrete model of transponder/decoder. In all these schemes, an individual transponder key is derived from a manufacturer's key MK and some transponder-specific

[1] A message from the transponder ($\mathcal{T}$) to the decoder ($\mathcal{V}$) consisting of an n-bit value A is denoted as:

$$\mathcal{T} \to \mathcal{V}: \quad A \quad (n).$$

information during the learning phase. The individual (KeeLoq encryption) key K is stored in the EEPROM of the transponder. The manufacturer's key MK is stored in the ROM and is fixed for large series of transponders. This enables each manufacturer to produce transponders that should not be cloned by competitors. For example, MK could remain the same for all immobilizers installed in a certain car model within one production year. This fact makes the manufacturer's key a valuable attack target. It turns out that one can deduce some information about the manufacturer key from an individual transponder key which can be found using cryptanalytic attacks on the KeeLoq block cipher described above. The concrete amount of gained information depends on the specific key derivation function.

There are two classes of key derivation functions used in KeeLoq systems: normal key generation and secure key generation. Here we concentrate on the XOR-based secure key generation and do not treat the normal key generation. The secure key derivation procedure has a variety of modes. Their general feature is that the individual transponder key depends on a randomly looking but fixed seed stored in the transponder and sent to the main system at the learning stage. The modes differ with respect to the length of the seed used.

To derive an individual key according to XOR-based secure key generation method, the transponder uses a seed S, its 28-bit transponder identifier $N_{27,0}$ and the 64-bit manufacturer's key $MK = MK_{63,0} = MK_{63,32}|MK_{31,0}$. Here $MK_{63,32}$ and $MK_{31,0}$ are the most significant and least significant 32-bit parts of MK, respectively.

There are three modes of the KeeLoq XOR-based secure key generation:

- *32-bit seed*: The 64-bit individual transponder key $K = K_{63,0} = K_{63,32}|K_{31,0}$ is obtained by XORing $MK_{31,0}$ with the seed $S = S_{31,0}$ and $MK_{63,32}$ with the zero-padded $N_{27,0}$ (Figure 5.3).
- *48-bit seed*: The seed $S_{47,0}$ is split into two parts - $S_{47,32}$ and $S_{31,0}$. $K_{31,0}$ is obtained by XORing $MK_{31,0}$ with $S = S_{31,0}$. $K_{63,32}$ is the XOR of the zero-padded $N_{11,0}|S_{47,32}$ with $MK_{31,0}$ (Figure 5.4).
- *60-bit seed*: In this case, the individual transponder key K does not depend on the transponder identifier SN. K is obtained by XORing MK with the zero-padded $S = S_{59,0}$ (Figure 5.5).

5.3 Attacks on the KeeLoq Algorithm

5.3.1 Basic Linear Slide Attack on KeeLoq Algorithm

Our basic attack is based on the following weaknesses of the algorithm: self-similar key schedule scheme, relatively short blocks of 32 bits, and existence of an efficient

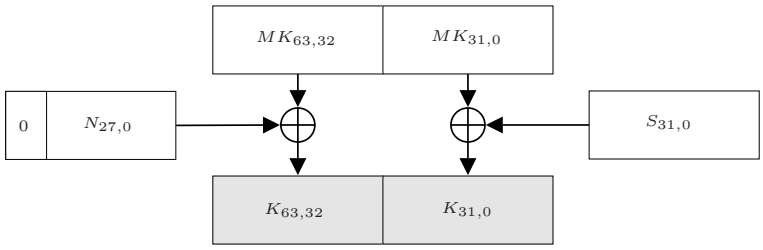


Figure 5.3: XOR-based secure key generation with a 32-bit seed

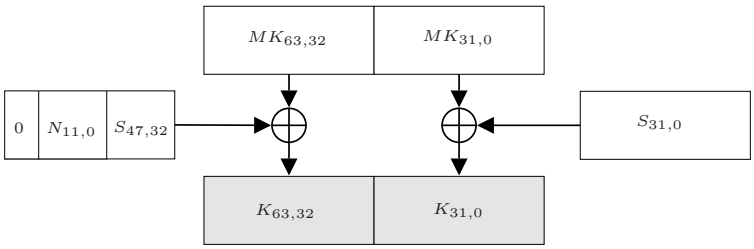


Figure 5.4: XOR-based secure key generation with a 48-bit seed

linear approximation of the NLF.

The attack can be outlined in the following way. For each subkey

$$K' = (k_{15}, \ldots, k_0)$$

and for a random 32-bit input $I_0 \in \mathbb{F}_2^{32}$ guess the corresponding output $O_0 \in \mathbb{F}_2^{32}$ after the 64 clock cycles which depends on the other 48 key bits $(k_{63}, \ldots, k_{16})$. Using the periodic structure of the KeeLoq key schedule generate several other pairs $(I_i, O_i) \in (\mathbb{F}_2^{32})^2$, $i = 1, \ldots, N-1$ (*sliding step*). For a successful attack N has to be about 2^8. For each number of such pairs we mount a distinguishing attack to obtain linear relations on some unknown key bits with a high probability due to the fact that the KeeLoq NLF is not 2-resilient (*linear correlation step*). In this way it is possible to determine $(k_{47}, \ldots, k_{16})$ bit by bit. After this an input/output pair for 16 encryption cycles can be represented as a triangular system of linear equations with the remaining bits $(k_{63}, \ldots, k_{48})$ of K as variables. It can be solved using 16 simple computational operations (*linear step*).

Sliding step. Using a single input/output pair (I_0, O_0) for the full round of 64 cycles and knowing the first 16 key bits $K' = (k_{15}, \ldots, k_0)$ one can produce an arbitrary number of other input/output pairs for this round. This is possible due to the fact that (almost) all rounds in KeeLoq are identical permutations which is the

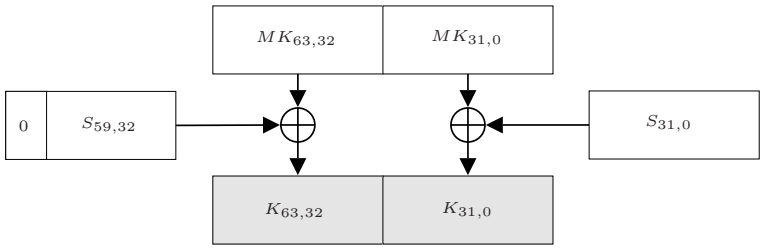


Figure 5.5: XOR-based secure key generation with a 60-bit seed

property on which the slide attacks by Biryukov and Wagner are substantially based [32], [31]. Once a pair (I_0, O_0) is known, the next input/output pair is produced by encrypting I_0 and O_0 with the key to be recovered (it is a chosen plaintext attack) and obtaining (I_1', O_1') as ciphertext. Then I_1' and O_1' are decrypted using the guessed partial key $K' = (k_{15}, \ldots, k_0)$. The resulting plaintexts form the needed pair (I_1, O_1), see Figure 5.6. From one pair (I_i, O_i), $i = 0, 1, 2, \ldots$, an arbitrary number of pairs (I_j, O_j), $j > i$ can be derived for a certain value of K' by iteratively encrypting I_i, O_i using KeeLoq and decrypting them with K', thus, obtaining (I_{j+1}, O_{j+1}). We call the set of pairs (I_i, O_i) needed for determining the key a *pseudo-slide group.*

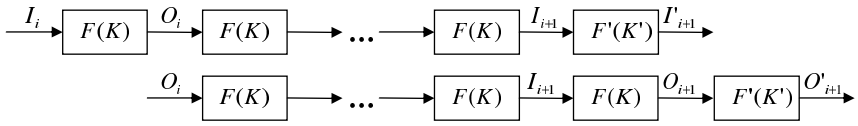


Figure 5.6: Generating input/output pairs using sliding techniques

As the sliding has to be performed for each guess of $K' = (k_{15}, \ldots, k_0)$ and each round output (2^{47} times on average) in our basic attack, its complexity is crucial for the efficiency of the whole attack.

Correlation step. Once the pseudo-slide group has been generated in the sliding step, the following weakness of the KeeLoq NLF with respect to correlation attacks is used due to the fact that the NLF is 1-resilient, but not 2-resilient.

Lemma 9. *For uniformly distributed $x_4, x_3, x_2 \in \mathbb{F}_2$ the following holds:*

- $\Pr\{NLF(x_4, x_3, x_2, x_1, x_0) = 0 \mid x_0 \oplus x_1 = 0\} = \frac{5}{8}$,
- $\Pr\{NLF(x_4, x_3, x_2, x_1, x_0) = 1 \mid x_0 \oplus x_1 = 1\} = \frac{5}{8}$.

This means that the NLF can be efficiently approximated by $x_0 \oplus x_1$. So, if x_0, x_1 are known and x_4, x_3, x_2 are random and unknown, we can determine $f(K)$ by

statistically filtering out the contribution of $NLF(x_4, x_3, x_2, x_1, x_0)$ to the equation

$$NLF(x_4, x_3, x_2, x_1, x_0) \oplus f(K) = 0$$

using a very limited number of such samples. $f(K)$ is a key-dependent Boolean function remaining constant for all samples.

Here we show how to obtain k_{16} and k_{32} from I_i and O_i. The remaining key bits $(k_{47}, \ldots, k_{33})$ and $(k_{31}, \ldots, k_{17})$ can be obtained in the same way by using k_{32}, k_{16} and shifting input and output bits.

We denote $I_i = Y^{(0)}$ and $O_i = Y^{(64)}$ for each i. The idea is to make use of the correlation weakness of the dependency between the output bits $y_0^{(64)}$, $y_1^{(64)}$ and the input bits $Y^{(0)}$. One can compute $Y^{(16)}$ from $Y^{(0)}$, since $K' = (k_{15}, \ldots, k_0)$ is known. For the next bit $y_{31}^{(17)}$, which is the first key-dependent bit, one has the following equation:

$$\begin{aligned} y_{16}^{(32)} = y_{31}^{(17)} &= NLF(y_{31}^{(16)}, y_{26}^{(16)}, y_{20}^{(16)}, y_9^{(16)}, y_1^{(16)}) \oplus y_0^{(16)} \oplus y_{16}^{(16)} \oplus k_{16} = \\ &= c_0 \oplus k_{16}, \end{aligned} \tag{5.2}$$

where $c_0 \in \mathbb{F}_2$ denotes the key-independent part of (5.2).

After 32 encryption cycles the following holds:

$$(y_{15}^{(32)}, y_{14}^{(32)}, \ldots, y_0^{(32)}) = (y_{31}^{(16)}, y_{30}^{(16)}, \ldots, y_{16}^{(16)}) \in \mathbb{F}_2^{16}\,.$$

Thus, the least significant half of $Y^{(32)}$ is known. Then $y_0^{(64)}$ can be represented as:

$$\begin{aligned} y_0^{(64)} &= NLF(y_{31}^{(32)}, y_{26}^{(32)}, y_{20}^{(32)}, y_9^{(32)}, y_1^{(32)}) \oplus y_0^{(32)} \oplus y_{16}^{(32)} \oplus k_{32} = \\ &= NLF(y_{31}^{(32)}, y_{26}^{(32)}, y_{20}^{(32)}, y_9^{(32)}, y_1^{(32)}) \oplus y_0^{(32)} \oplus (c_0 \oplus k_{16}) \oplus k_{32}, \end{aligned} \tag{5.3}$$

where $y_0^{(64)}$, $y_0^{(32)}$, $y_1^{(32)}$, $y_9^{(32)}$, c_0 are known and $y_{31}^{(32)}$, $y_{26}^{(32)}$, $y_{20}^{(32)}$, k_{32}, k_{16} are unknown. As the first two inputs of the NLF are known, its contribution to (5.3) can be replaced with the random variate ε using Lemma 9:

$$NLF(y_{31}^{(32)}, y_{26}^{(32)}, y_{20}^{(32)}, y_9^{(32)}, y_1^{(32)}) \oplus y_9^{(32)} \oplus y_1^{(32)} = \varepsilon \tag{5.4}$$

with

$$\Pr\{\varepsilon = 0\} = \frac{5}{8}. \tag{5.5}$$

Then the following holds:

$$y_0^{(64)} \oplus y_0^{(32)} \oplus c_0 \oplus y_9^{(32)} \oplus y_1^{(32)} = \varepsilon \oplus k_{16} \oplus k_{32}. \tag{5.6}$$

In order to determine $k_{16} \oplus k_{32}$ one has to distinguish between the following two cases: $k_{16} \oplus k_{32} = 0$ and $k_{16} \oplus k_{32} = 1$. In the first case:

$$\Pr\{y_0^{(64)} \oplus y_0^{(32)} \oplus c_0 \oplus y_9^{(32)} \oplus y_1^{(32)} = 0\} = \frac{5}{8}.$$

Otherwise, this probability is 3/8.

Thus, the bias δ of the first random variable with respect to the second one is $\delta = \frac{1}{4}$. Our experiments show that about 2^7 equations (5.6) for different pairs (I_i, O_i), $i = 0, \ldots, 2^7 - 1$, are needed to recover $\alpha = k_{16} \oplus k_{32}$ with an acceptable error probability (for all 32 key-dependent linear combinations to be determined in this way)

Next we consider $y_1^{(64)}$ and its dependencies from the input and key bits. Similar to (5.2) one has:

$$\begin{aligned} y_{16}^{(33)} &= NLF(y_{31}^{(17)}, y_{27}^{(16)}, y_{21}^{(16)}, y_{10}^{(16)}, y_2^{(16)}) \oplus y_1^{(16)} \oplus y_{17}^{(16)} \oplus k_{17} = \\ &= NLF(c_0 \oplus k_{16}, y_{27}^{(16)}, y_{21}^{(16)}, y_{10}^{(16)}, y_2^{(16)}) \oplus y_1^{(16)} \oplus y_{17}^{(16)} \oplus k_{17} = \\ &= c_1' \oplus c_2 k_{16} \oplus y_1^{(16)} \oplus y_{17}^{(16)} \oplus k_{17} = c_1 \oplus c_2 k_{16} \oplus k_{17}, \end{aligned} \tag{5.7}$$

where $c_1' \in \mathbb{F}_2$ is the free term of NLF, $c_2 \in \mathbb{F}_2$ is its linear term with respect to k_{16}, and $c_1 = c_1' \oplus y_1^{(16)} \oplus y_{17}^{(16)} \in \mathbb{F}_2$. Here c_1 and c_2 are known and depend on $Y^{(0)}$. Then the second output bit $y_1^{(64)}$ is represented as follows:

$$\begin{aligned} y_1^{(64)} &= NLF(y_{31}^{(33)}, y_{26}^{(33)}, y_{20}^{(33)}, y_9^{(33)}, y_1^{(33)}) \oplus y_0^{(33)} \oplus y_{16}^{(33)} \oplus k_{33} = \\ &= (\varepsilon \oplus y_9^{(33)} \oplus y_1^{(33)}) \oplus y_0^{(33)} \oplus (c_1 \oplus c_2 k_{16} \oplus k_{17}) \oplus k_{33}, \end{aligned} \tag{5.8}$$

where the random variate ε is assigned in a way similar to (5.4) and c_0, c_1, c_2, $y_0^{(33)}$, $y_9^{(33)}$, $y_1^{(33)}$ are known. To determine $k_{17} \oplus k_{33}$ pairs (I_i, O_i) with $c_2 = 0$ are selected[2]. Then ε in (5.8) is filtered out statistically, which recovers $\beta = k_{17} \oplus k_{33}$. After this the remaining pairs (I_i, O_i) (with $c_2 = 1$) are used to obtain $\gamma = k_{16} \oplus k_{17} \oplus k_{33}$ in the same way. Thus, $k_{16} = \beta \oplus \gamma$ and $k_{32} = \alpha \oplus k_{16}$.

Now k_{16}, k_{32} and $k_{17} \oplus k_{33}$ are known. In the next step we determine $k_{18} \oplus k_{34}$ and $k_{17} \oplus k_{18} \oplus k_{34}$ using the same plaintext/ciphertext pairs (I_i, O_i) and the same statistical recovery method. In this way all 32 key bits $(k_{47}, \ldots, k_{16})$ are obtained in only 16 rather simple computational steps.

Linear step and key verification. The remaining key bits $(k_{63}, \ldots, k_{48}) \in \mathbb{F}_2^{32}$ can be recovered as follows. As $(k_{47}, \ldots, k_0)$ are known, $Y^{(48)}$ can be computed for each pair (I_i, O_i). $y_{16}^{(64)}$ can be expressed as:

$$y_{16}^{(64)} = NLF(y_{31}^{(48)}, y_{26}^{(48)}, y_{20}^{(48)}, y_9^{(48)}, y_1^{(48)}) \oplus y_{16}^{(48)} \oplus y_0^{(48)} \oplus k_{48}, \tag{5.9}$$

which reveals k_{48} since the entire state $Y^{(48)}$ is known. Now $Y^{(49)}$ can be completely calculated which leads to the value of k_{49} using $y_{17}^{(64)}$, and so on. In this way the rest of the key is recovered.

[2]Note that for random inputs I_i the probability of $c_2 = 0$ is 0.5. Therefore about $N/2$ out of N known pairs (I_i, O_i) will lead to $c_2 = 0$. This raises the required number of plaintext/ciphertext pairs to about 2^8.

At the end of the key recovery procedure we expect to obtain a number of key candidates. The upper bound for their average quantity is $2^{64-32} = 2^{32}$ due to the known plaintext unicity distance [179], since the block length is 32 bit and the key length is 64 bit. Thus, we need to verify each key candidate against max. $\lceil \frac{64+4}{32} \rceil = 3$ plaintext-ciphertext pairs for all 528 encryption cycles.

Attack complexity and experiments. The attack consists of the following stages:

- Compute all plaintext-ciphertext pairs for the whole cipher;
- Guess the partial key K' and the output O_0 after one round for some input I_0;
- For each pair of guesses (K', O_0) do the following:
  - Obtain $2^8 - 1$ other pairs (I_i, O_i); thus, the cardinality of the pseudo-slide group is 2^8 for this attack;
  - Determine $k_{16} \oplus k_{32}$ by evaluating c_0 for the first 2^7 pairs of the pseudo-slide group;
  - Determine $(k_{47}, \ldots, k_{16})$ by evaluating c_1 and c_2 2^8 times;
  - Determine $(k_{63}, \ldots, k_{48})$ by evaluating 2^4 nonlinear boolean functions;
- Verify max. 2^{32} candidate keys using at most 3 plaintext-ciphertext pairs for the whole cipher and 3 full encryption operations.

If one step is equivalent to a single full KeeLoq encryption (528 encryption cycles), 2^{32} steps are needed for generating 2^{32} plaintext-ciphertext pairs. Each element has to be stored in a memory of 2^{32} 32-bit words.

For each guess of (I_0, O_0) and K' operations of the following complexity have to be be performed:

- $2^9 - 2$ memory accesses for obtaining (I_i, O_i), $i = 1, \ldots, 2^8 - 1$. We assume a single memory access equivalent to 4 encryption cycles. This leads to approximately 2^2 steps required to perform the memory accesses.
- 2^7 evaluations of c_0 and $k_{16} \oplus k_{32}$, each evaluation being equivalent to one encryption cycle. The complexity of this stage is then $2^7/528 \approx 2^{-2}$ steps.
- $16 \cdot 2^8 = 2^{12}$ evaluations of c_1 and c_2 for determining $(k_{47}, \ldots, k_{16})$. Each evaluation is computationally equivalent to one encryption cycle. This stage requires about $2^{12}/528 \approx 2^3$ steps.
- 2^4 evaluations of a boolean function to determine $(k_{63}, \ldots, k_{48})$. Each evaluation is equivalent to one encryption cycle which leads to a complexity of about $2^4/528 \approx 2^{-5}$ steps.

Max. 2^{32} candidate keys have to be verified using at most 3 full encryptions which requires max. 2^{34} steps. Thus, the overall computational complexity of the attack is

$$2^{32} + \frac{2^{32} \cdot 2^{16}}{2} \cdot (2^2 + 2^{-2} + 2^3 + 2^{-5}) + 2^{34} \approx 2^{50.6} \text{ steps.}$$

The memory complexity is quite reasonable and is 2^{32} 32-bit words (16 GByte). This enables an attacker to place all plaintext-ciphertext values into RAM which substantially accelerates the implementation of the attack. Most computations in our attack are perfectly parallelizable.

5.3.2 Cycle-Based Attack on KeeLoq Algorithm

In this subsection, we first outline some parallel work on the cryptanalysis of KeeLoq including algebraic attacks and cycle structure analysis. Then we combine our basic linear slide attack with the cycle structure analysis and obtain the fastest attack on KeeLoq working for the whole key space, which requires about 2^{40} KeeLoq encryptions.

Algebraic attack. Courtois and Bard [70] used the idea of sliding KeeLoq, but employed algebraic techniques to obtain the key from a slid pair. The attack requires only one slid pair. This eliminates the necessity to guess the partial key K'.

The attack works as follows. By exploring 2^{16} random known plaintext-ciphertext pairs, the attacker expects to have at least one slid pair (I_i, I'_{i+1}), (O_i, O'_{i+1}) with $O_i = F(I_i)$. This forms the first 32 equations of the non-linear system, which are not sufficient for uniquely determining the key. To obtain more equations, the attacker can use the fact that the ciphertexts I'_{i+1} and O'_{i+1} are related by $F'[F[F'^{-1}[I'_{i+1}]]] = O'_{i+1}$ (see Figure 5.6), which gives the other 32 binary equations. That is, the ciphertexts in the slid pair are one round (64 KeeLoq cycles) apart in the same KeeLoq cipher with K rotated by 16 bits - $(k_{16}, k_{17}, \ldots, k_{63}, k_0, \ldots, k_{15})$.

After this, the 64 equations are solved using the SAT solver MiniSat. The procedure has to be performed 2^{31} times on average (for each combination of the available plaintext-ciphertext pairs). The total complexity of the attack is about 2^{53} KeeLoq encryptions and it requires only 2^{16} known plaintexts. However, our basic attack is faster than this algebraic attack, though requiring more known plaintexts.

Cycle structure attack. The second attack from [70] makes use of the specific cycle structure imposed by the fact that the first 8 KeeLoq rounds (64 clock cycles each) are exactly the same.

For a random permutation on n-bit words, there are on average about $\ln 2^n$ cycles. That is, for $n = 32$ the expected number of cycles is about 22, approximately half of them being even. If the same permutation is applied twice, the cycles of

even size split into two classes - even and odd cycles. The odd cycles of the original permutation persist.

In KeeLoq, the same permutation F, which we consider as random, is applied 8 times (8 full rounds of KeeLoq) followed by a quarter round F'. That is, the permutation $F^8(\cdot)$ has about $11/2^{\log 8} \approx 1.4$ cycles of even size[3]. At the same time, a random permutation would have about 11 cycles of even size.

Thus, one can determine the correct value of K' by decrypting all ciphertexts one quarter round and counting the number of cycles of even size. If there are more than 6 even cycles, this can be seen as a random permutation and the guess of K' is wrong. Otherwise, the guessed K' is correct. This test allows one to determine the correct value of K' with a high probability. The complexity of this step is about 2^{40} KeeLoq encryptions.

Extended attack. Now we can determine the quarter key $K' = (k_{15}, \ldots, k_0)$ using the cycle structure attack described above. This requires 2^{32} known plaintext-ciphertext pairs and about 2^{40} KeeLoq encryptions. Then we just apply our basic attack from Section 5.3.1.

Actually, we do not have to perform the whole attack, as the first 16 key bits are already known. The attacker chooses two plaintext-ciphertext pairs and builds the corresponding pseudo-slide group of size 2^8. Then the correlation and linear steps of our basic attack are directly applied to determine the remaining 48 bits of the key. This operation has to be performed for approximately 2^{31} random plaintext-ciphertext pairs to achieve a high success probability. That is, the complexity of these steps is about $2^{35.5}$ KeeLoq encryptions.

Thus, we have built an attack of complexity 2^{40} operations working for the whole key space and requiring 2^{32} known plaintexts. This is the fastest known attack on the KeeLoq block cipher applicable to the whole key space.

Though it might seem that the data requirements make the attack unpractical, we show in the next section that our attacks can have practical relevance due to some severe weaknesses of the key management schemes used in the real-world KeeLoq systems.

5.3.3 Further Attacks and Comparison

Since the publication of these results, some further cryptanalysis of the KeeLoq algorithm has been conducted. In Table 5.1 we put our attacks into the framework of these more recent attacks and show that the cycle-structure based version of our attack is still the fastest one working for the whole key space.

[3]Note that we believe that [70] reports an incorrect expected number of even cycles in this case.

Table 5.1: Different attacks on the KeeLoq block cipher

	data	time	%keys	memory
slide-linear, this chapter	2^{32} KP	$2^{50.6}$	100%	16 GB
cycle-linear, this chapter	2^{32} KP	2^{40}	100%	18 GB
self-similarity [69]	2 KP	$2^{60.4}$	100%	16 GB
slide-algebraic [70]	2^{16} KP	2^{53}	63%	negl.
slide-meet-in-the-middle [19]	2^{16} KP	2^{46}	63%	3 MB
slide-meet-in-the-middle [19]	2^{16} CP	2^{45}	63%	3 MB
slide-determine [72]	2^{32} KP	2^{31}	63%	16 GB
slide-determine [72]	2^{32} KP	2^{28}	30%	16 GB

5.4 Attacks on KeeLoq Key Management

The protocols and key derivation schemes described above allow one to deduce some information about the manufacturer's key MK from a single individual transponder key.

In the case of KeeLoq IFF, the attacker can freely collect plaintext-ciphertext pairs choosing challenges and recording the responses. The PIC16 controllers that KeeLoq systems base upon run at a frequency of 4 MHz. As the KeeLoq cipher is implemented in hardware, every of its 528 clocks takes about 2^{-18} s. That is, it can be theoretically possible to encrypt all 2^{32} plaintexts within about 100 days on a PIC16 controller. Other controllers using KeeLoq can be clocked faster, thus reducing the time needed to collect the plaintext-ciphertext pairs. For instance, if a KeeLoq implementation was clocked with a moderate frequency of 64 MHz, the attacker could theoretically collect all needed pairs within about 6 days. Note that the transponder identifier is known as it is transmitted in plaintext before each protocol run.

Thus, one can apply an efficient attack on the KeeLoq block cipher and recover the individual transponder key K. As the identifier is known, the attacker obtains 32, 16, or 4 bits of MK by XORing the found K with $N_{27,0}$ or simply taking the 4 most significant bits, depending on the used key derivation function. This reduces the security of all other KeeLoq systems using the same manufacturer's key MK to 32, 48, or 60 bits, respectively.

Moreover, the attacker can have access to the seed used. For instance, he can intercept the transmission of the seed from the transponder to the decoder during the learn procedure, as the seed is sent in plaintext. Note that the attacker can also put an transponder into the learn mode by activating some of the PIC16 pins. In this case, the attacker pulls off the seed from the individual transponder key by XORing these two numbers and obtains the full manufacturer's key MK. This all other KeeLoq systems using the same manufacturer's key totally insecure, if the corresponding seeds are known.

5.5 Generic Attacks on KeeLoq IFF Protocol

In this section, we briefly discuss the security of the KeeLoq IFF protocol in the context of generic attacks.

A simple challenge-response protocol can be described as:

$$\begin{array}{lll} \mathcal{V} \rightarrow \mathcal{T}: & R & (n) \\ \mathcal{T} \rightarrow \mathcal{V}: & enc(R) & (m), \end{array} \tag{5.10}$$

where R is an n-bit challenge and enc is a keyed cryptographic function $enc : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ (e.g. an n-bit block cipher with a truncated m-bit output).

Basically, there are two main generic attack approaches for simple challenge-response protocols:

Scan Attacks (SA): Scan attacks as applied to the challenge-response protocol (5.10) can be described in the following way. First, the challenge (with or without an authentication field) is received by the attacker $\mathcal{A}$ from the main system and discarded. Then, $\mathcal{A}$ sends back a constant m-bit value R':

$$\begin{array}{lll} \mathcal{V} \rightarrow \mathcal{A}: & R & (n) \\ \mathcal{A} \rightarrow \mathcal{V}: & R' & (m). \end{array}$$

After $C_{SA} = 2^{m-1}$ trials, it is expected that the response has been accepted by the main system at least once.

Simple Two-Step Dictionary Attacks (DA): The two-step dictionary attack is the most powerful attack against generic challenge-response protocols:

$$\begin{array}{lll} \text{Step 1 } (\mathcal{A} \leftrightarrow \mathcal{T}): & & \\ & \mathcal{A} \rightarrow \mathcal{T}: \quad R & (n) \\ & \mathcal{T} \rightarrow \mathcal{V}: \quad enc(R) & (m) \\ & \mathcal{A}: \quad \mathbb{S} \leftarrow \mathbb{S} \cup (R, enc(R)) & \\ \text{Step 2 } (\mathcal{A} \leftrightarrow \mathcal{V}): & & \\ & \mathcal{V} \rightarrow \mathcal{A}: \quad R & (n) \\ \text{If } (R, \cdot) \in \mathbb{S}, \text{ then} & \mathcal{A} \rightarrow \mathcal{V}: \quad enc(R) & (m) \end{array}$$

In the first step, the attacker sends random challenges R to the component as well as receives and logs the responses together with the corresponding challenges $enc(R)$. After $2^{n/2}$ challenge-response pairs $(R, enc(R))$ have been sampled and recorded into the set $\mathbb{S}$, the second step is started: The main system is triggered to send challenges R. After $2^{n/2}$ challenges have been received from the main system, the probability that the attacker has at least one intersection with the $2^{n/2}$ recorded challenge-response pairs $\mathbb{S}$ is about 0.5 by the birthday paradox. That is, the overall complexity of the attack is about $C_{DA} = 2^{n/2} + 2^{n/2} = 2^{n/2+1}$ communication sessions.

The implications of these attacks for the KeeLoq IFF protocol are as follows. The security level of KeeLoq IFF is about 2^{17} communication sessions due to the standard dictionary attack. The scan attack would require about 2^{31} communication sessions. Note that attacks with a communication complexity under 2^{20} sessions can be performed within few hours in practice, even if one session requires 50 ms to succeed. Many attacks, including scan and dictionary-type ones, can be significantly hampered by additional organizational measures implemented in the main system.

5.6 Conclusions

In this chapter we proposed two key-recovery attacks on KeeLoq – a block cipher used in numerous automotive applications as well as in various property identification systems.

Our cryptanalysis uses techniques of guess-and-determine, sliding, linear and cycle structure attacks. Our basic attack works with complexity of $2^{50.6}$ KeeLoq encryptions (while KeeLoq uses a 64-bit key). It requires all 2^{32} plaintexts and a memory of 2^{32} 32-bit words. Our second attack uses the cycle structure techniques to determine the first 16 bits of the key and then applies our first attack. It has a complexity of 2^{40} encryptions and requires 2^{32} known plaintexts, while being applicable to the whole key space. This is the fastest known attack on the KeeLoq block cipher working for the whole key space.

We demonstrated that several real-world applications are vulnerable to attacks on the KeeLoq block cipher, including KeeLoq IFF systems and KeeLoq hopping codes. After attacking one instance of KeeLoq, one can reduce the effective key length of all other KeeLoq systems of the same series to 32, 48, or 60 bits depending on the key management scheme applied. In some cases, the attacker is even able to obtain the encryption key instantly. All this imposes a real threat on the KeeLoq systems. We also considered some generic attacks on the KeeLoq IFF authentication protocol.

Chapter 6

Practical Cryptanalysis of A5/2 with Special-Purpose Hardware

A5/2 is widely applied for protecting GSM communication. It is a synchronous stream cipher. Several powerful attacks [12, 106] against A5/2 exist. In this chapter the ciphertext-only attack [12] is accelerated by proposing a hardware engine to generate and solve the relevant equation systems. The approach of [46, 47] is used for implementing the parallelized Gauss-Jordan elimination. With this hardware architecture, the attack instantly determines the initial internal state of A5/2. This allows one to efficiently recover the plaintexts sent in all frames of a communication session using a few ciphertext frames only. The hardware-assisted attack does not require any precomputations or memory and targets the GSM speech channel (TCH/FS and TCH/EFS) directly. 16 ciphertext frames and about 1 second are required. Similar techniques are also applicable to other GSM channels (such as SDCCH/8).

This chapter is mainly based on the joint work of the author with Thomas Eisenbarth and Andy Rupp published in [42] that in turn relies on the hardware engine stemming from the paper [46] published by the author jointly with Marius Mertens, Christof Paar, Jan Pelzl and Andy Rupp.

6.1 Introduction

The GSM (Global System for Mobile communications) standard specifies algorithms for data encryption and authentication. Initially the stream cipher A5/1 was standardized for encryption purposes. The stream cipher A5/2, an intentionally weaker version of A5/1, was developed for deploying GSM outside Europe. The designs of both ciphers were revealed in [52].

The security of A5/1 has been extensively analyzed, e.g., in [11, 17, 30, 92, 107, 178]. However, in this chapter the real-world security of A5/2 is studied which is still important for the overall security of GSM communication. Partially, this is due to some flaws in the GSM protocols that allow to take advantage of attacks on A5/2 even if a stronger encryption algorithm (say, A5/1 or A5/3) is used [12]. These weaknesses can be exploited whenever the cell phone supports A5/2.

A known-plaintext attack on A5/2 was presented in [106]. The attack itself requires only two plaintext frames, but these frames have to be exactly 1326 frames apart to fulfill a certain property. In [213] another attack on A5/2 was proposed which requires 4 arbitrary plaintext frames and allows to decrypt most of the remaining communication (the internal state is not recovered).

In [12] a guess-and-determine attack on A5/2 was proposed that requires 4 plaintext frames for session key recovery. The idea of this attack is to guess 16 bits of the internal state of the cipher and then express the output as a degree-2 function of the remaining unknown 61 initial state bits. In this way each known plaintext frame yields 114 quadratic equations. Given 4 plaintext frames, one obtains a system of linear equations of dimension 456×655 by linearizing the equations. Though the system is underdetermined, experiments show that this number of equations is

enough to resolve the 61 original linear variables. This attack can be also turned into a ciphertext-only attack. Here, due to the fact that GSM employs error correction before encryption, the attacker possesses knowledge of the values of certain linear combinations of the stream bits. The attack consists of a precomputation phase and an online phase. In the precomputation phase, the equation systems for all guesses are computed in advance. In the online phase, this data is used to solve the equations for the specific input frames. These guesses also concern the initialization vectors that are used to set up A5/2 and which are derived from the frame numbers. Estimations of a full-optimized attack against GSM control channel SDDCH/8 are provided. In this case the precomputation can be done in about 11 hours on a PC requiring 1GB of RAM and producing 4GB of data. In the online phase, eight consecutive ciphertext frames are needed to find the session key in about 1 second.

In this chapter we show that a hardware-only attack against A5/2 leads to significant improvements in terms of time, memory and flexibility compared to current software attacks, although existing attacks are already quite efficient. Our general approach is similar to the ciphertext-only attack described in [12]. However, no precomputation is required and the ciphertext frames that can be used to mount the attack do not need to satisfy any special properties (e.g., appropriate differences of initial vectors). In contrast to the software implementation in [12], we designed our architecture to directly attack the speech channel. Namely, it uses ciphertext frames from the GSM speech traffic channel (TCH/FS and TCH/EFS) instead of a specific control channel (e.g., SDCCH/8 in [12]) to mount the attack. The advantage is that eavesdropping can start immediately at any time during a call (not only at the set-up of the call) without waiting until appropriate data is transmitted over the specific control channel. However, since the proposed architecture is quite generic, also other GSM channels (e.g., SDCCH/8) can be attacked (with minor changes). Based on our architecture, even a hardware device is conceivable where the target channel can be chosen at runtime.

The basic hardware architecture for attacking the speech channel requires 16 consecutive ciphertext frames as input and outputs the recovered secret initial state of A5/2. This initial state is sufficient to decrypt all frames of a session and to recover the key. The core blocks of the architecture are 3 equation generators and the solver for linear systems of equations. As an implementation of the latter building block, we have chosen the solver for linear equation systems recently proposed in [46, 47]. As a basis for estimating the chip size and average power consumption of an ASIC implementation, all relevant parts of the architecture design have been implemented in VHDL and synthesized. As a result, one obtains approx. 9.3 million gate equivalents for the entire hardware engine. At 256 MHz for the main chip component and 512 MHz for the rest of the engine, the architecture consumes roughly 12.8 Watts of energy and completes one attack instance in about one second on average. Note that the design requires less than 15% of the area and consumes less

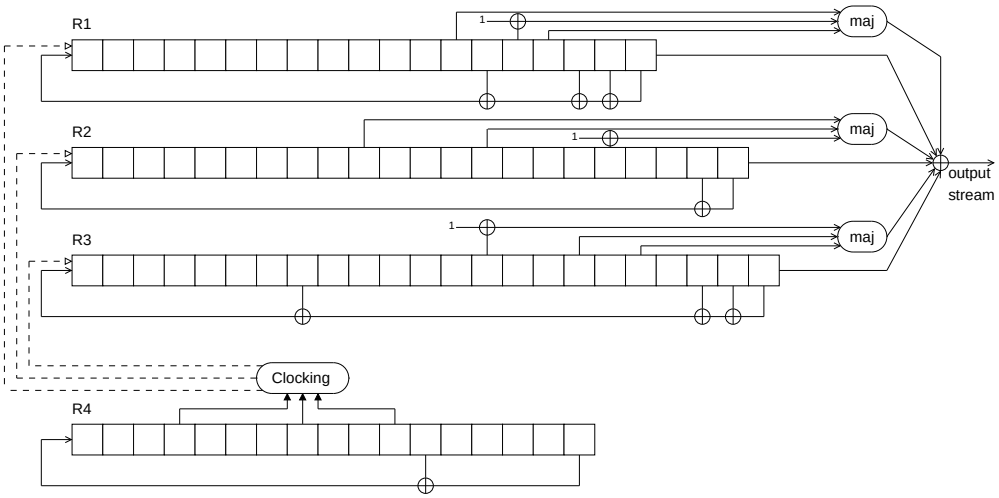


Figure 6.1: A5/2

than one third of the power of the Core 2 Duo "Conroe" processor.

6.2 A Short Description of A5/2

A5/2 is a synchronous stream cipher working with a 64-bit key K and a 22-bit initial vector IV:

$$K = (k_0, \ldots, k_{63}) \in \mathbb{F}_2^{64},$$

$$IV = (v_0, \ldots, v_{21}) \in \mathbb{F}_2^{22}.$$

IV is derived from the 22-bit frame number which is publicly known. It uses four linear feedback shift registers (LFSRs) $R1$, $R2$, $R3$ and $R4$ of lengths 19, 22, 23 and 17 bits, respectively, as its main building blocks (see Figure 6.1). The taps of the LFSRs correspond to primitive polynomials and, therefore, the registers produce sequences of maximal periods. $R1$, $R2$ and $R3$ are clocked irregularly based on the current state of $R4$. The A5/2 stream cipher has three phases: initialization phase, warm-up phase and encryption phase. In the warm-up and encryption phases, the registers $R1$, $R2$ and $R3$ are clocked irregularly.

The stop/go clocking is determined by the bits $R4[3]$, $R4[7]$ and $R4[10]$ in each clock cycle as follows: the majority $maj(a, b, c)$ of the three bits is computed where $maj(a, b, c) = ab \oplus ac \oplus bc$. $R1$ is clocked iff $R4[10]$ agrees with the majority. $R2$ is clocked iff $R4[3]$ agrees with the majority. $R3$ is clocked iff $R4[7]$ agrees with the majority. In each cycle at least two of the three registers are clocked. After these clockings, $R4$ is (regularly) clocked, and an output bit is generated from the values

of $R1$, $R2$, and $R3$ by adding their rightmost bits to three majority values, one for each register (see Figure 6.1).

In the initialization phase, all registers are set to 0. Then the *key setup* and the *IV setup* are performed. Here the key resp. IV bits are cyclically added to the registers modulo 2. At the end of the initialization phase one bit in each register is set to 1:

$R1 \leftarrow 0,\ R2 \leftarrow 0,\ R3 \leftarrow 0,\ R4 \leftarrow 0;$
for $i = 0...63$ do
  -Clock $R1, R2, R3, R4$;
  -$R1[0] \leftarrow R1[0] \oplus k_i, R2[0] \leftarrow R2[0] \oplus k_i, R3[0] \leftarrow R3[0] \oplus k_i, R4[0] \leftarrow R4[0] \oplus k_i$;
for $i = 0...21$ do
  -Clock $R1, R2, R3, R4$;
  -$R1[0] \leftarrow R1[0] \oplus v_i, R2[0] \leftarrow R2[0] \oplus v_i, R3[0] \leftarrow R3[0] \oplus v_i, R4[0] \leftarrow R4[0] \oplus v_i$;
$R1[15] \leftarrow 1,\ R2[16] \leftarrow 1,\ R3[18] \leftarrow 1,\ R4[10] \leftarrow 1$

In the warm-up phase, $R4$ is clocked 99 times irregularly, the output being discarded. In the encryption phase, A5/2 produces 228 output bits, one per clock cycle. 114 of them are used to encrypt uplink traffic, while the remaining bits are used to decrypt downlink traffic. In this chapter we treat only a half of this keystream bits used for traffic encryption in one direction.

6.3 Attack Outline

The approach pursued in this chapter is similar to that of [12] where a ciphertext-only attack is proposed. At the same time, we do not require any precomputation. Moreover, the ciphertext frames that can be used to mount the attack without any special properties (e.g. only those possessing suitable frame numbers are allowed to be used in some attacks). The attack needs the ciphertext of any l frames encrypted using the same session key K. The parameter l depends on the channel that should be attacked. For example, the attack requires about $l = 16$ frames in the the speech channel scenario as shown in this chapter and about $l = 8$ frames for the SDCCH/8 channel.

The major idea of the approach is that the internal state of the register $R4$ is guessed right after initialization (there are 2^{16} possible states). Then every bit of the generated key stream is written in terms of the initial states of the registers $R1, R2$ and $R3$. We then use certain information about the key stream bits – which are provided by the error correction coding of the GSM channel – to construct an overdetermined quadratic system of equations. This system is linearized and then solved using Gaussian elimination. Above procedure is repeated for different guesses of $R4$ until the correct solution is found. Using this solution, we can easily construct

the internal state of A5/2 after initialization for an arbitrary frame that has been encrypted using K. This is already sufficient to decrypt all frames of a session, since we can construct the respective states and load them into the A5/2 machine. However, by reversing the initialization phase, we can also recover the session key.

Below our attack is treated in more detail. So we first introduce the basic notation. We denote the l known ciphertext frames by $C_0, \ldots, C_{l-1}$ and the corresponding (unknown) plaintext frames by $P_0, \ldots, P_{l-1}$. For each frame C_h (or P_h) we denote the respective initialization vector by $IV_h = (v_{h,0}, \ldots, v_{h,21})$ and the key stream by $S_h = (s_{h,0}, \ldots, s_{h,113})$. Furthermore, let $R1^{(h)}$, $R2^{(h)}$, $R3^{(h)}$ and $R4^{(h)}$ be the internal states of the registers of A5/2 during a certain cycle when generating S_h.

6.3.1 Key-Stream as a Function of Internal State

Consider the frame C_h and the corresponding stream generation for this frame. At the beginning of the initialization phase the registers $R1^{(h)}$, $R2^{(h)}$, $R3^{(h)}$ and $R4^{(h)}$ are all set to zero. Then the key setup is performed for 64 clock cycles, where in each cycle the first bit of each LFSR is set to the sum of the respective feedback value and one of the key bits (see Section 6.2). After that, due to the linearity of the feedback functions the bits of the three registers can be written as certain linear combinations of K, e.g., $R1^{(h)}[0] = k_0 \oplus k_{19} \oplus k_{38} \oplus k_{47}$. In the subsequent initialization step, the IV setup, the initialization vector IV_h is added to the content of the registers in an analogous manner. Thus, the resulting register bits are (known) linear combinations of the key bits and the IV bits. Finally, certain bits of the registers are set to 1. More precisely, after initialization the registers $R1^{(h)}$ to $R4^{(h)}$ can be written as

$$\begin{array}{rcl} R1^{(h)} & = & (\alpha_0 \oplus \sigma_{h,0}, \ldots, \alpha_{14} \oplus \sigma_{h,14}, 1, \alpha_{15} \oplus \sigma_{h,15}, \ldots, \alpha_{17} \oplus \sigma_{h,17}), \\ R2^{(h)} & = & (\alpha_{18} \oplus \sigma_{h,18}, \ldots, \alpha_{33} \oplus \sigma_{h,33}, 1, \alpha_{34} \oplus \sigma_{h,34}, \ldots, \alpha_{38} \oplus \sigma_{h,38}), \\ R3^{(h)} & = & (\alpha_{39} \oplus \sigma_{h,39}, \ldots, \alpha_{56} \oplus \sigma_{h,56}, 1, \alpha_{57} \oplus \sigma_{h,57}, \ldots, \alpha_{60} \oplus \sigma_{h,60}), \\ R4^{(h)} & = & (\alpha_{61} \oplus \sigma_{h,61}, \ldots, \alpha_{70} \oplus \sigma_{h,70}, 1, \alpha_{71} \oplus \sigma_{h,71}, \ldots, \alpha_{76} \oplus \sigma_{h,76}), \end{array} \tag{6.1}$$

where $\alpha_i \in \mathsf{span}(k_0, \ldots, k_{63})$ and $\sigma_{h,i} \in \mathsf{span}(v_{h,0}, \ldots, v_{h,21})$. Observe that since IV_h is known, the values $\sigma_{h,0}$ to $\sigma_{h,76}$ can be considered as known constants. So only the α_i values are unknowns. Note that we have the *same* α_i's for *all* frames C_h. In the following, we guess the values of $\alpha_{61}, \ldots, \alpha_{76}$, determine the initial secret state $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_{60}) \in \mathbb{F}_2^{61}$ and verify this solution.[1] This procedure has to be repeated at most 2^{16} times until $\alpha_{61}, \ldots, \alpha_{76}$ take on the correct values.

The bits of the key stream S_h for each frame C_h are re-written in terms of α in order to determine α. Then certain information about these bits is used to construct a linear system of equations which is then solved by Gaussian elimination. Before

[1] Since the registers $R1^{(h)}, R2^{(h)}$ and $R3^{(h)}$ are clocked irregularly after initialization based on certain bits of $R4^{(h)}$, the clocking of these registers are fully determined by guessing α_{61} to α_{76}.

the first stream bit for C_h is generated the warm-up phase is executed running for 99 cycles. After warm-up, a stream bit is generated from the current internal states of $R1^{(h)}, R2^{(h)}$ and $R3^{(h)}$ every cycle. In an *arbitrary* cycle of A5/2 (after initialization), these states can be written as

$$\begin{array}{lcl} R1^{(h)} & = & (\beta_{h,0} \oplus \delta_{h,0}, \ldots, \beta_{h,18} \oplus \delta_{h,18}), \\ R2^{(h)} & = & (\beta_{h,19} \oplus \delta_{h,19}, \ldots, \beta_{h,40} \oplus \delta_{h,40}), \\ R3^{(h)} & = & (\beta_{h,41} \oplus \delta_{h,41}, \ldots, \beta_{h,63} \oplus \delta_{h,63}), \end{array} \tag{6.2}$$

where

$$\beta_{h,0}, \ldots, \beta_{h,18} \in \mathsf{span}(\alpha_0, \ldots, \alpha_{17}),$$

$$\beta_{h,19}, \ldots, \beta_{h,40} \in \mathsf{span}(\alpha_{18}, \ldots, \alpha_{38}),$$

$$\beta_{h,41}, \ldots, \beta_{h,63} \in \mathsf{span}(\alpha_{39}, \ldots, \alpha_{60}), \text{ and}$$

$$\delta_{h,i} \in \mathsf{span}(v_{h,0}, \ldots, v_{h,21}, 1).$$

Note that the linear combinations $\beta_{h,i}$ depend on the specific frame C_h, since the clocking of the registers now depends on IV_h. (Certainly, $\beta_{h,i}$ and $\delta_{h,i}$ also depend on the specific clock cycle.) However, it is important to observe that we know the specific linear combination of α_j's each $\beta_{h,i}$ is composed of as well as the concrete value of each $\delta_{h,i}$, since we know IV_h and fix some values for $\alpha_{61}, \ldots, \alpha_{76}$.

A stream bit $s_{h,k}$ ($k \in \{0, \ldots, 113\}$) is generated by summing up the output of the three majority functions and the rightmost bits of the registers $R1^{(h)}, R2^{(h)}$ and $R3^{(h)}$ (see Fig. 6.1). More precisely, in terms of the current state (k clock cycles after warm-up) of these registers the output bit can be written as

$$\begin{array}{lcl} s_{h,k} & = & maj(\beta_{h,12} \oplus \delta_{h,12}, \beta_{h,14} \oplus \delta_{h,14} \oplus 1, \beta_{h,15} \oplus \delta_{h,15}) \\ & \oplus & maj(\beta_{h,28} \oplus \delta_{h,28}, \beta_{h,32} \oplus \delta_{h,32}, \beta_{h,35} \oplus \delta_{h,35} \oplus 1) \\ & \oplus & maj(\beta_{h,54} \oplus \delta_{h,54} \oplus 1, \beta_{h,57} \oplus \delta_{h,57}, \beta_{h,59} \oplus \delta_{h,59}) \\ & \oplus & \beta_{h,18} \oplus \delta_{h,18} \oplus \beta_{h,40} \oplus \delta_{h,40} \oplus \beta_{h,63} \oplus \delta_{h,63}\,. \end{array} \tag{6.3}$$

It is important to note that due to the majority function, each output bit is a quadratic function in $\alpha_0, \ldots, \alpha_{60}$. More precisely, it has the general form

$$\begin{array}{lcl} s_{h,k} & = & \displaystyle\sum_{0 \le i < j \le 17} b_{i,j}\alpha_i\alpha_j \oplus \sum_{18 \le i < j \le 38} b_{i,j}\alpha_i\alpha_j \\ & \oplus & \displaystyle\sum_{39 \le i < j \le 60} b_{i,j}\alpha_i\alpha_j \oplus \sum_{0 \le i \le 60} a_i\alpha_i \oplus c, \end{array} \tag{6.4}$$

for some $b_{i,j}, a_i, c \in \{0, 1\}$.

To linearize above relations we simply replace each quadratic term $\alpha_i\alpha_j$ by a new variable $\gamma_{i,j}$. In this way we obtain $\frac{18 \cdot 17}{2} + \frac{21 \cdot 20}{2} + \frac{22 \cdot 21}{2} = 594$ new variables. Thus, each stream bit can be described by at most 655 variables (and a constant).

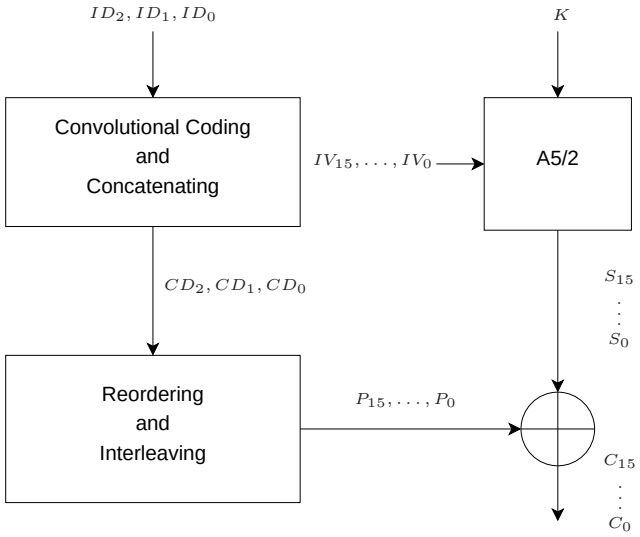


Figure 6.2: GSM coding, interleaving and A5/2 encryption

6.3.2 Attacking the Speech Channel

Now, we describe how a ciphertext-only attack using data from the speech traffic channel can be mounted. In the case of a ciphertext-only attack the direct output stream of A5/2 is not (partly) known. However, we have access to certain linear combinations of the output bits. This is due to the fact that A5/2 is encrypting linearly in the plaintext (as any synchronous stream cipher) and to the linear error-correction coding procedure that is performed *before* encryption. The applied error-correction procedure is however specific to the GSM channel (see [94] for details on GSM channel coding). How this procedure can be exploited in the case of the GSM control channel SDCCH/8 is sketched in [12]. We analyze how this can be done in the case of the full-rate speech traffic channel (TCH/FS and TCH/EFS) where a different interleaving procedure is used.

A multi-stage error-correction procedure performed to protect a 260-bit block of speech data produced by the speech coder against transmission errors This procedure increases the data size by adding redundant data in each stage and also reorders bits. We are interested in the last two stages of this procedure which are depicted in Figure 6.2. Here the 267-bit blocks ID_i containing some intermediate data are input to a so-called non-recursive binary convolutional encoder (of rate 1/2 with memory length 4 and constant length 5). The outputs of the convolutional coder are the 456-bit blocks CD_i. The function CC computed by the convolution encoder can

be described as follows:

$$CC : ID_i = (id_{i,0}, \ldots, id_{i,266}) \mapsto (cd_{i,0}, \ldots, cd_{i,455}) = CD_i\,, \text{ where}$$
$$cd_{i,j} = \begin{cases} id_{i,k} \oplus id_{i,k-3} \oplus id_{i,k-4}, & 0 \le j \le 377 \text{ and } j = 2k \\ id_{i,k} \oplus id_{i,k-1} \oplus id_{i,k-3} \oplus id_{i,k-4}, & 0 \le j \le 377 \text{ and } j = 2k+1 \\ id_{i,182+(j-378)}, & 378 \le j \le 455 \end{cases} \tag{6.5}$$

Note that the last 78 bits of ID_i are actually not protected by a convolutional code. Rather these bits are just copied unchanged to the tail of CD_i. The important property of the convolutional code bits of an arbitrary block CD_i (bits 0–377) – that is exploited later on – are the following linear dependencies that hold for $1 \le j \le 184$:

$$cd_{i,2j} \oplus cd_{i,2j+1} \oplus cd_{i,2j+2} \oplus cd_{i,2j+3} \oplus cd_{i,2j+6} \oplus cd_{i,2j+8} \oplus cd_{i,2j+9} = 0 \tag{6.6}$$

As we can see in Figure 6.2, the blocks CD_i are not directly encrypted. Prior to encryption, they are first reordered and interleaved "block diagonal". The result of the interleaving is a distribution of the reordered 456 bits of a given data block CD_i over the eight 114-bit blocks $P_{4i+0}, \ldots, P_{4i+7}$ using the even numbered bits of the first 4 blocks and odd numbered bits of the last 4 blocks. The reordered bits of the next data block CD_{i+1}, use the even numbered bits of the blocks $P_{4i+4}, \ldots, P_{4i+7}$ and the odd numbered bits of the blocks $P_{4i+8}, \ldots, P_{4i+11}$. The interleaving of CD_{i+2} and subsequent blocks is done analogously. So new data starts every 4th block and is distributed over 8 blocks. Considering the example in Figure 6.2, this means that each of the blocks $P_0, \ldots, P_3$ contains 57 bits of data from CD_0, $P_4, \ldots, P_7$ each contains 57 bits from CD_0 and 57 bits from CD_1, $P_8, \ldots, P_{11}$ each contains 57 bits from CD_1 and 57 bits from CD_2 and finally each of the blocks $P_{12}, \ldots, P_{15}$ contains 57 bits of CD_2.

More precisely, the following function can be used to describe the reordering and interleaving of data blocks:

$$\begin{gathered} f : \mathbb{N} \times \{0, \ldots, 455\} \to \mathbb{N} \times \{0, \ldots, 113\} \\ (i, j) \mapsto (4i + (j \bmod 8),\ 2(49j \bmod 57) + (j \bmod 8) \operatorname{div} 4) \end{gathered} \tag{6.7}$$

Then we have the following relation between the bits CD_i and the output blocks $P_{(4i+0)}, \ldots, P_{(4i+7)}$:

$$cd_{i,j} = p_{f(i,j)}, \tag{6.8}$$

where the right-hand side denotes the bit with index $f(i, j)$ belonging to block $P_{(4i+(j \bmod 8))}$.

A 114-bit block P_i produced by the interleaver is then encrypted by computing the bitwise XOR with the output stream S_i resulting in the ciphertext frame C_i. The linear dependencies of the convolutional code bits seen in Equation 6.6 also propagate to the ciphertext because the encryption is linear in the plaintext and the

keystream. So taking the interleaving and reordering into account, we can exploit this property to obtain equations of the form

$$\begin{aligned} & c_{f(i,2j)} \oplus c_{f(i,2j+1)} \oplus c_{f(i,2j+2)} \oplus c_{f(i,2j+3)} \oplus c_{f(i,2j+6)} \oplus c_{f(i,2j+8)} \oplus c_{f(i,2j+9)} \oplus \\ & s_{f(i,2j)} \oplus s_{f(i,2j+1)} \oplus s_{f(i,2j+2)} \oplus s_{f(i,2j+3)} \oplus s_{f(i,2j+6)} \oplus s_{f(i,2j+8)} \oplus s_{f(i,2j+9)} \\ & = \\ & p_{f(i,2j)} \oplus p_{f(i,2j+1)} \oplus p_{f(i,2j+2)} \oplus p_{f(i,2j+3)} \oplus p_{f(i,2j+6)} \oplus p_{f(i,2j+8)} \oplus p_{f(i,2j+9)} \\ & = \\ & cd_{i,2j} \oplus cd_{i,2j+1} \oplus cd_{i,2j+2} \oplus cd_{i,2j+3} \oplus cd_{i,2j+6} \oplus cd_{i,2j+8} \oplus cd_{i,2j+9} = 0 \end{aligned} \tag{6.9}$$

for $0 \leq j \leq 184$. It is important to note that the ciphertext and stream bits in the above equation do not belong to a *single* ciphertext block respectively stream. Rather, for fixed i eight consecutive ciphertext blocks and corresponding streams are involved in the 185 equations. A single equation involves bits from 5 different blocks and streams. These effects are due to the interleaving and reordering (and make an efficient hardware implementation somewhat tricky).

Hence, given 16 consecutive ciphertext blocks we can setup LSEs with 555 equations and 655 unknowns using the results from the previous section[2]. Though the LSEs are underdetermined, we found out by experiments (similar to [12]) that this number of equations is always sufficient to determine the 61 original linear variables α using Gaussian elimination. Having determined the values of these variables, merely the consistency with the quadratic equations needs to be checked to identify the correct secret initial state.

6.4 Hardware Acceleration of Attacks on A5/2

Our hardware architecture is sketched in Figure 6.3. It accepts 16 ciphertext frames and the 16 corresponding IVs as input. The hardware calculates and outputs the recovered 77-bit state $\alpha_0, \ldots, \alpha_{76}$.

The given ciphertext frames and IVs are stored in the Ciphertext Module (CM). Each of the three Equation Generators (EGs) generates 185 linear equations with the secret state bits α_i ($0 \leq i \leq 60$) as variables (cf. (6.9)). The EGs receive the required IVs and ciphertext bits from the CM. A generated equation is passed to the buffer of the LSE Solver. This buffer is needed because the LSE Solver accepts only one equation per clock cycle, but the three EGs produce their equations simultaneously. After the LSE Solver is filled with 555 equations, it proceeds to the solving step and produces a candidate for the secret state. The secret state candidate is sent from the LSE Solver to the Key Tester (KT) that verifies whether the correct state has been found. This verification process is done in parallel to the determination of a

[2]Assuming that the first 8 blocks contain the encrypted data of a whole convolutional code block.

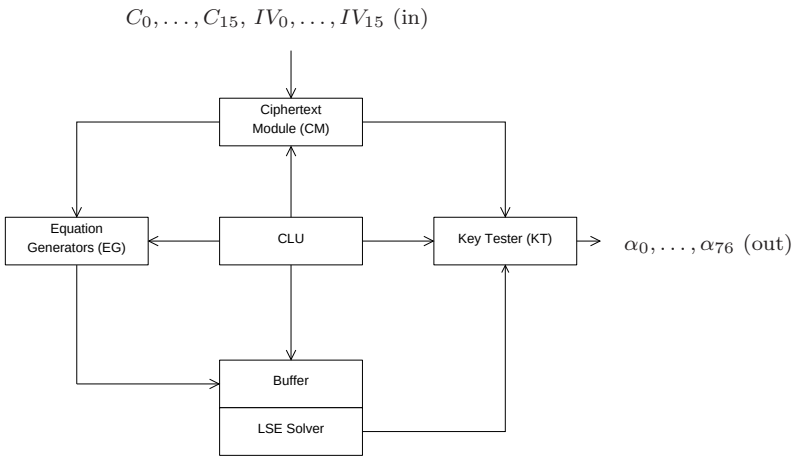


Figure 6.3: Overview of the proposed architecture

new candidate. More precisely, while equations for the j-th candidate are generated by the EGs the $(j-1)$-th candidate is tested by the KT. All processes are controlled by the Control Logic Unit (CLU) that performs synchronization and clocking for the CM, EGs, the LSE Solver, and the KT. Its main task is to ensure that the right stream and ciphertext bits are combined (within the EGs and also the KT) to form the desired equations as described in Section 6.3.2 in (6.9).

6.4.1 Equation Generators

Three Equation Generators (EGs) are used to generate the system of linear equations for the LSE Solver. Each EG is associated with one of the 3 convolutional code blocks $CD_0, \ldots, CD_2$ whose data is spread - due to interleaving - over 8 of the 16 given ciphertext blocks C_h (cf. Section 6.3.2). By means of the CM an EG has access to the required 8 ciphertext blocks and the corresponding IVs and generates 185 equations from this data.

As shown in Figure 6.4, an EG consists of eight Stream Generators (SGs) and one Stream Combiner (SC) which are all controlled by the CLU. Each of the eight SG is associated with one of the eight ciphertext frames C_h related to its EG. More precisely, Stream Generator SG_j $(0 \leq j \leq 7)$ belonging to Equation Generator EG_i $(0 \leq i \leq 2)$ is associated with frame $C_{(4i+j)}$. The SG for a frame C_h consists of an expanded A5/2 engine and can produce linearized terms for the 114 stream bits $s_{h,k}$ (cf. (6.4)). For instance, SG_0 in EG_1 is associated with C_4 and is able to generate linear terms for the stream bits $s_{4,0}, \ldots s_{4,113}$. Each EG also contains another type of component, the Stream Combiner, that takes care of adding the right stream bit terms and the right ciphertext bits together in order to get the final equations that

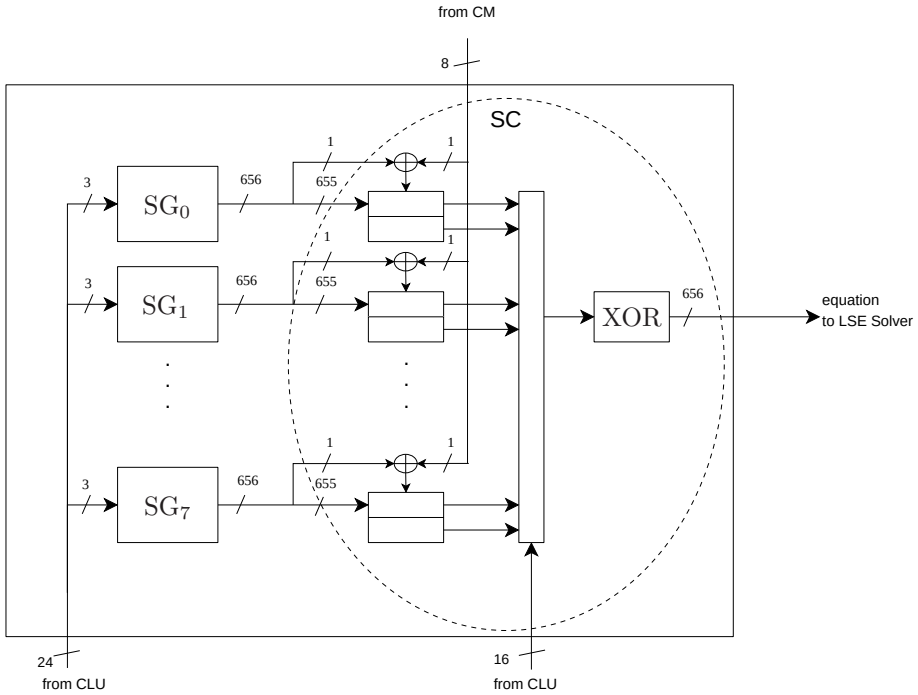


Figure 6.4: Equation Generator with its components Stream Generator (SG) and Stream Combiner (SC)

are then passed to the LSE Solver.

Stream Generator. The Stream Generator (SG) unit consists of an A5/2 engine where the states of the LFSRs R1, R2 and R3 are represented by vectors of α_i's instead of single bits. We implemented the Ri's as *vector LFSRs* instead of standard scalar LFSRs to obtain linear expressions in the variables α_i (every LFSR binary addition and shift operation is applied to a linear combination of α_i's). Figure 6.5 shows an example of this representation for R1 right after the initialization phase. Each column vector gives the dependence of the corresponding bit of the simple LFSR on the α_i's and a constant (equation (6.1) describes exactly the same state of the vector LFSR). Hence the rows of the matrix indicate the dependence on the corresponding α_i while the columns indicate the position in the LFSR. The row at the bottom corresponds to the constants $\sigma_{h,i}$. Right after the initialization phase, each column (ignoring the bottom row) only contains a single 1, because before warm-up each position of each *Ri* depends only on a single α_i (cf. (6.1)). The only exception are the three positions in R1 through R3 that are set to one. Here the respective positions do not depend on any α_i but the respective constant part is 1. Note that no clock cycles need to be wasted to actually perform the initialization

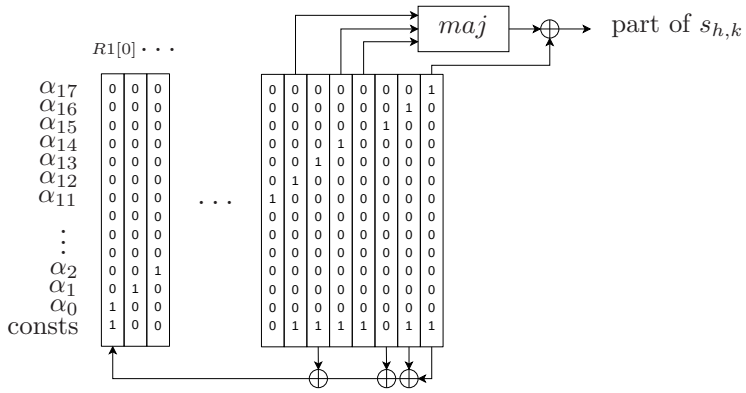


Figure 6.5: Detailed view of R1 represented within a SG after initialization

phase for vector LFSRs, since we can precalculate the IV's influence on each LFSR position

Each SG performs the warm-up phase where its vector LFSRs are clocked 99 times. Every time a vector LFSR is clocked (forward), all columns are shifted one position to the right, the last column is dropped and the first column is calculated as an XOR of the columns according to the feedback term. After warm-up, the CLU can query one of its 114 outputs. To produce this output, the SG is clocked as many times as necessary to reach the desired linear expression for $s_{h,k}$. An SG can be clocked forward as well as backward[3], resulting in an average of 36 clock cycles required for generating one output equation. The output is generated by XORing the result of the majority function as described in (6.4).

The majority function performs a pairwise "multiplication" of all three input vectors and binary adds the intermediate results and the vector that directly enters the equation (e.g, R1[18] in Figure 6.5). The multiplication of two column vectors is done by binary multiplying each element of one vector with each element of the other vector. The resulting term for one key bit $s_{h,k}$ is linearized by assigning each quadratic variable to a new variable (represented by a "new" signal line). We implemented the multiplication of (6.4) to be performed in one clock cycle. Instead of rotating a number of registers several times, we directly tap and combine all input bits of these registers that are required for the computation of a specific output bit.

Since the domain of each instance of the majority function is restricted to one register, its action is local and one obtains three smaller quadratic equations with disjunct variables (except for the constant term) before the final XOR. That is, the results of the different vector LFSRs do not have to be XORed and can be directly output (the first 655 data lines). The only operation one has to perform to compute

[3]Figure 6.5 depicts only the control logic for forward clocking. For simplicity reasons we also omitted certain other control and data signals in the figure.

this XOR is to add the three constant bits (the last single-bit output of each SG). Note that one does to have to linearize the local quadratic equations, since we already use the linearized form to represent quadratic equations in hardware (each linear or quadratic term is represented by one bit).

Each of the 8 SGs accepts 3 control signals from the CLU indicating the clocking direction (forward or backward), the stop-go command (as the vector LFSRs belonging to different SGs need to be clocked a different number of times), and the initialization command (increment R4 and perform warm-up).

Stream Combiner. The Stream Combiner (SC) combines the results of the SGs with the right ciphertext bits from the CM to produce the equations for the LSE Solver. More precisely, it works as follows: the output of an SG are 656 signals representing a certain stream bit $s_{h,k}$. The signal representing the constant value c of $s_{h,k}$ (cf. (6.4)) is then XORed with the respective ciphertext bit $c_{h,k}$ provided by the CM. By having a closer look at (6.9) and the involved function f, we can see that this 656-bit result is sometimes needed for the generation of two consecutive equations. Moreover, note that sometimes also the current and the previous result of an SG are required at the same time to build an equation. To this end the result of an SG is buffered in the SC (see Figure 6.5). A signal of the CLU is used to decide which of the 8 previous and 8 current results are XORed together. The resulting equation is now passed as new equation to the buffer of the LSE Solver.

6.4.2 Ciphertext Module

The Ciphertext Modul (CM) stores the 16 ciphertext blocks and IVs and provides them to the SCs and the KT in the required order. It consists of 24 memory blocks for storing ciphertexts and 16 memory blocks for storing the IVs. The content of the ciphertext memory blocks can be cyclicly shifted in both directions. The ciphertexts $C_0, \ldots, C_{15}$ are initially stored in the bit order as they are recorded from air. $C_0, \ldots, C_7$ is put in the first 8 memory blocks, $C_4, \ldots, C_{11}$ is put in the next 8 memory blocks and $C_8, \ldots, C_{15}$ is put in the last 8 blocks.

Each EG and the KT has parallel access to the 8 required IVs. Each of the SCs needs only access to 8 of the 24 ciphertext memory blocks. More precisely, the SC belonging to EG_i is provided with the first bit of memory block $4i + 0$ to $4i + 7$ respectively (i.e, the positions where the bits $c_{4i+j,0}$ are initially stored). The content of these memory blocks needs to be rotated in the same way as the vector LFSRs within the 8 SGs of EG_i. To this end the CM receives the same control signals from the CLU as the SGs. Finally, the KT accesses the first bit of the ciphertext memory blocks 0 to 7 respectively, i.e., the same bits as EG_0.

6.4.3 LSE Solver

The LSE Solver is controlled by the CLU. Each time an equation in an Equation Generator is ready, the LSE Solver obtains a signal for loading the equation. As all orders have been processed and all equations loaded into the LSE Solver, it receives a command from the CLU to start Gaussian elimination. When the LSE Solver is ready, it writes the 61-bit result into a buffer. After this it signals that a solution is ready. The CLU informs the Key Tester that the secret state candidate is in the buffer. It is then read out by the Key Tester. We decided to use the SMITH architecture presented in [46, 47] to realize the LSE Solver module.

SMITH. The SMITH architecture implements a hardware-optimized variant of the Gauss-Jordan[4] algorithm over $\mathbb{F}_2$. The architecture is described in [46] for LSEs of dimension $m \times n$ where $m \geq n$. Its average running time for systems of dimension $n \times n$ with uniformly distributed coefficients is about $2n$ (clock cycles) as opposed to about $\frac{1}{4}n^3$ in software.

Though SMITH was not originally designed for performing Gauss-Jordan on underdetermined LSEs ($m < n$), using minor modifications it can also handle this type of LSEs which is required for our purposes. Due to page limitations, we omit describing these straightforward adaptions. In the remainder of this section, we sketch a simple enhancement of the architecture that significantly reduces the total number of clock cycles (in our case) for doing Gauss-Jordan elimination.

The required number of clock cycles is determined by two operations which are applied $\min(m, n) = m$ times: pivoting and elimination. Pivoting roughly means the search for a non-zero element in a certain column of the respective coefficient matrix which is then used during the elimination operation to zero-out all other "1" entries in this column. While the elimination operations consume a fixed amount of m clock cycles in total, a variable number of clock cycles for pivoting is required. This number highly depends on the characteristics of the coefficient matrix. Roughly speaking, if the matrix columns contain many zero entries pivoting requires many clock cycles and may dominate the total costs. In our case, we initially have dense matrices that however contain many linearly dependent row vectors resulting in many zero columns while the algorithm proceeds. More precisely, our experiments show that each of our matrices contains about 160 to 190 linearly dependent equations. It is important to note that if if the pivoting is to be applied to a zero column, no pivoting and no subsequent elimination is actually required. Thus, by performing zero column detection (in an efficient manner), we can save these clock cycles and proceed immediately with the next column. Since, in the case of the SMITH architecture the logic for pivoting is physically located in a *single* column (the 1st column), we could

[4] In this version of Gaussian elimination the backward substitution steps are combined with the elimination steps. In this way one immediately obtains the solution vector without doing any post-processing.

efficiently realize this detection by computing the OR over all entries of this single column. Using this simple adaption, the pivoting operations for the whole matrix consume about 4000-4500 cycles (instead of more than 60000). Thus, a solution is computed after about 5000 cycles.

6.4.4 Key Tester

The Key Tester (KT) receives the output of the LSE Solver, i.e. the secret state candidate, and checks its correctness. The KT is built-up as a modified EG. The determined candidate is written into the SG-engines of the KT, which are normal A5/2 engines that can be clocked in both directions. Hence the size of this modified SGs is much smaller and they produce single bits as output. For the verification of a candidate, the output bits $s_{h,k}$ generated by the SGs are combined with the ciphertext bits according to (6.9) like it is done within a regular EG. If all resulting XOR-sums are equal to 0, the correct secret state has been found and is written out.

6.4.5 Control Logic Unit

The Control Logic Unit (CLU) controls all other components and manages the data flow. It ensures that the right stream bit expressions are generated, combined with the ciphertext bits and passed to the LSE Solver. Once the LSE Solver is filled with 555 equations, it stops the EGs and starts the LSE Solver. When the LSE is solved, the candidate is passed to the KT. The KT is operated in parallel to the generation of the new LSE.

The CLU generates the same sequence of 24-bit control signals for each of the three EGs and the KT (cf. Figure 6.4). Remember that each SG belonging to an EG receives 3 of these signals. They determine the direction in which the engine is clocked, whether the engine is clocked at all and whether the R4 register need to be increased (to generate equations for a new candidate). The generation of these signals can be immediately derived from (6.9). More precisely, the CLU internally generates an *order* for each of the 185 equations an EG should produce. From such an order the required signals can be easily derived. The orders for the first 21 equations are shown in Table 6.1. Each order thereby consists of 7 pairs (fr_i, cl_i) $(0 \leq i \leq 6)$. The number fr_i equals the index of a ciphertext/key-stream block modulo 8 (cf. (6.9)) required in an equation. So this number addresses one of the 8 SGs belonging to an EG. The number cl_i (which can be negative) is the relative position of the required bit within the fr_i-th ciphertext/key-stream block. "Relative" means that this position is given relatively to the position of the bit of this block that was required just before. This number can be used to signal how often and in which direction an SG should be clocked. Considering the columns of Table 6.1, we see that these pairs occur (almost) periodically. So orders can be generated easily in hardware.

Table 6.1: Orders required for the first 21 equations ($(j, 0)$ means that the current output of Stream Generator j is needed)

Equation	Orders						
0	(0, 0),	(0, 100),	(1, 98),	(1, -14),	(2, 82),	(3, 66),	(6, 19)
1	(0, 0),	(2, 0),	(2, -14),	(3, 0),	(3, -14),	(4, 51),	(5, 35)
2	(2, 0),	(4, 0),	(4, -14),	(5, 0),	(5, -14),	(6, 0),	(7, 3)
3	(0, 0),	(1, 0),	(4, 0),	(6, 0),	(6, -14),	(7, 0),	(7, 100)
4	(0, 0),	(0, -14),	(1, 0),	(1, -14),	(2, 0),	(3, 0),	(6, 0)
5	(0, 0),	(2, 0),	(2, -14),	(3, 0),	(3, -14),	(4, 0),	(5, 0)
6	(2, 0),	(4, 0),	(4, -14),	(5, 0),	(5, -14),	(6, 0),	(7, 0)
7	(0, 0),	(1, 0),	(4, 0),	(6, 0),	(6, 100),	(7, 0),	(7, -14)
8	(0, 0),	(0, -14),	(1, 0),	(1, -14),	(2, 0),	(3, 0),	(6, 0)
9	(0, 0),	(2, 0),	(2, -14),	(3, 0),	(3, -14),	(4, 0),	(5, 0)
10	(2, 0),	(4, 0),	(4, -14),	(5, 0),	(5, 100),	(6, 0),	(7, 0)
11	(0, 0),	(1, 0),	(4, 0),	(6, 0),	(6, -14),	(7, 0),	(7, -14)
12	(0, 0),	(0, -14),	(1, 0),	(1, -14),	(2, 0),	(3, 0),	(6, 0)
13	(0, 0),	(2, 0),	(2, -14),	(3, 0),	(3, -14),	(4, 0),	(5, 0)
14	(2, 0),	(4, 0),	(4, 100),	(5, 0),	(5, -14),	(6, 0),	(7, 0)
15	(0, 0),	(1, 0),	(4, 0),	(6, 0),	(6, -14),	(7, 0),	(7, -14)
16	(0, 0),	(0, -14),	(1, 0),	(1, -14),	(2, 0),	(3, 0),	(6, 0)
17	(0, 0),	(2, 0),	(2, -14),	(3, 0),	(3, 100),	(4, 0),	(5, 0)
18	(2, 0),	(4, 0),	(4, -14),	(5, 0),	(5, -14),	(6, 0),	(7, 0)
19	(0, 0),	(1, 0),	(4, 0),	(6, 0),	(6, -14),	(7, 0),	(7, -14)
20	(0, 0),	(0, -14),	(1, 0),	(1, -14),	(2, 0),	(3, 0),	(6, 0)

Besides the three control signals for each SG, the CLU has to produce a 16-bit mask to control which outputs of the SGs are XORed within an SC (cf. Figure 6.4). As can be derived from Table 6.1, only 7 bits of the mask are simultaneously set to 1. Finally, the CLU also "orders" the needed ciphertext bits from the CM which is done in the same way as stream bits are "ordered" from the SGs.

Operating procedure. During the setup of our attack engine, all components are being initialized and 16 ciphertext frames and 16 corresponding IVs are read into the CM. The R4 registers of the GEs are set to the initialization value 0.

After the initialization the equations are generated, solved, and tested for all different possible states of R4, until the right state is found. Hence the following steps are performed 2^{15} times on average:

1. The registers R4 are incremented and the warm-up is executed in the SGs and in the KT. The SGs are now ready to generate the linearized terms for the stream bits $s_{h,k}$ when queried.

2. The LSE Solver gets filled with 555 equations. The CLU queries each of the three EGs 185 times to receive these equations. The CLU plays an important role in this, because it controls each SG to provide the right $s_{h,k}$ terms, which

are then combined by the SCs and passed to the buffer of the LSE Solver. The SGs inside the EGs need to be clocked 36 times on average to produce the necessary terms.

3. Once all equations are generated, the LSE Solver is started. It takes roughly 5000 cycles until the result is calculated.
4. The determined candidate is fed into the KT and the warm-up is executed.
5. The CLU queries the KT 185 times to generate the output bits. If all parity checks in the KT succeed, the recovered 77-bit state is passed to the output.

Since the KT and the EGs have the same components the warm-up and equation generation of both can be performed in parallel. Hence, steps 1 and 4 as well as 2 and 5 are performed in parallel. Furthermore, setup and warm-up (steps 1 & 4) for the new state candidate can be performed while the LSE Solver is determining the previous candidate (step 3).

6.5 FGPA Results and ASIC Estimates

Due to the size of the architecture, an ASIC realization seems most realistic. We decided to keep the operating speed at 256 MHz for the LSE Solver to maintain a decent power consumption at still reasonable performance time. Since the remaining components are smaller than the LSE Solver and there are periods where those components are idle, they are clocked at twice the speed (512MHz). This way the LSE Solver still accounts for two thirds of the overall power consumption and heat development. At these clock rates one key is recovered on average in about 1 second.

To evaluate the requirements on chip size and average power consumption, we implemented our design in VHDL and synthesized it using the Virtual Silicon (VST) standard cell library based on the UMC L180 0.18μ 1P6M Logic process. We used Synopsys Design Compiler version Y-2006.06 for synthesis and estimation of power consumption. Mentor Graphics Modelsim SE was used for simulation. Due to the huge size of the whole system, simulation and synthesis were done component-wise. A synthesis of the whole design should further decrease the needed area.

All critical components were implemented and synthesized. Table 6.2 shows the synthesis results. For each component the area it needs is given as well as the consumed average power. The area is given in gate equivalents (GE). One GE is equal to the area needed by one NAND-gate in the appropriate process. Power is given in μW. The first column shows at which clock frequency a component is operated. For few uncritical components like the Stream Combiner and the Ciphertext Module, area and power consumption were estimated rather than synthesized. Conservative assumptions on the needed number of flip-flops and area for control-logic were transformed into area and power estimations. Estimated components are indicated by *.

Table 6.2: Simulation results for the implementation sorted by components used. SC, CM, and CLU data stems from synthesis estimations. The figures for the entire design have also been estimated

Component name	Area requirements	Power consumption
Stream Generator, at 512 MHz	28.9 kGE	129.9 mW
Key Tester, at 512 MHz	0.7 kGE	2.9 mW
LSE Solver, at 256 MHz	8,205.3 kGE	8,360.8 mW
Stream Combiner, at 512 MHz	95.5 kGE	431.8 mW
Ciphertext Module , at 512 MHz	16.6 kGE	27.3 mW
Control Logic, at 512 MHz	4.6 kGE	20.1 mW
Entire Architecture, at 256/512 MHz	9,316.8 kGE	12,833.7 mW

Obviously solving linear equation systems of requires the most considerable portion of chip area (roughly 90% of the entire design area requirements). The remaining design parts account for approx. one third of the whole device power consumption which is because of the higher operating frequency of the corresponding modules. Some modules appear several times in the hardware architecture. The resulting architecture requires approx. 9.3×10^6 GEs and needs approx. 12.8 W for its operation which is totally acceptable in the view of the state-of-the-art semiconductor technology.

Chapter 7

Collision Attacks Using Side-Channel Leakage

This chapter contains multiple contributions to the methods of side-channel collisions attacks. All techniques are thoroughly studied for the case of attacking AES-128. Many of these techniques apply directly to other cryptographic algorithms such as block ciphers and message authentication codes. Some methods need to be adapted to be applicable to other constructions. We illustrate such an adaptation of the standard techniques in the case of Alpha-MAC which is an AES-based message authentication code.

The most important contributions of this chapter are as follows: notion of generalized internal collisions, linear collision-based key recovery, algebraic collision-based key recovery as well as numerous methods for robust collision detection including several multiple-differential collision detection techniques. Techniques are evaluated using both simulated and real-world side-channel traces.

Collision attacks combined with linear and algebraic key recovery represent a new powerful cryptanalytic method for block ciphers, which is based on side-channel leakage and allows for low measurement counts needed for a successful key recovery in case of AES. As opposed to many other side-channel attacks, these techniques are essentially based on the internal structure of the attacked cryptographic algorithm, namely, on the algebraic properties of AES.

On the collision detection side, the probability distributions of Euclidean distance for collisions and non-collisions are derived. On this basis, a statistical framework for finding the instances of side-channel traces leaking most key information in collision attacks is proposed. Moreover, two efficient multiple-differential methods to detect collisions in the presence of strong noise are proposed - binary and ternary voting. We refer to this combination of the collision detection methods and collision-based cryptanalytic techniques as multiple-differential collision attacks (MDCA).

This chapter is mainly based on the independent research of the author published in [37] and [39] as well as on his joint work with Ilya Kizhvatov and Andrey Pyshkin partially published in [43] and on his joint work with Alex Biryukov, Dmitry Khovratovich and Timo Kasper published in [26].

7.1 Introduction

The major motivation of this chapter is to develop a framework minimizing the number of online measurements needed for a successful key recovery in a real-world noisy environment. This is to a certain extent equivalent to extracting a maximum amount of key information from the given side-channel signal, which is the central question of side-channel cryptanalysis. In practice, this setting is important where the attacker has very restricted access to the device due to organizational policies or where only few cryptographic operations with the same key are allowed, which is used as a side-channel countermeasure in some real-world systems. An independent line of motivation we pursue is to come up with an efficient and practical alternative

to such well-known side-channel techniques as differential power analysis (DPA) [153] and template attacks [60], [167] which would be free of their main natural limitations: Dependency on a certain leakage model for DPA and the necessity of thoroughly characterizing the attacked device for template attacks.

Side-channel *collision attacks* are a well-suited base for the solution of these problems due to the inherently low numbers of needed measurements, the absence of any concrete leakage model, and the possibility to build collision templates without detailed knowledge of the target, which will be shown in this chapter.

The idea of collision attacks. Side-channel attacks have become mainstream since their first publication in [152]. Differential power analysis (DPA) [153] and correlation power analysis (CPA) [53], a generalization of DPA, are probably the most wide-spread practical attacks on numerous cryptographic embedded systems such as smart-card microcontrollers [181] and dedicated lightweight ASICs [202].

Collision attacks represent another class of side-channel attack techniques being essentially based on the cryptanalytic properties of attacked cryptographic algorithms. Collision attacks on block ciphers were proposed in [233] for DES. The idea is due to Hans Dobbertin and was also discussed in the early work [257]. Since then there has been quite a bit of research in this area: [165] improves the collision attack on DES, [232] applies the technique to AES, [27] combines collision attacks on AES with differential cryptanalysis to overcome several masked rounds.

An internal collision, as defined in [233] and [232], occurs, if a function f within a cryptographic algorithm processes different input arguments, but returns an equal output argument. As applied to AES, the work [232] considers the column transforms of the MixColumns operation of the first AES round as colliding functions f. To detect collisions, power consumption curves bytewisely corresponding to separate S-box operations in the second round at a certain internal state position after the key addition are compared.

Basic collision attacks extended to Alpha-MAC. The idea of basic collision attacks is also applicable to other cryptographic constructions such as message authentication codes. We illustrate this by extending the basic collision attack from AES to the AES-based MAC construction Alpha-MAC. First, the internal state of the unkeyed part of Alpha-MAC is recovered using side-channel collision attacks. Then this information is used to perform selective message-forgery. The practicability of these attacks is illustrated for an Alpha-MAC implementation on a PIC controller.

Generalized collisions and linear key recovery. The major idea of improved collision attacks on AES is that one can detect equal inputs to various S-boxes by comparing the corresponding power consumption curves. This turns out to

be possible not only for the outputs of the same function f: Using this technique, it can be possible to detect whether two inputs to the AES S-box are equal within the same AES execution as well as for different AES runs. Each of such collisions can be considered as a (generally) non-linear equation over $\mathbb{F}_2^8$. The set of all detected collisions corresponds to a system of non-linear equations with respect to the key bytes. The notion of a generalized collision is introduced that occurs if two S-boxes at some arbitrary positions of some arbitrary rounds process an equal byte value within several runs.

This chapter first explores the question of how to solve these equations linearly. To be able to linearize, we restrict our consideration to the first round. As DPA, this attack also works in the known-plaintext model, while the attack in [232] is applicable in the chosen-plaintext scenario only. However, collision attacks do need one plaintext-ciphertext pair for choosing the correct key from a set of key candidates in the offline stage. Note that this input-output pair does not have to be one of the those for which the measurements have been performed. We use both theoretical and experimental tools for estimating the efficiency of our attacks. Linear systems of equations are rewritten in terms of associated undirected graphs. As the resulting equation systems never possess the full rank, combinatorial methods are applied to solve these systems. The complexity of these methods can be analyzed through connected components of those graphs. The expected number of edges in such a graph is computed theoretically. The number of connected components, which defines the overall complexity of the offline attack stage, is estimated using thorough computer simulations for the numbers of edges obtained theoretically.

Nonlinear collisions and algebraic collision-based key recovery. Additionally to linear collisions, this chapter considers nonlinear collisions that are defined as generalized collisions comprising several rounds. They deliver extra information contained in further AES rounds. Each such collision can be treated as a nonlinear equation over a finite field. The set of all detected collisions corresponds to a system of nonlinear equations with respect to the key. Such a system can be solved using techniques closely related to the algebraic cryptanalysis of AES with a reduced number of rounds, which are referred to as algebraic collision-based key recovery. The attacks require a very low number of inputs for the key recovery procedure to succeed with a high probability and within a feasible time span.

Methods of basic collision detection. For the basic collision detection (comparing two traces), the Euclidean distance between two real-valued vectors is used. We obtain probability distributions of this statistic in the univariate Gaussian noise model. We show that for large numbers of points in the side-channel trace these two Euclidean distances can be approximated by normal distributions with different parameters. This allows us to define a statistical metric for the time instants of the

trace leaking most information for collision detection. It turns out that these points may be quite different from those leaking key information in standard DPA.

Multiple-differential collision detection. Two collision detection techniques are proposed called *binary* and *ternary voting*. We refer to the combination of these statistical collision detection methods and cryptanalytic collision-based key recovery as *multiple-differential collision attacks (MDCA)*. We apply MDCA to a hardware implementation of AES for a wide range of noise amplitudes using advanced power consumption simulation.

MDCA works in the two scenarios: where profiling is either allowed (ternary voting) or not allowed (ternary voting without profiling and binary voting). Note that the notion of profiling for these collision detection techniques is different from that for template attacks [60], [5]. While template attacks require detailed knowledge of the implementation in the profiling stage, the only information needed in the profiling stage of the collision detection methods is the time interval when an S-box is executed.

A further advantage of the proposed collision detection techniques combined with the linear collision-based key recovery is that that they work with secret S-boxes. Moreover, ternary voting with profiling also requires neither keys nor inputs/outputs to be known in the profiling stage. However, as already mentioned, the attacker has to know when the S-boxes are executed within the implementation.

Practical results. Combining these improvements, we achieve a considerable reduction of the number of online measurements needed for a successful key recovery. We implemented the attacks for an Atmel AVR ATmega16 microcontroller. The practical results can be found in Table 7.1. In a version of the attack, we additionally use collisions from ternary voting, a multiple-differential collision detection technique. Neither plaintexts, ciphertexts nor keys have to be known in the profiling stage.

This indicates that the algebraic collision attacks on AES without profiling are superior to standard DPA in terms of the number of measurements needed. Rather surprisingly, the efficiency of our collision techniques *without profiling* is comparable to the stochastic methods [167] *with profiling* (one of best known template-based attacks) for low numbers of profiling curves. Moreover, if profiling is allowed for collision attacks, the number of online measurements can be further reduced. Note that all the implemented collision techniques (both with and without profiling) do use the knowledge of the time instances leaking most information.

Organization of the chapter. The remainder of the chapter is organized as follows. Section 7.2 provides preliminary information on collision attacks and introduces some notations. Section 7.3 outlines the basic collision attack on AES. In

Table 7.1: Summary of results: Hamming-weight based CPA, basic collision attack (on our ATmega16 AES implementation) without profiling and stochastic methods with profiling (on an ATM163 AES implementation [167]) vs. collision attacks based on FL-collisions with and without profiling for $C_{\text{offline}} \leq 2^{40}$ ($\mathcal{P}$ – success probability, C_{online} – number of online measurements, $C_{\text{profiling}}$ – number of profiling measurements, C_{offline} – number of offline operations for key recovery)

	$\mathcal{P}$	C_{online}	$C_{\text{profiling}}$
Hamming weight based CPA	0.8	61	0
Basic collision attack [232]	0.85	300	0
Stochastic methods for ATM163 [167]	0.82	10	200
FL-collisions, this chapter	0.76	16	0
FL-collisions, this chapter	0.72	12	625

Section 7.4 the principle of basic collision attacks is applied to the AES-based MAC construction Alpha-MAC. Section 7.5 rigorously introduces the notion of a generalized internal collision for AES as well as specifies and analyzes linear collision-based key recovery. Section 7.6 deals with nonlinear collisions and algebraic collision-based key recovery. Section 7.7 introduces some techniques to improve the quality of the Euclidean distance based binary comparison test which is used to detect collisions. Section 7.8 presents the multiple-differential collision detection techniques and theoretically investigates some of their properties. Section 7.9 characterizes the underlying least-square based binary comparison test for an AES hardware implementation, applies MDCA to this implementation and compares the results to DPA. In Section 7.10 we discuss the technical framework and practical feasibility of our attacks on a real-world hardware platform. We conclude in Section 7.11.

7.2 Preliminaries

7.2.1 Basic Notation

In this chapter we perform our collision attacks at the example of AES and AES-based constructions. In the first part of the chapter, the following notation is used. $P = (p_{ij})$ with $i, j \in \{0, \ldots, 3\}$, $p_{ij} \in \mathbb{F}_2^8$, and $K = (k_{ij})$ with $i, j \in \{0, \ldots, 3\}$, $k_{ij} \in \mathbb{F}_2^8$, denote the plaintext block and the first subkey, respectively, of AES. P and K are 4×4 byte matrices with i and j being row and column numbers, respectively.

In the second part of the chapter, we will be using a slightly different notation to represent the variables of AES. $K = \{k_j\}_{j=1}^{16}$, $k_j \in \mathbb{F}_2^8$, is the 16-byte user-supplied key (the initial AES subkey). $X = \{x_j\}_{j=1}^{16}$, $Y = \{y_j\}_{j=1}^{16}$ and $Z = \{z_j\}_{j=1}^{16}$, $x_j, y_j, z_j \in \mathbb{F}_2^8$ are the first, next to the last and last 16-byte AES subkeys, respec-

tively. AES plaintexts are denoted by $P = \{p_j\}_{j=1}^{16}$, $p_j \in \mathbb{F}_2^8$, and ciphertexts by $C = \{c_j\}_{j=1}^{16}$, $c_j \in \mathbb{F}_2^8$.

Collision-based key recovery methods for AES are mainly parametrized by the number γ of random plaintexts and/or ciphertexts needed to obtain the cryptographic key with success probability $\mathcal{P}$. In our collision attacks, γ can be chosen between 4 and 20 in the majority of cases. We are interested in success probabilities $\mathcal{P} \geq 0.5$. Further notations and refinements of the notations mentioned here will be introduced whenever needed.

7.2.2 Attack Flows

All collision attacks have two stages: an *online stage*, where measurements on the target device implementing the attacked cryptographic algorithm are performed, and an *offline stage*, where the cryptographic key is obtained from the traces acquired in the online stage. Additionally, a collision attack can be enhanced to have a *profiling stage*, where some profiling traces are obtained from an implementation of the attacked cryptographic algorithm.

In the *online stage*, some random known 16-byte plaintexts P_i are sent to the attacked device implementing AES, where they are added with the first 16-byte subkey K. Then each of the 16 bytes of the result is processed by the AES S-box. An *online trace* $T = \{\tau_j\}_{j=1}^{16}$, $\tau_j = (\tau_{j,1}, \ldots, \tau_{j,l}) \in \mathbb{R}^l$ is acquired by the measurement equipment (e.g. they can contain such side-channel parameters as power consumption or electromagnetic radiation). Each T corresponds to its own AES execution and its specific AES round number in this execution.

In the optional *profiling stage*, the device is triggered to perform a number of cryptographic operations with some unknown profiling inputs for some unknown keys. The *profiling traces* are acquired by the measurement equipment. The profiling stage takes place before the online stage and can be reused by several attacks on the same implementation.

The *offline stage* recovers the key. This occurs in two steps. First, collisions are detected in the online traces by means of signal processing. The collision detection with profiling additionally uses the profiling traces. Second, an AES key candidate is obtained using the detected collisions and the corresponding plaintexts. For some types of key recovery techniques based on nonlinear collisions the corresponding ciphertexts are additionally required. Note that one or several plaintext-ciphertext pairs produced with the attacked key may be needed to identify the correct key candidate in the offline stage.

If averaging is applied, the attacker has to be able to send several unknown equal inputs to the device and to fix some unknown key for these measurements in the profiling stage. Additionally, he has to be able to send several copies of the known random plaintexts to the implementation in the online stage.

The attack complexity is defined by three parameters. $C_{\text{profiling}}$ is the number of inputs to AES for which measurements have to be performed in the profiling stage (number of profiling measurements). Obviously, $C_{\text{profiling}} = 0$ for collision attacks without profiling. C_{online} is the number of inputs to AES for which measurements have to be performed in the online stage (number of online measurements). C_{offline} is the computational complexity of the key recovery, that is, the number of operations needed to solve the resulting systems of linear or nonlinear equations and to identify the most probable solution. It is assumed that the complexity of the offline stage is mainly determined by the key recovery step.

7.3 Basic Collision Attacks on AES

AES was attacked using side-channel collision techniques in [232]. This attack is based on detecting internal one-byte collisions in the MixColumns transformation of the first AES round. The basic idea is to identify pairs of plaintexts leading to the same byte value in an output byte after the MixColumns transformation of the first round and to use these pairs to deduce information about some key bytes involved into the transformation.

Let $A = (a_{ij})$ with $i, j \in \{0, \ldots, 3\}$ and $a_{ij} \in \mathbb{F}_2^8$ be the internal state in the first AES round after key addition, byte substitution and the ShiftRows operation. Let $B = (b_{ij})$ with $i, j \in \{0, \ldots, 3\}$ and $b_{ij} \in \mathbb{F}_2^8$ be the internal state after the MixColumns transformation, $B = \text{MixColumns}(A)$, where the MixColumns transformation is defined for each column j as follows[1]:

$$\begin{pmatrix} b_{0j} \\ b_{1j} \\ b_{2j} \\ b_{3j} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} a_{0j} \\ a_{1j} \\ a_{2j} \\ a_{3j} \end{pmatrix}. \tag{7.1}$$

Here all operations are performed over $\mathbb{F}_2^8$. Then b_{00} can be represented as:

$$\begin{aligned} b_{00} &= 02 \cdot a_{00} + 03 \cdot a_{10} + 01 \cdot a_{20} + 01 \cdot a_{30} \\ &= 02 \cdot S(p_{00} + k_{00}) + 03 \cdot S(p_{11} + k_{11}) + \\ &\quad 01 \cdot S(p_{22} + k_{22}) + 01 \cdot S(p_{33} + k_{33}). \end{aligned} \tag{7.2}$$

For two plaintexts P and P' with $p_{00} = p_{11} = p_{22} = p_{33} = \delta$ and $p'_{00} = p'_{11} = p'_{22} = p'_{33} = \epsilon$, $\delta \neq \epsilon$, one obtains the following, provided $b_{00} = b'_{00}$:

$$\begin{aligned} & 02 \cdot S(k_{00} + \delta) + 03 \cdot S(k_{11} + \delta) + 01 \cdot S(k_{22} + \delta) + 01 \cdot S(k_{33} + \delta) \\ = \; & 02 \cdot S(k_{00} + \epsilon) + 03 \cdot S(k_{11} + \epsilon) + 01 \cdot S(k_{22} + \epsilon) + 01 \cdot S(k_{33} + \epsilon). \end{aligned} \tag{7.3}$$

[1]Here and below any byte $uv = u \cdot 16 + v = \sum_{i=0}^{7} d_i \cdot 2^i$ is interpreted as the element of $\mathbb{F}_2^8 = \mathbb{F}_2[\omega]$ using a polynomial representation $\sum_{i=0}^{7} d_i \cdot \omega^i$, where $\omega^8 + \omega^4 + \omega^3 + \omega + 1 = 0$ holds.

Let $C_{\delta,\epsilon}$ be the set of all key bytes $k_{00}, k_{11}, k_{22}, k_{33}$ that lead to a collision (7.3) with plaintexts (δ, ϵ). Such sets are pre-computed and stored for all 2^{16} pairs (δ, ϵ). Each set contains on average 2^{24} candidates for the four key bytes. Actually, every set $C_{\epsilon,\delta}$ can be computed from the set $C_{\epsilon+\delta,0}$ using some relations between the sets. Due to some dependencies within the sets, this optimization reduces the required disk space to about 540 megabytes.

The attack on the single internal state byte b_{00} is then as follows. The attacker generates random values (ϵ, δ) and inputs them to the AES module as described above. The power consumption curve for the time interval, where b_{00} is processed, is stored. Then the attacker proceeds with other random values (ϵ', δ'), measures the power profile, stores it and correlates it with all stored power curves. And so on. One needs about 4 collisions (one in each output byte of a column) to recover the four bytes involved into the MixColumns transformation. The probability that after N operations at least one collision $b_{00} = b'_{00}$ occurs in a single byte is:

$$p_N = 1 - \prod_{l=0}^{N-1} (1 - l/2^8). \tag{7.4}$$

Actually, the attack can be parallelized to search for collisions in all four columns of B in parallel. In this case the attacker needs at least 16 collisions, 4 for each column of B, so $p_N^{16} \geq 1/2$ and $N \approx 40$. Once the required number of collisions was detected, he uses the pre-computed tables $C_{\epsilon+\delta,0}$ to recover all four key bytes for each column by intersecting the pre-computed key sets corresponding to the collisions (ϵ, δ) detected. Thus, on average one has to perform about 40 measurements to get all 16 collisions needed and to determine all 16 key bytes. Note that since the cardinality of the intersections for the sets $C_{\epsilon,\delta}$ is not always 1, there are a number of key candidates to be tested using known plaintext-ciphertext pairs.

7.4 Basic Collision Attack on Alpha-MAC

In this section we show how to obtain the internal state of Alpha-MAC using side-channel collision attacks and differential power analysis. Our attacks are aimed at the internal state rather than at K for the following reason: We attack implementations of Alpha-MAC under the assumption that keyed AES transformations (before and after the message injections) are protected against side-channel attacks (e.g. through masking, etc.) and the internal injection rounds are not. Developers in the real-world applications might choose to protect keyed transformations and keep the unkeyed rounds which perform the bulk of the hashing unprotected or weakly protected due to performance concerns. For instance, processing a (relatively short) 10-Kbyte message with Alpha-MAC requires a total of 2580 AES-128 rounds, from which only 20 are keyed.

7.4.1 Alpha-MAC

Message authentication code construction Alred and its AES-based instance Alpha-MAC were introduced in [78]. We show that under certain assumptions about its implementation (namely that keyed parts are perfectly protected against side-channel attacks but bulk hashing rounds are not) one can efficiently attack this function. We propose a side-channel collision attack on this MAC recovering its internal state just after 29 measurements in the known-message scenario which is to be compared to 40 measurements required by collision attacks on AES in the chosen-plaintext scenario. Having recovered the internal state, we mount a selective forgery attack using 4 to 1 round collisions working with negligible memory and time complexity.

Alpha-MAC is an iterative MAC function operating the state that is changed by consecutive "injections" of message blocks. The secret key of Alpha-MAC is used as a key of two AES transformations, which are applied at the beginning and at the end of computation, respectively.

Alpha-MAC suggests some significant benefits when implemented on embedded systems. First, Alpha-MAC can be easily implemented, if the AES algorithm is already on board, which has been becoming common practice in the real world. Moreover, the advantage of Alpha-MAC over traditional MAC constructions such as CBC-MAC or OMAC is its performance. For CBC-MAC/OMAC based on AES-128 the number of rounds per one processed 128-bit message block is 10, and for Alpha-MAC it is $4 + \frac{20}{t}$, where t is the number of processed blocks. That is, for sufficiently long messages Alpha-MAC outperforms CBC-MAC/OMAC by factor 2.5, which affects both the runtime and the power consumption.

In [78] several theorems are provided that substantiate the resistance of Alpha-MAC against adaptively-chosen-message attacks. They proved that any forgery attack on Alpha-MAC not involving internal collisions may be easily extended to an attack on the AES itself. Furthermore, they showed that any colliding messages of the same size have to be at least 5 blocks long.

Recently Huanga et al. [126] have shown how to construct 5-block collisions given a complete internal state or a secret key. Unfortunately, they did not consider how to derive that information in real-world applications.

Here we follow most of the notations from the original paper [78]. An Alpha-MAC structure is illustrated in Figure 7.1.

Let M be $4n$ byte long message. A computation of a MAC for M is a three step process. First, AES with the key K (of length 16, 24, or 32 bytes) is used to encrypt a zero block. Then n AES rounds are applied with the subkey of the i-th round being of the form:

$$\textsc{Injection:} \quad K^i = \begin{pmatrix} m^i_{00} & 0 & m^i_{02} & 0 \\ 0 & 0 & 0 & 0 \\ m^i_{20} & 0 & m^i_{22} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

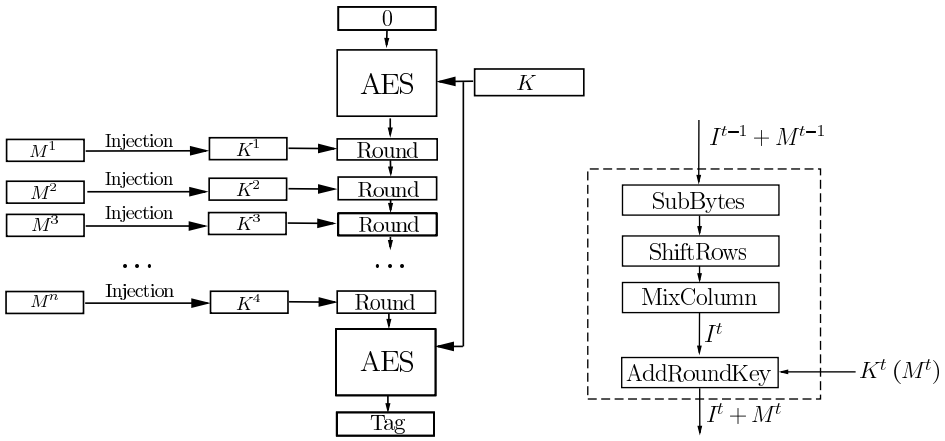


Figure 7.1: Alpha-MAC: Internal structure and the t-th round of message-dependent transformations

where $(m^i_{00}, m^i_{02}, m^i_{20}, m^i_{22}) = M^i$ — the i-th block of M. After n such rounds the result is again AES encrypted with the same key K. Here and later we will denote an internal state I^t modified by AddRoundKey(K^t) by $I^t + M^t$ instead of $I^t + K^t$ for convenience.

In [78] it was also allowed to add a Truncation block which crops some bytes from the final result. Truncation does not affect our attack so we omit it for simplicity.

7.4.2 Recovering Internal State

Here it is assumed that the attacker can observe messages that are to be processed by Alpha-MAC and is able to measure the power consumption of the computing device. Moreover, we require the observed messages to look random. This assumption is substantially different from that for the basic collision attack in [232], where the chosen-message possibility is required.

The basic collision attack on AES described above does not apply to Alpha-MAC directly, since only 4 fixed bytes out of 16 input bytes K^i can vary. The other 12 bytes are zero. However, the collision attack can be enhanced for Alpha-MAC in a way that requires a reduced number of measurements (29 instead of 40) and requires no pre-computations. Note that our side-channel collision attack itself does not need chosen plaintexts. However, at the end there are about 2^8 state candidates, which have to be tested by constructing collisions and verifying them using access to the device computing Alpha-MAC.

The ideas that we use are:

- Having detected several byte collisions, we treat them as a nonlinear system

of equations over $\mathbb{F}_2^8$. Then we solve these systems by brute-force. This allows not to use pre-computations and memory.

- We look for collisions in three consecutive injection rounds instead of working with only a single round. This is possible due to the fact that no entropy is introduced in the injection rounds. We show that this method requires less measurements than that of [232].

Let $I^1 = (i^1_{rs})$, $r, s \in \{0, \ldots, 3\}$, denote the internal state of Alpha-MAC directly before the first injection. I^1 is $E_K(0)$ after the SubBytes, ShiftRows and MixColumns transformations of the first injection round. We also similarly define I^2, I^3, and I^4. The goal of our attack is to find I^1. The initial internal state $E_K(0)$ can be then easily computed from I^1, since all AES round transformations are bijective. We denote the i-th injection by K^i:

$$K^i = \begin{pmatrix} k^i_{00} & k^i_{01} & k^i_{02} & k^i_{03} \\ k^i_{10} & k^i_{11} & k^i_{12} & k^i_{13} \\ k^i_{20} & k^i_{21} & k^i_{22} & k^i_{23} \\ k^i_{30} & k^i_{31} & k^i_{32} & k^i_{33} \end{pmatrix} = \begin{pmatrix} m^i_{00} & 0 & m^i_{02} & 0 \\ 0 & 0 & 0 & 0 \\ m^i_{20} & 0 & m^i_{22} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{7.5}$$

for $i \in \{1, \ldots, 4\}$.

Our side-channel collision attack on Alpha-MAC works as follows. We treat the states $I^j + K^j$ for $j \in \{2, \ldots, 4\}$ and try to detect collisions in some of the column bytes for a number of messages. Note that this differs from the basic collision attack, where one looks for collisions directly after the MixColumns transform and before key addition. Collisions in $I^2 + K^2$ and $I^3 + K^3$ reveal eight bytes of the internal state I^1. Collisions in $I^4 + K^4$ recover eight linear state-dependent relations over $\mathbb{F}_2^8$. These are linearly independent and can be uniquely solved, which reveals the remaining eight bytes of I^1.

After the MixColumns transform and message addition in the second injection round we have 8 bytes in I^2+K^2 that can collide: $i^2_{r0}+k^2_{r0}$ and $i^2_{r2}+k^2_{r2}$, $r \in \{0, \ldots, 3\}$. For instance, if a collision occurs in $i^2_{00} + k^2_{00}$, one has the following relation:

$$02 \cdot S(i^1_{00} + m^1_{00}) + S(i^1_{22} + m^1_{22}) + m^2_{00} = 02 \cdot S(i^1_{00} + z^1_{00}) + S(i^1_{22} + z^1_{22}) + z^2_{00}, \tag{7.6}$$

where M^1, Z^1 and M^2, Z^2 are message blocks injected into the first and second injection rounds, respectively, which result in this collision. Note that the other bytes do not depend on the message and, thus, cancel out. in this equation. After a further collision of type (7.6) has been detected in another byte of the 0th column, one has two nonlinear equations over $\mathbb{F}_2^8$ with two binary variables $i^1_{00}, i^1_{22} \in \mathbb{F}_2^8$. These equations are solved by brute-force. One gets similar equations by detecting two one-byte collisions in the second column. These can be solved in the same way and yield i^1_{02} and i^1_{20}.

Next, we detect collisions after the MixColumns transform and message addition in the third injection round. Note that four bytes of I^1 are already known. If a collision is detected in $i^3_{00} + k^3_{00}$, the following relation holds:

$$\begin{gathered} 02 \cdot S(03 \cdot S(i^1_{11}) + S(i^1_{33}) + c_1 + m^2_{00}) \\ +S(S(i^1_{13}) + 03 \cdot S(i^1_{31}) + c_2 + m^2_{22}) + m^3_{00} \\ = \\ 02 \cdot S(03 \cdot S(i^1_{11}) + S(i^1_{33}) + c'_1 + z^2_{00}) \\ +S(S(i^1_{13}) + 03 \cdot S(i^1_{31}) + c'_2 + z^2_{22}) + z^3_{00} \end{gathered} \tag{7.7}$$

for some injected blocks Z^2, M^2, Z^3, M^3 and known constants[2] $c_1, c_2, c'_1, c'_2 \in \mathbb{F}_2^8$. Two collisions in two bytes of the 0th column in the third injection round give two relations of type (7.7) which are solved with respect to $03 \cdot S(i^1_{11}) + S(i^1_{33})$ and $S(i^1_{13}) + 03 \cdot S(i^1_{31})$. Two further collisions in the second column of the same round deliver two other relations which yield $03 \cdot S(i^1_{13}) + S(i^1_{31})$ and $S(i^1_{11}) + 03 \cdot S(i^1_{33})$. These four relations can be uniquely solved with respect to i^1_{11}, i^1_{33}, i^1_{13} and i^1_{31}.

At the time one arrives at the MixColumns transform in the fourth injection round, 8 bytes of I^1 are known. Let us again focus on the 0th column. Its state before the MixColumns operation is as follows:

$$\begin{pmatrix} S(03 \cdot S(f_2) + S(g_4) + c_{00} + m^3_{00}) \\ S(S(f_1) + 03 \cdot S(g_3) + c_{10}) \\ S(S(g_2) + 03 \cdot S(f_4) + c_{20} + m^3_{22}) \\ S(03 \cdot S(g_1) + S(f_3) + c_{30}) \end{pmatrix}, \tag{7.8}$$

where

$$\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} S(i^1_{01}) \\ S(i^1_{12}) \\ S(i^1_{23}) \\ S(i^1_{30}) \end{pmatrix}$$

and

$$\begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} S(i^1_{03}) \\ S(i^1_{10}) \\ S(i^1_{21}) \\ S(i^1_{32}) \end{pmatrix}. \tag{7.9}$$

The constants c_{00}, c_{10}, c_{20} and c_{30} in (7.8) depend on the message injections K^1 and K^2 as well as on the previously recovered state bytes. These constants can be seen

[2]These constants depend on known message injections bytes and on already recovered bytes of I^1.

random. Thus, if a collision is detected in $i^4_{00} + k^4_{00}$, the following equation holds:

$$\begin{gathered} 02 \cdot S(03 \cdot S(f_2) + S(g_4) + c_{00} + m^3_{00}) + \\ 03 \cdot S(S(f_1) + 03 \cdot S(g_3) + c_{10}) + \\ S(S(g_2) + 03 \cdot S(f_4) + c_{20} + m^3_{22}) + \\ S(03 \cdot S(g_1) + S(f_3) + c_{30}) + m^4_{00} \\ = \\ 02 \cdot S(03 \cdot S(f_2) + S(g_4) + c'_{00} + z^3_{00}) + \\ 03 \cdot S(S(f_1) + 03 \cdot S(g_3) + c'_{10}) + \\ S(S(g_2) + 03 \cdot S(f_4) + c'_{20} + z^3_{22}) + \\ S(03 \cdot S(g_1) + S(f_3) + c'_{30}) + z^4_{00} \end{gathered} \tag{7.10}$$

for some appropriate Z^3, K^3, Z^4, K^4. The attacker can obtain the variables $03 \cdot S(f_2) + S(g_4)$, $S(f_1) + 03 \cdot S(g_3)$, $S(g_2) + 03 \cdot S(f_4)$ and $03 \cdot S(g_1) + S(f_3)$, if all four bytes of the column collide. Then he has four equations of type (7.10) that can be solved by brute force (2^{32} operations). Another 4-byte collision in column 1 will lead to the values $02 \cdot S(f_1) + S(g_3)$, $02 \cdot S(g_2) + S(f_4)$, $S(g_1) + 02 \cdot S(f_3)$ and $02 \cdot S(f_2) + S(g_4)$. These reveal f_j, g_j, $j \in \{1, \ldots, 4\}$. Note that this occurs for any pair of columns in this injection round. Thus, the attacker needs two 4-byte collisions in any of the four columns, a single 4-byte collision for a column meaning four 1-byte collisions in the same column which can occur in different messages.

After f_j, g_j, $j \in \{1, \ldots, 4\}$ have been recovered, the linear systems of equations (7.9) can be solved and uniquely deliver the rest of the variables i^1_{01}, i^1_{12}, i^1_{23}, i^1_{30}, i^1_{03}, i^1_{10}, i^1_{21} and i^1_{32}, since one deals with the invertible MixColumns transform. Thus, the whole state I^1 is recovered.

Our thorough simulations show that two collisions in a column for rounds 2 and 3 do not allow for a unique solution of the two involved unknown bytes, even if these collisions occur in different column bytes. In this case one has about two solutions, averaged over all pairs of unknown bytes for different random injections. In round 4 each of the two 4-byte collisions deliver approximately 2^3 candidates for the intermediate variables. Thus, the attacker has in average 2^8 candidates for I^1 at the end. The correct one can be identified in the next step by trying to construct a collision as described in Section 7.4.4.

Now the number of needed measurements is estimated. In injection rounds 2 and 3 at least two collisions are needed in the 0th and second columns (collisions in some two bytes of each of the both columns). In a single column two collisions have to be detected. Thus, the following probability needs to be computed:

$$\begin{gathered} P = \Pr\{\text{at least two collisions in 4 column bytes}\} = \\ = 1 - \Pr\{A = \text{no collisions in all 4 bytes}\} - \\ - \Pr\{B = \text{exactly one collision in one of the 4 bytes}\} = \\ = 1 - P_A - P_B. \end{gathered} \tag{7.11}$$

If p is the probability that no collisions occurred in a specific column byte after N measurements, then:

$$p = \prod_{j=0}^{N-1} \left(1 - \frac{j}{256}\right) \text{ and } P_A = p^4. \tag{7.12}$$

Let q_i, $i = 2, \ldots, N$, be the probability that exactly one collision is detected in the i-th measurement and no collisions occurred in all the remaining measurements. Then:

$$q_i = \frac{i-1}{256} \prod_{j=0}^{N-2} \left(1 - \frac{j}{256}\right) \text{ and } P_B = 4p^3 \sum_{i=2}^{N} q_i. \tag{7.13}$$

In injection round 4 any two columns have to yield a 4-byte collision each. This probability can be calculated using p_N from (7.4) as follows:

$$P' = \binom{4}{2} p_N^8 (1 - p_N^4)^2 + \binom{4}{3} p_N^{12} (1 - p_N^4) + \binom{4}{4} p_N^{16}. \tag{7.14}$$

Thus, $P^4 P' \geq 1/2$ must be fulfilled for a successful attack, which is achieved for $N = 29$ with $P^4 P' \approx 0.560$. This is to be compared to 40 measurements needed for the key-recovery in [232].

7.4.3 Experiments

As a proof of concept, we have implemented Alpha-MAC on an 8-bit microcontroller. We opted for a PIC16F687 low power CMOS microcontroller [188], clocked at 4 MHz employing its internal oscillator and needing four internal clock cycles (1 μs) to execute an instruction, which is in between the twelve internal cycles of a standard i8051 controller and the single clock cycle an Atmel AVR processor needs for carrying out one command. Communication takes place via a serial port, while the actual power consumption is detected single-ended by means of a 240 Ω resistor inserted between the supply ground and the ground pin of the microcontroller. Care has to be taken to avoid ground loops and assure stable power supply and proper shielding of the measurement setup, for reduction of noise, as measurements with a high accuracy are required to detect collisions of bytes being processed, compared to only observing their Hamming weight.

Our implementation of Alpha-MAC is written in PIC assembly language using ideas from [164]. Due to the RAM in the PIC being restricted to only 128 bytes, we stored the 256 bytes of the S-box in the program memory, as proposed in [244]. One pin of the microcontroller is used as a trigger output, thus easing the alignment of the data. While the PIC is computing the MAC, power traces are acquired by the 8-Bit ADC (Analog to Digital Converter) of an Agilent Infinium 54832D oscilloscope at a sampling rate of 4 GSa/s. The data is stored and evaluated on a PC with Matlab.

In order to detect the collisions at the end of the i-th injection round, we identified four distinct sequences of instructions which looked valuable for a power analysis in the relevant parts of the $(i+1)$-th injection round. The following code uses the target byte directly:

```
...
;SubBytes, target byte in accu
  movf  A1,w          ;
  call  SBOX          ;
  movwf A1            ;
...
;ShiftRows (for rows 1,2 and 3 only)
  movf  A1,w          ;
  movwf A2            ;
...
;MixColumns
  movf  A2,w          ;
  xorwf R1,w          ;
...
  movf  A2,w          ;
  xorwf R2,f          ;
...
```

Note that the code between the listed fragments delivers random states of the accumulator register before each sequence. Moreover, values R1 and R2 in the MixColumns transform can be seen as uncorrelated with the target byte. The similarity of two power traces τ_1 and τ_2 was detected for all discrete points belonging to the above mentioned code fragments by finding a minimum of $\sum (\tau_{1,i} - \tau_{2,i})^2$.

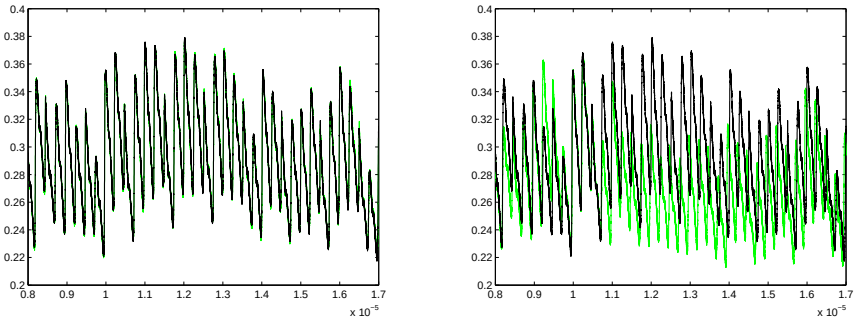


Figure 7.2: Power consumption curves for equal (left) and different (right) bytes

The result is presented in Figure 7.2, where two power traces out of the section involving the S-box lookup are compared. On the left side, the indices for the table lookups are equal to each other and a difference between the power consumption is

almost not noticeable, while on the right side two distinct table lookups exhibit an obvious difference.

The left part of Figure 7.3 shows the difference curve, i.e., $\tau_1 - \tau_2$, when a correlation is detected. The right half of Figure 7.3 depicts the difference curve in case the measured power consumption curves do not correlate.

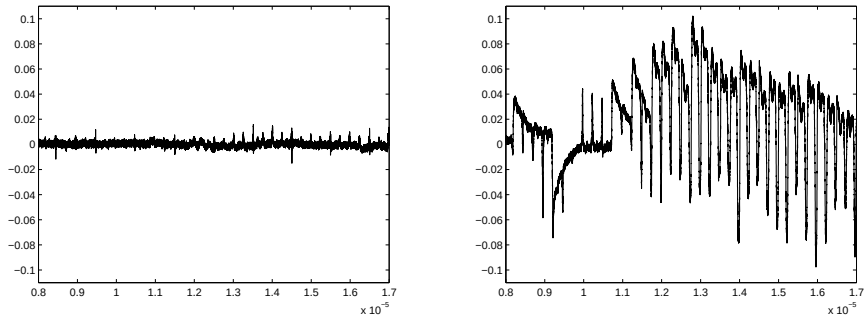


Figure 7.3: Difference curves in case a collision is detected (left) and no collision is detected (right)

Note that the PIC microcontroller leaks a lot of collision-relevant information. This to a large extent contributed to the measurement results presented here. For many other microcontroller platforms, one has to deal with much more noise.

7.4.4 Message Forgery for Alpha-MAC

Here, following [26], it is shown how one can exploit the knowledge of the internal state in order to construct collisions for Alpha-MAC.

Some properties of the AES round function. Let us take another look at Figure 7.1, more precisely, at the AES/Alpha-MAC round function. MixColumns is the only transformation that provides diffusion [79]. It is a linear transformation over $(\mathbb{F}_2^8)^{16}$ and acts on groups of 4 bytes, so it may be considered as four linear transformations over $(\mathbb{F}_2^8)^4$. Remind that SubBytes acts on individual bytes, and ShiftRows is just a permutation. As a result, we can divide the state $I^{t-1} + K^{t-1}$ into 4 groups of bytes (denote them by A_i, $i \in \{1, \ldots, 4\}$) and the state I^t into other 4 groups (B_i, respectively). Then the following property holds.

Observation 1. *B_i bytes are linear combinations over $\mathbb{F}_2^8$ of* SubBytes *transformed A_i bytes.*

This may be illustrated on the following scheme:

$$
\begin{array}{cccccccccccc}
 & I^{t-1}+K^{t-1} & & & & & I^t & & & & & \\
A_1: & a_{00} & a_{11} & a_{22} & a_{33} & \xrightarrow{L_1\circ S} & b_{00} & b_{01} & b_{02} & b_{03} & : B_1 & \\
A_2: & a_{10} & a_{21} & a_{32} & a_{03} & \xrightarrow{L_2\circ S} & b_{10} & b_{11} & b_{12} & b_{13} & : B_2 & \\
A_3: & a_{20} & a_{31} & a_{02} & a_{13} & \xrightarrow{L_3\circ S} & b_{20} & b_{21} & b_{22} & b_{23} & : B_3 & \\
A_4: & a_{30} & a_{01} & a_{12} & a_{23} & \xrightarrow{L_4\circ S} & b_{30} & b_{31} & b_{32} & b_{33} & : B_4 &
\end{array}
\tag{7.15}
$$

Here S denotes the S-box transformation and L_i — the i-th subfunction of MixColumns. For example, byte 7 in I^t is a linear over $\mathbb{F}_2^8$ function of S-boxed bytes 4, 9, 14, and 3 of $I^{t-1}+K^{t-1}$.

The following property also holds.

Observation 2. *Given any 4 bytes of a row of (7.15) one may compute the other 4 bytes of the row fast (4 S-box transformations and* ≈ 20 *additions and multiplications in the field).*

The proof may be easily derived by the substitution of $S(x)$ into x for all x from A_i.

How to construct a 4-1 collision. The side-channel collision attack gives us the internal state I^1. As a result, for every 4-byte M' we can compute J — the internal state after one round. The authors of Alpha-MAC proved that any 128-bit J can be reached from any other internal state after 4 rounds with an appropriate choice of $M=(M^1,M^2,M^3,M^4)$. More formally, we use the following lemma.

Lemma 10 ([78])**.** *Given* I^1*, the state value before iteration* 1*, the map*

$$s:(M^1,M^2,M^3,M^4)\to I^5$$

from the sequence of 4 message blocks (M^1,M^2,M^3,M^4) *to the state value before iteration 5 is a bijection.*

This lemma was proven in a non-constructive way so we show how to compute the message $M=(M^1,M^2,M^3,M^4)$. A brief scheme of computing M is presented in Figure 7.4. Now we show how to compute unknown state values step by step.

Step 0. We already computed $I^5=J$.

Step 1. A_2 and A_4 bytes are not modified by K^1 (the result of the injection M^1) so we can compute B_2 and B_4 bytes. Analogously for M^4.

Remark 1. *Let us notice that* $I^5=J$ *implies* $I^4+K^4=I^1+K'$ *so known bytes of* I^4 *are exactly bytes of* I^1*.*

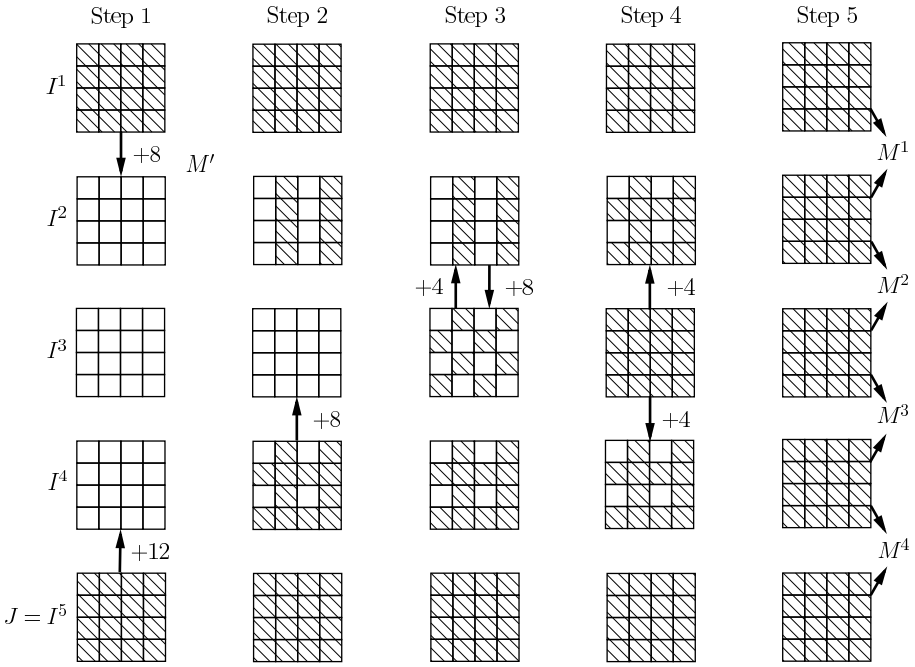


Figure 7.4: Computation of the collision: $I^1 \xrightarrow[M^1||M^2||M^3||M^4]{M'} I^5$

Step 2. Given 12 bytes of I^4 we compute 8 bytes of I^3.

Step 3. Now we know 8 bytes of I^2 (and thus the same bytes of $I^2 + K^2$) and 8 bytes of I^3. One can easily check that we know 2 bytes from each A_i and 2 bytes from each B_i. Then we use Observation 2 and obtain all bytes of I^3 and 12 bytes of I^2.

Step 4. We use Observation 2 to derive values of last unknown bytes in I^2 and I^4.

Step 5. First we compute M^1 and M^2 using pairs (I^1, I^2) and (I^2, I^3). Secondly, we do the same to obtain M^3 and M^4.

As a result we have a 4–1 collision: $M^1||M^2||M^3||M^4$ collides with M'.

On N–1 collisions. Collisions of type 3–1 and 2–1 may theoretically exist. Given I^1 one can reduce our construction to the search of such collisions. Due to Remark 1 we do not need to know M' to detect whether a collision exist. As a result, only some linear computations are required to check whether short collisions are possible.

The collisions that we showed in this section were not considered by designers or by [126]. It seems that in the case of AES it may be difficult to construct such

collisions in less than 2^{64} steps without the knowledge of the internal state. However formally proving it seems challenging. Note also that in the case of Alred used with Feistel ciphers (ex. Triple-DES, proposed in [78]) one has to be more careful, since involutional structure of Feistel-ciphers makes it easier to find partial fixed points required in such collisions.

Selective forgery of Alpha-MAC We have shown very efficient way to construct 4 to 1 block collisions in Alpha-MAC, if the internal state is known. We estimate the complexity of finding collisions as 2^{11} operations in $\mathbb{F}_2^8$.

The way we construct these collisions allows to perform selective forgery attack on Alpha-MAC. The attack would be as follows: the attacker obtains measurements of authentication for 29 arbitrary known messages. From these he derives 2^8 candidates for the 128-bit internal state. He then picks arbitrary victim message-tag pair (M, σ). The attacker picks a message M' that he would like to authenticate (with exception of a suffix δ of 16 bytes). For each candidate of the internal state (and thus for each candidate of $E_K(0)$) and M' the attacker can evaluate the internal states I and I' of Alpha-MAC after the injection of M and M', respectively. He then computes the 16 injection bytes δ which transform I' into I. Then a pair $((M'||\delta), \sigma)$ would be a properly authenticated message-tag pair.

7.4.5 Implications for Pelican-MAC

In [80] another AES-based MAC called Pelican has been proposed, which is very similar to Alpha-MAC. The main difference is that in Pelican there is a single 128-bit message injection every 4 rounds instead of a 32-bit message injection every round. Pelican is also sensitive to attacks shown in this chapter. The adversary first learns the internal state at some unprotected round, if such round exists. From that point the attacker has full control of the internal state: i.e. he can create arbitrary meaningful colliding messages by calculating a proper 128-bit injection at the end. Thus, one should mask the internal rounds of Pelican-MAC to hamper such attacks.

7.5 Linear Collision-Based Key Recovery for AES

7.5.1 Generalized Internal Collisions

We introduce the notion of *a generalized internal collision* for AES that occurs within one or several AES runs, if there are two equal input bytes to the S-box operation in some (possibly different) rounds at some (possibly different) byte positions for one or several inputs.

In other words, we take all applications of the S-box transform within a number of AES executions and compare them pairwisely to each other. AES-128 performs 160 S-box operations in the data path for each run, which are different for different inputs, and 40 additional S-box computations in the key schedule, which remain the same for a given key. If two of these S-box instances in one or two distinct runs process the same value, there is a generalized internal collision. The power of the improved collision attacks on AES originates from the fact that the number of generalized collisions grows quadratically with the linear increase of the number of unique inputs considered. So, even if the key schedule is ignored, there are about 40.9 colliding S-boxes for just one input and already about 555.2 collisions for 5 inputs.

Starting from this point, it is not that important anymore to be aware of where a certain byte is positioned in the 4×4 byte matrix of the AES internal state. Therefore, we will be using a slightly different notation in the remainder of the chapter for simplicity: $K = \{k_j\}_{j=1}^{16}$, $k_j \in \mathbb{F}_2^8$ is the 16-byte AES key and $P = \{p_j\}_{j=1}^{16}$, $p_j \in \mathbb{F}_2^8$ is the 16-byte plaintext.

7.5.2 Linear Collisions in AES and Linear Equations

In round $i \in \{1, \ldots, 10\}$, AES-128 performs the SubBytes operation (16 parallel S-box applications) on the output bytes of the previous round XORed with the i-th round subkey. A generalized internal AES collision occurs, if there are two S-boxes within the same AES execution or within several AES runs accepting the same byte value as their input.

In Figure 7.5 a collision within the first round of two different AES executions (number 1 and 3) is illustrated, where p_j^l, $j \in \{1, \ldots, 16\}$, are plaintext bytes for the lth measurement. In the example of Figure 7.5, byte 4 in the first execution and byte 11 in the third execution collide.

A detected collision in the S-box layer of the first round in bytes j_1 and j_2 with $j_1, j_2 \in \{1, \ldots, 16\}$ corresponds to the following linear equation:

$$S(k_{j_1} \oplus p_{j_1}^{l_1}) = S(k_{j_2} \oplus p_{j_2}^{l_2}), \tag{7.16}$$

$$k_{j_1} \oplus k_{j_2} = \Delta_{j_1,j_2} = p_{j_1}^{l_1} \oplus p_{j_2}^{l_2} \tag{7.17}$$

for some known plaintext bytes $p_{j_1}^{l_1}$ and $p_{j_2}^{l_2}$ with some positive integers l_1, l_2 indicating measurement numbers. In the same way, one can rewrite equations resulting from collisions within some other round $i \in \{1, \ldots, 10\}$. In this case one has some unknown key- and plaintext-dependent byte variables instead of the plaintext bytes $p_{j_1}^{l_1}$ and $p_{j_2}^{l_2}$. We will study such collisions in the next section.

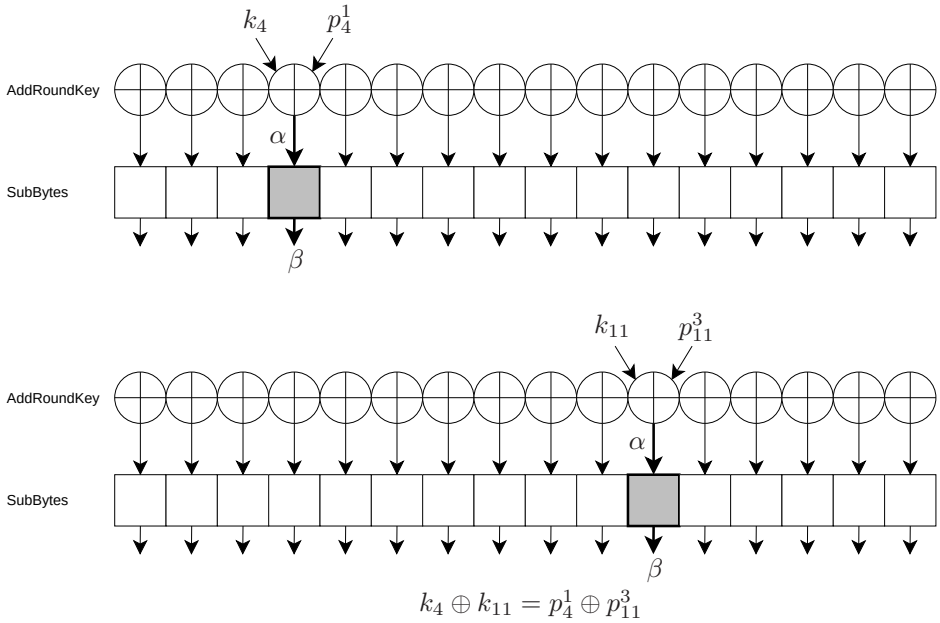


Figure 7.5: Generalized internal collision within the first round of two AES runs

7.5.3 Systems of Equations and Associated Graphs

Given a linear collision (within the first round of AES), one obtains a linear equation with respect to the key over $\mathbb{F}_2^8$. If m collisions have been detected, they can be interpreted as a system of linear binomial equations over $\mathbb{F}_2^8$.

The structure of a random system of m linear equations of type (7.17) resulting from a number of collisions detected within the S-box layer in the first round is as follows:

$$S_m : \begin{cases} k_{j_1} \oplus k_{j_2} = \Delta_{j_1,j_2} \\ k_{j_3} \oplus k_{j_4} = \Delta_{j_3,j_4} \\ \dots \\ k_{j_{2m-1}} \oplus k_{j_{2m}} = \Delta_{j_{2m-1},j_{2m}}. \end{cases} \tag{7.18}$$

Note that this system has 16 variables (bytes of the first round subkey). In system (7.18) the key byte numbers and the variables are not necessarily distinct. This system cannot have the full rank (see Proposition 6). Moreover, for small numbers of inputs to AES the system is not connected and it can be divided into a set of smaller independent (with disjunct variables) connected subsystems with respect to the parts of the key.

A straightforward proposition holds about the rank of S_m:

Proposition 6. *The maximal rank of S_m is 15, $rank(S_m) \leq 15$.*

Proof. The maximal rank of 15 is attained, for instance, for

$$\begin{cases} k_1 \oplus k_2 = \Delta_{1,2} \\ k_2 \oplus k_3 = \Delta_{2,3} \\ \dots \\ k_{14} \oplus k_{15} = \Delta_{14,15} \\ k_{15} \oplus k_{16} = \Delta_{15,16}. \end{cases} \tag{7.19}$$

It is easy to see that the XOR of any other pair of variables can be obtained as a sum of two of the 15 equations in (7.19). Thus, 15 is the maximal rank for S_m □

We use a graph representation of S_m for our analysis.

Definition 11. *A random graph $G_m = \langle V, E \rangle$ is associated with the random system S_m of linear equations, where $V = \{k_1, k_2, \dots, k_{16}\}$ is the set of 16 vertices of G_m and the edge (k_{j_1}, k_{j_2}) belongs to the edge set E iff the binomial equation*

$$k_{j_1} \oplus k_{j_2} = \Delta_{j_1,j_2}$$

belongs to the system S_m, $|E| = m$.

Among others, the associated graph possesses the following properties:

Proposition 7. *The system S_m is of the maximal rank 15 iff its associated graph G_m is connected.*

Proposition 8. *Let $G = \langle V, E \rangle$ be a non-directed graph with n vertices, $|V| = n$. If*

$$|E| > \begin{pmatrix} n-1 \\ 2 \end{pmatrix},$$

the graph G is connected.

For G_m Proposition 8 implies that if $|E| > 105$, G_m is necessarily connected and, thus, S_m has the maximal rank of 15. A system of type (7.18) having the maximal rank can be solved by assigning a byte value to some variable k_j (which is equivalent to adding a further, linearly non-dependent equation $k_j = \Delta_j$ to the system) and uniquely solving the system

$$S_m \cup \{k_j = \Delta_j\}$$

of rank 16. Then another byte value is assigned to that variable. And so on. The correct key is identified using a known plaintext-ciphertext pair.

Generally speaking, it is not necessary for S_m to have the maximal rank. If there are several isolated subsystems within S_m, then each of them can be solved independently as described above. If there are q independent subsystems $SS_m^1, \dots, SS_m^q$ in S_m, then S_m can be represented as a union of these subsystems:

$$S_m = SS_m^1 \cup \dots \cup SS_m^q, \ SS_m^i \cap SS_m^j = \varnothing, \ i \neq j.$$

To solve S_m in this case, one has to assign q byte values to some q variables in the subsystems $\{SS_m^i\}_{i=1}^q$. At the end there are 2^{8q} key candidates to be tested. The correct key is identified using a known plaintext-ciphertext pair as outlined above.

It is clear that the independent subsystems $\{SS_m^i\}_{i=1}^q$ of S_m correspond to the q connected components of the associated graph G_m.

The number of connected components of a random graph has the following asymptotic behaviour:

Proposition 9. *Let G be a random graph with n labeled vertices and*

$$N = \lfloor \frac{1}{2} n \log n + cn \rfloor$$

for some constant c. Let $q = q_{n,N}$ be the number of connected components in G. Then:

$$\lim_{n\to\infty} \Pr\{q = i+1\} = \frac{(e^{-2c})^i}{i!} \exp\left\{-e^{-2c}\right\}.$$

Proof. See Theorem 2.3 in [230] □

Unfortunately, the estimate of Proposition 9 for the number of connected components cannot be directly applied for S_m, since its associated graph has only 16 vertices.

7.5.4 Expected Number of Random Binomial Equations

The number of edges in the associated graph G_m can be estimated using the following

Proposition 10. *The expected number $E(m)$ of edges in G_m (equivalently, the expected number of distinct binomial equations in S_m) for the first round of AES after $\gamma \geq 1$ random inputs is*

$$E(m) = 120 \cdot \left(1 - \left(\frac{119}{120}\right)^{16\gamma - 256 + 256 \cdot \exp\left\{16\gamma \cdot \ln \frac{255}{256}\right\}}\right).$$

Proof. The expected number of generalized collisions within the first round after γ inputs can be computed as:

$$N_{1\mathrm{R}} = 16\gamma - 256 + 256 \cdot \left(\frac{255}{256}\right)^{16\gamma}, \tag{7.20}$$

where 16γ is the number of S-box operations in one AES round for γ inputs. This equation is a reformulation of the birthday paradox.

The expected number of edges in a random graph with n labeled vertices after N_{1R} random selections of edges (after N_{1R} generalized collisions) can be interpreted

Table 7.2: Number of collisions and edges in G_m according to Proposition 10

Inputs, γ	4	5	6	7	8	9	11	29
# 1R collisions, N_{1R}	7.27	11.18	15.82	21.14	28.12	33.70	48.55	249.64
# edges, $E(m)$	7.09	10.72	14.88	19.46	24.36	29.49	40.07	105.14

as the expected number of filled boxes after N_{1R} random shots in the classical shot problem, which was studied e.g. in Chapter 1 of [154]. In the case of a graph, one deals with $\binom{n}{2}$ boxes (possible graph edges) and the expected number of edges after N_{1R} collisions is

$$E(m) = \binom{n}{2} \left(1 - \left(1 - \frac{1}{\binom{n}{2}} \right)^{N_{1R}} \right). \tag{7.21}$$

As $n = 16$ for the case of AES, one obtains the claim of the proposition by combining (7.20) and (7.21) □

Table 7.2 contains theoretical estimations for the numbers of first round collisions N_{1R} and edges $E(m)$ depending on the number of inputs γ.

Note that according to Proposition 8, it is expected that after 29 inputs one obtains 105 edges which provide the maximal rank of S_m. However, on average a lower number of edges are sufficient for the G_m to be connected with a high probability (see the next subsection).

7.5.5 Number of Connected Components in Associated Graphs

In order to estimate the number q of connected components for G_m accounting for the offline complexity, statistical simulation was applied consisting of generating a random graph on 16 vertices corresponding to γ inputs as well as counting the number of connected components q using a variation of Karp and Tarjan's algorithm

Table 7.3: Offline complexity and success probabilities of linear key-recovery

Inputs, γ	4	5	6	7	8	9	11	29
# edges in G_m, m	7.09	10.72	14.88	19.46	24.36	29.49	40.07	105.14
# connected components of G_m, q	8.81	5.88	3.74	2.20	1.43	1.15	1.04	1.00
C_{offline} for ≤ 40 bit	34.70	37.34	37.15	34.74	30.32	21.36	12.11	8
Success probability π for ≤ 40 bit	0.037	0.372	0.854	0.991	0.999	1.000	1.000	1.000
C_{offline} for ≤ 48 bit	43.90	45.50	44.30	41.14	30.32	21.36	12.11	8
Success probability π for ≤ 48 bit	0.092	0.548	0.927	0.997	0.999	1.000	1.000	1.000

[138] for finding connected components of a graph. Note that the expected complexity of this algorithm is linear in the number of vertices. For each number of inputs we performed 2^{16} simulations with random graphs.

The results of our simulations are shown in Table 7.3. The first and second rows of the table represent input numbers and average numbers of edges in G_m according to Proposition 10 (see also Table 7.2), respectively. The offline complexity is given for two cases: for ≤ 40 bit and for ≤ 48 bit. In the first case, only offline complexities $\leq 2^{40}$ are considered in the computation of the average offline complexity value. For each number of inputs the probability is provided that the overall offline complexity is $\leq 2^{40}$. In the second case, the upper bound for the offline complexities taken into account is 2^{48}. The corresponding success probabilities are also provided in the table.

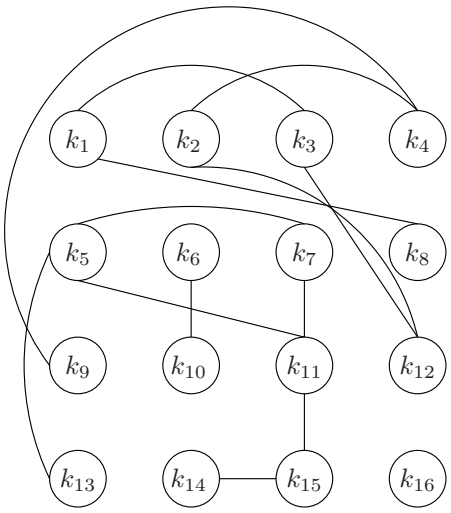


Figure 7.6: A typical random graph for linear collision-based key recovery with $\gamma = 6$ (13 edges and 4 connected components)

A low-complexity offline stage of the attack becomes probable after 5 inputs ($2^{45.5}$ steps with success probability 0.548). Practically all instances of linear systems resulting from 7 inputs are easily solvable with average complexity $2^{34.74}$ steps (with success probability 0.99). After 11 inputs the expected offline attack complexity is about $2^{12.11}$.

Figure 7.6 shows a typical random graph G_m associated with a random system S_m of linear equations with $m = 13$ and 4 independent subsystems (4 connected components): $\{k_1, k_{12}, k_2, k_3, k_4, k_8, k_9\}$, $\{k_{11}, k_{13}, k_{14}, k_{15}, k_5, k_7\}$, $\{k_6, k_{10}\}$, $\{k_{16}\}$.

7.6 Algebraic Collision-Based Key Recovery for AES

In this section we identify (Subsection 7.6.1) types of nonlinear generalized collisions enabling efficient algebraic representation that give rise to efficient key recovery. Then the corresponding systems of nonlinear equations are constructed (Subsection 7.6.2) and solved (Subsection 7.6.3). We first assume that all collisions are detected correctly. We deal with collision detection errors in Sections 7.7 and 7.8.

Throughout this section, $K = \{k_j\}_{j=1}^{16}$, $k_j \in \mathbb{F}_2^8$ is the 16-byte AES key. $X = \{x_j\}_{j=1}^{16}$, $Y = \{y_j\}_{j=1}^{16}$ and $Z = \{z_j\}_{j=1}^{16}$, $x_j, y_j, z_j \in \mathbb{F}_2^8$ are the first, next to the last and last 16-byte AES subkeys, respectively. AES plaintexts are denoted by $P = \{p_j\}_{j=1}^{16}$, $p_j \in \mathbb{F}_2^8$ and ciphertexts by $C = \{c_j\}_{j=1}^{16}$, $c_j \in \mathbb{F}_2^8$.

7.6.1 Nonlinear Collisions

FS-collisions. Generalized collisions in the first two AES rounds occurring between bytes of the first two rounds are called *FS-collisions*. If input bytes $a_{j_1}^{i_1}$ and $a_{j_2}^{i_2}$ of two S-boxes collide, one has the simple linear equation over $\mathbb{F}_2^8$:

$$a_{j_1}^{i_1} \oplus a_{j_2}^{i_2} = 0.$$

If a_j^i lies in the S-box layer of the first round, then $a_j^i = k_j \oplus p_j^i$, for some i, j. Otherwise, one has

$$a_j^i = x_j \oplus m_{(j-1)\ div\ 4} \cdot b^i_{(j-1)\ mod\ 4+1} \oplus m_{j\ div\ 4} \cdot b^i_{j\ mod\ 4+5} \oplus$$
$$m_{(j+1)\ div\ 4} \cdot b^i_{(j+1)\ mod\ 4+9} \oplus m_{(j+2)\ div\ 4} \cdot b^i_{(j+2)\ mod\ 4+13},$$

where $m = (m_0, m_1, m_2, m_3) = (02, 03, 01, 01)$ and $b_j^i = S(k_j \oplus p_j^i)$.

We distinguish between the following three types of FS-collisions: linear collisions in the first round, nonlinear collisions between bytes of the first two rounds, and nonlinear collisions within the second round. These three collision types are illustrated in Figure 7.7. S-boxes where collisions occur (*active* S-boxes) are marked by the numbers of collisions they account for. The input byte p_j^i is characterized by its position $j \in \{1, \ldots, 16\}$ within the plaintext block and the number $i = 1, 2, \ldots$ of the plaintext block it belongs to.

Collision 1 occurs between two bytes of the first round, linearly binding k_1 and k_{13}. Collision 2 occurs between the S-box number 7 of the second round and the S-box number 1 of the first round. It binds 6 key bytes: k_1, k_3, k_8, k_9, k_{14}, and k_7. Collision 3 algebraically connects two MixColumns expressions on 8 key bytes after the S-box layer with two bytes of the second subkey in a linear manner. The

algebraic expressions in this example are the following:

$$
\begin{aligned}
1: &\quad k_1 \oplus p_1^1 = k_{13} \oplus p_{13}^1 \\
2: &\quad k_1 \oplus p_1^2 = S^{-1}(s_7^2) = \\
&\quad x_7 \oplus 01 \cdot S(k_2 \oplus p_2^2) \oplus 02 \cdot S(k_8 \oplus p_8^2) \oplus 03 \cdot S(k_9 \oplus p_9^2) \oplus 01 \cdot S(k_{14} \oplus p_{14}^2) \\
3: &\quad s_7^3 = s_{16}^3, \\
&\quad k_7 \oplus 01 \cdot S(k_3 \oplus p_3^3) \oplus 02 \cdot S(k_8 \oplus p_8^3) \oplus 03 \cdot S(k_9 \oplus p_9^3) \oplus 01 \cdot S(k_{14} \oplus p_{14}^3) = \\
&\quad x_{16} \oplus 03 \cdot S(k_4 \oplus p_4^3) \oplus 01 \cdot S(k_5 \oplus p_5^3) \oplus 01 \cdot S(k_{10} \oplus p_{10}^3) \oplus 02 \cdot S(k_{15} \oplus p_{15}^3)
\end{aligned}
$$

Note that there are also mirrored collisions occurring between the S-boxes of the last round (number 10) and the round next to the last one (number 9). Such collisions are called *LN-collisions*.

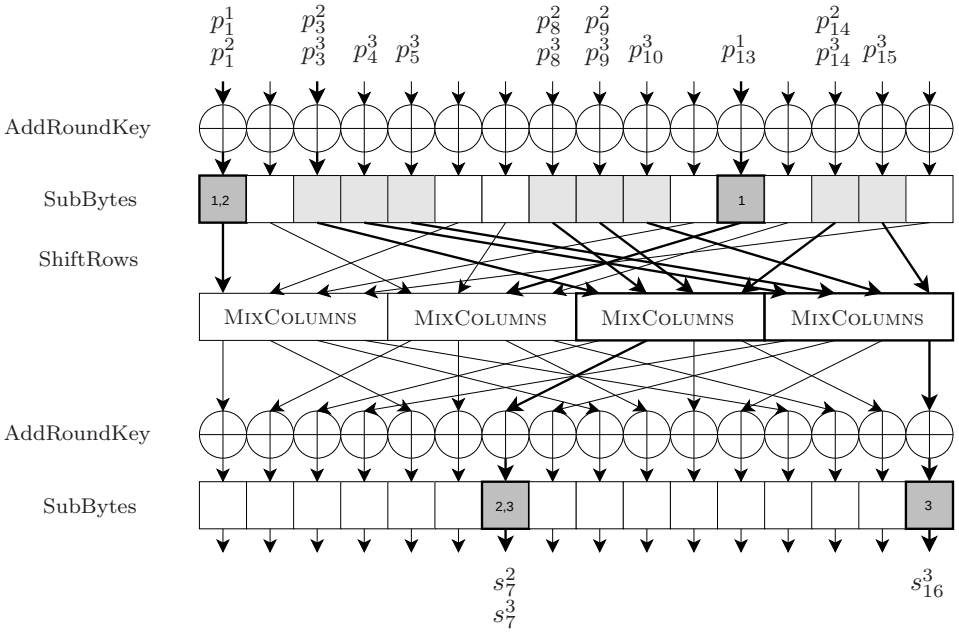


Figure 7.7: FS-collisions

FL-collisions. *FL-collisions* are generalized collisions between bytes of the first and last rounds. If both plaintexts and ciphertexts are known, FL-collisions lead to simple nonlinear equations. Linear collisions within the first round as well as those within the last round can be additionally used.

Figure 7.8 illustrates these three types of collisions. Collision 1 occurs between the 2nd byte of the first round and the 5th byte of the last round for some input and output with p_2^1 and c_8^1 (note that the bytes do not have to belong to the same input/output pair). Input $y_2 \oplus a^1$ to S-box 5 in the last round can be expressed as

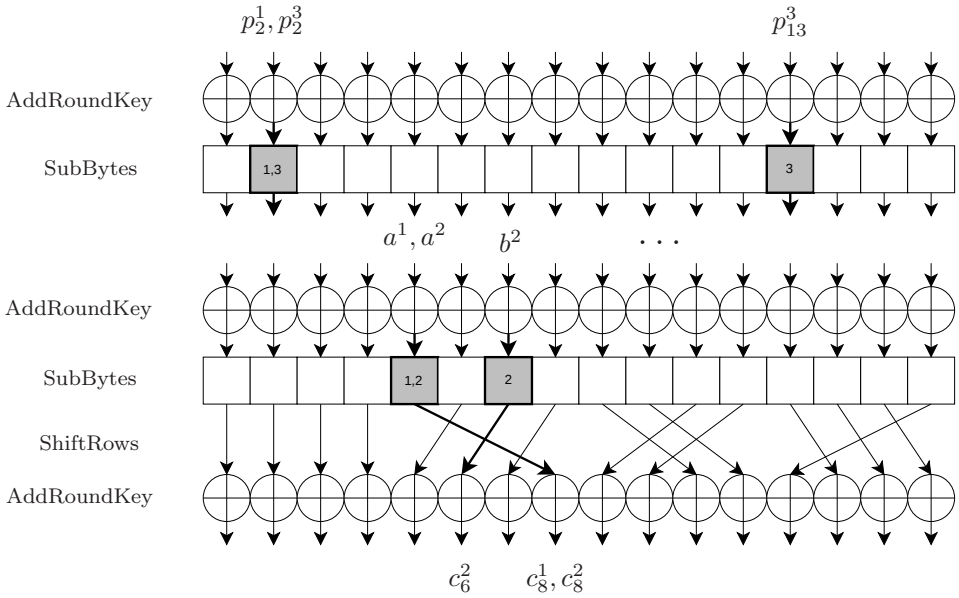


Figure 7.8: FL-collisions

$S^{-1}(z_8 \oplus c_8^1)$ using the corresponding ciphertext and last subkey bytes. Collision 2 of Figure 7.8 is a linear collision within the last AES round. Collision 3 is a standard linear collision within the first AES round. The following equations result from these collisions:

$$\begin{aligned} 1: &\quad k_2 \oplus p_2^1 = y_5 \oplus a^1 = S^{-1}(z_8 \oplus c_8^1), S(k_2 \oplus p_2^1) = z_8 \oplus c_8^1 \\ 2: &\quad y_5 \oplus a^2 = y_7 \oplus b^2,\ z_8 \oplus c_8^2 = z_6 \oplus c_6^2 \\ 3: &\quad k_2 \oplus p_2^3 = k_{13} \oplus p_{13}^3. \end{aligned}$$

7.6.2 Constructing Systems of Equations for FS- and FL-Collisions

Equations for FS-collisions. The application of the Faugère F4 algorithm to a system of equations constructed in Subsection 7.6.1 for FS-collisions gives results that are superior to the linear collision attack and are summarized in Table 7.4.

The system of nonlinear equations is considered over $\mathbb{F}_2$. For γ inputs there are 128 variables of the first subkey K, 128 variables of the second subkey X and $128 \cdot \gamma$ intermediate variables for the output bits of the first round S-box layer. The collision-independent part of the system consists of S-box equations for the first round and linear equations for the key schedule. Since the AES S-box can be implicitly expressed as 39 equations of degree 2 [64], we have $39 \cdot 16 \cdot \gamma$ quadratic equations over $\mathbb{F}_2$ connecting the inputs and outputs of the first round S-boxes, and $4 \cdot 39 = 156$ quadratic and $12 \cdot 8 = 96$ linear equations connecting K and X using

the key schedule relations. Each of the three types of FS-collisions adds 8 linear equations to the system, resulting in $8 \cdot D$ equations if D collisions occurred.

Equations for FL-collisions. FL-collisions can be also obviously expressed as a system of quadratic equations over $\mathbb{F}_2$. Now we show how to derive a system of quadratic equations over $\mathbb{F}_2^8$ for these collisions. One way is to use the BES expression [64]. However one would have 8 variables per one key byte in this case. We describe a simpler system, which has only 32 variables.

It is clear that linear collisions in the first or the last round can be interpreted as linear equations over $\mathbb{F}_2^8$. Let us consider a nonlinear FL-collision of type 1 (see example above). Its algebraic expression is given by $S(k_{j_1} \oplus p_{j_1}^{i_1}) = z_{j_2} \oplus c_{j_2}^{i_2}$ for some $j_1, j_2 \in \{1, \ldots, 16\}$, $i_1, i_2 = 1, 2, \ldots$ Recall that the AES S-box is the composition of the multiplicative inverse in the finite field $\mathbb{F}_2^8$, an $\mathbb{F}_2$-linear mapping, and the XOR-addition of the constant 63. The $\mathbb{F}_2$-linear mapping is invertible, and its inverse is given by the following polynomial over $\mathbb{F}_2^8$:

$$f(x) = 6e \cdot x^{2^7} + db \cdot x^{2^6} + 59 \cdot x^{2^5} + 78 \cdot x^{2^4} + 5a \cdot x^{2^3} + 7f \cdot x^{2^2} + fe \cdot x^2 + 05 \cdot x.$$

Hence we have

$$(k_{j_1} \oplus p_{j_1}^{i_1})^{-1} = f(z_{j_2} \oplus c_{j_2}^{i_2} \oplus 63) = f(z_{j_2}) \oplus f(c_{j_2}^{i_2} \oplus 63).$$

If we replace $f(z_{j_2})$ by a new variable u_{j_2}, we obtain the quadratic equation

$$(k_{j_1} \oplus p_{j_1}^{i_1})(u_{j_2} \oplus f(c_{j_2}^{i_2} \oplus 63)) = 1,$$

which holds with probability $\frac{255}{256}$. The following proposition follows:

Proposition 11. *Solutions to the equation* $S(k_{j_1} \oplus p_{j_1}^{i_1}) = z_{j_2} \oplus c_{j_2}^{i_2}$ *coincides with solutions to the equation*

$$(k_{j_1} \oplus p_{j_1}^{i_1})(u_{j_2} \oplus f(c_{j_2}^{i_2} \oplus 63)) = 1$$

under the change of variables $u_{j_2} = f(z_{j_2})$ *with a probability of* $\frac{255}{256}$.

Moreover, if $z_{j_2} \oplus z_{j_3} = \Delta_{j_2,j_3} = c_{j_2}^{i_2} \oplus c_{j_3}^{i_3}$, then we have

$$f(z_{j_2}) \oplus f(z_{j_3}) = u_{j_2} \oplus u_{j_3} = f(\Delta_{j_2,j_3}).$$

Thus, we derive for FL-collisions the system $\mathbb{S}$ of quadratic equations over $\mathbb{F}_2^8$ in 32 variables $\mathcal{K} = \{k_j, u_j\}_{1 \le j \le 16}$. Furthermore, each equation of the resulting system has only two variables and, thus, is a binomial equation.

We say that a subset of variables $\mathcal{K}' \subset \mathcal{K}$ is connected, if for any non-trivial partition of $\mathcal{K}' = A \cup B$ there is an equation in $\mathbb{S}$ in two variable $v \in A$ and $w \in B$. Thus $\mathcal{K}$ can be devided into disjoint subsets $\mathcal{K}_i$ with respect to $\mathbb{S}$, where $\mathcal{K}_i$ is either connected or singleton. Each $\mathcal{K}_i$ corresponds to an unique subsystem $\mathbb{S}_i$, and we call $(\mathcal{K}_i, \mathbb{S}_i)$ a *chain*.

Table 7.4: Solving equation systems for FS-collisions over $\mathbb{F}_2$

Inputs, γ	5	5	4	4
Success probability π	0.425	0.932	0.042	0.397
C_{offline} (time), s	142.8	7235.8	71.5	6456.0
Memory limit, MB	500	500	500	500
# variables	896	896	768	768
# linear/quadratic equations	96+8D/3276	96+8D/3276	96+8D/2652	96+8D/2652

7.6.3 Solving Systems for FS- and FL-Collisions

Solving equations for FS-collisions. The system of nonlinear equations for FS-collisions is solved in the following way. First the system is passed to the F4 algorithm without modifications. If it is not solvable, one guesses the largest connected linear component as in linear collision-based recovery (that is, 8 bits per connected component, see Subsection 7.5.3), adds the corresponding linear equation to the system and tries to solve the system again. The memory limit for the Magma program was set to 500 MB. It can be seen from Table 7.4 that for 5 inputs most ($> 93\%$) instances of the FS-system can be solved within several hours on a PC. For 4 inputs, less systems are solvable (about 40%) within approx. 2 hours on a standard PC.

Solving equations for FL-collisions. FL-collisions lead, as a rule, to better results. Since one deals with binomial equations introduced in Subsection 7.6.2 for FL-collisions, each equation binds only two $\mathbb{F}_2^8$-variables. There are 32 variables $\mathcal{K}$ over $\mathbb{F}_2^8$. The algebraic relations on these variables are much simpler, since one has both plaintext and ciphertext bytes (more information related to the detected collisions).

Moreover, for the system we have a set of independent chains. Let $(\mathcal{K}_i, \mathbb{S}_i)$ be a chain, and $v \in \mathcal{K}'$. Since $\mathcal{K}'$ is connected, there exists a relation between v and any other variable of $\mathcal{K}'$. It is not hard to prove that this relation can be expressed as linear or quadratic equation in two variables. Further, some chain can have a non-linear equation such that the corresponding variables are still connected also without this equation. In this case we call this chain a cycle. For any cycle the system has at most two solutions. Thus, there are often nonlinear subsystems solvable independently.

On average, there are about 1.02 independently solvable subsystems covering 30.08 out of 32 $\mathbb{F}_2^8$-variables for $\gamma = 5$ inputs and 0.99 cycles covering 20.08 out of 32 $\mathbb{F}_2^8$-variables for $\gamma = 4$ inputs. Statistically there are 43.58 collisions for $\gamma = 5$ inputs and 29.66 collisions for $\gamma = 4$ inputs.

Table 7.5 contains the results for applying the F4 algorithm to FL-systems of nonlinear equations averaged over 10000 samples. After resolving the nonlinear subsystems using F4, as for FS-collisions, we guess variables defining the remaining

Table 7.5: Solving equation systems over $\mathbb{F}_2^8$ for FL-collisions

Inputs, γ	5	4
Success probability π	1.00	0.85
C_{offline}	$\leq 2^{40}$	$\leq 2^{40}$
Memory limit, MB	500	500
# variables	32	32
Average # equations	43.58	29.66

bytes in a way similar to the linear key-recovery. With $C_{\text{offline}} \leq 2^{40}$, practically all FL-systems are solvable for 5 inputs, an FL-system being solvable with probability 0.85 for $\gamma = 4$ inputs.

7.6.4 A Sample Equation System for FL-Collisions

Equations. We derived a system of $\mathbb{F}_2^8$-equations for the following secret key K and $\gamma = 4$ plaintexts/ciphertexts P_j/C_j, $j = 1, 2, 3, 4$:

$$K = (\texttt{4F}, \texttt{4D}, \texttt{32}, \texttt{3D}, \texttt{DC}, \texttt{BA}, \texttt{7C}, \texttt{1C}, \texttt{6A}, \texttt{7E}, \texttt{80}, \texttt{A1}, \texttt{19}, \texttt{EB}, \texttt{A1}, \texttt{A8})$$

$$\begin{aligned}
P_1 &= (\texttt{3D}, \texttt{72}, \texttt{41}, \texttt{0D}, \texttt{C6}, \texttt{AC}, \texttt{A1}, \texttt{EE}, \texttt{1F}, \texttt{88}, \texttt{74}, \texttt{84}, \texttt{9E}, \texttt{E2}, \texttt{BA}, \texttt{06}),\\
C_1 &= (\texttt{D5}, \texttt{E1}, \texttt{2A}, \texttt{BF}, \texttt{D7}, \texttt{DB}, \texttt{FF}, \texttt{B4}, \texttt{45}, \texttt{F9}, \texttt{4C}, \texttt{2E}, \texttt{30}, \texttt{94}, \texttt{64}, \texttt{D3}),\\
P_2 &= (\texttt{A8}, \texttt{14}, \texttt{6F}, \texttt{9D}, \texttt{01}, \texttt{4F}, \texttt{21}, \texttt{A7}, \texttt{9A}, \texttt{50}, \texttt{B6}, \texttt{72}, \texttt{9A}, \texttt{C2}, \texttt{A2}, \texttt{43}),\\
C_2 &= (\texttt{DE}, \texttt{EB}, \texttt{1C}, \texttt{A1}, \texttt{8F}, \texttt{63}, \texttt{B4}, \texttt{06}, \texttt{5C}, \texttt{21}, \texttt{19}, \texttt{C2}, \texttt{0F}, \texttt{3F}, \texttt{AA}, \texttt{24},\\
P_3 &= (\texttt{C9}, \texttt{2E}, \texttt{9D}, \texttt{52}, \texttt{99}, \texttt{5F}, \texttt{30}, \texttt{32}, \texttt{AF}, \texttt{79}, \texttt{A8}, \texttt{A0}, \texttt{5D}, \texttt{4D}, \texttt{82}, \texttt{44}),\\
C_3 &= (\texttt{9D}, \texttt{3A}, \texttt{A2}, \texttt{76}, \texttt{3B}, \texttt{1C}, \texttt{4C}, \texttt{CE}, \texttt{4D}, \texttt{56}, \texttt{82}, \texttt{8F}, \texttt{43}, \texttt{BF}, \texttt{C1}, \texttt{8F},\\
P_4 &= (\texttt{21}, \texttt{AA}, \texttt{E9}, \texttt{2E}, \texttt{53}, \texttt{20}, \texttt{E1}, \texttt{B3}, \texttt{B8}, \texttt{0A}, \texttt{0C}, \texttt{E4}, \texttt{9D}, \texttt{30}, \texttt{53}, \texttt{B5}),\\
C_4 &= (\texttt{FF}, \texttt{38}, \texttt{F2}, \texttt{AD}, \texttt{86}, \texttt{67}, \texttt{63}, \texttt{48}, \texttt{9A}, \texttt{2E}, \texttt{45}, \texttt{D1}, \texttt{91}, \texttt{F0}, \texttt{50}, \texttt{96}).
\end{aligned}$$

The corresponding system of equations consists of the following 26 equations resulting from 26 different collisions (13 linear and 13 nonlinear):

- 9 linear equations for collisions between bytes of the first round

$$\begin{aligned}
k_7 + k_5 &= \texttt{A0} & k_3 + k_7 &= \texttt{4E} & k_{14} + k_{16} &= \texttt{43}\\
k_8 + k_{10} &= \texttt{62} & k_1 + k_2 &= \texttt{02} & k_3 + k_8 &= \texttt{2E}\\
k_5 + k_{12} &= \texttt{7D} & k_3 + k_{14} &= \texttt{D9} & k_8 + k_{15} &= \texttt{BD}
\end{aligned}$$

- 4 linear equations for collisions between bytes of the last round

$$u_1 + u_4 = \texttt{18} \qquad u_{14} + u_8 = \texttt{F0} \qquad u_{11} + u_{12} = \texttt{B5} \qquad u_4 + u_{13} = \texttt{0C}$$

- 13 quadratic equations for collisions between bytes of the first and last rounds

$$
\begin{aligned}
k_2 \cdot u_{12} + \texttt{A3} \cdot k_2 + \texttt{72} \cdot u_{12} &= \texttt{70} & k_{14} \cdot u_{16} + \texttt{C8} \cdot k_{14} + \texttt{E2} \cdot u_{16} &= \texttt{89} \\
k_9 \cdot u_{10} + \texttt{9A} \cdot u_{10} + \texttt{47} \cdot k_9 &= \texttt{7D} & k_{12} \cdot u_6 + \texttt{72} \cdot u_6 + \texttt{9A} \cdot k_{12} &= \texttt{1C} \\
k_{13} \cdot u_3 + \texttt{DA} \cdot k_{13} + \texttt{9E} \cdot u_3 &= \texttt{16} & k_3 \cdot u_1 + \texttt{9D} \cdot u_1 + \texttt{F9} \cdot k_3 &= \texttt{F9} \\
k_7 \cdot u_7 + \texttt{30} \cdot u_7 + \texttt{E4} \cdot k_7 &= \texttt{5C} & k_1 \cdot u_{11} + \texttt{B4} \cdot k_{12} + \texttt{C9} \cdot u_{11} &= \texttt{5F} \\
k_{13} \cdot u_{12} + \texttt{85} \cdot k_{13} + \texttt{00} \cdot u_{12} &= \texttt{01} & k_{15} \cdot u_9 + \texttt{A6} \cdot k_{15} + \texttt{82} \cdot u_9 &= \texttt{4A} \\
k_4 \cdot u_5 + \texttt{2E} \cdot u_5 + \texttt{64} \cdot k_4 &= \texttt{7B} & k_{15} \cdot u_{15} + \texttt{33} \cdot k_{15} + \texttt{BA} \cdot u_{15} &= \texttt{2B} \\
k_1 \cdot u_{16} + \texttt{39} \cdot k_1 + \texttt{C9} \cdot u_{16} &= \texttt{B6}
\end{aligned}
$$

Note that for k_6, k_{11}, and u_2 there are no equations.

Graph representation and solving the system. To describe the intuition behind our method of solving the system of equations for FL-collisions, we represent it as a graph with 32 vertices (each corresponding to one of 32 $\mathbb{F}_2^8$-variables), see Figure 7.9, step 1. Note that in order to find the key, it suffices to determine either all variables k_i or u_i for $i \in \{1, \ldots, 16\}$ (either first or last AES subkey, respectively). Thus, at the beginning, the entropy of the system is maximal and equal to 16 bytes.

Next we identify the connected components of the graph with respect to the linear equations (linear collisions denoted by dashed lines in Figure 7.9, step 2). There are five such connected components - two on k_i and three on u_i. Note that the entropy in each of these connected components is 8 bit, since fixing one variable of a connected component to a certain value unambiguously defines the remaining variables of this component. Thus, we can 'glue' the variables (vertices) in each of these five connected components and treat them as five intermediate variables k'_1, k'_5, u'_1, u'_8, u'_{11}. After this step, the entropy of the system is reduced to 7 bytes, which is primarily due to the large component of k'_5.

Then we consider the graph with glued variables and add edges corresponding to quadratic equations (denoted by solid lines in Figure 7.9, step 3) to it. The resulting graph contains a *cycle* between variables u'_{11} and k'_1. That is, these are connected by *two* quadratic equations of the same two variables. Then we obtain a quadratic equation of *one* variable having a maximum of two solutions. Thus, u'_{11} and k'_1 are now resolved with an entropy of one bit.

After this, we observe that all variables of the (nonlinear) connected component containing the cycle can be resolved unambiguously, if the values of u'_{11} and k'_1 are fixed. In other words, the entropy of the whole component containing the cycle is one bit. Among other variables, we determine k'_1, k'_5 and k_{13} in this way. Note that k_6, k_{11}, k_4 and k_9 have to be guessed independently. This step mainly defines the overall solving complexity for this system as $2 \cdot 2^{4 \cdot 8} = 2^{33}$.

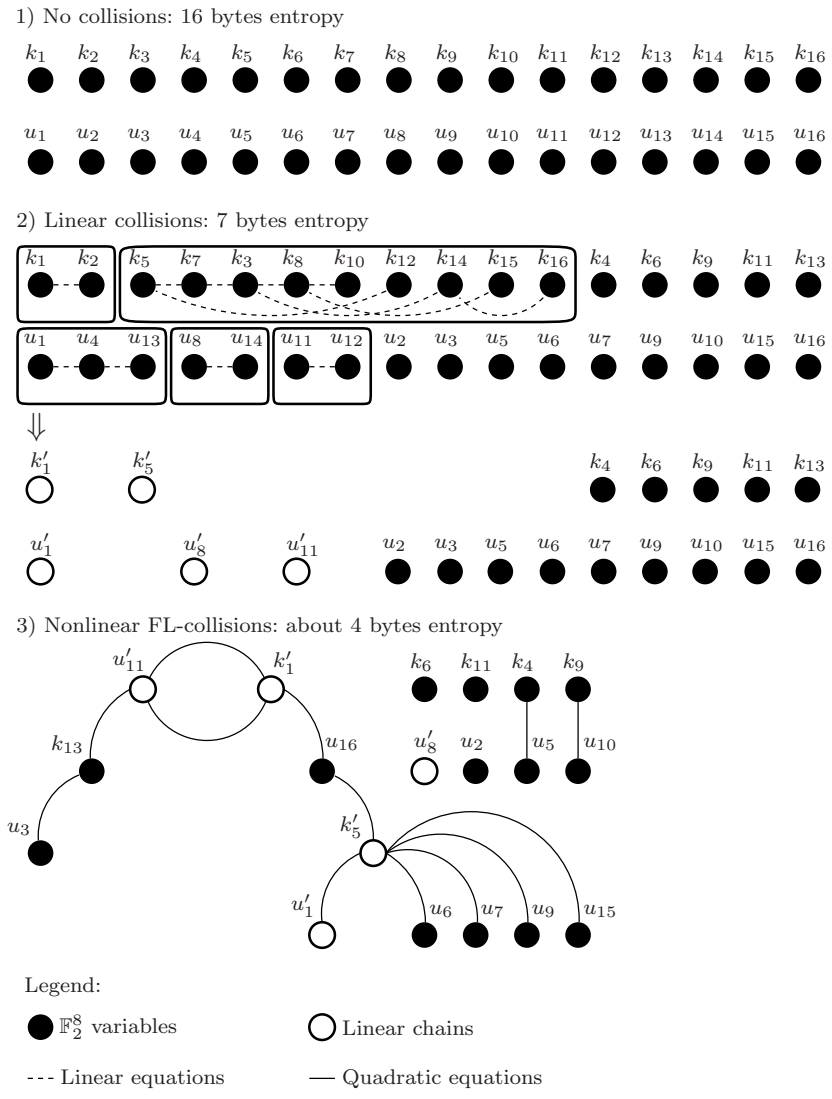


Figure 7.9: Solving a sample system of equations for FL-collisions

7.7 Towards Reliable Collision Detection

7.7.1 Euclidean Distance for Trace Comparison

Typical side channels are the power consumption of devices as well as their electromagnetic radiation, which manifest some data and key dependency in many cases. There are several ways of deciding if two S-boxes accept equal inputs using the side-channel information obtained from the implementation of an attacked cryptographic algorithm. In this work we use the Euclidean distance of two traces as real-valued vectors for this purpose.

Definition. Given two traces $\tau_{j_1}^{i_1} = (\tau_{j_1,1}^{i_1}, \ldots, \tau_{j_1,l}^{i_1}) \in \mathbb{R}^l$ and $\tau_{j_2}^{i_2} = (\tau_{j_2,1}^{i_2}, \ldots, \tau_{j_2,l}^{i_2}) \in \mathbb{R}^l$, respectively corresponding to S-box j_1 for plaintext P_{i_1} processing value $a_{j_1}^{i_1}$ and to S-box j_2 for plaintext P_{i_2} processing value $a_{j_2}^{i_2}$, one has to decide if $a_{j_1}^{i_1} = a_{j_2}^{i_2}$ for collision detection. The binary comparison test $\mathfrak{T}^{BC}$ based on Euclidean distance can be defined as:

$$\mathfrak{T}^{BC}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}) = \begin{cases} 0 \text{ (no collision)}, \text{if } \mathfrak{S}^{BC}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}) > Y^{BC} \\ 1 \text{ (collision)}, \text{ if } \mathfrak{S}^{BC}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}) \leq Y^{BC}, \end{cases}$$

where Y^{BC} is a decision threshold and

$$\mathfrak{S}^{BC}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}) = \sum_{r=1}^{l} (\tau_{j_1,r}^{i_1} - \tau_{j_2,r}^{i_2})^2,$$

which can be seen as a correlation characteristic of two reduced templates. Let $\mathfrak{T}^{BC}$ be characterized by the following type I and II error probabilities[3]:

$$\alpha_1 = \Pr\{\mathfrak{T}^{BC}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}) = 0 | a_{j_1}^{i_1} = a_{j_2}^{i_2}\},$$
$$\alpha_2 = \Pr\{\mathfrak{T}^{BC}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}) = 1 | a_{j_1}^{i_1} \neq a_{j_2}^{i_2}\}.$$

Note that α_1 and α_2 depend on the implementation and the value of Y^{BC}. Of course, there is a strong dependency on the noise as well.

Combination with averaging. To increase the resolution of the collision detection one can use averaging. That is, each plaintext is sent t times to the device. Respectively, t measurements are performed for each plaintext. Then the obtained traces for each distinct plaintext are averaged. If the noise is due to normal distribution with the zero mean value and a standard deviation σ, then the noise amplitude of a trace averaged t times will be $\sigma/\sqrt{t}$.

[3]Note that α_1 and α_2 strongly depend on the statistical properties of the traces (among many other factors on the noise amplitude) and the choice of the threshold value.

7.7.2 Key Recovery Robust to Type I Collision Detection Errors

Both algebraic and linear collision-based key recovery methods can be made tolerant to non-zero type I error probabilities of collision detection: A non-zero value of the type I collision detection error probability α_1 is equivalent to omitting some collisions. If the number of inputs γ is somewhat increased, this is easily tolerated by the methods, since the number of true (apriori) collisions grows quadratically with the increase of the number of inputs.

Under the assumption that the type II error probability α_2 is negligibly low, we performed this trade-off between α_1, γ and the success probability $\mathcal{P}$ of the entire attack for FS-, FL- and linear collisions. The results can be found in Figure 7.10 and show that the FL-collision based method is superior to FS- and linear collision based methods even in the presence of strong noise. For example, while type I error probabilities up to $\alpha_1 = 0.65$ are well tolerated with only $\gamma = 7$ inputs for FL-collisions, one needs at least $\gamma = 8$ and $\gamma = 9$ inputs to achieve a comparable tolerance level for FS- and linear collisions, respectively.

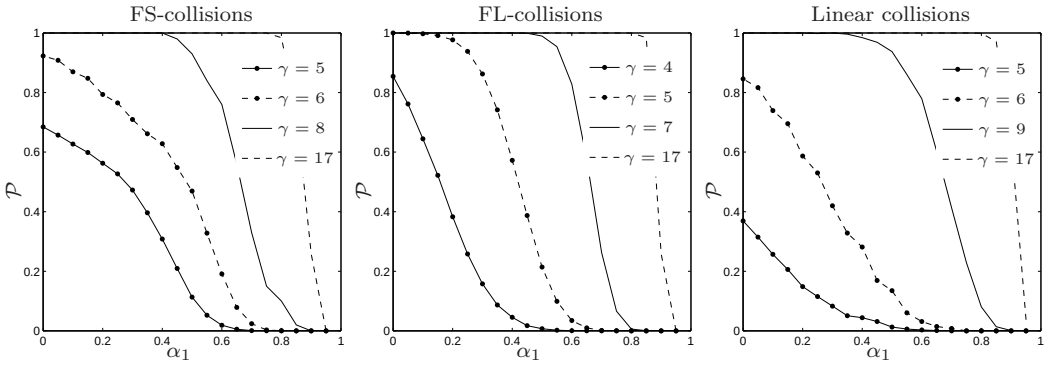


Figure 7.10: Success probability $\mathcal{P}$ of the entire attack against type I error probability α_1 in FS-, FL- and linear collision-based key recovery for different numbers γ of random inputs

7.7.3 Probability Distribution of Euclidean Distance

Given two traces $\tau_1 = (\tau_{1,1}, \ldots, \tau_{1,l}) \in \mathbb{R}^l$ and $\tau_2 = (\tau_{2,1}, \ldots, \tau_{2,l}) \in \mathbb{R}^l$, we assume that each point $\tau_{i,j}$ can be statistically described as $\tau_{i,j} = s_{i,j} + r_{i,j}$, where $s_{i,j}$ is signal constant (without noise) for the given time point i as well as some fixed input to the S-box, and $r_{i,j}$ is Gaussian noise due to univariate normal distribution[4] with

[4]The real measured power consumption is often due to the generic multivariate normal distribution. However, almost all entries of the corresponding covariance matrix are close to

mean 0 and some variance σ^2 remaining the same for all time instances in our rather rough model. Let τ_1 and τ_2 correspond to some S-box inputs a_1 and a_2.

If $a_1 = a_2$, the corresponding deterministic signals are equal (that is, $s_{1,j} = s_{2,j}$ for all j's) and one has:

$$1/H(\tau_1, \tau_2)_{a_1=a_2} = \sum_{j=1}^{l} (\tau_{1,j} - \tau_{2,j})^2 = \sum_{j=1}^{l} \xi_j^2 = 2\sigma^2 \sum_{j=1}^{l} \eta_j^2,$$

where $\xi_j = r_{1,j} - r_{2,j}$, $\xi_j \sim \mathcal{N}\left(0, 2\sigma^2\right)$ and $\eta_j \sim \mathcal{N}(0,1)$. It follows that statistic $1/H(\tau_1, \tau_2)_{a_1=a_2}$ follows the chi-square distribution with l degrees of freedom up to the coefficient $2\sigma^2$. As the chi-square distribution is approximated by normal distribution for high degrees of freedom, one has the following

Proposition 12. *Statistic* $1/H(\tau_1, \tau_2)_{a_1=a_2} = \sum_{j=1}^{l} (\tau_{1,j} - \tau_{2,j})^2$ *for* $\tau_i = (\tau_{i,1}, \ldots, \tau_{i,l}) \in \mathbb{R}^l$ *with* $\tau_{i,j} \sim \mathcal{N}\left(s_{i,j}, \sigma^2\right)$ *can be approximated by normal distribution* $\mathcal{N}(2\sigma^2 l, 8\sigma^4 l)$ *for sufficiently large* l*'s.*

Alternatively, if $a_1 \neq a_2$, one has

$$1/H(\tau_1, \tau_2)_{a_1 \neq a_2} = \sum_{j=1}^{l} (\tau_{1,j} - \tau_{2,j})^2 = \sum_{j=1}^{l} \left(\delta_j^{(1,2)} + \xi_j\right)^2 = 2\sigma^2 \sum_{j=1}^{l} \nu_j^2,$$

where $\delta_j^{(1,2)} = s_{1,j} - s_{2,j}$, $\xi_j = r_{1,j} - r_{2,j}$, $\xi_j \sim \mathcal{N}\left(0, 2\sigma^2\right)$ and $\nu_j \sim \mathcal{N}\left(\delta_j^{(1,2)}/\sqrt{2}\sigma, 1\right)$. That is, statistic $1/H(\tau_1, \tau_2)_{a_1 \neq a_2}$ follows the *noncentral* chi-square distribution with l degrees of freedom and $\lambda = \sum_{j=1}^{l} \left(\delta_j^{(1,2)}/\sqrt{2}\sigma\right)^2$ up to the coefficient $2\sigma^2$. Again, we have an approximation using

Proposition 13. *Statistic* $1/H(\tau_1, \tau_2)_{a_1 \neq a_2} = \sum_{j=1}^{l} (\tau_{1,j} - \tau_{2,j})^2$ *for* $\tau_i = (\tau_{i,1}, \ldots, \tau_{i,l}) \in \mathbb{R}^l$ *with* $\tau_{i,j} \sim \mathcal{N}\left(s_{i,j}, \sigma^2\right)$ *can be approximated by normal distribution* $\mathcal{N}\left(2\sigma^2(l+\lambda), 8\sigma^4(l+2\lambda)\right)$ *with* $\lambda = \sum_{j=1}^{l} \left(\delta_j^{(1,2)}/\sqrt{2}\sigma\right)^2$ *for sufficiently large* l*'s.*

7.7.4 Selection of Most Informative Trace Points

In the direct binary comparison, we try to distinguish between the distributions $1/H(\tau_1, \tau_2)_{a_1 \neq a_2}$ and $1/H(\tau_1, \tau_2)_{a_1=a_2}$. As described above these statistics approximately follow normal distribution for large numbers of trace points. That is, to

zero. Thus, the model with independent multivariate normal distribution seems to be quite realistic.

efficiently distinguish between these two statistics it is crucial to decrease their variances while keeping the difference of their means high. For this purpose, to increase the success probability of the Euclidean distance test, we propose to discard points of traces with small minimal contribution to the difference of means.

To illustrate this method of point selection, we assume for the moment that $\delta_j^{(1,2)} = 0$ for $j > l/2$ and $\delta_j^{(1,2)} \neq 0$ for $j \leq l/2$ with l even, that is, the second half of the trace does not contain any data dependent information. Then we can discard the second halves of the both traces τ_1 and τ_2 in the direct binary comparison function and compute two related statistics on the rest of the points:

$$1/H'(\tau_1, \tau_2)_{a_1=a_2} = \sum_{j=1}^{l/2} (\tau_{1,j} - \tau_{2,j})^2, 1/H'(\tau_1, \tau_2)_{a_1 \neq a_2} = \sum_{j=1}^{l/2} (\tau_{1,j} - \tau_{2,j})^2.$$

This will adjust the means and variances of the approximating normal distributions: $\mathcal{N}\left(\sigma^2 l, 4\sigma^4 l\right)$ and $\mathcal{N}\left(2\sigma^2(l/2+\lambda), 8\sigma^4(l/2+2\lambda)\right)$, respectively. Note that the difference of means remains unaffected and equal to $2\sigma^2\lambda$. At the same time both variances are reduced, one of them by factor 2, which allows one to distinguish between these two distributions more efficiently and, thus, to detect collisions more reliably.

More generally speaking, for AES we have to reliably distinguish between inputs in each (a_{i_1}, a_{i_2}) of the $\binom{256}{2}$ pairs of byte values, $a_{i_1}, a_{i_2} \in \mathbb{F}_2^8$. Thus, the most informative points j of the traces are those with maximal minimums of $\delta_j^{(i_1,i_2)}$ over all pairs of different inputs, that is, points j with maximal values of

$$\min_{a_{i_1} \neq a_{i_2}} \delta_j^{(i_1,i_2)}.$$

We estimated these values for all time instances j of our AES implementation on ATmega16 and compared this to the signal variance in the same time points on the same platform, $\mathrm{var}(s_{i,j})$, $a_i \in \mathbb{F}_2^8$, which is known to be a good indicator of the points leaking information in DPA. This comparison is represented in Figure 7.11 for two clock cycles of our 8-bit table look-up operation.

7.7.5 Adaptive Decision Threshold

To further increase the success probability of the collision attacks, it is crucial to minimize the type II error probability α_2 of collision detection: A high type II error probability α_2 would result in an erroneous system of equations, which would be either inconsistent or deliver incorrect keys, the probability that a wrong system of equations leads to a correct solution being negligible.

In a real-world noisy environment, there are always collision detection errors occurring with certain probabilities. The decision threshold for accepting a pair of

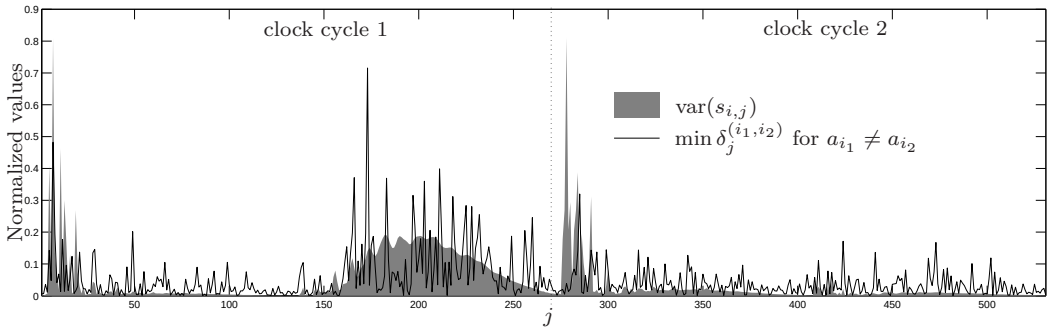


Figure 7.11: Informative points for collision detection and DPA

S-box instances as a collision can be selected in a way minimizing the type II error probability (the probability that a non-collision has been detected as a collision). For a given attack, it can be even chosen to eliminate all type II errors, since there are only a finite number of S-box pairs occurring in this attack. Of course, this is only possible at the expense of increasing the type I error probability α_1 (the probability of omitting a true collision).

Thus, in the presence of collision detection errors, we adjust the decision threshold Y^{BC} of the collision detection procedure $\mathfrak{T}^{BC}$ in a way minimizing the type II error probability α_2 and somewhat increase the number of inputs γ (this is an *adaptive threshold*). In the real-world setting, an optimal decision threshold Y^{BC} can be found by trying a number of thresholds in some area of values and running the collision detection several times. Most of the non-optimal threshold values can be filtered out by looking at the number of detected collisions, without performing the key recovery.

7.8 Multiple-Differential Collision Detection

The goal of the collision detection is to decide if two S-box instances in AES have had equal inputs based on side-channel traces.

For the direct binary comparison of S-box instances, the least-square based test is which is essentially a computation of the Euclidean distance between two real-valued traces. Its resolution can be increased by suppressing noise through averaging.

However, there are other collision detection methods substantially using the simple binary comparison, two of which – binary voting and ternary voting – we propose in this section. Both methods can be combined with averaging. Additionally, the ternary voting test enables performance gains through profiling.

7.8.1 Binary Voting Test

In this subsection we propose a more efficient method to suppress noise which is called *binary voting*. Like in averaging, traces for multiple copies of the same plaintexts are first obtained. However, instead of averaging, the attacker tries to detect collisions using binary comparison for each pair of the traces and applies voting to filter for correct ones.

Definition. We have to reliably detect collisions for γ different plaintexts. Then each of these plaintexts is sent M^{BV} times to the device. So we have a group $\tilde{\tau}_j^i = \{\tau_j^{i,m}\}_{m=1}^{M^{BV}}$, $\tau_j^{i,m} \in \mathbb{R}^l$, of traces for each S-box instance and each plaintext. That is, the direct application of binary voting requires $C_{\text{online}} = \gamma \cdot M^{BV}$ measurements.

The binary voting test is based on the following statistic which uses a binary comparison test (for instance, the one defined above):

$$\mathfrak{S}^{BV}(\tilde{\tau}_{j_1}^{i_1}, \tilde{\tau}_{j_2}^{i_2}) = \sum_{m=1}^{M^{BV}} \mathfrak{T}^{BC}(\tau_{j_1}^{i_1,m}, \tau_{j_2}^{i_2,m}),$$

where the multiple traces for two S-box instances are pairwisely compared to each other. The test $\mathfrak{T}^{BV}$ to decide if there has been a collision is then defined as

$$\mathfrak{T}^{BV}(\tilde{\tau}_{j_1}^{i_1}, \tilde{\tau}_{j_2}^{i_2}) = \begin{cases} 0 \text{ (no collision)}, \text{if } \mathfrak{S}^{BV}(\tilde{\tau}_{j_1}^{i_1}, \tilde{\tau}_{j_2}^{i_2}) < Y^{BV} \\ 1 \text{ (collision)}, \text{ if } \mathfrak{S}^{BV}(\tilde{\tau}_{j_1}^{i_1}, \tilde{\tau}_{j_2}^{i_2}) \geq Y^{BV}, \end{cases}$$

where Y^{BV} is a decision threshold. The idea is that the distribution of statistic $\mathfrak{S}^{BV}$ will be different for $a_{j_1}^{i_1} = a_{j_2}^{i_2}$ and for $a_{j_1}^{i_1} \neq a_{j_2}^{i_2}$.

Properties. Recall that α_1 and α_2 are type I and II error probabilities, respectively, for $\mathfrak{T}^{BC}$. As the individual binary comparisons are independent, the distribution of $\mathfrak{S}^{BV}$ is due to the binomial law with M^{BV} experiments and success probability p. If $a_{j_1}^{i_1} = a_{j_2}^{i_2}$, the success probability is $p = p_e = 1 - \alpha_1$. If $a_{j_1}^{i_1} \neq a_{j_2}^{i_2}$, it is $p = p_{ne} = \alpha_2$. For sufficiently large group sizes M^{BV}, the distribution of $\mathfrak{S}^{BV}$ can be approximated by a normal distribution $\mathcal{N}(M^{BV}p, M^{BV}p(1-p))$. That is, the problem of collision detection is reduced to the problem of distinguishing between two normal distributions in this case. Thus, the required value of M^{BV} can be obtained using

Proposition 14. *Let α_1 and α_2 be type I and II error probabilities, respectively, for $\mathfrak{T}^{BC}$. Then the number of S-box traces in each group needed to distinguish between $a_{j_1}^{i_1} = a_{j_2}^{i_2}$ and $a_{j_1}^{i_1} \neq a_{j_2}^{i_2}$ using binary voting test $\mathfrak{T}^{BV}$ can be estimated as*

$$M^{BV} \approx \frac{(u_{1-\beta_1}\sqrt{\alpha_1(1-\alpha_1)} + u_{1-\beta_2}\sqrt{\alpha_2(1-\alpha_2)})^2}{(1-\alpha_1-\alpha_2)^2},$$

where:

- β_1 *and* β_2 *are the required type I and II error probabilities for* $\mathfrak{T}^{BV}$,
- $u_{1-\beta_1}$ *and* $u_{1-\beta_2}$ *are quantiles of the standard normal distribution* $\mathcal{N}(0,1)$.

Combination with averaging. The required value of M^{BV} depends on α_1 and α_2 which in turn can be seen as functions of the noise amplitude σ. For this reason we will write $M^{BV}(\sigma)$ where this dependency is important.

The binary voting technique can be combined with averaging. The traces are first averaged t times. Then the statistic $\mathfrak{S}^{BV}$ is computed. That is, one deals with $M^{BV}(\sigma/\sqrt{t})$ instead of $M^{BV}(\sigma)$.

Since each plaintext P_i is sent $t \cdot M^{BV}(\sigma/\sqrt{t})$ times to the device, binary voting with averaging requires $C_{\text{online}} = \gamma \cdot t \cdot M^{BV}(\sigma/\sqrt{t})$ measurements. Depending on the concrete implementation and on the range of σ, the measurement complexity can be reduced, if $\gamma \cdot t \cdot M^{BV}(\sigma/\sqrt{t}) < \gamma \cdot M^{BV}(\sigma)$ for some t. In the sequel, we will refer to binary voting with averaging simply as binary voting, since binary voting with averaging for $t = 1$ corresponds to the basic binary voting.

7.8.2 Ternary Voting Test

Ternary voting is another statistical technique we propose to reliably detect collisions. It is based on indirect comparisons of traces, where two given S-box traces (*target traces*, a subset of online traces) are compared through a pool of other ones (*reference traces*, profiling traces if any and possibly a subset of online traces).

While the ternary voting test is less efficient than the binary voting one in terms of the overall number of traces needed, it allows for profiling. That is, the reference traces can be acquired in the profiling stage and shared by several attacks, which can significantly amplify the performance of the online stage.

Definition. Let N^{TV} be the number of S-box instances whose (reference) traces $\{\tau_m\}_{m=1}^{N^{TV}}$, $\tau_m \in \mathbb{R}^l$, are available to the attacker for some random unknown inputs $\{a_m\}_{m=1}^{N^{TV}}$, $a_m \in \mathbb{F}_2^8$. Let $\tau_{j_1}^{i_1}$ and $\tau_{j_2}^{i_2}$ be the traces for two further S-box instances for which we have to decide if $a_{j_1}^{i_1} = a_{j_2}^{i_2}$. Then the ternary voting test can be defined as follows:

$$\mathfrak{T}^{TV}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}) = \begin{cases} 0 \text{ (no collision)}, \text{if } \mathfrak{S}^{TV}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}) < Y^{TV} \\ 1 \text{ (collision)}, \text{ if } \mathfrak{S}^{TV}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}) \geq Y^{TV}, \end{cases}$$

where

$$\mathfrak{S}^{TV}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}) = \sum_{m=1}^{N^{TV}} F(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}, \tau_m)$$

with

$$F(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}, \tau_m) = \mathfrak{T}^{BC}(\tau_{j_1}^{i_1}, \tau_m) \cdot \mathfrak{T}^{BC}(\tau_{j_2}^{i_2}, \tau_m)$$

and Y^{TV} is some decision threshold. The key idea of ternary voting is similar to that of binary voting: The distributions of $\mathfrak{S}^{TV}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2})$ for $a_{j_1}^{i_1} = a_{j_2}^{i_2}$ and for $a_{j_1}^{i_1} \neq a_{j_2}^{i_2}$ will be different. Typically, $\mathfrak{S}^{TV}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2})$ will be higher for $a_{j_1}^{i_1} = a_{j_2}^{i_2}$ than for $a_{j_1}^{i_1} \neq a_{j_2}^{i_2}$. To decide if there has been a collision, the attacker needs to statistically distinguish between these two cases.

Properties. To explore the behaviour of F, it is not sufficient to know the type I and II error probabilities for the binary comparison test. Let $\mathfrak{T}^{BC}$ be characterized by the simultaneous distribution of the test results depending on the relations between $a_{j_1}^{i_1}$, $a_{j_2}^{i_2}$ and a_m:

$$\begin{aligned}
\chi_1 &= \Pr\{\mathfrak{T}^{BC}(\tau_{j_1}^{i_1}, \tau_m) = 1, \mathfrak{T}^{BC}(\tau_{j_2}^{i_2}, \tau_m) = 1 | a_{j_1}^{i_1} = a_{j_2}^{i_2} = a_m\},\\
\chi_2 &= \Pr\{\mathfrak{T}^{BC}(\tau_{j_1}^{i_1}, \tau_m) = 1, \mathfrak{T}^{BC}(\tau_{j_2}^{i_2}, \tau_m) = 1 | a_{j_1}^{i_1} = a_{j_2}^{i_2} \neq a_m\},\\
\chi_3 &= \Pr\{\mathfrak{T}^{BC}(\tau_{j_1}^{i_1}, \tau_m) = 1, \mathfrak{T}^{BC}(\tau_{j_2}^{i_2}, \tau_m) = 1 | a_{j_1}^{i_1} \neq a_{j_2}^{i_2}, a_{j_1}^{i_1} = a_m, a_{j_2}^{i_2} \neq a_m\},\\
\chi_4 &= \Pr\{\mathfrak{T}^{BC}(\tau_{j_1}^{i_1}, \tau_m) = 1, \mathfrak{T}^{BC}(\tau_{j_2}^{i_2}, \tau_m) = 1 | a_{j_1}^{i_1} \neq a_{j_2}^{i_2}, a_m \neq a_{j_1}^{i_1}, a_m \neq a_{j_2}^{i_2}\}.
\end{aligned}$$

Then the probabilities

$$p_e = \Pr\{F(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}, \tau_m) = 1 | a_{j_1}^{i_1} = a_{j_2}^{i_2}\}$$

and

$$p_{ne} = \Pr\{F(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}, \tau_m) = 1 | a_{j_1}^{i_1} \neq a_{j_2}^{i_2}\}$$

can be computed using

Proposition 15. *If $a_{j_1}^{i^1}, a_{j_2}^{i_2}, a_m \in \mathbb{F}_2^8$ are uniformly distributed and mutually independent, then*

$$p_e = \tfrac{1}{2^8}\chi_1 + \tfrac{2^8-1}{2^8}\chi_2$$

and

$$p_{ne} = \tfrac{2}{2^8}\chi_3 + \tfrac{2^8-2}{2^8}\chi_4.$$

Proof. If $a_{j_1}^{i_1} = a_{j_2}^{i_2}$, two cases are possible for $F(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}, \tau_m) = 1$:

- $a_{j_1}^{i_1} = a_{j_2}^{i_2} = a_m$ which happens with probability of $1/2^8$, and
- $a_{j_1}^{i_1} = a_{j_2}^{i_2} \neq a_m$ which happens with probability $\frac{2^8-1}{2^8}$.

If $a_{j_1}^{i_1} \neq a_{j_2}^{i_2}$, there are three cases leading to $F(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2}, \tau_m) = 1$:

- $a_{j_1}^{i_1} = a_m$, $a_{j_2}^{i_2} \neq a_m$ with probability $1/2^8$,
- $a_{j_2}^{i_2} = a_m$, $a_{j_1}^{i_1} \neq a_m$ with probability $1/2^8$, and
- $a_{j_1}^{i_1} \neq a_m$, $a_{j_2}^{i_2} \neq a_m$ with probability $(2^8 - 2)/2^8$.

The claims of the proposition follow. □

For the sake of simplicity, we first study the properties of $\mathfrak{T}^{TV}$ under the assumption that all applications of F to compute $\mathfrak{S}^{TV}$ are mutually independent. Under this assumption, $\mathfrak{S}^{TV}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2})$ would have a binomial distribution with N^{TV} being the number of experiments and success probability $p = p_e$, if $a_{j_1}^{i_1} = a_{j_2}^{i_2}$, or $p = p_{ne}$, if $a_{j_1}^{i_1} \neq a_{j_2}^{i_2}$. Thus, for sufficiently large values of N^{TV}, $\mathfrak{S}^{TV}(\tau_{j_1}^{i_1}, \tau_{j_2}^{i_2})$ could be approximated by normal distribution $\mathcal{N}(N^{TV}p, N^{TV}p(1-p))$. Thus, similarly to binary voting, the number N^{TV} of S-box reference instances needed to distinguish between $a_{j_1}^{i_1} = a_{j_2}^{i_2}$ and $a_{j_1}^{i_1} \neq a_{j_2}^{i_2}$ could be estimated as

$$N^{TV} \approx \frac{(u_{1-\beta_1}\sqrt{p_e(1-p_e)} + u_{1-\beta_2}\sqrt{p_{ne}(1-p_{ne})})^2}{(p_e - p_{ne})^2},$$

where β_1 and β_2 are the required type I and II error probabilities for $\mathfrak{T}^{TV}$, $u_{1-\beta_1}$ and $u_{1-\beta_2}$ are quantiles of the standard normal distribution $\mathcal{N}(0,1)$.

However, the applications of F are dependent and this result can be only used to obtain a rough estimation of N^{TV}.

Procedure, complexity, averaging. Now we can describe the basic procedure of ternary voting in the case that the target key is fixed in the device and the plaintexts are random and known. This is what we call *ternary voting without profiling*.

The number N^{TV} of S-box reference instances as well as the number M^{TV} of different inputs for which reference traces have to be acquired depend on the noise level σ. We will write $N^{TV}(\sigma)$ and $M^{TV}(\sigma)$, whenever this dependency is crucial for understanding.

First, the attacker obtains traces for $M^{TV}(\sigma)$ random plaintexts. This yields τ_m for $N^{TV}(\sigma) = 160 \cdot M^{TV}(\sigma)$ different S-box instances for AES-128, if the key schedule is not considered and all the $16 \cdot 10$ S-box traces within each AES run are acquired at a time. Then, if $M^{TV}(\sigma) \geq \gamma$, no further measurements are needed. Otherwise, the

attacker acquires traces for further $\gamma - M^{TV}(\sigma)$ plaintexts. Note that some of the reference traces can be interpreted as target traces (16 S-box traces corresponding to the first round in each of some γ executions of AES).

This yields the complexity of $C_{\text{online}} = \max(\gamma, M^{TV}(\sigma))$ measurements, where

$$M^{TV}(\sigma) = \left\lceil \frac{N^{TV}(\sigma)}{160} \right\rceil .$$

Like binary voting, ternary voting can be combined with averaging to achieve better resolution. In this case each trace has to be averaged t times. Thus, the complexity of ternary voting with averaging is $C_{\text{online}} = t \cdot \max(\gamma, M^{TV}(\sigma/\sqrt{t}))$. In the sequel we refer to ternary voting both with and without averaging simply as ternary voting.

Profiling. Now we are ready to describe what we refer to as *ternary voting with profiling*. Unlike binary voting, the method of ternary voting allows for profiling. In the profiling stage, reference traces are acquired only, for which the attacker has to know *neither* the key used *nor* the plaintexts. Moreover, this also works if keys are changed between blocks of t executions. The target traces are obtained in the online phase and compared based on the pre-measured reference traces.

Thus, $C_{\text{profiling}} = t \cdot M^{TV}(\sigma/\sqrt{(t)})$ measurements have to be performed in the profiling stage, each measurement comprising all 10 rounds of AES-128. Then only $C_{\text{online}} = t \cdot \gamma$ measurements are needed in the online stage, each measurement comprising only the first round for the linear key recovery. For the latter measurements we do have to know inputs. Moreover, they have to be performed with the key to be recovered.

7.8.3 On the Error Probabilities of Collision Detection

The measurement complexity of the binary and ternary voting methods depends on the success probability to be achieved. Let us take $\mathcal{P}$ as a desirable success probability of the entire attack and estimate the required type II error probabilities β_2 for binary and ternary voting. Recall that π is the success probability of the cryptanalytic collision attack used to recover the key after the collisions have been detected.

In the linear key recovery, there are 16γ S-box instances between which a collision can occur. That is, the voting has to be performed $w = \binom{16\gamma}{2}$ times. If one assumes the applications of tests to be statistically independent, then β_2 can computed as $\beta_2 = 1 - (\mathcal{P}/\pi)^{1/w}$. For instance, if $\gamma = 6$ and $\mathcal{P} = 0.5$, one obtains $\beta_2 \approx 1.174 \cdot 10^{-4}$, in other words, it has to be very close to zero. Additionally, β_1 has to be low enough to enable the detection of a sufficient number of collisions.

7.9 Experiments with Simulated Traces

The purpose of this section is to estimate the efficiency of different MDCA variants based on an AES implementation example and to compare the methods to the standard Hamming-weight based DPA for the same AES implementation. In order to be able to perform this comparison for different noise levels σ, we simulated the deterministic power consumption in Nanosim using dedicated power simulation libraries and added Gaussian noise of different amplitudes to it. The main results of the section are summarized in Table 7.6.

Note that we deal with an MDCA-friendly hardware implementation of AES. That is, the quantitative results of this section should be interpreted as proof-of-concept results, though indicating that multiple-differential collision detection is highly efficient in some cases.

7.9.1 Implementation and Simulation

The characteristics of $\mathfrak{T}^{BC}$ strongly depend on the signal-to-noise ratio of the implementation. To perform the estimations for a variety of noise levels, a serial VHDL implementation of the AES S-box has been performed (that is, only one S-box is calculated at a time). The deterministic power consumption for all 2^8 inputs was simulated using Synopsys Nanosim with the Dolphin Integration power consumption library SESAME-LP2 based on a 250nm technology by IHP [89]. The design was clocked at 10 MHz. The sampling rate was set to 10 Gsamples/s.

The S-box was implemented as combinatorial logic on the basis of an 8-bit register. Each S-box calculation $y = S(x)$ occurs in two clocks. In the first clock, the input x is read from the register and the output y is computed. In the second clock, the register is set to zero and the calculated output y is written to the register.

The simulated deterministic power traces obtained are noise-free. That is, there is neither electronic noise (power supply noise, clock generator noise, conducted emissions, radiated emissions, etc.) nor algorithmic noise (since only the relevant part of the circuit is considered) in these traces. To model noise we added random values due to univariate normal distribution[5] with the zero mean value and a standard deviation σ whose value characterizes the noise amplitude.

Note also that the simulated signal was not subject to a low-pass filter as it would have been the case for the real-world measurements of power consumption due to the presence of capacitances within the chip as well as on the circuit board where the power consumption measurements are performed. This would have cut

[5]Normal distribution is a sound noise model [175]. As a matter of fact, the noise is often distributed due to the multivariate normal distribution [175], [167]. However, only a few co-variances in the co-variance matrix of this multivariate normal distribution significantly differ from zero [167] for many implementations.

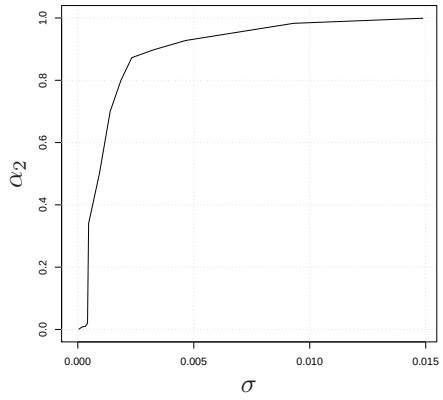


Figure 7.12: Type II error probability α_2 for $\mathfrak{T}^{BC}$ as a function σ

off the high-frequency contribution to the signal reducing the advantage of high-resolution measurements. However, the effect of this circumstance is rather limited for the measurements of the electromagnetic radiation. A major limitation in this case is the bandwidth of the oscilloscope. Thus, we believe that the simulated traces with added Gaussian noise can be used for an initial analysis of the efficiency of our collision detection techniques. The main advantage of using the simulated power consumption is that one can add noise of different amplitudes to model the behaviour of attack methods for different devices and physical conditions.

To evaluate α_2 for this implementation, we chose Y^{BC} in $\mathfrak{T}^{BC}$ so that α_1 becomes sufficiently low by shifting Y^{BC} to the right. For this value of α_1, the type II error probability α_2 was estimated experimentally by executing $\mathfrak{T}^{BC}$ for random equal and unequal inputs to the S-box. We performed that for several noise amplitudes σ. The results can be found in Figure 7.12. Though this cannot be seen as a complete characterization of $\mathfrak{T}^{BC}$, the figure is meant to illustrate the intuition behind the multiple-differential collision detection methods.

7.9.2 Reference Figures for DPA

We compared the efficiency of MDCA with binary and ternary voting to DPA [53] based on Hamming weight. The Hamming-weight power consumption model is sound for the implementation in question, since the register is first set to zero and then re-written with the target byte value. DPA was applied to the same simulated traces with the same noise amplitudes as MDCA. The number of measurements needed by DPA is denoted by C_{DPA}.

For our comparison, it was assumed that traces for all 16 S-boxes in the first round are acquired within one measurement. This is very similar to MDCA based

on linear key recovery considered in this chapter: The traces corresponding to the 16 S-box calculations in the first round are acquired at a time in the online stage for binary voting and ternary voting with profiling.

The number of measurements needed for DPA can be potentially reduced if guessing entropy is allowed in the offline stage of DPA. To treat this point, we assumed that DPA is successful, if it returns a correct 8-bit key chunk with probability 0.5. At the same time, it was assumed for all collision attacks that the needed success probability of the complete attack is $\mathcal{P} = 0.5$. That is, a collision attack on AES is successful, iff it returns the correct 16-byte key with probability ≥ 0.5.

Note that the type of leakage is also important for collision attacks. The right choice of a power consumption model allows the attacker to perform binary comparison more efficiently. Here the consideration was restricted to the Euclidean distance of two real-valued vectors. However, other binary comparison tests can turn out to be more consistent with the power consumption of other implementations. The choice of binary comparison test implicitly fixes a power consumption model for collision detection. In other words, different binary comparison tests may turn out optimal for different types of leakage.

7.9.3 Online and Profiling Complexity of MDCA

In this subsection, C_{online} and $C_{\text{profiling}}$ for MDCA based on binary voting and ternary voting both with and without profiling are experimentally derived for the given implementation. The estimations are performed for the linear key recovery method with $\gamma = 6$.

Table 7.6: C_{online} against different values of σ for $\mathfrak{T}^{BV}$, $\mathfrak{T}^{TV}$ without profiling, $\mathfrak{T}^{TV}$ with profiling and C_{DPA}

$10^3\sigma$	0.46	0.93	2.32	3.25	4.65	6.97	9.30	11.62	13.95
C_{online}, $\mathfrak{T}^{BV}$	60	192	276	468	960	1290	1872	2976	4242
C_{online}, $\mathfrak{T}^{TV}$ w/o profiling	80	390	2605	5200	10640	23840	42320	66080	95200
C_{online}, $\mathfrak{T}^{TV}$ with profiling	6	6	6	6	6	18	30	60	120
C_{DPA}, HW based DPA	163	349	1645	4192	6912	15676	26341	39348	56025

Binary voting. Figure 7.13 and Table 7.6 give experimental values of C_{online} for the binary voting test in a range of noise amplitudes. The values of t have been chosen that minimize the resulting number of traces needed. If σ' is the noise amplitude to be attained by averaging and σ is the given noise level, then one has to average about $t = (\sigma/\sigma')^2$ times. Thus, $C_{\text{online}} \approx \gamma \frac{\sigma^2}{\sigma'^2} M^{BV}(\sigma')$. The results demonstrate that binary voting is well-suited for our implementation providing an advantage of factor 2.7 to 13.2 for a wide range of σ.

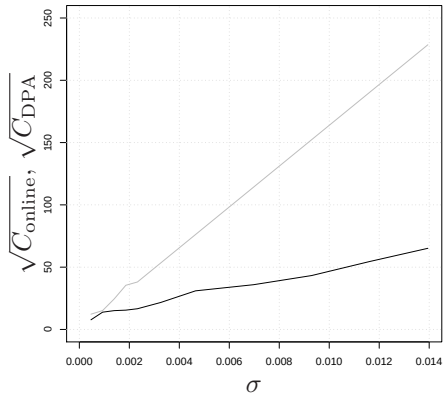


Figure 7.13: Binary voting test against DPA: C_{online} (black line) and C_{DPA} (grey line) as functions of σ

Ternary voting without profiling. Figure 7.14 and Table 7.6 give concrete values of C_{online} in this case for a range of noise amplitudes. Values of t were chosen that minimize C_{online}. The performance of the ternary voting test without profiling is comparable to DPA. However, ternary voting without profiling does not exhibit any advantages over DPA in terms of measurement complexity.

Ternary voting with profiling. For a given σ, the attacker can reduce t which leads to a linear decrease of C_{online} and to a considerable growth of $C_{\text{profiling}}$ due to the slope of M^{TV} as a function of the noise amplitude (see Figure 7.14 for this dependency). We assumed that $\leq 10^6$ measurements in the profiling stage are feasible. To obtain the lowest possible online complexity within this bound on the profiling complexity, we chose t that minimizes C_{online} with $C_{\text{profiling}} \leq 10^6$ for each interesting value of σ. The resulting values of C_{online} and $C_{\text{profiling}}$ are depicted in Figure 7.15. The values of C_{online} can be also found in Table 7.6. Note that there is a wide spectrum of parameter choices: If there are more severe limits on $C_{\text{profiling}}$, then t and C_{online} increase. And the other way round: If the attack scenario admits for higher values of $C_{\text{profiling}}$, C_{online} can be further reduced.

The complexity estimations for ternary voting were performed under the assumption that the attacker is able to acquire the reference traces for all S-boxes in each of the 10 AES rounds at a time. If one deals with a short-memory oscilloscope, $C_{\text{profiling}}$ increases in a linear way with respect to the decrease of the available memory volume. However, only measurements for the first round are needed for the target traces, if the linear key recovery is used.

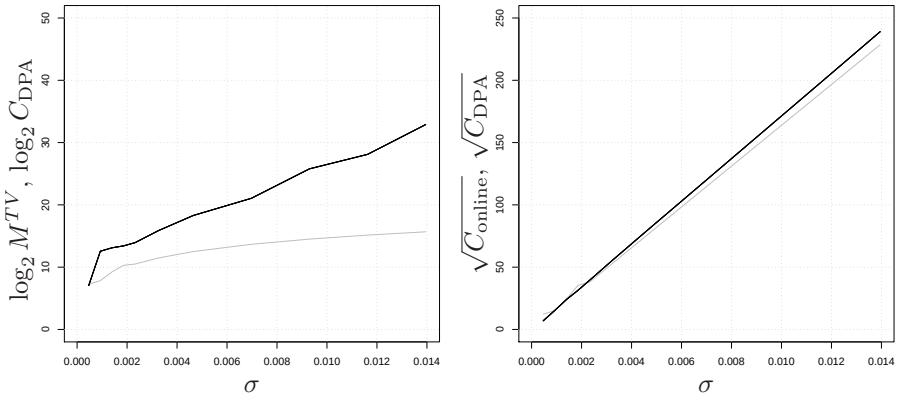


Figure 7.14: Ternary voting test without profiling against DPA: $M^{TV}(\sigma)$ (on the left, black line) and C_{online} (on the right, black line) as well as C_{DPA} (both graphics, grey lines) as functions of σ

7.10 Real-World Experimental Validation

7.10.1 AES Implementation and Measurement Set-Up

We performed our attacks for a typical AES implementation on the Atmel ATmega16 microcontroller, an RISC microcontroller from the 8-bit AVR family with a Harvard architecture. 8-bit AVR microcontrollers are widely used in embedded devices. To run a collision attack, the attacker has to know when the AES S-boxes are executed. So we measured the power consumption of the table look-ups corresponding to the relevant S-box applications. These include instances in SubBytes, ShiftRows, and MixColumns operations.

The typical AES operations involving a single state byte were implemented in the AVR assembly as follows.

```
; SubBytes
  mov  ZL, R16   ; load input byte, 1 cycle
  lpm            ; perform S-Box lookup, 3 cycles
  st Y, R0       ; store S-Box output byte into SRAM, 2 cycles
; ShiftRows
  ld R1, Y       ; load state byte, 2 cycles
  st X, R2       ; store it to another location in SRAM, 2 cycles
; pre-MixColumns
  ld R2, X       ; load it again, 2 cycles
```

S-box lookup table was stored in the program memory and accessed with the `LPM` instruction. SRAM was used to store the state byte. The trigger signal was raised by

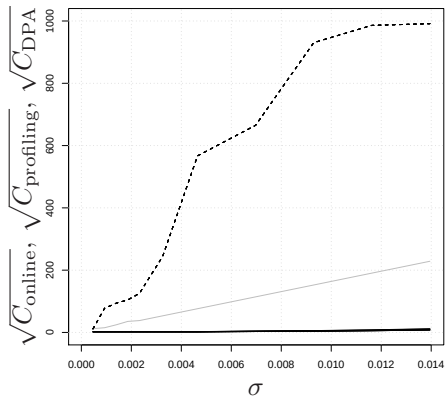


Figure 7.15: Ternary voting test with profiling: C_{online} (solid black line), $C_{\text{profiling}} \leq 10^6$ (dashed black line) and C_{DPA} (solid grey line) as functions of σ

the implementation on a pin of the microcontroller prior to execution of the above instructions.

The microcontroller was clocked at 3.68 MHz and supplied with an operating voltage of 5V from a standard laboratory power source. The variations of the power consumption were observed on a shunt resistor of 5.6 Ohm inserted into the ground line of the microcontroller. The measurements were performed with a LeCroy WaveRunner 104MXi DSO equipped with ZS1000 active probe. The DSO has 8-bit resolution and 1 GHz input bandwidth (with the specified probe). The acquisitions were performed at the maximum sampling rate of 10 GS/s without any input filters. We stress that this measurement setup is *not* noise-optimized[6].

With the given shunt resistor and 160 mV peak-to-peak vertical resolution, variations of the consumed current as small as 110 μA could be observed, while the average power consumption of the microcontroller in the given configuration was about 12 mA. The DSO, the power source and the target device shared the common ground, also connected to the Faraday cage.

The DSO was controlled remotely via Gigabit Ethernet connection by the MATLAB script running on the host PC. The same script was sending the inputs to the

[6]As in case of DPA [61], collision detection methods tend to be sensitive to the preprocessing of measured signals. To denoise the traces, we proceed in two steps. First, the traces are decimated by applying a low-pass filter to the original traces and subsequently resampling them at a lower rate. Additionally to noise reduction, this weakens time jitter. Second, the decimated traces are denoised by applying a wavelet decomposition at a certain level, thresholding the detail coefficients, and subsequent wavelet reconstruction. Our experiments show that symlets proposed by Daubechies (first of all, the 'sym3' wavelet) are most suitable for this operation.

microcontroller over the serial line. The DSO was set to the sequence acquisition mode, which enabled to achieve the acquisition rate of 150 traces per second. Each trace consisted of 46000 samples and was decimated 10 times to 4600 samples and denoised with wavelets. Pre-processed traces were stored along with the corresponding input bytes.

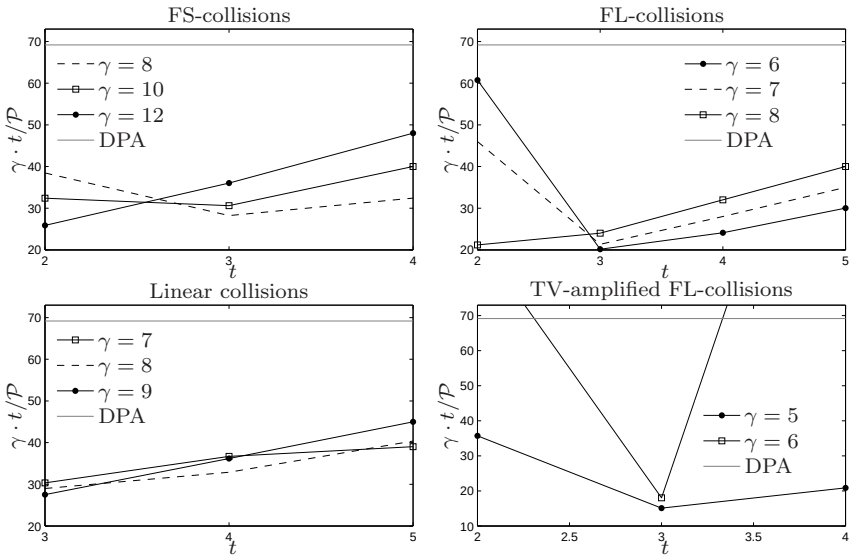


Figure 7.16: Performance of collision attacks based on FS-, FL- and linear collisions without profiling as well as of FL-collision based attacks with profiling (ternary voting with 625 profiling measurements) vs. Hamming-weight based DPA on the same traces

7.10.2 Attack Scenarios and Results

Performance metric. We use the following efficiency metric to compare the performance of all these attacks: $t \cdot \gamma/\mathcal{P}$, where t is the number of averagings, γ is the number of different inputs, and $\mathcal{P}$ is the success probability of the attack. In case of the Hamming-weight based DPA we apply a similar metric: n/p^{16}, where n is the number of measurements needed to determine a single-byte chunk of the AES key with probability p. These metrics characterize the expected number of measurements needed to recover the whole 16-byte key. The performance results for DPA can be found in Figure 7.16.

Collision attacks without profiling. In the online stage, an attacker observes executions of AES for γ random inputs, each repeated t times. Then, the $t \cdot \gamma$ traces are averaged t times and one obtains γ averaged traces for γ random inputs. These

are used to detect collisions – linear, FS- or FL-collisions (see Sections 7.5 and 7.6) depending on the key-recovery method – with the direct binary comparison (see Subsection 7.7.1). All key-recovery methods need AES plaintexts to be known. Additionally, if the attack is based on FL-collisions, the corresponding AES ciphertexts have to be known.

Note that since the adaptive threshold Y^{BC} is used in the direct binary comparison which eliminates all type II errors, many true collisions are omitted due to the increased type I error probability by shifting Y^{BC} to the right. That is, for given γ and α_1, the success probability $\mathcal{P}$ of the whole attack follows the dependencies illustrated in Figure 7.10 for different key-recovery methods.

Profiling-amplified collision attacks. If profiling is possible prior to the online stage, the ternary voting method (see Subsection 7.8.2) can be used to detect additional collisions, which are omitted by the direct binary comparison due to the application of the adaptive threshold. Taking additional collisions is equivalent to the decrease of α_1, which further improves the performance metric $t \cdot \gamma / P$. In our profiling-amplified attacks we used $N = 10^5$ profiling S-box traces τ_i which is equivalent to about $C_{\text{profiling}} = 10^5/160 = 625$ executions of AES in the profiling stage. See Figure 7.16 for concrete results of collision attacks with profiling.

7.11 Conclusions

This chapter provides a many-fold technical framework considerably improving on the efficiency of side-channel collision attacks on AES. We have demonstrated how collision attacks can be extended to other-type cryptographic constructions based on AES, introduced the idea of generalized internal collisions and applied it to produce linear and nonlinear systems of equations for AES as well as given a toolbox for the robust detection of collisions in the presence of strong noise.

The combination of these improvements results in a performance of side-channel attacks being comparable and in many cases even superior to that of differential side-channel analysis without profiling and template-like attacks with profiling. Most interestingly, our collision attacks are essentially based on the internal structure of the attacked algorithms. Moreover, the types of side-channel leakage leading to successful collision detection can differ greatly.

Techniques similar to the ones described in this chapter might turn out applicable to other symmetric constructions such as stream ciphers or asymmetric constructions such as digital signature schemes. There can be also some potential in using collision-based methods to overcome certain random masking schemes for block ciphers.

Bibliography

[1] Advanced Encryption Standard. FIPS. Publication 197. National Bureau of Standards, U.S. Department of Commerce, 2001.

[2] M. Albrecht, C. Cid. Algebraic Techniques in Differential Cryptanalysis. FSE'09, LNCS, Springer-Verlag, 2009.

[3] Y. An and S. Oh. RFID System for User's Privacy Protection. IEEE Asia-Pacific Conference on Communications, IEEE Computer Society, 2005.

[4] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, T. Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. SAC'00, Springer-Verlag, 2000.

[5] C. Archambeau, E. Peeters, F.-X. Standaert, J.-J. Quisquater. Template Attacks in Principal Subspaces. CHES'06, LNCS, Springer-Verlag, 2006.

[6] G. Avoine and P. Oechslin. A Scalable and Provably Secure Hash-based RFID Protocol. 3rd IEEE PerCom'05, IEEE Computer Society, 2005.

[7] T. Baigneres, P. Junod, S. Vaudenay. How Far Can We Go beyond Linear Cryptanalysis? ASIACRYPT'04, LNCS, Springer-Verlag, 2004.

[8] S. Balasubramanian, A. Bogdanov, A. Rupp, J. Ding, H. W. Carter. Fast Multivariate Signature Generation in Hardware: The Case of Rainbow. ASAP'08, 2008.

[9] M. Bardet, J.-C. Faugere, B. Salvy. Complexity of Grobner Basis Computation for Semi-Regular Overdetermined Sequences over $\mathbb{F}_2$ with Solutions in $\mathbb{F}_2$. Technical Report 5049, INRIA, 2003.

[10] P. Baretto, V. Rijmen. The Whirlpool Hashing Function, `paginas.terra.com.br/informatica/-paulobarreto/WhirlpoolPage.html`.

[11] E. Barkan, E. Biham. Conditional Estimatores: An Effective Attack on A5/1. SAC'05, LNCS, Springer-Verlag, 2006.

[12] E. Barkan, E. Biham, N. Keller. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communications. CRYPTO'03, LNCS, Springer-Verlag, 2003.

[13] D.J. Bernstein. Cache-Timing Attacks on AES. Available from: `cr.yp.to/antiforgery/cachetiming-20050414.pdf`, 2005.

[14] E. Biham. How to Decrypt or Even Substitute DES-Encrypted Messages in 2^{28} Steps. Information Processing Letters, 84, 2002.

[15] E. Biham. New Types of Cryptanalytic Attacks Using Related Keys. J. Cryptology, vol. 7, no. 4, pages 229–246, 1994.

[16] E. Biham, A. Biryukov, A. Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. EUROCRYPT'99, LNCS, Springer-Verlag, 1999.

[17] E. Biham, O. Dunkelman. Cryptanalysis of the A5/1 GSM Stream Cipher. INDOCRYPT'00, LNCS, Springer-Verlag, 2000.

[18] E. Biham, O. Dunkelman. A Framework for Iterative Hash Functions - HAIFA. Presented at Second NIST Cryptographic Hash Workshop, 2006. Available via `csrc.nist.gov/groups/ST/hash/`.

[19] E. Biham, O. Dunkelman, S. Indesteege, N. Keller, B. Preneel. How To Steal Cars – A Practical Attack on KeeLoq. EUROCRYPT'08, LNCS, Springer-Verlag, 2008.

[20] E. Biham, O. Dunkelman, N. Keller. Improved Slide Attacks. FSE'07, LNCS, Springer-Verlag, 2007.

[21] E. Biham, O. Dunkelman, N. Keller. The Rectangle Attack - Rectangling the Serpent. EUROCRYPT'01, LNCS, Springer-Verlag, 2001.

[22] E. Biham, L.R. Knudsen, R.J. Anderson. Serpent: A New Block Cipher Proposal. FSE'98, LNCS, Springer-Verlag, 1998.

[23] E. Biham, A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. J. Cryptology, 4, pages 3–72, 1991.

[24] E. Biham, A. Shamir. Differential Cryptanalysis of the Data Encryption Standard. Springer-Verlag, 1993.

[25] E. Biham, A. Shamir. Power Analysis of the Key Scheduling of the AES Candidates. In Proceedings of the Second Advanced Encryption Standard (AES) Candidate Conference, Rome, 1999. Available from: `csrc.nist.gov/CryptoToolkit/aes/round1/conf2/papers/biham3.pdf`, 1999.

[26] A. Biryukov, A. Bogdanov, D. Khovratovich, T. Kasper. Collision Attacks on AES-Based MAC: Alpha-MAC. CHES'07, LNCS, Springer-Verlag, 2007.

[27] A. Biryukov, D. Khovratovich. Two New Techniques of Side-Channel Cryptanalysis. CHES'07, LNCS, Springer-Verlag, 2007.

[28] A. Biryukov, S. Mukhopadhyay, P. Sarkar. Improved Time-Memory Trade-offs with Multiple Data. SAC'05, LNCS, Springer-Verlag, 2005.

[29] A. Biryukov, A. Shamir. Structural Cryptanalysis of SASAS. EUROCRYPT'01, LNCS, Springer-Verlag, 2001.

[30] A. Biryukov, A. Shamir, D. Wagner. Real Time Cryptanalysis of A5/1 on a PC. FSE'00, LNCS, Springer-Verlag, 2001.

[31] A. Biryukov, D. Wagner. Advanced Slide Attacks. EUROCRYPT'00, LNCS, Springer-Verlag, 2000.

[32] A. Biryukov, D. Wagner. Slide Attacks. FSE'99, LNCS, Springer-Verlag, 1999.

[33] J. Black, P. Rogaway A Block-Cipher Mode of Operation for Parallelizable Message Authentication. EUROCRYPT '02, LNCS, Springer-Verlag, 2002.

[34] J. Black, P. Rogaway. CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. CRYPTO'00, LNCS, Springer-Verlag, 2000.

[35] J. Black, P. Rogaway, T. Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. CRYPTO'02, LNCS, Springer-Verlag, 2002.

[36] A. Bogdanov. Attacks in the KeeLoq Block Cipher and Authentication Systems. RFIDSec 2007, Malaga, Spain, 2007.

[37] A. Bogdanov. Improved Side-Channel Collision Attacks on AES. SAC'07, LNCS, Springer-Verlag, 2007.

[38] A. Bogdanov. Linear Slide Attacks on the KeeLoq Block Cipher. INSCRYPT'07, LNCS, Springer-Verlag, 2007.

[39] A. Bogdanov. Multiple-Differential Side-Channel Collision Attacks on AES. CHES'08, LNCS, Springer-Verlag, 2008.

[40] A. Bogdanov. On the Differential Trails of Unbalanced Feistel Networks with Contracting MDS Diffusion. WCC'09, Bergen, 2009.

[41] A. Bogdanov On Unbalanced Fesitel Networks with Contracting MDS Diffusion. Designs, Codes, and Cryptography, Springer-Verlag, 2010. To appear.

[42] A. Bogdanov, T. Eisenbarth, A. Rupp. A Hardware-Assisted Realtime Attack on A5/2 without Precomputations. CHES'07, LNCS, Springer-Verlag, 2007.

[43] A. Bogdanov, I. Kizhvatov, A. Pyshkin. Algebraic Methods in Side-Channel Collision Attacks and Practical Collision Detection. INDOCRYPT'08, LNCS, Springer-Verlag, 2008.

[44] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. CHES'07, LNCS, Springer-Verlag, 2007.

[45] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin. Hash Functions and RFID Tags : Mind The Gap. CHES'08, LNCS, Springer-Verlag, 2008.

[46] A. Bogdanov, M. Mertens, C. Paar, J. Pelzl, and A. Rupp. A Parallel Hardware Architecture for Fast Gaussian Elimination over GF(2). FCCM'06, IEEE Computer Society, 2006.

[47] A. Bogdanov, M. Mertens, C. Paar, J. Pelzl, A. Rupp. SMITH - a Parallel Hardware Architecture for Fast Gaussian Elimination over GF(2). SHARCS'06, 2006.

[48] A. Bogdanov, C. Paar. On the Security and Efficiency of Real-World Lightweight Authentication Protocols. SECSI'08, Berlin, 2008.

[49] A. Bogdanov, A. Pyshkin. Algebraic Side-Channel Collision Attacks on AES. Available from `eprint.iacr.org/2007/477`, 2007.

[50] S. Bono, M. Green, A. Stubblefield, A. Rubin, A. Juels, M. Szydlo. Security Analysis of a Cryptographically Enabled RFID device. 14th USENIX Security Symposium, 2005.

[51] J. Borst, L. R. Knudsen, V. Rijmen. Two Attacks on Reduced IDEA. EUROCRYPT'97, LNCS, Springer-Verlag, 1997.

[52] M. Briceno, I. Goldberg, D. Wagner. A Pedagogical Implementation of the GSM A5/1 and A5/2 "Voice Privacy" Encryption Algorithms. `cryptome.org/gsm-a512.html`, 1999.

[53] E. Brier, C. Clavier, F. Olivier. Correlation Power Analysis with a Leakage Model. CHES'04, LNCS, Springer-Verlag, 2004.

[54] L. Brown, J. Pieprzyk, J. Seberry. LOKI - A Cryptographic Primitive for Authentication and Secrecy Applications. AUSCRYPT'90, LNCS, Springer-Verlag, 1990.

[55] B. Buchberger. Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. Ph.D. thesis, Universitaet Innsbruck, 1965.

[56] C. De Canniere, I. Dinur, A. Shamir. New Generic Attacks Which are Faster Than Exhaustive Search. Presented at the Rump Session of FSE'09, 2009.

[57] C. De Canniere, O. Dunkelman, M. Knezevic: KATAN and KTANTAN – A Family of Small and Efficient Hardware-Oriented Block Ciphers. In C. Clavier and K. Gaj, editors, CHES'09, vol. 5747 of LNCS, Springer-Verlag (2009) 272–288

[58] C. de Cannière, B. Preneel. Trivium. Available via `www.ecrypt.eu.org`.

[59] D. Chang. A Practical Limit of Security Proof in the Ideal Cipher Model: Possibility of Using the Constant As a Trapdoor In Several Double Block Length Hash Functions. IACR Cryptology ePrint Archive, Report 2006/481. Available from `eprint.iacr.org/2006/481`, 2006.

[60] S. Chari, J. R. Rao, P. Rohatgi. Template Attacks. CHES'02, LNCS, Springer-Verlag, 2002.

[61] X. Charvet, H. Pelletier. Improving the DPA Attack Using Wavelet Transform. NIST Physical Security Testing Workshop, 2005.

[62] B. Chor, O. Goldreich, J. Hastad, J. Fridman, S. Rudich, R. Smolensky. The Bit Extraction Problem or t-Resilient Functions. 26th Symposium on Foundations of Computer Science, 1985.

[63] C. Cid, G. Leurent. An Analysis of the XSL Algorithm. ASIACRYPT'05, LNCS, Springer-Verlag, 2005.

[64] C. Cid, S. Murphy, M. Robshaw. Algebraic Aspects of the Advanced Encryption Standard. Springer-Verlag, 2006.

[65] C. Cid, S. Murphy, M.J.B. Robshaw. Small Scale Variants of the AES. FSE'05, LNCS, Springer-Verlag, 2005.

[66] B. Collard, F.-X. Standaert. A Statistical Saturation Attack against the Block Cipher PRESENT. CT-RSA'09, LNCS, Springer-Verlag, 2009.

[67] D. Coppersmith. The Data Encryption Standard (DES) and Its Strength against Attacks. IBM Journal of Research and Development 38 (3), May 1994.

[68] D. Coppersmith, S. Pilpel, C.H. Meyer, S.M. Matyas, M.M. Hyden, J. Oseas, B. Brachtl, M. Schilling. Data Authentication Using Modification Detection Codes Based on a Public One Way Encryption Function. U.S. Patent No. 4,908,861, March 13, 1990.

[69] N. Courtois. Self-Similarity Attacks on Block Ciphers and Application to KeeLoq. WCC'09, Ullensvang, 2009.

[70] N. Courtois, G.V. Bard. Algebraic and Slide Attacks on KeeLoq. Available at `eprint.iacr.org/2007/062`, 2007.

[71] N.T. Courtois, G.V. Bard, A. Bogdanov. Periodic Ciphers with Small Blocks and Cryptanalysis of KeeLoq. Tatra Mt. Math. Publ. 41, 2008.

[72] N. Courtois, G. V. Bard, D. Wagner. Algebraic and Slide Attacks on KeeLoq. FSE'08, LNCS, Springer-Verlag, 2008.

[73] N. Courtois, A. Klimov, J. Patarin, A. Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. EUROCRYPT'00, LNCS, Springer-Verlag, 2000.

[74] N. Courtois, J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. ASIACRYPT'02, LNCS, Springer-Verlag, 2002.

[75] D. Cox, J. Little, D. O'Shea. Using Algebraic Geometry. Graduate Texts in Mathematics, vol. 185, Springer-Verlag, 2nd edition, 2004.

[76] J. Daemen. Cipher and Hash Function Design Strategies Based on Linear and Differential Cryptanalysis. PhD Thesis, K.U.Leuven, March 1995.

[77] J. Daemen, L.R. Knudsen, V. Rijmen. The Block Cipher Square. FSE'97, LNCS, Springer-Verlag, 2005.

[78] J. Daemen, V. Rijmen. A New MAC Construction Alred and a Specific Instance Alpha-MAC. FSE'05, LNCS, Springer-Verlag, 2005.

[79] J. Daemen, V. Rijmen. The Design of Rijndael — The Advanced Encryption Standard, Springer-Verlag, 2002.

[80] J. Daemen, V. Rijmen. The Pelican MAC Function. Available at `eprint.iacr.org/2005/088.pdf`, 2005.

[81] I. Damgård. A Design Principle for Hash Functions. CRYPTO'89, LNCS, Springer-Verlag, 1989.

[82] Data Encryption Standard: FIPS. National Bureau of Standards, U.S. Department of Commerce, 1977.

[83] R.D. Dean. Formal Aspects of Mobile Code Security. Ph.D. thesis, Princeton University, 1999.

[84] C. Diem. The XL-Algorithm and a Conjecture from Commutative Algebra. Asiacrypt'04, LNCS, Springer-Verlag, 2004.

[85] W. Diffie, M.E. Hellman. New Directions in Cryptography. IEEE Transactions on Information Theory (22:6), pages 644–654, 1976.

[86] J. Dillon: APN polynomials: An update. In *Fq9, the 9th International Conference on Finite Fields and Applications*, Dublin, Ireland, Invited talk (2009)

[87] T. Dimitriou. A Lightweight RFID Protocol to Protect against Traceability and Cloning Attacks. IEEE International Conference on Security and Privacy of Emerging Areas in Communication Networks (SecureComm'05), IEEE Computer Society, 2005.

[88] I. Dinur, A. Shamir. Cube Attacks on Tweakable Black Box Polynomials. EUROCRYPT'09, LNCS, Springer-Verlag, 2009.

[89] Dolphin. Description of the Standard Cells for the Process IHP 0.25 μm. ViC Specifications. SESAME-LP2, version 1.1, 2005.

[90] ECRYPT Network of Excellence. The Stream Cipher Project: eSTREAM. Available via `www.ecrypt.eu.org/stream`.

[91] W.F. Ehrsam, C.H.W. Meyer, J.L. Smith, W.L. Tuchman. Message Verification and Transmission Error Detection by Block Chaining. International Business Machines Corporation. US Patent 4074066, 1976.

[92] P. Ekdahl, T. Johansson. Another Attack on A5/1. IEEE Transactions on Information Theory, 49(1):284–289, 2003.

[93] EM Microelectronics. EM4170 - 125kHz CRYPTO READ/WRITE contactless identification device. EM Microelectronic-Marin SA. Available from `www.datasheetarchive.com`, 2002.

[94] European Telecommunications Standards Institute. Digital Cellular Telecommunications System (Phase 2+); Channel Coding (GSM 05.03 Version 8.5.1 Release 1999). `www.etsi.org`, 1999.

[95] J.-C. Faugere. A New Efficient Algorithm for Computing Groebner Bases without Reduction to Zero (F5). International Symposium on Symbolic and Algebraic Computation - ISSAC'02, 2002.

[96] J.-C. Faugere. A New Efficient Algorithm for Computing Groebner Bases (F4). Journal of Pure and Applied Algebra, 139:61-88, 1999.

[97] J.-C. Faugere, P. Gianni, D. Lazard, T. Mora. Efficient Computation of Zero-dimensional Groebner Bases by Change of Ordering. Journal of Symbolic Computation, 16(4):329-344, 1993.

[98] H. Feistel. Cryptography and Computer Privacy. Scientific American, vol. 228, pages 15–23, 1973.

[99] M. Feldhofer, S. Dominikus, J. Wolkerstorfer. Strong Authentication for RFID Systems Using the AES Algorithm. CHES'04, LNCS, Springer-Verlag, 2004.

[100] M. Feldhofer C. Rechberger. A Case Against Currently Used Hash Functions in RFID Protocols. IS'06, LNCS, Springer-Verlag, 2006.

[101] FIPS PUB 186-2. Digital Signature Standard. Federal Information Processing Standards Publication, January 2000.

[102] X. Gao, Z. Xian, H. Wang, J. Shen, J. Huang, S. Song. An Approach to Security and Privacy of RFID System for Supply Chain. IEEE International Conference on E-Commerce Technology for Dynamic E-Business, IEEE Computer Society, 2004.

[103] M.R. Garey, D.S. Johnson. Computers and Intractability - A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, 1979.

[104] B. Gierlichs, L. Batina, P. Tuyls, B. Preneel. Mutual Information Analysis. A Generic Side-Channel Distinguisher. CHES'08, LNCS, Springer-Verlag, 2008.

[105] H. Gilbert, M. Minier. A Collision Attack on 7 Rounds of Rijndael. 3rd Advanced Encryption Standard Conference, National Institute of Standards and Technology, 2000.

[106] I. Goldberg, D. Wagner, L. Green. The Real-Time Cryptanalysis of A5/2. Presented at the Rump Session of Crypto'99, 1999.

[107] J. Golic. Cryptanalysis of Alleged A5 Stream Cipher. EUROCRYPT'97, LNCS, 1997.

[108] T. Good, W. Chelton, M. Benaissa. Hardware Results for Selected Stream Cipher Candidates. SASC'07, 2007. Available for download via `www.ecrypt.eu.org/stream/`.

[109] H. Handschuh, L.R. Knudsen, M.J.B. Robshaw. Analysis of SHA-1 in Encryption Mode. CT-RSA'01, LNCS, Springer-Verlag, 2001.

[110] H. Handschuh, B. Preneel. Blind Differential Cryptanalysis for Enhanced Power Attacks. SAC'06, LNCS, Springer-Verlag, 2006.

[111] M. Hell, T. Johansson, W. Meier. Grain – a Stream Cipher for Constrained Environments. Available via `www.ecrypt.eu.org`.

[112] M.E. Hellman. A Cryptanalytic Time – Memory Trade-off. IEEE Transactions on Information Theory, vol. IT-26, No. 4, July 1980.

[113] M.E. Hellman, S.K. Langford. Differential-Linear Cryptanalysis. CRYPTO'94, LNCS, Springer-Verlag, 1994.

[114] D. Henrici, J. Götze, P. Müller. A Hash-based Pseudonymization Infrastructure for RFID Systems. SecPerU'06, IEEE Computer Society Press, 2006.

[115] C. Herbst, E. Oswald, S. Mangard. An AES Implementation Resistant to Power Analysis Attacks. ACNS'06, LNCS, Springer-Verlag, 2006.

[116] H. Heys. A Tutorial on Differential and Linear Cryptanalysis. Available via `www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.pdf`.

[117] H. Heys, S. Tavares. Substitution-Permutation Networks Resistant to Differential and Linear Cryptanalysis. Journal of Cryptology, vol.9, no.1, pages 1–21, 1996.

[118] S. Hirose. How to Construct Double-Block-Length Hash Functions. Second Cryptographic Hash Workshop, Santa Barbara, 2006.

[119] S. Hirose. Provably Secure Double-Block-Length Hash Functions in a Black-Box Model. ICISC'04, LNCS, Springer-Verlag, 2004.

[120] S. Hirose. Some Plausible Constructions of Double-Block-Length Hash Functions. FSE'06, LNCS, Springer-Verlag, 2006.

[121] B. Hochet, P. Quinton, Y. Robert. Systolic Gaussian Elimination over GF(p) with Partial Pivoting. IEEE Transactions on Computers, 38(9):1321–1324, 1989.

[122] W. Hohl, X. Lai, T. Meier, C. Waldvogel. Security of Iterated Hash Function Based on Block Ciphers. CRYPTO'93, LNCS, Springer-Verlag, 1994.

[123] HomeLink: Homelink and KeeLoq-based Rolling Code Garage Door Openers. Available from `www.homelink.com/home/keeloq.tml`, 2006.

[124] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, S. Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. CHES'06, LNCS, Springer-Verlag, 2006.

[125] X.-d. Hou: Affinity of permutations of $\mathbb{F}_2^n$. In Proceedings of the Workshop on Coding and Cryptography WCC 2003 (2003) 273–280. Completed version in Discrete Applied Mathematics 154, Issue 2 (2006) 313–325

[126] J. Huanga, J. Seberry, W. Susilo. On the Internal Structure of Alpha-MAC. VIETCRYPT'06, LNCS, Springer-Verlag, 2006.

[127] Intel Corporation. Intel Unveils World's Best Processor. Press Release, July 27, 2006.

[128] ISO/IEC 9797-1:1999. Information Technology – Security Techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms Using a Block Cipher, 1999.

[129] T. Iwata, K. Kurosawa. OMAC: One-Key CBC MAC. FSE'03, LNCS, Springer-Verlag, 2003.

[130] E. Jaulmes, A. Joux, F. Valette. On the Security of Randomized CBC-MAC beyond the Birthday Paradox Limit. A New Construction. FSE'02, LNCS, Springer-Verlag, 2002.

[131] A. Joux. Multi-Collisions in Iterated Hash Functions. Application to Cascaded Constructions. CRYPTO'04, LNCS, Springer-Verlag, 2004.

[132] A. Juels, S.A. Weis. Authenticating Pervasive Devices with Human Protocols. CRYPTO'05, LNCS, Springer-Verlag, 2005.

[133] C.S. Jutla. Generalized Birthday Attacks on Unbalanced Feistel Networks. CRYPTO'98, LNCS, Springer-Verlag, 1998.

[134] B.S. Kaliski, M.J.B. Robshaw. Linear Cryptanalysis Using Multiple Approximations and FEAL. FSE'94, LNCS, Springer-Verlag, 1994.

[135] M. Kanda. Practical Security Evaluation against Differential and Linear Cryptanalyses for Feistel Ciphers with SPN Round Function. SAC'00, LNCS, Springer-Verlag, 2001

[136] O. Kara, C. Manap. A New Class of Weak Keys for Blowfish. FSE'07, LNCS, Springer-Verlag, 2007.

[137] O. Kara. Reflection Cryptanalysis of Some Ciphers. INDOCRYPT'08, LNCS, Springer-Verlag, 2008.

[138] R.M. Karp, R.E. Tarjan. Linear Expected-Time Algorithms for Connectivity Problems. J. Algorithms, vol. 1, 1980.

[139] J. Kelsey, T. Kohno, B. Schneier. Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. FSE'00, LNCS, Springer-Verlag, 2000.

[140] J. Kelsey, B. Schneier. Second Preimages on n-Bit Hash Functions for Much Less Than 2^n Work. EUROCRYPT'05, LNCS, Springer-Verlag, 2005.

[141] A. Kipnis, A. Shamir. Cryptanalysis of the HFE Public Key Cryptosystem. Crypto'99, LNCS, Springer-Verlag, 1999.

[142] L.R. Knudsen. A Chosen-Plaintext Linear Attack on DES. FSE'01, LNCS, Springer-Verlag, 2001.

[143] L.R. Knudsen. Contemporary Block Ciphers. Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School, Aarhus, 1998. LNCS, Springer-Verlag, 1999.

[144] L.R. Knudsen. Truncated and Higher Order Differentials. FSE'94, LNCS, Springer-Verlag, 1994.

[145] L.R. Knudsen, T. Berson. Truncated Differentials of SAFER. FSE'96, LNCS, Springer-Verlag, 1996.

[146] L.R. Knudsen, X. Lai. New Attacks on all Double Block Length Hash Functions of Hash Rate 1, Including the Parallel-DM. EUROCRYPT'94, LNCS, Springer-Verlag, 1994.

[147] L.R. Knudsen, F. Mendel, C. Rechberger, S.S. Thomsen. Cryptanalysis of MDC-2. EUROCRYPT'09, LNCS, Springer-Verlag, 2009.

[148] L.R. Knudsen, V. Rijmen. Known-Key Distinguishers for Some Block Ciphers. ASIACRYPT'07, Springer-Verlag, 2007.

[149] L.R. Knudsen, M.J.B. Robshaw. Non-Linear Approximations in Linear Cryptanalysis. EUROCRYPT'96, LNCS, Springer-Verlag, 1996.

[150] L.R. Knudsen, M.J.B. Robshaw, D. Wagner. Truncated Differentials and Skipjack. CRYPTO'99, LNCS, Springer-Verlag, 1999.

[151] L.R. Knudsen, D. Wagner. Integral Cryptanalysis. FSE'02, LNCS, Springer-Verlag, 2002.

[152] P.C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems. CRYPTO'96, Springer-Verlag, 1996.

[153] P.C. Kocher, J. Jaffe, B. Jun. Differential Power Analysis. CRYPTO'99, LNCS, Springer-Verlag, 1999.

[154] V.F. Kolchin, B. Sevastyanov, V.P. Chistyakov. Random Allocations. V. H. Winston & Sons, 1978.

[155] I.V. Konovaltsev. On an Algorithm for Solving Systems of Linear Equations in Finite Fields. Problemy Kibernetiky 19, 1967.

[156] K. Kurosawa, T. Iwata. TMAC: Two-Key CBC MAC. CT-RSA'03, LNCS, Springer-Verlag, 2003.

[157] X. Lai, J.L. Massey. A Proposal for a New Block Encryption Standard, EUROCRYPT'90, LNCS, Springer-Verlag, 1991.

[158] X. Lai, J.L. Massey. Hash Functions Based on Block Ciphers. EUROCRYPT'92, LNCS, Springer-Verlag, 1992.

[159] X. Lai, J. Massey, S. Murphy. Markov Ciphers and Differential Cryptanalysis. Eurocrypt'91, LNCS, Springer-Verlag, 1991.

[160] X. Lai, C. Waldvogel, W. Hohl, T. Meier. Security of Iterated Hash Functions Based on Block Ciphers. CRYPTO'93, LNCS, Springer-Verlag, 1993.

[161] S.K. Langford, M.E. Hellman. Differential-Linear Cryptanalysis. CRYPTO'94, LNCS, Springer-Verlag, 1994.

[162] G. Leander, C. Paar, A. Poschmann, K. Schramm. A Family of Lightweight Block Ciphers Based on DES Suited for RFID Applications. FSE'07, LNCS, Springer-Verlag, 2007.

[163] G. Leander, A. Poschmann. On the Classification of 4 Bit S-boxes. WAIFI'07, LNCS, Springer-Verlag, 2007.

[164] J. Lechner, M. Tatzgern. Efficient Implementation of the AES Encryption Algorithm for Smart-Cards. Available at `www.iaik.tugraz.at`, 2004.

[165] H. Ledig, F. Muller, F. Valette. Enhancing Collision Attacks. CHES'04, LNCS, Springer-Verlag, 2004.

[166] S. Lee, Y. Hwang, D. Lee, J. Lim. Efficient Authentication for Low-Cost RFID Systems. ICCS'05, LNCS, Springer-Verlag, 2005.

[167] K. Lemke-Rust. Models and Algorithms for Physical Cryptanalysis. Ph.D. thesis, Ruhr University Bochum, 2007.

[168] M. Li, P. M. B. Vitanyi. An Introduction to Kolmogorov Complexity and Its Applications. Springer-Verlag, 1997.

[169] R. Lidl, H. Niederreiter. Finite Fields. Encyclopedia of Mathematics and Its Applications 20, Cambridge University Press, 1997.

[170] C. Lim, T. Korkishko. mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. WISA'05, LNCS, Springer-Verlag, 2005.

[171] S. Lucks. Faster Luby-Rackoff Ciphers. FSE'96, LNCS, Springer-Verlag, 1996.

[172] S. Lucks. The Saturation Attack – A Bait for Twofish. FSE'01, LNCS, Springer-Verlag, 2001.

[173] MAGMA v2.12. Computational Algebra Group, School of Mathematics and Statistics, University of Sydney, 2005. `magma.maths.usyd.edu.au`.

[174] S. Mangard. A Simple Power-Analysis (SPA) Attackon Implementations of the AES Key Expansion. ICISC'02, LNCS, Springer-Verlag, 2002.

[175] S. Mangard, E. Oswald, T. Popp. Power Analysis Attacks and Countermeasures for Cryptographic Smart Cards: Revealing the Secrets of Smart Cards. Springer-Verlag, 2007.

[176] M. Matsui. Linear Cryptanalysis Method for DES Cipher. EUROCRYPT'93, LNCS,Springer-Verlag, 1994.

[177] M. Matsui. Linear Cryptanalysis of the Data Encryption Standard. EUROCRYPT'93, LNCS, Springer-Verlag, 1994.

[178] A. Maximov, T. Johansson, S. Babbage. An Improved Correlation Attack on A5/1. SAC'04, LNCS, Springer-Verlag, 2005.

[179] A.J. Menezes, S.A. Vanstone, P.C. Van Oorschot. Handbook of Applied Cryptography. CRC Press Inc., Boca Raton, FL, USA, 1996.

[180] R.C. Merkle. One Way Hash Functions and DES. CRYPTO'89, LNCS, Springer-Verlag, 1989.

[181] T.S. Messerges, E.A. Dabbish, R.H. Sloan. Investigations of Power Analysis Attacks on Smartcards. SMARTCARD'99, USENIX Association, 1999.

[182] Microchip. An Introduction to KeeLoq Code Hopping. Available from `ww1.microchip.com/downloads/en/AppNotes/91002a.pdf`, 1996.

[183] Microchip. HCS101 Fixed Code Encoder Data Sheet. Available from `ww1.microchip.com/downloads/en/DeviceDoc/41115c.pdf`, 2001.

[184] Microchip. HCS301 KeeLoq Code Hopping Encoder Data Sheet. Available from `ww1.microchip.com/downloads/en/devicedoc/21143b.pdf`, 2002.

[185] Microchip. HCS410 Keeloq Code Hopping Encoder and Transponder. Available from ww1.microchip.com/downloads/en/DeviceDoc/40158e.pdf, 2001.

[186] Microchip. Hopping Code Decoder using a PIC16C56, AN642. Available from en.wikipedia.org/wiki/KeeLoq and www.keeloq.boom.ru/decryption.pdf, 1998.

[187] Microchip: PIC12F635/PIC16F636/PIC16F639 Cryptographic Module General Overview, TB086. Available from ww1.microchip.com/downloads/en/DeviceDoc/91086A.pdf, 2005.

[188] Microchip: PIC16F687 Microcontroller Data Sheet ,2007.

[189] Microchip. Using KeeLoq to Validate Subsystem Compatibility, AN827. Available from ww1.microchip.com/downloads/en/AppNotes/00827a.pdf, 2002.

[190] M. Nandi. Towards Optimal Double-Length Hash Functions. INDOCRYPT'05, LNCS, Springer-Verlag, 2005.

[191] M. Naor, O. Reingold. On the Construction of Pseudorandom Permutations: Luby-Rackoff Revisited. J. Cryptology, 12(1), pages 29–66, 1999.

[192] National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard, August 2002. csrc.nist.gov.

[193] National Institute of Standards and Technology. FIPS 197: Advanced Encryption Standard (AES), November 2001. csrc.nist.gov.

[194] National Institute of Standards and Technology. FIPS 198: The Keyed-Hash Message Authentication Code, March 2002. csrc.nist.gov.

[195] National Institute of Standards and Technology. FIPS 46-3: Data Encryption Standard (DES), October 1999. csrc.nist.gov.

[196] National Institute of Standards and Technology. SP800-38A: Recommendation for block cipher modes of operation. December 2001. Available via csrc.nist.gov.

[197] NIST. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST Special Publication 800-38B, 2005.

[198] K. Nyberg. Differentially Uniform Mappings for Cryptography. EUROCRYPT'93, LNCS, Springer-Verlag, 1994.

[199] K. Nyberg: Generalized Feistel Networks. In K. Kim and T. Matsumoto, editors, ASIACRYPT'96, vol. 1163 of LNCS, Springer-Verlag (1996) 91–104

[200] K. Nyberg. Linear Approximation of Block Ciphers. EUROCRYPT'94, LNCS, Springer-Verlag, 1994.

[201] P. Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-off. CRYPTO'03, LNCS, Springer-Verlag, 2003.

[202] S.B.Örs, F. Gürkaynak, E.Oswald, B.Preneel. Power-Analysis Attack on an ASIC AES Implementation. ITCC'04, IEEE Computer Society, 2004.

[203] M. Ohkubo, K. Suzuki, S. Kinoshita. Cryptographic Approach to Privacy-Friendly Tags. RFID Privacy Workshop, MIT, 2003.

[204] K. Okeya. Side Channel Attacks against HMACs Based on Block-Cipher Based Hash Functions. ACISP'06, LNCS, Springer-Verlag, 2006.

[205] D.A. Osvik, A. Shamir, T. Tromer. Cache Attacks and Countermeasures: The Case of AES. CT-RSA'06, LNCS, Springer-Verlag, 2006.

[206] E. Oswald, S. Mangard, N. Pramstaller, V. Rijmen. A Side-Channel Analysis Resistant Description of the AES S-box. FSE'05, LNCS, Springer-Verlag, 2005.

[207] E. Oswald, K. Schramm. An Efficient Masking Scheme for AES Software Implementations. WISA'05, LNCS, Springer-Verlag, 2006.

[208] O. Ozen, K. Varici, C. Tezcan, C. Kocair. Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT. `www.cosic.esat.kuleuven.be/publications/article-1234.pdf`, 2009.

[209] D. Page. Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel. Cryptology ePrint Archive 2002/169, 2002.

[210] J. Patarin, V. Nachef, C. Berbain. Generic Attacks on Unbalanced Feistel Schemes with Contracting Functions. ASIACRYPT'06, LNCS, Springer-Verlag 2006.

[211] J. Patarin, V. Nachef, C. Berbain. Generic Attacks on Unbalanced Feistel Schemes with Expanding Functions. ASIACRYPT'07, LNCS, Springer-Verlag, 2007.

[212] E. Petrank, C. Rackoff. CBC MAC for Real-Time Data Sources. J. Cryptology 13(3): 315–338, 2000.

[213] S. Petrovic, A. Fuster-Sabater. Cryptanalysis of the A5/2 Algorithm. IACR ePrint Report 200/52, `eprint.iacr.org`, 2000.

[214] T. Peyrin, H. Gilbert, F. Muller, M.J.B. Robshaw. Combining Compression Functions and Block Cipher-Based Hash Functions. ASIACRYPT'06, LNCS, Springer-Verlag, 2006.

[215] T. Pornin, J. Stern. Software-hardware Trade-offs: Application to A5/1 Cryptanalysis. CHES'00, LNCS, Springer-Verlag, 2000.

[216] B. Preneel. Ph.D. thesis. Katholieke Universiteit Leuven, 1993.

[217] B. Preneel, A. Bosselaers, R. Govaerts, J. Vandewalle. Collision-Free Hash Functions Based on Block Cipher Algorithms. International Carnahan Conference on Security Technology, IEEE, 1989.

[218] B. Preneel, R. Govaerts, J. Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. CRYPTO'93, LNCS, Springer-Verlag, 1994.

[219] J.-J. Quisquater, M. Girault. 2n-Bit Hash-Functions Using n-Bit Symmetric Block Cipher Algorithms. EUROCRYPT'89, LNCS, Springer-Verlag, 1989.

[220] J.-J. Quisquater, D. Samyde. Electromagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. Proceedings of Smart Card Programming and Security E-smart'01 LNCS, Springer-Verlag, 2001.

[221] M. Renauld, F.-X. Standaert, N. Veyrat-Charvillon. Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA. CHES'09, LNCS, Springer-Verlag, 2009.

[222] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, E. De Win. The cipher Shark. FSE'96, LNCS, Springer-Verlag, 1996.

[223] V. Rijmen, B. Preneel, E. De Win. On Weaknesses of Non-Surjective Round Functions. Des. Codes Cryptography (DCC) 12(3):253-266, 1997.

[224] R. Rivest. The RC5 Encryption Algorithm. FSE'94, LNCS, Springer-Verlag, 1994.

[225] R.L. Rivest. RFC 1321: The MD5 Message-Digest Algorithm, April 1992. `www.ietf.org/rfc/rfc1321.txt`.

[226] R. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21 (2): 120–126, 1978.

[227] M.J.B. Robshaw. Searching for Compact Algorithms: CGEN. VIETCRYPT'06, Springer-Verlag, 2006.

[228] P. Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. ASIACRYPT'04, LNCS, Springer-Verlag, 2004.

[229] C. Rolfes, A. Poschmann, G. Leander, C. Paar. Ultra-Lightweight Implementations for Smart Devices – Security for 1000 Gate Equivalents. CARDIS'08, LNCS, Springer-Verlag, 2008.

[230] V.N. Sachkov. Probabilistic Methods in Combinatorial Analysis. Encyclopedia of Mathematics and Its Applications, vol. 56, Cambridge University Press, 1997.

[231] B. Schneier, J. Kelsey. Unbalanced Feistel Networks and Block Cipher Design. FSE'96 (D. Gollmann, ed.), vol. 1039 of LNCS, pages 121–144, Springer-Verlag, 1996.

[232] K. Schramm, G. Leander, P. Felke, C. Paar. A Collision-Attack on AES: Combining Side Channel- and Differential-Attack. CHES'04, LNCS, Springer-Verlag, 2004.

[233] K. Schramm, T.J. Wollinger, C. Paar. A New Class of Collision Attacks and Its Application to DES. FSE'03, LNCS, Springer-Verlag, 2003.

[234] A.A. Selcuk. On Probability of Success in Linear and Differential Cryptanalysis. J. Cryptology 21(1): 131-147, 2008.

[235] Y. Seurin, T. Peyrin. Security Analysis of Constructions Combining FIL Random Oracles. FSE'07, LNCS, Springer-Verlag, 2007.

[236] A. Shamir. On the Security of DES. CRYPTO'85, LNCS, Springer-Verlag, 1985.

[237] A. Shamir. SQUASH - a New MAC with Provable Security Properties for Highly Constrained Devices Such As RFID Tags. FSE'08, LNCS, Springer-Verlag, 2008.

[238] C.E. Shannon. Communication Theory of Secrecy Systems, Bell System Technical Journal, vol. 28, pages 656–715, 1949. The material in this paper appeared originally in a confidential report 'A Mathematical Theory of Cryptography', dated Sept. 1, 1945.

[239] T. Shirai, B. Preneel. On Feistel Ciphers Using Optimal Diffusion Mappings across Multiple Rounds. ASIACRYPT'04, LNCS, Springer-Verlag, 2004.

[240] T. Shirai, K. Shibutani. Improving Immunity of Feistel Ciphers against Differential Cryptanalysis by Using Multiple MDS Matrices. FSE'04, LNCS, Springer-Verlag, 2004.

[241] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, T. Iwata. The 128-Bit Block-cipher CLEFIA (Extended Abstract). FSE'07, LNCS, Springer-Verlag, 2007.

[242] T. Siegenthaler. Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications. IEEE Trans. on Inform. Theory, IT-30, 1984.

[243] T. Siegenthaler. Decrypting a Class of Stream Ciphers Using Ciphertext Only. IEEE Trans. on Computers 34, 1985.

[244] S. D'Souza. AN556 – Implementing a Table Read. Technical report, Microchip Technology Inc., Application Note, 2000.

[245] F.-X. Standaert, C. Archambeau. Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages. CHES'08, LNCS, Springer-Verlag, 2008.

[246] F.-X. Standaert, G. Piret, N. Gershenfeld, J.-J. Quisquater. SEA: A Scalable Encryption Algorithm for Small Embedded Applications. CARDIS'06, LNCS, Springer-Verlag, 2006.

[247] J. P. Steinberger. The Collision Intractability of MDC-2 in the Ideal-Cipher Model. EUROCRYPT'07, LNCS, Springer-Verlag, 2007.

[248] V. Strassen. Gaussian Elimination Is Not Optimal. Numer. Math. 13, 1969.

[249] A. Tardy-Corfdir, H. Gilbert. A Known Plaintext Attack of FEAL-4 and FEAL-6. CRYPTO'91, LNCS, Springer-Verlag, 1991.

[250] I. Verbauwhede, F. Hoornaert, J. Vandewalle, H. De Man. Security and Performance Optimization of a New DES Data Encryption Chip. IEEE Journal of Solid-State Circuits, vol.23, no.3, pages 647–656, 1988.

[251] D. Wagner. The Boomerang Attack. FSE'99, LNCS, Springer-Verlag, 1999.

[252] M. Wang. Differential Cryptanalysis of Reduced-Round PRESENT. AFRICACRYPT'08, LNCS, Springer-Verlag, 2008.

[253] X. Wang, Y.L. Yin, H. Yu. Finding Collisions in the Full SHA-1. CRYPTO'05, LNCS, Springer-Verlag, 2005.

[254] X. Wang, H. Yu. How to Break MD5 and Other Hash Functions. EUROCRYPT'05, LNCS, Springer-Verlag, 2005.

[255] D. Wheeler, R. Needham. TEA, a Tiny Encryption Algorithm. FSE'94, LNCS, Springer-Verlag, 1994.

[256] D. Wheeler, R. Needham. TEA extensions. October 1997. (Also Correction to XTEA. October, 1998.) Available via `www.ftp.cl.cam.ac.uk/ftp/users/djw3/`.

[257] A. Wiemers. Collision Attacks for Comp128 on Smartcards. ECC-Brainpool Workshop on Side-Channel Attacks on Cryptographic Algorithms, Bonn, Germany, 2001.

[258] Wikipedia: Keeloq algorithm. Available from `en.wikipedia.org/wiki/KeeLoq`, 2006.

[259] H. Yoshida, D. Watanabe, K. Okeya, J. Kitahara, J. Wu, O. Kucuk, B. Preneel. MAME: A Compression Function with Reduced Hardware Requirements. CHES'07, LNCS, Springer-Verlag, 2007.

[260] Y. Yu, Y. Yang, Y. Fan, H. Min. Security Scheme for RFID Tag. Auto-ID Labs white paper WP-HARDWARE-022. `www.autoidlabs.org/`.

[261] Z. Yuan, K. Jia, W. Wang, X. Wang Distinguishing and Forgery Attacks on Alred and Its AES-Based Instance Alpha-MAC. Cryptology ePrint Archive: Report 2008/516, 2008.

[262] M. Reza Zaba, H. Raddum, M. Henricksen, E. Dawson. Bit-Pattern Based Integral Attack. FSE'08, LNCS, Springer-Verlag, 2008.

[263] Y. Zheng, T. Matsumoto, and H. Imai: On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses. In G. Brassard, editor, CRYPTO'89, vol. 435 of LNCS, Springer-Verlag (1990) 461–480

List of Tables

List of Figures

Curriculum Vitae

12.2006–07.2009	Research and teaching assistant Ruhr-University Bochum, Bochum (Germany), Faculty of Electrical Engineering and Information Technology, Horst Görtz Institute for IT Security, Chair for Embedded Security
01.2008–06.2009	Senior security engineer escrypt GmbH – Embedded Security, Bochum (Germany)
10.2005–12.2007	Security developer escrypt GmbH – Embedded Security, Bochum (Germany)
03.2005–06.2005	External graduant University Duisburg-Essen, Essen (Germany), Institute for Experimental Mathematics, research group "Number Theory"
04.2004–09.2004	Research intern Siemens AG, Munich (Germany), Corporate Technology, Information & Communications, Security Technologies (CT IC 3)
10.2003–12.2004	Stipendiary, Siemens scholarship Humboldt University Berlin, Berlin (Germany), Institute for Computer Science, Chair for Cryptography and Complexity
09.2000–06.2005	Diploma degree (M.Sc.) in IT security (with distinction) Russian State University for the Humanities, Moscow (Russia), Institute for Computer Science and Security Technologies, Faculty of Information Security, Chair for Fundamental and Applied Mathematics
09.1997–06.2000	Secondary school diploma (with silver medal and distinction) Technical Lyceum 8 in Electrostal (Russia), special group for mathematics and physics

Publications

Journal Papers and Book Chapters

Gregory V. Bard, Andrey Bogdanov and Nicolas T. Courtois. Periodic Ciphers with Small Blocks and Cryptanalysis of KeeLoq. Tatra Mt. Math. Publ. 41 (2008), 2008 (alphabetical order).

Andrey Bogdanov and Ilya Kizhvatov. Cryptanalysis of the NiVa stream cipher. Journal of Information Technology Security, Vol. 3/2007, MEPhI, 2007 (in Russian).

Andrey Bogdanov, Thomas Eisenbarth, Christof Paar, Marco Wolf. Trusted Computing in Automotive Systems. Chapter in "Trusted Computing", N. Pohlmann and H. Reimer (Eds.), Vieweg-Verlag, 2007.

Research Papers on International Conferences and Workshops

Andrey Bogdanov. On the Differential Trails of Unbalanced Feistel Networks with Contracting MDS Diffusion. International Workshop on Coding and Cryptography (WCC 2009), Ullensvang, Norway, 2009.

Andrey Bogdanov, Ilya Kizhvatov, Andrey Pyshkin. Algebraic Methods in Side-Channel Collision Attacks and Practical Collision Detection. Progress in Cryptology - INDOCRYPT 2008, LNCS, Springer-Verlag, 2008.

Andrey Bogdanov. Multiple-Differential Side-Channel Collision Attacks on AES. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2008), LNCS, Springer-Verlag, 2008.

Andrey Bogdanov, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J.B. Robshaw, Yannick Seurin. Hash Functions and RFID Tags : Mind the Gap. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2008), LNCS, Springer-Verlag, 2008.

Andrey Bogdanov, Thomas Eisenbarth, Andy Rupp, Christopher Wolf. Time-Area Optimized Public-Key Engines: MQ-Cryptosystems as Replacement for Elliptic Curves? Workshop on Cryptographic Hardware and Embedded Systems (CHES 2008), LNCS, Springer-Verlag, 2008 (**Best Paper Award**). Solicited for Journal of Cryptology.

Sundar Balasubramanian, Andrey Bogdanov, Andy Rupp, Jintai Ding and Harold W. Carter. Fast Multivariate Signature Generation in Hardware: The Case of Rainbow (poster). 16th IEEE Symposium on Field- Programmable Custom Computing Machines (FCCM 2008). IEEE Computer Society, 2008.

Sundar Balasubramanian, Andrey Bogdanov, Andy Rupp, Jintai Ding and Harold W. Carter. Fast Multivariate Signature Generation in Hardware: The Case of Rainbow, 19th IEEE International Conference Application-Specific Systems, Architectures and Processors (ASAP 2008), IEEE Computer Society Press, 2008.

Andrey Bogdanov and Christof Paar. On the Security and Efficiency of Real-World Lightweight Authentication Protocols, 1st Workshop on Secure Component and System Identification (SECSI 2008), Berlin, 2008.

Andrey Bogdanov. Linear Slide Attacks on the KeeLoq Block Cipher. The 3rd SKLOIS Conference on Information Security and Cryptology (INSCRYPT 2007), LNCS, Springer-Verlag, 2007.

Andrey Bogdanov. Improved Collision Attacks on AES. The 14th Annual Workshop on Selected Areas in Cryptography (SAC 2007), LNCS, Springer-Verlag, 2007.

Andrey Bogdanov. Attacks on the Keeloq Block Cipher and Authentication Systems. The 3rd Conference on RFID Security (RFIDSec 2007), Malaga, Spain, 2007.

Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J.B. Robshaw, Yannick Seurin, Charlotte Vikkelsoe. Small-Footprint Block Cipher Design - How Far Can You Go? The 3rd Conference on RFID Security (RFIDSec 2007), Malaga, Spain, 2007.

Nicolas T. Courtois, Gregory V. Bard, Andrey Bogdanov. Sliding Attacks, Ciphers with Small Blocks and Recent Attacks on KeeLoq. The 7th Central European Conference on Cryptology (TATRACRYPT 2007), Slovakia, 2007.

Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J.B. Robshaw, Yannick Seurin, Charlotte Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007), LNCS, Springer-Verlag, 2007.

Alex Biryukov, Andrey Bogdanov, Dmitry Khovratovich, Timo Kasper. Collision Attacks on AES-Based MAC: Alpha-MAC. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007), LNCS, Springer-Verlag, 2007.

Andrey Bogdanov, Thomas Eisenbarth, Andy Rupp. A Hardware-Assisted Real-Time Attack on A5/2 without Precomputations. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007), LNCS, Springer-Verlag, 2007.

Andrey Bogdanov, Marius Mertens, Christof Paar, Jan Pelzl, Andy Rupp. SMITH – a Parallel Hardware Architecture for Fast Gaussian Elimination over GF(2). Special-purpose Hardware for Attacking Cryptographic Systems (SHARCS 2006), Cologne, Germany, 2006.

Andrey Bogdanov, Marius Mertens, Christof Paar, Jan Pelzl, Andy Rupp. A Parallel Hardware Architecture for Fast Gaussian Elimination over GF(2). IEEE Proceedings on Field-Programmable Custom Computing Machines (FCCM 2006), IEEE Computer Society Press, 2006.

Vladimir Anashin, Andrey Bogdanov, Ilya Kizhvatov. Security and Implementation Properties of ABC v.2. Stream Ciphers Revised (SASC 2006), Leuven, Belgium, 2006.

Vladimir Anashin, Andrey Bogdanov, Ilya Kizhvatov, Sandeep Kumar. ABC: A New Fast Flexible Stream Cipher. Symmetric Key Encryption Workshop (SKEW 2005), Aarhus, Denmark, 2005.

Other Papers (Selected)

Andrey Bogdanov, Jan Pelzl, Thomas Wollinger. Embedded Security in Automobilen: Chancen und Risiken. Warum die konventionelle IT-Sicherheit für PCs im Automobil nicht funktioniert, VDI/VW-Gemeinschaftstagung Automotive Security, Wolfsburg, VDI-Verlag, 2007.

Andrey Bogdanov, Thomas Eisenbarth, Marko Wolf, Thomas Wollinger. Trusted Computing for Automotive Systems: New Approaches to Enforce Security for Electronic Systems in Vehicles, VDI/VW-Gemeinschaftstagung Automotive Security, Wolfsburg, VDI-Verlag, 2007.

Andrey Bogdanov, Andre Weimerskirch, Thomas Wollinger, Dario Carluccio. Embedded Security Solutions for Automotive Applications. 11th International Forum on Advanced Microsystems for Automotive Applications (AMAA 2007), Springer-Verlag, VDI, 2007.