Information Security and Cryptography
Texts and Monographs

Springer-Verlag Berlin Heidelberg GmbH

Joan Daemen • Vincent Rijmen

The Design of Rijndael

AES – The Advanced Encryption Standard

With 48 Figures and 17 Tables

Springer

Joan Daemen
Proton World International (PWI)
Zweefvliegtuigstraat 10
1130 Brussels, Belgium

Vincent Rijmen
Cryptomathic NV
Lei 8a
3000 Leuven, Belgium

Library of Congress Cataloging-in-Publication Data

Daemen, Joan, 1965–
The design of Rijndael: AES - The Advanced Encryption Standard/Joan Daemen, Vincent Rijmen.
p. cm.
Includes bibliographical references and index.

1. Computer security - Passwords. 2. Data encryption (Computer science) I. Rijmen, Vincent, 1970– II. Title

QA76.9.A25 D32 2001
005.8–dc21 2001049851

ACM Subject Classification (1998): E.3, C.2, D.4.6, K.6.5

ISBN 978-3-642-07646-6 ISBN 978-3-662-04722-4 (eBook)
DOI 10.1007/978-3-662-04722-4

Springer-Verlag Berlin Heidelberg New York,
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Originally published by Springer-Verlag Berlin Heidelberg New York in 2002.
Softcover reprint of the hardcover 1st edition 2002

Typesetting: Camera-ready by the authors
Cover Design: KünkelLopka, Heidelberg
Printed on acid-free paper SPIN 10851372 - 06/3142SR - 5 4 3 2 1 0

Foreword

Rijndael was the surprise winner of the contest for the new Advanced Encryption Standard (AES) for the United States. This contest was organized and run by the National Institute for Standards and Technology (NIST) beginning in January 1997; Rijndael was announced as the winner in October 2000. It was the "surprise winner" because many observers (and even some participants) expressed scepticism that the U.S. government would adopt as an encryption standard any algorithm that was not designed by U.S. citizens.

Yet NIST ran an open, international, selection process that should serve as model for other standards organizations. For example, NIST held their 1999 AES meeting in Rome, Italy. The five finalist algorithms were designed by teams from all over the world.

In the end, the elegance, efficiency, security, and principled design of Rijndael won the day for its two Belgian designers, Joan Daemen and Vincent Rijmen, over the competing finalist designs from RSA, IBM, Counterpane Systems, and an English/Israeli/Danish team.

This book is the story of the design of Rijndael, as told by the designers themselves. It outlines the foundations of Rijndael in relation to the previous ciphers the authors have designed. It explains the mathematics needed to understand the operation of Rijndael, and it provides reference C code and test vectors for the cipher.

Most importantly, this book provides justification for the belief that Rijndael is secure against all known attacks. The world has changed greatly since the DES was adopted as the national standard in 1976. Then, arguments about security focussed primarily on the length of the key (56 bits). Differential and linear cryptanalysis (our most powerful tools for breaking ciphers) were then unknown to the public. Today, there is a large public literature on block ciphers, and a new algorithm is unlikely to be considered seriously unless it is accompanied by a detailed analysis of the strength of the cipher against at least differential and linear cryptanalysis.

This book introduces the "wide trail" strategy for cipher design, and explains how Rijndael derives strength by applying this strategy. Excellent resistance to differential and linear cryptanalysis follow as a result. High efficiency is also a result, as relatively few rounds are needed to achieve strong security.

The adoption of Rijndael as the AES is a major milestone in the history of cryptography. It is likely that Rijndael will soon become the most widely-used cryptosystem in the world. This wonderfully written book by the designers themselves is a "must read" for anyone interested in understanding this development in depth.

Ronald L. Rivest
Viterbi Professor of Computer Science
MIT

Preface

This book is about the design of Rijndael, the block cipher that became the Advanced Encryption Standard (AES). According to the 'Handbook of Applied Cryptography' [68], a block cipher can be described as follows:

> A block cipher is a function which maps n-bit plaintext blocks to n-bit ciphertext blocks; n is called the block length. [...] The function is parameterized by a key.

Although block ciphers are used in many interesting applications such as e-commerce and e-security, this book is *not* about applications. Instead, this book gives a detailed description of Rijndael and explains the design strategy that was used to develop it.

Structure of this book

When we wrote this book, we had basically two kinds of readers in mind. Perhaps the largest group of readers will consist of people who want to read a full and unambiguous description of Rijndael. For those readers, the most important chapter of this book is Chap. 3, that gives its comprehensive description. In order to follow our description, it might be helpful to read the preliminaries given in Chap. 2. Advanced implementation aspects are discussed in Chap. 4. A short overview of the AES selection process is given in Chap. 1.

A large part of this book is aimed at the readers who want to know *why* we designed Rijndael in the way we did. For them, we explain the ideas and principles underlying the design of Rijndael, culminating in our wide trail design strategy. In Chap. 5 we explain our approach to block cipher design and the criteria that played an important role in the design of Rijndael. Our design strategy has grown out of our experiences with linear and differential cryptanalysis, two cryptanalytical attacks that have been applied with some success to the previous standard, the Data Encryption Standard (DES). In Chap. 6 we give a short overview of the DES and of the differential and the linear attacks that are applied to it. Our framework to describe linear cryptanalysis is explained in Chap. 7; differential cryptanalysis is described

in Chap. 8. Finally, in Chap. 9, we explain how the wide trail design strategy follows from these considerations

Chapter 10 gives an overview of the published attacks on reduced-round variants of Rijndael. Chapter 11 gives an overview of ciphers related to Rijndael. We describe its predecessors and discuss their similarities and differences. This is followed by a short description of a number of block ciphers that have been strongly influenced by Rijndael and its predecessors.

In Appendix A we show how linear and differential analysis can be applied to ciphers that are defined in terms of finite field operations rather than Boolean functions. In Appendix B we discuss extensions of differential and linear cryptanalysis. To assist programmers, Appendix C lists some tables that are used in various descriptions of Rijndael, Appendix D gives a set of test vectors, and Appendix E consists of an example implementation of Rijndael in the C programming language.

See Fig. 1 for a graphical representation of the different ways to read this book.

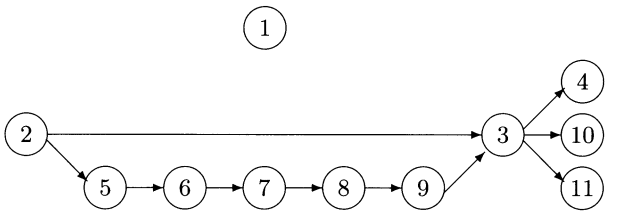

Fig. 1. Logical dependence of the chapters.

Large portions of this book have already been published before: Joan's PhD thesis [18], Vincent's PhD thesis [80], our submission to AES [26], and our paper on linear frameworks for block ciphers [22].

Acknowledgements

This book would not have been written without the support and help of many people. It is impossible for us to list all people who contributed along the way. Although we probably will make oversights, we would like to name some of our supporters here.

First of all, we would like to thank the many cryptographers who contributed to developing the theory on the design of symmetric ciphers, and from who we learned much of what we know today. We would like to mention explicitly the people who gave us feedback in the early stages of the design

process: Johan Borst, Antoon Bosselaers, Paulo Barreto, Craig Clapp, Erik De Win, Lars R. Knudsen, and Bart Preneel.

Elaine Barker, James Foti and Miles Smid, and all the other people at NIST, who worked very hard to make the AES process possible and visible.

The moral support of our family and friends, without whom we would never have persevered.

Brian Gladman, who provided test vectors.

Othmar Staffelbach, Elisabeth Oswald, Lee McCulloch and other proof-readers who provided very valuable feedback and corrected numerous errors and oversights.

The financial support of K.U.Leuven, the Fund for Scientific Research – Flanders (Belgium), Banksys, Proton World and Cryptomathic is also greatly appreciated.

November 2001 *Joan Daemen and Vincent Rijmen*

Contents

1. The Advanced Encryption Standard Process 1
1.1 In the Beginning 1
1.2 AES: Scope and Significance 1
1.3 Start of the AES Process 2
1.4 The First Round 3
1.5 Evaluation Criteria 4
1.5.1 Security 4
1.5.2 Costs 4
1.5.3 Algorithm and Implementation Characteristics 4
1.6 Selection of Five Finalists 5
1.6.1 The Second AES Conference 5
1.6.2 The Five Finalists 6
1.7 The Second Round 7
1.8 The Selection 7

2. Preliminaries 9
2.1 Finite Fields 10
2.1.1 Groups, Rings, and Fields 10
2.1.2 Vector Spaces 11
2.1.3 Fields with a Finite Number of Elements 13
2.1.4 Polynomials over a Field 13
2.1.5 Operations on Polynomials 14
2.1.6 Polynomials and Bytes 15
2.1.7 Polynomials and Columns 16
2.2 Linear Codes 17
2.2.1 Definitions 17
2.2.2 MDS codes 19
2.3 Boolean Functions 19
2.3.1 Bundle Partitions 20
2.3.2 Transpositions 21
2.3.3 Bricklayer Functions 22
2.3.4 Iterative Boolean Transformations 22
2.4 Block Ciphers 23
2.4.1 Iterative Block Ciphers 24

2.4.2 Key-Alternating Block Ciphers ... 25
2.5 Block Cipher Modes of Operation ... 27
2.5.1 Block Encryption Modes ... 27
2.5.2 Key-Stream Generation Modes ... 27
2.5.3 Message Authentication Modes ... 28
2.5.4 Cryptographic Hashing ... 29
2.6 Conclusions ... 29

3. Specification of Rijndael ... 31
3.1 Differences between Rijndael and the AES ... 31
3.2 Input and Output for Encryption and Decryption ... 31
3.3 Structure of Rijndael ... 33
3.4 The Round Transformation ... 33
3.4.1 The `SubBytes` Step ... 34
3.4.2 The `ShiftRows` Step ... 37
3.4.3 The `MixColumns` Step ... 38
3.4.4 The Key Addition ... 40
3.5 The Number of Rounds ... 41
3.6 Key Schedule ... 43
3.6.1 Design Criteria ... 43
3.6.2 Selection ... 43
3.7 Decryption ... 45
3.7.1 Decryption for a Two-Round Rijndael Variant ... 45
3.7.2 Algebraic Properties ... 46
3.7.3 The Equivalent Decryption Algorithm ... 48
3.8 Conclusions ... 50

4. Implementation Aspects ... 53
4.1 8-Bit Platforms ... 53
4.1.1 Finite Field Multiplication ... 53
4.1.2 Encryption ... 54
4.1.3 Decryption ... 55
4.2 32-Bit Platforms ... 56
4.3 Dedicated Hardware ... 59
4.3.1 Decomposition of S_{RD} ... 60
4.3.2 Efficient Inversion in $GF(2^8)$ ... 61
4.4 Multiprocessor Platforms ... 61
4.5 Performance Figures ... 62
4.6 Conclusions ... 62

5. Design Philosophy ... 63
5.1 Generic Criteria in Cipher Design ... 63
5.1.1 Security ... 63
5.1.2 Efficiency ... 64
5.1.3 Key Agility ... 64

5.1.4 Versatility 64
5.1.5 Discussion 64
5.2 Simplicity 65
5.3 Symmetry 65
5.3.1 Symmetry Across the Rounds 66
5.3.2 Symmetry Within the Round Transformation 66
5.3.3 Symmetry in the D-box 67
5.3.4 Symmetry and Simplicity in the S-box 68
5.3.5 Symmetry between Encryption and Decryption 68
5.3.6 Additional Benefits of Symmetry 68
5.4 Choice of Operations 69
5.4.1 Arithmetic Operations 70
5.4.2 Data-Dependent Shifts 70
5.5 Approach to Security 71
5.5.1 Security Goals 71
5.5.2 Unknown Attacks Versus Known Attacks 72
5.5.3 Provable Security Versus Provable Bounds 73
5.6 Approaches to Design 73
5.6.1 Non-Linearity and Diffusion Criteria 73
5.6.2 Resistance against Differential and Linear Cryptanalysis 73
5.6.3 Local Versus Global Optimization 74
5.7 Key-Alternating Cipher Structure 76
5.8 The Key Schedule 76
5.8.1 The Function of a Key Schedule 76
5.8.2 Key Expansion and Key Selection 77
5.8.3 The Cost of the Key Expansion 77
5.8.4 A Recursive Key Expansion 78
5.9 Conclusions 79

6. The Data Encryption Standard 81
6.1 The DES 81
6.2 Differential Cryptanalysis 83
6.3 Linear Cryptanalysis 85
6.4 Conclusions 87

7. Correlation Matrices 89
7.1 The Walsh-Hadamard Transform 89
7.1.1 Parities and Selection Patterns 89
7.1.2 Correlation 89
7.1.3 Real-valued Counterpart of a Binary Boolean Function 90
7.1.4 Orthogonality and Correlation 90
7.1.5 Spectrum of a Binary Boolean Function 91
7.2 Composing Binary Boolean Functions 93
7.2.1 XOR 93
7.2.2 AND 93

7.2.3 Disjunct Boolean Functions 94
7.3 Correlation Matrices 94
7.3.1 Equivalence of a Boolean Function and its Correlation Matrix 95
7.3.2 Iterative Boolean Functions 96
7.3.3 Boolean Permutations 96
7.4 Special Boolean Functions 98
7.4.1 XOR with a Constant 98
7.4.2 Linear Functions 98
7.4.3 Bricklayer Functions 98
7.5 Derived Properties 99
7.6 Truncating Functions 100
7.7 Cross-correlation and Autocorrelation 101
7.8 Linear Trails 102
7.9 Ciphers 103
7.9.1 General Case 103
7.9.2 Key-Alternating Cipher 104
7.9.3 Averaging over all Round Keys 105
7.9.4 The Effect of the Key Schedule 106
7.10 Correlation Matrices and Linear Cryptanalysis Literature 108
7.10.1 Linear Cryptanalysis of the DES 108
7.10.2 Linear Hulls 109
7.11 Conclusions 111

8. Difference Propagation 113
8.1 Difference Propagation 113
8.2 Special Functions 114
8.2.1 Affine Functions 114
8.2.2 Bricklayer Functions 114
8.2.3 Truncating Functions 115
8.3 Difference Propagation Probabilities and Correlation 115
8.4 Differential Trails 117
8.4.1 General Case 117
8.4.2 Independence of Restrictions 117
8.5 Key-Alternating Cipher 118
8.6 The Effect of the Key Schedule 119
8.7 Differential Trails and Differential Cryptanalysis Literature 119
8.7.1 Differential Cryptanalysis of the DES Revisited 119
8.7.2 Markov Ciphers 120
8.8 Conclusions 122

9. The Wide Trail Strategy 123
9.1 Propagation in Key-alternating Block Ciphers 123
9.1.1 Linear Cryptanalysis 123
9.1.2 Differential Cryptanalysis 125
9.1.3 Differences between Linear Trails and Differential Trails 126
9.2 The Wide Trail Strategy 126
9.2.1 The $\gamma\lambda$ Round Structure in Block Ciphers 127
9.2.2 Weight of a Trail 129
9.2.3 Diffusion 130
9.3 Branch Numbers and Two-Round Trails 131
9.3.1 Derived Properties 133
9.3.2 A Two-Round Propagation Theorem 133
9.4 An Efficient Key-Alternating Structure 134
9.4.1 The Diffusion Step θ 134
9.4.2 The Linear Step Θ 136
9.4.3 A Lower Bound on the Bundle Weight of Four-Round Trails 136
9.4.4 An Efficient Construction for Θ 137
9.5 The Round Structure of Rijndael 138
9.5.1 A Key-Iterated Structure 138
9.5.2 Applying the Wide Trail Strategy to Rijndael 142
9.6 Constructions for θ 143
9.7 Choices for the Structure of $\mathcal{I}$ and π 145
9.7.1 The Hypercube Structure 145
9.7.2 The Rectangular Structure 147
9.8 Conclusions 147

10. Cryptanalysis 149
10.1 Truncated Differentials 149
10.2 Saturation Attacks 149
10.2.1 Preliminaries 150
10.2.2 The Basic Attack 150
10.2.3 Influence of the Final Round 152
10.2.4 Extension at the End 153
10.2.5 Extension at the Beginning 153
10.2.6 Attacks on Six Rounds 153
10.2.7 The Herds Attack 154
10.3 Gilbert–Minier Attack 154
10.3.1 The Four-Round Distinguisher 154
10.3.2 The Attack on Seven Rounds 155
10.4 Interpolation Attacks 156
10.5 Symmetry Properties and Weak Keys as in the DES 156
10.6 Weak keys as in IDEA 157
10.7 Related-Key Attacks 157
10.8 Implementation Attacks 157

10.8.1 Timing Attacks 157
10.8.2 Power Analysis 158
10.9 Conclusion 160

11. Related Block Ciphers 161
11.1 Overview 161
11.1.1 Evolution 161
11.1.2 The Round Transformation 162
11.2 SHARK 163
11.3 Square 165
11.4 BKSQ 168
11.5 Children of Rijndael 171
11.5.1 Crypton 171
11.5.2 Twofish 172
11.5.3 ANUBIS 172
11.5.4 GRAND CRU 173
11.5.5 Hierocrypt 173
11.6 Conclusion 173

Appendices

A. Propagation Analysis in Galois Fields 175
A.1 Functions over $\mathrm{GF}(2^n)$ 176
A.1.1 Difference Propagation 177
A.1.2 Correlation 177
A.1.3 Functions that are Linear over $\mathrm{GF}(2^n)$ 179
A.1.4 Functions that are Linear over $\mathrm{GF}(2)$ 180
A.2 Functions over $(\mathrm{GF}(2^n))^\ell$ 181
A.2.1 Difference Propagation 182
A.2.2 Correlation 182
A.2.3 Functions that are Linear over $\mathrm{GF}(2^n)$ 182
A.2.4 Functions that are Linear over $\mathrm{GF}(2)$ 183
A.3 Representations of $\mathrm{GF}(p^n)$ 184
A.3.1 Cyclic Representation of $\mathrm{GF}(p^n)$ 184
A.3.2 Vector Space Representation of $\mathrm{GF}(p^n)$ 184
A.3.3 Dual Bases 185
A.4 Boolean Functions and Functions in $\mathrm{GF}(2^n)$ 186
A.4.1 Differences in $\mathrm{GF}(2)^n$ and $\mathrm{GF}(2^n)$ 186
A.4.2 Relationship Between Trace Patterns and Selection Patterns 187
A.4.3 Relationship Between Linear Functions in $\mathrm{GF}(p)^n$ and $\mathrm{GF}(p^n)$ 187
A.4.4 Illustration 190
A.5 Rijndael-GF 192

B. Trail Clustering 195
B.1 Transformations with Maximum Branch Number 196
B.2 Bounds for Two Rounds 199
B.2.1 Difference Propagation 200
B.2.2 Correlation 202
B.3 Bounds for Four Rounds 204
B.4 Two Case Studies 205
B.4.1 Differential Trails 205
B.4.2 Linear Trails 207

C. Substitution Tables 211
C.1 S_{RD} 211
C.2 Other Tables 212
C.2.1 `xtime` 212
C.2.2 Round Constants 212

D. Test Vectors 215
D.1 `KeyExpansion` 215
D.2 Rijndael(128,128) 215
D.3 Other Block Lengths and Key Lengths 217

E. Reference Code 221

Bibliography 229

Index 235

1. The Advanced Encryption Standard Process

The main subject of this book would probably have remained an esoteric topic of cryptographic research — with a name unpronounceable to most of the world — without the Advanced Encryption Standard (AES) process. Therefore, we thought it proper to include a short overview of the AES process.

1.1 In the Beginning ...

In January 1997, the US National Institute of Standards and Technology (NIST) announced the start of an initiative to develop a new encryption standard: the AES. The new encryption standard was to become a Federal Information Processing Standard (FIPS), replacing the old Data Encryption Standard (DES) and triple-DES.

Unlike the selection process for the DES, the Secure Hash Algorithm (SHA-1) and the Digital Signature Algorithm (DSA), NIST had announced that the AES selection process would be open. Anyone could submit a candidate cipher. Each submission, provided it met the requirements, would be considered on its merits. NIST would not perform any security or efficiency evaluation itself, but instead invited the cryptology community to mount attacks and try to cryptanalyse the different candidates, and anyone who was interested to evaluate implementation cost. All results could be sent to NIST as public comments for publication on the NIST AES web site or be submitted for presentation at AES conferences. NIST would merely collect contributions using them to base their selection. NIST would motivate their choices in evaluation reports.

1.2 AES: Scope and Significance

The official scope of a FIPS standard is quite limited: the FIPS only applies to the US Federal Administration. Furthermore, the new AES would only be used for documents that contain *sensitive but not classified* information.

However, it was anticipated that the impact of the AES would be much larger than this: for AES is the successor of the DES, the cipher that ever since its adoption has been used as a worldwide de facto cryptographic standard by banks, administrations and industry.

Rijndael's approval as a government standard gives it an official 'certificate of quality'. AES has been submitted to the International Organization for Standardization (ISO) and the Internet Engineering Task Force (IETF) as well as the Institute of Electrical and Electronics Engineers (IEEE) are adopting it as a standard. Still, even before Rijndael was selected to become the AES, several organizations and companies declared their adoption of Rijndael. The European Telecommunications Standards Institute (ETSI) uses Rijndael as a building block for its MILENAGE algorithm set, and several vendors of cryptographic libraries had already included Rijndael in their products.

The major factors for a quick acceptance for Rijndael are the fact that it is available royalty-free, and that it can be implemented easily on a wide range of platforms without reducing bandwidth in a significant way.

1.3 Start of the AES Process

In September 1997, the final request for candidate nominations for the AES was published. The minimum functional requirements asked for symmetric block ciphers capable of supporting block lengths of 128 bits and key lengths of 128, 192 and 256 bits. An early draft of the AES functional requirements had asked for block ciphers also supporting block sizes of 192 and 256 bits, but this requirement was dropped later on. Nevertheless, since the request for proposals mentioned that extra functionality in the submissions would be received favourably, some submitters decided to keep the variable block length in the designs. (Examples include RC6 and Rijndael.)

NIST declared that it was looking for a block cipher *as secure as triple-DES, but much more efficient.* Another mandatory requirement was that the submitters agreed to make their cipher available on a world wide royalty-free basis, if it would be selected as the AES. In order to qualify as an official AES candidate, the designers had to provide:

1. A complete written specification of the block cipher in the form of an algorithm.
2. A reference implementation in ANSI C, and mathematically optimized implementations in ANSI C and Java.
3. Implementations of a series of known-answer and Monte Carlo tests, as well as the expected outputs of these tests for a correct implementation of their block cipher.

4. Statements concerning the estimated computational efficiency in both hardware and software, the expected strength against cryptanalytic attacks, and the advantages and limitations of the cipher in various applications.
5. An analysis of the cipher's strength against known cryptanalytic attacks.

It turned out that the required effort to produce a 'complete and proper' submission package would already filter out several of the proposals. Early in the submission stage, the Cryptix team announced that they would provide Java implementations for all submitted ciphers, as well as Java implementations of the known-answer and Monte Carlo tests. This generous offer took some weight off the designers' shoulders, but still the effort required to compile a submission package was too heavy for some designers. The fact that the AES Application Programming Interface (API), which all submissions were required to follow, was updated two times during the submission stage, increased the workload. Table 1.1 lists (in alphabetical order) the 15 submissions that were completed in time and accepted.

Table 1.1. The 15 AES candidates accepted for the first evaluation round.

Submissions	Submitter(s)	Submitter type
CAST-256	Entrust (CA)	Company
Crypton	Future Systems (KR)	Company
DEAL	Outerbridge, Knudsen (USA–DK)	Researchers
DFC	ENS-CNRS (FR)	Researchers
E2	NTT (JP)	Company
Frog	TecApro (CR)	Company
HPC	Schroeppel (USA)	Researcher
LOKI97	Brown et al. (AU)	Researchers
Magenta	Deutsche Telekom (DE)	Company
Mars	IBM (USA)	Company
RC6	RSA (USA)	Company
Rijndael	Daemen and Rijmen (BE)	Researchers
SAFER+	Cylink (USA)	Company
Serpent	Anderson, Biham, Knudsen (UK–IL–DK)	Researchers
Twofish	Counterpane (USA)	Company

1.4 The First Round

The selection process was divided into several stages, with a public workshop to be held near the end of each stage. The process started with a *submission*

stage, which ended on 15 May 1998. All accepted candidates were presented at *The First Advanced Encryption Standard Candidate conference*, held in Ventura, California, on 20-22 August 1998. This was the official start of the first evaluation round, during which the international cryptographic community was asked for comments on the candidates.

1.5 Evaluation Criteria

The evaluation criteria for the first round were divided into three major categories: security, cost and *algorithm and implementation characteristics*. NIST invited the cryptology community to mount attacks and try to cryptanalyse the different candidates, and anyone interested to evaluate implementation cost. The result could be sent to NIST as public comments or be submitted for presentation at the second AES conference. NIST collected all contributions and would use these to select five finalists. In the following sections we discuss the evaluation criteria.

1.5.1 Security

Security was the most important category, but perhaps the most difficult to assess. Only a small number of candidates showed some theoretical design flaws. The large majority of the candidates fell into the category 'no weakness demonstrated'.

1.5.2 Costs

The 'costs' of the candidates were divided into different subcategories. A first category was formed by costs associated with intellectual property (IP) issues. First of all, each submitter was required to make his cipher available for free if it would be selected as the AES. Secondly, each submitter was also asked to make a signed statement that he would not claim ownership or exercise patents on ideas used in *another submitter's proposal* that would eventually be selected as AES. A second category of 'costs' was formed by costs associated with the implementation and execution of the candidates. This covers aspects such as computational efficiency, program size and working memory requirements in software implementations, and chip area in dedicated hardware implementations.

1.5.3 Algorithm and Implementation Characteristics

The category *algorithm and implementation characteristics* grouped a number of features that are harder to quantify. The first one is *versatility*, meaning

the ability to be implemented efficiently on different platforms. At one end of the spectrum should the AES fit 8-bit micro-controllers and smart cards, which have limited storage for the program and a very restricted amount of RAM for working memory. At the other end of the spectrum the AES should be implementable efficiently in dedicated hardware, e.g. to provide on-the-fly encryption/decryption of communication links at gigabit-per-second rates. In between there is the whole range of processors that are used in servers, workstations, PCs, palmtops etc., which are all devices in need of cryptographic support. A prominent place in this range is taken by the Pentium family of processors due to its presence in most personal computers.

A second feature is *key agility*. In most block ciphers, key set up takes some processing. In applications where the same key is used to encrypt large amounts of data, this processing is relatively unimportant. In applications where the key often changes, such as the encryption of Internet Protocol (IP) packets in Internet Protocol Security (IPSEC), the overhead due to key setup may become quite relevant. Obviously, in those applications it is an advantage to have a fast key setup.

Finally, there is the criterion of *simplicity*, that may even be harder to evaluate than cryptographic security. Simplicity is related to the size of the description, the number of different operations used in the specification, symmetry or lack of symmetry in the cipher and the ease with which the algorithm can be understood. All other things equal, NIST considered it to be an advantage for an AES candidate to be more simple for reasons of ease of implementation and confidence in security.

1.6 Selection of Five Finalists

In March 1999, the second AES conference was held in Rome, Italy. The remarkable fact that a US Government department organized a conference on a future US Standard in Europe is easily explained. NIST chose to combine the conference with the yearly Fast Software Encryption Workshop that had for the most part the same target audience and that was scheduled to be in Rome.

1.6.1 The Second AES Conference

The papers presented at the conference ranged from crypto-attacks, cipher cross-analysis, smart-card-related papers, and so-called *algorithm observations*. In the session on cryptographic attacks, it was shown that FROG, Magenta and LOKI97 did not satisfy the security requirements imposed by NIST. For DEAL it was already known in advance that that the security requirements were not satisfied. For HPC weaknesses had been demonstrated in a paper previously sent to NIST. This eliminated five candidates.

Some cipher cross-analysis papers focused on performance evaluation. The paper of B. Gladman [37], a researcher who had no link with any submission, considered performance on the Pentium processor. From this paper it became clear that RC6, Rijndael, Twofish, MARS and Crypton where the five fastest ciphers on this processor. On the other hand, the candidates DEAL, Frog, Magenta, SAFER+ and Serpent appeared to be problematically slow. Other papers by the Twofish team (Bruce Schneier et al.) [84] and a French team of 12 cryptographers [5] essentially confirmed this.

A paper by E. Biham warned that the security margins of the AES candidates differed greatly and that this should be taken into account in the performance evaluation [7]. The lack of speed of Serpent (with E. Biham in the design team) was seen to be compensated with a very high margin of security. Discussions on how to measure and take into account security margins lasted until after the third AES conference.

In the session on smart cards there were two papers comparing the performance of AES candidates on typical 8-bit processors and a 32-bit processor: one by G. Keating [48] and one by G. Hachez et al. [40]. From these papers and results from other papers, it became clear that some candidates simply do not fit into a smart card and that Rijndael is by far the best suited for this platform. In the same session there were some papers that discussed power analysis attacks and the suitability of the different candidates for implementations that can resist against these attacks [10, 15, 27].

Finally, in the *algorithm observations* session, there were a number of papers in which AES submitters re-confirmed their confidence in their submission by means of a considerable amount of formulas, graphs and tables and some *loyal cryptanalysis* (the demonstration of having found no weaknesses after attacks of its own cipher).

1.6.2 The Five Finalists

After the workshop there was a relatively calm period that ended with the announcement of the five candidates by NIST in August 1999. The finalists were (in alphabetical order): MARS, RC6, Rijndael, Serpent and Twofish.

Along with the announcement of the finalists, NIST published a status report [72] in which the selection was motivated. The choice coincided with the top five that resulted from the response to a questionnaire handed out at the end of the second AES workshop. Despite its moderate performance, Serpent made it thanks to its high security margin. The candidates that had not been eliminated because of security problems were not selected mainly for the following reasons:

1. CAST-256: comparable to Serpent but with a higher implementation cost.

2. Crypton: comparable to Rijndael and Twofish but with a lower security margin.
3. DFC: low security margin and bad performance on anything other than 64-bit processors.
4. E2: comparable to Rijndael and Twofish in structure, but with a lower security margin and higher implementation cost.
5. SAFER+: high security margin similar to Serpent but even slower.

1.7 The Second Round

After the announcement of the five candidates NIST made another open call for contributions focused on the finalists. Intellectual property issues and performance and chip area in dedicated hardware implementations entered the picture. A remarkable contribution originated from NSA, presenting the results of hardware performance simulations performed for the finalists. This third AES conference was held in New York City in April 2000. As in the year before, it was combined with the Fast Software Encryption Workshop.

In the sessions on cryptographic attacks there were some interesting results but no breakthroughs, since none of the finalists showed any weaknesses that could jeopardize their security. Most of the results were attacks on reduced-round versions of the ciphers. All attacks presented are only of academic relevance in that they are only slightly faster than an exhaustive key search. In the sessions on software implementations, the conclusions of the second workshop were confirmed.

In the sessions on dedicated hardware implementations there was attention for Field Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs). In the papers Serpent came out as a consistently excellent performer. Rijndael and Twofish also proved to be quite suited for hardware implementation while RC6 turned out to be expensive due to its use of 32-bit multiplication. Dedicated hardware implementations of MARS seemed in general to be quite costly. The Rijndael related results presented at this conference are discussed in more detail in Chap. 4 (which is on efficient implementations) and Chap. 10 (which is on cryptanalytic results).

At the end of the conference a questionnaire was handed out asking about the preferences of the attendants. Rijndael resoundingly voted as the public's favourite.

1.8 The Selection

On 2 October, 2000, NIST officially announced that Rijndael, without modifications, would become the AES. NIST published an excellent 116-page report

in which they summarize all contributions and motivate the choice [71]. In the conclusion of this report, NIST motivates the choice of Rijndael with the following words.

> Rijndael appears to be consistently a very good performer in both hardware and software across a wide range of computing environments regardless of its use in feedback or non-feedback modes. Its key setup time is excellent, and its key agility is good. Rijndael's very low memory requirements make it very well suited for restricted-space environments, in which it also demonstrates excellent performance. Rijndael's operations are among the easiest to defend against power and timing attacks. Additionally, it appears that some defense can be provided against such attacks without significantly impacting Rijndael's performance.
>
> Finally, Rijndael's internal round structure appears to have good potential to benefit from instruction-level parallelism.

2. Preliminaries

In this chapter we introduce a number of mathematical concepts and explain the terminology that we need in the specification of Rijndael (in Chap. 3), in the treatment of some implementation aspects (in Chap. 4) and when we discuss our design choices (Chaps. 5–9).

The first part of this chapter starts with a discussion of finite fields, the representation of its elements and the impact of this on its operations of addition and multiplication. Subsequently, there is a short introduction to linear codes. Understanding the mathematics is not necessary for a full and correct implementation of the cipher. However, the mathematics is necessary for a good understanding of our design motivations. Knowledge of the underlying mathematical constructions also helps for doing optimised implementations. Not all aspects will be covered in detail; where possible, we refer to books dedicated to the topics we introduce.

In the second part of this chapter we introduce the terminology that we use to indicate different common types of Boolean functions and block ciphers. Finally, we give a short overview of the modes of operation of a block cipher.

When the discussion moves from a general level to an example specific to Rijndael, the text is put in a grey box.

Notation. We use in this book two types of indexing:

subscripts: Parts of a larger, named structure are denoted with subscripts. For instance, the bytes of a state $\mathbf{a}$ are denoted by $a_{i,j}$ (see Chap. 3).

superscripts: In an enumeration of more or less independent objects, where the objects are denoted by their own symbols, we use superscripts. For instance the elements of a nameless set are denoted by $\{\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \ldots\}$, and consecutive rounds of an iterative transformation are denoted by $\rho^{(1)}, \rho^{(2)}, \ldots$ (see Sect. 2.3.4).

2.1 Finite Fields

In this section we present a basic introduction to the theory of finite fields. For a more formal and elaborate introduction, we refer to the work of Lidl and Niederreiter [58].

2.1.1 Groups, Rings, and Fields

We start with the formal definition of a group.

Definition 2.1.1. *An* Abelian group $< G, + >$ *consists of a set* G *and an operation defined on its elements, here denoted by '+':*

$$+ : G \times G \to G : (a, b) \mapsto a + b. \tag{2.1}$$

In order to qualify as an Abelian group, *the operation has to fulfill the following conditions:*

$$\begin{aligned}
\textit{closed:} &\ \forall\ a, b \in G : a + b \in G && (2.2)\\
\textit{associative:} &\ \forall\ a, b, c \in G : (a + b) + c = a + (b + c) && (2.3)\\
\textit{commutative:} &\ \forall\ a, b \in G : a + b = b + a && (2.4)\\
\textit{neutral element:} &\ \exists\, \mathbf{0} \in G, \forall\ a \in G : a + \mathbf{0} = a && (2.5)\\
\textit{inverse elements:} &\ \forall\ a \in G, \exists\ b \in G : a + b = \mathbf{0} && (2.6)
\end{aligned}$$

Example 2.1.1. The best-known example of an Abelian group is $< \mathbb{Z}, + >$: the set of integers, with the operation 'addition'. The structure $< \mathbb{Z}_n, + >$ is a second example. The set contains the integer numbers 0 to $n - 1$ and the operation is addition modulo n.

Since the addition of integers is the best known example of a group, usually the symbol '+' is used to denote an arbitrary group operation. In this book, both an arbitrary group operation and integer addition will be denoted by the symbol '+'. For some special types of groups, we will denote the addition operation by the symbol '$\oplus$' (see Sect. 2.1.3).

Both rings and fields are formally defined as structures that consist of a set of elements with *two* operations defined on these elements.

Definition 2.1.2. *A* ring $< R, +, \cdot >$ *consists of a set* R *with two operations defined on its elements, here denoted by '+' and '·'. In order to qualify as a ring, the operations have to fulfill the following conditions:*

1. *The structure* $< R, + >$ *is an Abelian group.*
2. *The operation '·' is closed, and associative over* R. *There is a neutral element for '·' in* R.

3. *The two operations '+' and '·' are related by the law of distributivity:*

$$\forall\ a, b, c \in R:\ (a + b) \cdot c = (a \cdot c) + (b \cdot c). \tag{2.7}$$

The neutral element for '·' is usually denoted by **1**. A ring $< R, +, \cdot >$ is called a *commutative ring* if the operation '·' is commutative.

Example 2.1.2. The best-known example of a ring is $< \mathbb{Z}, +, \cdot >$: the set of integers, with the operations 'addition' and 'multiplication'. This ring is a commutative ring. The set of matrices with n rows and n columns, with the operations 'matrix addition' and 'matrix multiplication' is a ring, but not a commutative ring (if $n > 1$).

Definition 2.1.3. *A structure* $< F, +, \cdot >$ *is a* field *if the following two conditions are satisfied:*

1. $< F, +, \cdot >$ *is a commutative ring.*
2. *For all elements of* F*, there is an inverse element in* F *with respect to the operation '·', except for the element* **0***, the neutral element of* $< F, + >$.

A structure $< F, +, \cdot >$ is a field iff both $< F, + >$ and $< F \backslash \{\mathbf{0}\}, \cdot >$ are Abelian groups and the law of distributivity applies. The neutral element of $< F \backslash \{\mathbf{0}\}, \cdot >$ is called the *unit element* of the field.

Example 2.1.3. The best-known example of a field is the set of real numbers, with the operations 'addition' and 'multiplication.' Other examples are the set of complex numbers and the set of rational numbers, with the same operations. Note that for these examples the number of elements is infinite.

2.1.2 Vector Spaces

Let $< F, +, \cdot >$ be a field, with unit element **1**, and let $< V, + >$ be an Abelian group. Let '$\odot$' be an operation on elements of F and V:

$$\odot:\ F \times V \to V. \tag{2.8}$$

Definition 2.1.4. *The structure* $< F, V, +, +, \cdot, \odot >$ *is a* vector space *over* F *if the following conditions are satisfied:*

1. *distributivity:*

$$\forall\ a \in F, \forall\ \mathbf{v}, \mathbf{w} \in V:\ a \odot (\mathbf{v} + \mathbf{w}) = (a \odot \mathbf{v}) + (a \odot \mathbf{w}) \tag{2.9}$$

$$\forall\ a, b \in F, \forall\ \mathbf{v} \in V:\ (a + b) \odot \mathbf{v} = (a \odot \mathbf{v}) + (a \odot \mathbf{v}) \tag{2.10}$$

2. *associativity:*

$$\forall\ a, b \in F, \forall\ \mathbf{v} \in V: (a \cdot b) \odot \mathbf{v} = a \odot (b \odot \mathbf{v}) \tag{2.11}$$

3. neutral element:

$$\forall\ \mathbf{v} \in V : \mathbf{1} \odot \mathbf{v} = \mathbf{v}. \tag{2.12}$$

The elements of V are called *vectors*, and the elements of F are the *scalars*. The operation '$+$' is called the *vector addition*, and '$\odot$' is the *scalar multiplication*.

Example 2.1.4. For any field F, the set of n-tuples $(a_0, a_1, \ldots, a_{n-1})$ forms a vector space, where '$+$' and '$\odot$' are defined in terms of the field operations:

$$(a_1, \ldots, a_n) + (b_1, \ldots, b_n) = (a_1 + b_1, \ldots, a_n + b_n) \tag{2.13}$$

$$a \odot (b_1, \ldots, b_n) = (a \cdot b_1, \ldots, a \cdot b_n). \tag{2.14}$$

A vector $\mathbf{v}$ is a *linear combination* of the vectors $\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \ldots, \mathbf{w}^{(s)}$ if there exist scalars $a^{(i)}$ such that:

$$\mathbf{v} = a^{(1)} \odot \mathbf{w}^{(1)} + a^{(2)} \odot \mathbf{w}^{(2)} + \cdots + a^{(s)} \odot \mathbf{w}^{(s)}. \tag{2.15}$$

In a vector space we can always find a set of vectors such that all elements of the vector space can be written in exactly one way as a linear combination of the vectors of the set. Such a set is called a *basis* of the vector space. We will consider only vector spaces where the bases have a finite number of elements. We denote a basis by

$$\mathsf{e} = \left[\mathbf{e}^{(1)}, \mathbf{e}^{(2)}, \ldots \mathbf{e}^{(n)}\right]^{\mathrm{T}}. \tag{2.16}$$

In this expression the T superscript denotes taking te transpose of the column vector e. The scalars used in this linear combination are called the *coordinates* of $\mathbf{x}$ with respect to the basis e:

$$\mathrm{co}(\mathbf{x}) = \boldsymbol{x} = (c_1, c_2, \ldots, c_n) \Leftrightarrow \mathbf{x} = \textstyle\sum_{i=1}^{n} c_i \odot \mathbf{e}^{(i)}. \tag{2.17}$$

In order to simplify the notation, from now on we will denote vector addition by the same symbol as the field addition ('$+$'), and the scalar multiplication by the same symbol as the field multiplication ('$\cdot$'). It should always be clear from the context what operation the symbols are referring to.

A function f is called *a linear function* of a vector space V over a field F, if it has the following properties:

$$\forall\ \mathbf{x}, \mathbf{y} \in V :\ f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y}) \tag{2.18}$$

$$\forall\ a \in F, \forall\ \mathbf{x} \in V :\ f(a\mathbf{x}) = af(\mathbf{x}). \tag{2.19}$$

The linear functions of a vector space $\mathrm{GF}(p)^n$ iff there exists a matrix M such that

$$\mathrm{co}(f(\mathbf{x})) = \mathsf{M} \cdot \boldsymbol{x}, \forall\ \mathbf{x} \in \mathrm{GF}(p)^n. \tag{2.20}$$

2.1.3 Fields with a Finite Number of Elements

A *finite field* is a field with a finite number of elements. The number of elements in the set is called the *order* of the field. A field with order m exists iff m is a *prime power*, i.e. $m = p^n$ for some integer n and with p a prime integer. p is called the *characteristic* of the finite field.

All finite fields used in the description of Rijndael have a characteristic of 2. By the symbol '$\oplus$', we will always denote the addition operation in a field with a characteristic of 2.

Fields of the same order are *isomorphic*: they display exactly the same algebraic structure differing only in the *representation* of the elements. In other words, for each prime power there is exactly one finite field, denoted by $\mathrm{GF}(p^n)$. From now on, we will only consider fields with a finite number of elements.

Perhaps the most intuitive examples of finite fields are the fields of prime order p. The elements of a finite field $\mathrm{GF}(p)$ can be represented by the integers $0, 1, \ldots, p-1$. The two operations of the field are then 'integer addition modulo p' and 'integer multiplication modulo p'.

For finite fields with an order that is not prime, the operations *addition* and *multiplication* cannot be represented by addition and multiplication of integers modulo a number. Instead, slightly more complex representations must be introduced. Finite fields $\mathrm{GF}(p^n)$ with $n > 1$ can be represented in several ways. The representation of $\mathrm{GF}(p^n)$ by means of polynomials over $\mathrm{GF}(p)$ is quite popular and is the one we have adopted in Rijndael and its predecessors. In the next sections, we explain this representation.

2.1.4 Polynomials over a Field

A polynomial over a field F is an expression of the form

$$b(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_2x^2 + b_1x + b_0, \tag{2.21}$$

x being called the *indeterminate* of the polynomial, and the $b_i \in F$ the *coefficients*.

We will consider polynomials as abstract entities only, which are never evaluated. Because the sum is never evaluated, we always use the symbol '+' in polynomials, even if they are defined over a field with characteristic 2.

The *degree* of a polynomial equals ℓ if $b_j = 0, \forall j > \ell$, and ℓ is the smallest number with this property. The set of polynomials over a field F is denoted by $F[x]$. The set of polynomials over a field F, which have a degree below ℓ, is denoted by $F[x]|_\ell$.

In computer memory, the polynomials in $F[x]|_\ell$ with F a finite field can be stored efficiently by storing the ℓ coefficients as a string.

Example 2.1.5. Let the field F be GF(2), and let $\ell = 8$. The polynomials can conveniently be stored as 8-bit values, or bytes:

$$b(x) \mapsto b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0. \tag{2.22}$$

Strings of bits are often abbreviated using the hexadecimal notation.

Example 2.1.6. The polynomial in $\mathrm{GF}(2)|_8$

$$x^6 + x^4 + x^2 + x + 1$$

corresponds to the bit string 01010111, or `57` in hexadecimal notation.

2.1.5 Operations on Polynomials

We define the following operations on polynomials.

Addition. Summing of polynomials consists of summing the coefficients with equal powers of x, where the summing of the coefficients occurs in the underlying field F:

$$c(x) = a(x) + b(x) \Leftrightarrow c_i = a_i + b_i, \quad 0 \leq i < n. \tag{2.23}$$

The neutral element for the addition $\mathbf{0}$ is the polynomial with all coefficients equal to 0. The inverse element of a polynomial can be found by replacing each coefficient by its inverse element in F. The degree of $c(x)$ is at most the maximum of the degrees of $a(x)$ and $b(x)$, hence the addition is closed. The structure $< F[x]|_\ell, + >$ is an Abelian group.

Example 2.1.7. Let F be the field GF(2). The sum of the polynomials denoted by `57` and `83` is the polynomial denoted by `D4`, since:

$$\begin{aligned}(x^6 + x^4 + x^2 + x + 1) &\oplus (x^7 + x + 1) \\ &= x^7 + x^6 + x^4 + x^2 + (1 \oplus 1)x + (1 \oplus 1) \\ &= x^7 + x^6 + x^4 + x^2.\end{aligned}$$

In binary notation we have: $01010111 \oplus 10000011 = 11010100$. Clearly, the addition can be implemented with the bitwise XOR instruction.

Multiplication. Multiplication of polynomials is associative (2.3), commutative (2.4) and distributive (2.7) with respect to addition of polynomials. There is a neutral element: the polynomial of degree 0 and with coefficient of x^0 equal to 1. In order to make the multiplication closed (2.2) over $F[x]|_\ell$, we select a polynomial $m(x)$ of degree ℓ, called the *reduction polynomial.*

The multiplication of two polynomials $a(x)$ and $b(x)$ is then defined as the algebraic product of the polynomials modulo the polynomial $m(x)$:

$$c(x) = a(x) \cdot b(x) \Leftrightarrow c(x) \equiv a(x) \times b(x) \pmod{m(x)}. \tag{2.24}$$

Hence, the structure $< F[x]|_\ell, +, \cdot >$ is a commutative ring. For special choices of the reduction polynomial $m(x)$, the structure becomes a field.

Definition 2.1.5. *A polynomial $d(x)$ is* irreducible *over the field* GF(p) *iff there exist no two polynomials $a(x)$ and $b(x)$ with coefficients in* GF(p) *such that $d(x) = a(x) \times b(x)$, where $a(x)$ and $b(x)$ are of degree > 0.*

The inverse element for the multiplication can be found by means of the extended Euclidean algorithm (see e.g. [68, p. 81]). Let $a(x)$ be the polynomial we want to find the inverse for. The extended Euclidean algorithm can then be used to find two polynomials $b(x)$ and $c(x)$ such that:

$$a(x) \times b(x) + m(x) \times c(x) = \gcd(a(x), m(x)). \tag{2.25}$$

Here $\gcd(a(x), m(x))$ denotes the greatest common divisor of the polynomials $a(x)$ and $m(x)$, which is always equal to **1** iff $m(x)$ is irreducible. Applying modular reduction to (2.25), we get:

$$a(x) \times b(x) \equiv \mathbf{1} \pmod{m(x)}, \tag{2.26}$$

which means that $b(x)$ is the inverse element of $a(x)$ for the definition of the multiplication '$\cdot$' given in (2.24).

Conclusion. Let F be the field GF(p). With a suitable choice for the reduction polynomial, the structure $< F[x]|_n, +, \cdot >$ is a field with p^n elements, usually denoted by GF(p^n).

2.1.6 Polynomials and Bytes

According to (2.22) a byte can be considered as a polynomial with coefficients in GF(2):

$$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \mapsto b(x) \tag{2.27}$$

$$b(x) = b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0. \tag{2.28}$$

The set of all possible byte values corresponds to the set of all polynomials with degree less than eight. Addition of bytes can be defined as addition of the corresponding polynomials. In order to define the multiplication, we need to select a reduction polynomial $m(x)$.

In the specification of Rijndael, we consider the bytes as polynomials. Byte addition is defined as addition of the corresponding polynomials. In order to define byte multiplication, we use the following irreducible polynomial as reduction polynomial:

$$m(x) = x^8 + x^4 + x^3 + x + 1. \tag{2.29}$$

Since this reduction polynomial is irreducible, we have constructed a representation for the field $\mathrm{GF}(2^8)$. Hence we can state the following: In the Rijndael specification, bytes are considered as elements of $\mathrm{GF}(2^8)$. Operations on bytes are defined as operations in $\mathrm{GF}(2^8)$.

Example 2.1.8. In our representation for $\mathrm{GF}(2^8)$, the product of the elements denoted by `57` and `83` is the element denoted by `C1`, since:

$$\begin{aligned}(x^6 + x^4 + x^2 + x + 1) &\times (x^7 + x + 1) \\ &= (x^{13} + x^{11} + x^9 + x^8 + x^7) \oplus (x^7 + x^5 + x^3 + x^2 + x) \\ &\quad \oplus (x^6 + x^4 + x^2 + x + 1) \\ &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1\end{aligned}$$

and

$$\begin{aligned}(x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1) \\ \equiv x^7 + x^6 + 1 \pmod{x^8 + x^4 + x^3 + x + 1}.\end{aligned}$$

As opposed to addition, there is no simple equivalent processor instruction.

2.1.7 Polynomials and Columns

In the Rijndael specification, 4-byte columns are considered as polynomials over $\mathrm{GF}(2^8)$, having a degree smaller than four. In order to define the multiplication operation, the following reduction polynomial is used:

$$l(x) = x^4 + 1. \tag{2.30}$$

This polynomial is reducible, since in $\mathrm{GF}(2^8)$

$$x^4 + 1 = (x + 1)^4. \tag{2.31}$$

In the definition of Rijndael, one of the inputs of the multiplication is a constant polynomial.

Since $l(x)$ is reducible over $\mathrm{GF}(2^8)$, not all polynomials have an inverse element for the multiplication modulo $l(x)$. A polynomial $b(x)$ has an inverse if the polynomial $x + 1$ does not divide it.

Multiplication with a fixed polynomial. We work out in more detail the multiplication with the fixed polynomial used in Rijndael.

Let $b(x)$ be the fixed polynomial with degree three:

$$b(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3 \tag{2.32}$$

and let $c(x)$ and $d(x)$ be two variable polynomials with coefficients c_i and d_i, respectively ($0 \leq i < 4$). We derive the matrix representation of the transformation that takes as input the coefficients of polynomial c, and produces as output the coefficients of the polynomial $d = b \times c$. We have:

$$\begin{aligned} & d = b \cdot c && (2.33) \\ & \Updownarrow \\ & (b_0 + b_1 x + b_2 x^2 + b_3 x^3) \times (c_0 + c_1 x + c_2 x^2 + c_3 x^3) \\ & \quad \equiv (d_0 + d_1 x + d_2 x^2 + d_3 x^3) \pmod{x^4 + 1} && (2.34) \end{aligned}$$

Working out the product and separating the conditions for different powers of x, we get:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} b_0 & b_3 & b_2 & b_1 \\ b_1 & b_0 & b_3 & b_2 \\ b_2 & b_1 & b_0 & b_3 \\ b_3 & b_2 & b_1 & b_0 \end{bmatrix} \times \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}. \tag{2.35}$$

2.2 Linear Codes

In this section we give a short introduction to the theory of linear codes. For a more detailed treatment, we refer the interested reader to the work of MacWilliams and Sloane [63]. In code theory textbooks, it is customary to write codewords as $1 \times n$ matrices, or *row vectors.* We will follow that custom here. In further chapters, one-dimensional arrays will as often be denoted as $n \times 1$ matrices, or *column vectors.*

2.2.1 Definitions

The *Hamming weight* of a codeword is defined as follows.

Definition 2.2.1. *The* Hamming weight $\mathrm{w_h}(\mathbf{x})$ *of a vector* $\mathbf{x}$ *is the number of nonzero components of the vector* $\mathbf{x}$.

Based on the definition of Hamming weight, we can define the *Hamming distance* between two vectors.

Definition 2.2.2. *The* Hamming distance *between two vectors* $\mathbf{x}$ *and* $\mathbf{y}$ *is* $\mathrm{w_h}(\mathbf{x}-\mathbf{y})$*, which is equal to the Hamming weight of the difference of the two vectors.*

Now we are ready to define linear codes.

Definition 2.2.3. *A* linear $[n,k,d]$ code *over* $\mathrm{GF}(2^p)$ *is a* k*-dimensional subspace of the vector space* $\mathrm{GF}(2^p)^n$*, where any two different vectors of the subspace have a Hamming distance of at least* d *(and* d *is the largest number with this property).*

The distance d of a linear code equals the minimum weight of a non-zero codeword in the code. A linear code can be described by each of the two following matrices:

1. A *generator* matrix G for an $[n,k,d]$ code $\mathcal{C}$ is a $k \times n$ matrix whose rows form a vector space basis for $\mathcal{C}$ (only generator matrices of full rank are considered). Since the choice of a basis in a vector space is not unique, a code has many different generator matrices that can be reduced to one another by performing elementary row operations. The *echelon form* of the generator matrix is the following:

$$\mathsf{G}_e = \left[\mathsf{I}_{k\times k}\ \mathsf{A}_{k\times(n-k)}\right], \tag{2.36}$$

   where $\mathsf{I}_{k\times k}$ is the $k \times k$ identity matrix.
2. A *parity-check* matrix H for an $[n,k,d]$ code $\mathcal{C}$ is an $(n-k) \times k$ matrix with the property that a vector $\mathbf{x}$ is a codeword of $\mathcal{C}$ iff

$$\mathsf{H}\mathbf{x}^{\mathrm{T}} = 0. \tag{2.37}$$

If G is a generator matrix and H a parity-check matrix of the same code, then

$$\mathsf{G}\mathsf{H}^{\mathrm{T}} = 0. \tag{2.38}$$

Moreover, if $\mathsf{G} = [\mathsf{I}\ \mathsf{C}]$ is a generator matrix of a code, then $\mathsf{H} = \left[-\mathsf{C}^{\mathrm{T}}\ \mathsf{I}\right]$ is a parity-check matrix of the same code.

The *dual code* $\mathcal{C}^{\perp}$ of a code $\mathcal{C}$ is defined as the set of vectors that are orthogonal to all the vectors of $\mathcal{C}$:

$$\mathcal{C}^{\perp} = \{\mathbf{x} \mid \mathbf{x}\mathbf{y}^{\mathrm{T}} = 0, \forall\ \mathbf{y} \in \mathcal{C}\}. \tag{2.39}$$

It follows that a parity-check matrix of $\mathcal{C}$ is a generator matrix of $\mathcal{C}^{\perp}$ and vice versa.

2.2.2 MDS codes

The theory of linear codes addresses the problems of determining the distance of a linear code and the construction of linear codes with a given distance. We review a few well-known results.

The Singleton bound gives an upper bound for the distance of a code with given dimensions.

Theorem 2.2.1 (The Singleton bound). *If $\mathcal{C}$ is an $[n, k, d]$ code, then $d \leq n - k + 1$.*

A code that meets the Singleton bound, is called a *maximal distance separable (MDS) code.* The following theorems relate the distance of a code to properties of the generator matrix G.

Theorem 2.2.2. *A linear code $\mathcal{C}$ has distance d iff every $d - 1$ columns of the parity check matrix H are linearly independent and there exists some set of d columns that are linearly dependent.*

By definition, an MDS-code has distance $n - k + 1$. Hence, every set of $n - k$ columns of the parity-check matrix are linearly independent. This property can be translated to a requirement for the matrix A:

Theorem 2.2.3 ([63]). *An $[n, k, d]$ code with generator matrix*

$$\mathsf{G} = \left[\mathsf{I}_{k\times k}\ \mathsf{A}_{k\times(n-k)}\right],$$

is an MDS code iff every square submatrix of A is nonsingular.

A well-known class of MDS codes is formed by the Reed-Solomon codes, for which efficient construction algorithms are known.

2.3 Boolean Functions

The smallest finite field has an order of 2: GF(2). Its two elements are denoted by **0** and **1**. Its addition is the integer addition modulo 2 and its multiplication is the integer multiplication modulo 2. Variables that range over GF(2) are called *Boolean variables*, or *bits* for short. The addition of 2 bits corresponds with the Boolean operation *exclusive or*, denoted by XOR. The multiplication of 2 bits corresponds to the Boolean operation AND. The operation of changing the value of a bit is called *complementation.*

A vector whose coordinates are bits is called a *Boolean vector.* The operation of changing the value of all bits of a Boolean vector is called complementation.

If we have two Boolean vectors $\mathbf{a}$ and $\mathbf{b}$ of the same dimension, we can apply the following operations:

1. Bitwise XOR: results in a vector whose bits consist of the XOR of the corresponding bits of $\mathbf{a}$ and $\mathbf{b}$.
2. Bitwise AND: results in a vector whose bits consist of the AND of the corresponding bits of $\mathbf{a}$ and $\mathbf{b}$.

A function $\mathbf{b} = \phi(\mathbf{a})$ that maps a Boolean vector to another Boolean vector is called a *Boolean function*:

$$\phi : \mathrm{GF}(2)^n \rightarrow \mathrm{GF}(2)^m : \mathbf{a} \mapsto \mathbf{b} = \phi(\mathbf{a}), \tag{2.40}$$

where $\mathbf{b}$ is called the output Boolean vector and $\mathbf{a}$ the input Boolean vector. This Boolean function has n input bits and m output bits.

A *binary Boolean function* $b = f(\mathbf{a})$ is a Boolean function with a single output bit, in other words $m = 1$:

$$f : \mathrm{GF}(2)^n \rightarrow \mathrm{GF}(2) : \mathbf{a} \mapsto b = f(\mathbf{a}), \tag{2.41}$$

where b is called the output bit. Each bit of the output of a Boolean function is itself a binary Boolean function of the input vector. These functions are called the *component binary Boolean functions* of the Boolean function.

A Boolean function can be specified by providing the output value for the 2^n possible values of the input Boolean vector. A Boolean function with the same number of input bits as output bits can be considered as operating on an n-bit *state*. We call such a function a *Boolean transformation*. A Boolean transformation is called *invertible* if it maps all input states to different output states. An invertible Boolean transformation is called a *Boolean permutation*.

2.3.1 Bundle Partitions

In several instances it is useful to see the bits of a state as being partitioned into a number of subsets, called *bundles*. Boolean transformations operating on a state can be expressed in terms of these bundles rather than in terms of the individual bits of the state. In the context of this book we restrict ourselves to bundle partitions that divide the state bits into a number of equally sized bundles.

Consider an n_b-bit state $\mathbf{a}$ consisting of bits a_i where $i \in \mathcal{I}$. $\mathcal{I}$ is called the index space. In its simplest form, the index space is just equal to $\{1, \ldots, n_\mathrm{b}\}$. However, for clarity the bits may be indexed in another way to ease specifications. A bundling of the state bits may be reflected by having an index with two components: one component indicating the bundle position within the state, and one component indicating the bit position within the bundle. In this representation, $a_{(i,j)}$ would mean the state bit in bundle i at bit position

j within that bundle. The value of the bundle itself can be indicated by a_i. On some occasions, even the bundle index can be decomposed. For example, in Rijndael the bundles consist of bytes that are arranged in a two-dimensional array with the byte index composed of a column index and a row index.

Examples of bundles are the 8-bit *bytes* and the 32-bit *columns* in Rijndael. The non-linear steps in the round transformations of the AES finalist Serpent [3] operate on 4-bit bundles. The non-linear step in the round transformation of 3-Way [20] and BaseKing [23] operate on 3-bit bundles. The bundles can be considered as representations of elements in some group, ring or field. Examples are the integers modulo 2^m or elements of GF(2^m). In this way, steps of the round transformation, or even the full round transformation can be expressed in terms of operations in these mathematical structures.

2.3.2 Transpositions

A transposition is a Boolean permutation that only moves the positions of bits of the state without affecting their value. For a transposition $\mathbf{b} = \pi(\mathbf{a})$ we have:

$$b_i = a_{p(i)}, \tag{2.42}$$

where $p(i)$ is a permutation over the index space.

A bundle transposition is a transposition that changes the positions of the bundles but leaves the positions of the bits within the bundles intact. This can be expressed as:

$$b_{(i,j)} = a_{(p(i),j)}. \tag{2.43}$$

An example is shown in Fig. 2.1. Figure 2.2 shows the pictogram that we will use to represent a bundle transposition in this book.

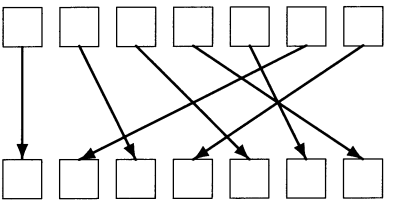

Fig. 2.1. Example of a bundle transposition.

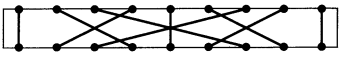

Fig. 2.2. Pictogram for a bundle transposition.

2.3.3 Bricklayer Functions

A *bricklayer function* is a function that can be decomposed into a number of Boolean functions operating independently on subsets of bits of the input vector. These subsets form a partition of the bits of the input vector. A bricklayer function can be considered as the parallel application of a number of Boolean functions operating on smaller inputs. If non-linear, these Boolean functions are called *S-boxes*. If linear, we use the term *D-box*, where D stands for *diffusion*.

A bricklayer function operating on a state is called a *bricklayer transformation*. As a bricklayer transformation operates on a number of subsets of the state independently, it defines a bundle partition. The component transformations of the bricklayer transformation operate independently on a number of bundles. A graphical illustration is given in Fig. 2.3. An invertible bricklayer transformation is called a bricklayer permutation. For a bricklayer transformation to be invertible, all of its S-boxes (or D-boxes) must be permutations. The pictogram that we will use is shown in Fig. 2.4.

For a bricklayer transformation $\mathbf{b} = \phi(\mathbf{a})$ we have:

$$(b_{(i,1)}, b_{(i,2)}, \ldots, b_{i,m}) = \phi_i(a_{(i,1)}, a_{(i,2)}, \ldots, a_{i,m}), \tag{2.44}$$

for all values of i. If the bundles within $\mathbf{a}$ and $\mathbf{b}$ are represented by a_i and b_i, respectively, this becomes:

$$b_i = \phi_i(a_i). \tag{2.45}$$

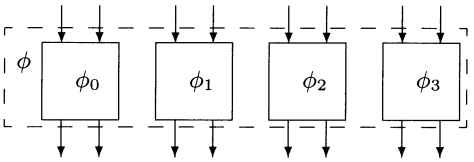

Fig. 2.3. Example of a bricklayer transformation.

Fig. 2.4. Pictogram for a bricklayer transformation.

2.3.4 Iterative Boolean Transformations

A Boolean vector can be transformed iteratively by applying a sequence of Boolean transformations, one after the other. Such a sequence is referred to

as an *iterative Boolean transformation*. If the individual Boolean transformations are denoted with $\rho^{(i)}$, an iterative Boolean transformation is of the form:

$$\beta = \rho^{(r)} \circ \ldots \circ \rho^{(2)} \circ \rho^{(1)}. \tag{2.46}$$

A schematic illustration is given in Fig. 2.5. We have $\mathbf{b} = \beta(\mathbf{d})$, where $\mathbf{d} = \mathbf{a}^{(0)}, \mathbf{b} = \mathbf{a}^{(m)}$ and $\mathbf{a}^{(i)} = \rho^{(i)}(\mathbf{a}^{(i-1)})$. The value of $\mathbf{a}^{(i)}$ is called the *intermediate state*. An iterative Boolean transformation that is a sequence of Boolean permutations is an iterative Boolean permutation.

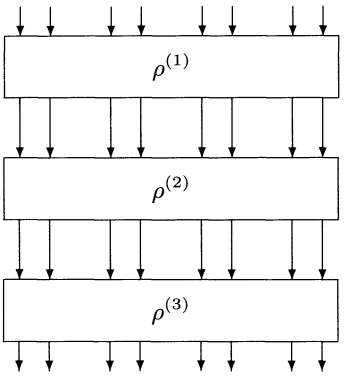

Fig. 2.5. Iterative Boolean transformation.

2.4 Block Ciphers

A *block cipher* transforms *plaintext blocks* of a fixed length n_b to *ciphertext blocks* of the same length under the influence of a cipher key k. More precisely, a block cipher is a set of Boolean permutations operating on n_b-bit vectors. This set contains a Boolean permutation for each value of the *cipher key* k. In this book we only consider block ciphers in which the cipher key is a Boolean vector. If the number of bits in the cipher key is denoted by n_k, a block cipher consists of 2^{n_k} Boolean permutations.

The operation of transforming a plaintext block into a ciphertext block is called *encryption*, and the operation of transforming a ciphertext block into a plaintext block is called *decryption*.

Usually, block ciphers are specified by an *encryption algorithm*, being the sequence of transformations to be applied to the plaintext to obtain the ciphertext. These transformations are operations with a relatively simple description. The resulting Boolean permutation depends on the cipher key

by the fact that key material, computed from the cipher key, is used in the transformations.

For a block cipher to be up to its task, it has to fulfil two requirements:

1. **Efficiency.** Given the value of the cipher key, applying the corresponding Boolean permutation, or its inverse, is efficient, preferably on a wide range of platforms.
2. **Security.** It must be impossible to exploit knowledge of the internal structure of the cipher in cryptographic attacks.

All block ciphers of any significance satisfy these requirements by iteratively applying Boolean permutations that are relatively simple to describe.

2.4.1 Iterative Block Ciphers

In an *iterative block cipher*, the Boolean permutations are iterative. The block cipher is defined as the application of a number of key-dependent Boolean permutations. The Boolean permutations are called the *round transformations* of the block cipher. Every application of a round transformation is called a *round*.

Example 2.4.1. The DES has 16 rounds. Since every round uses the same round transformation, we say the DES has only one round transformation.

We denote the number of rounds by r. We have:

$$B[\mathbf{k}] = \rho^{(r)}[\mathbf{k}^{(r)}] \circ \cdots \circ \rho^{(2)}[\mathbf{k}^{(2)}] \circ \rho^{(1)}[\mathbf{k}^{(1)}]. \tag{2.47}$$

In this expression, $\rho^{(i)}$ is called the ith *round* of the block cipher and $\mathbf{k}^{(i)}$ is called the ith round key.

The round keys are computed from the cipher key. Usually, this is specified with an algorithm. The algorithm that describes how to derive the round keys from the cipher key is called the *key schedule*. The concatenation of all round keys is called the *expanded key*, denoted by K:

$$\mathsf{K} = \mathbf{k}^{(0)}|\mathbf{k}^{(1)}|\mathbf{k}^{(2)}|\ldots|\mathbf{k}^{(r)} \tag{2.48}$$

The length of the expanded key is denoted by n_{K}. The iterative block cipher model is illustrated in Fig. 2.6. Almost all block ciphers known can be modelled this way. There is however a large variety in round transformations and key schedules. An iterative block cipher in which all rounds (with the exception of the initial or final round) use the same round transformation is called an *iterated block cipher*.

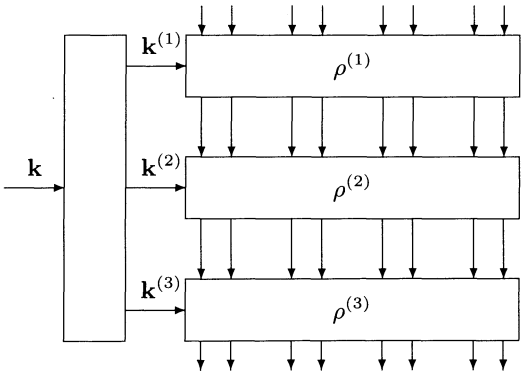

Fig. 2.6. Iterative block cipher with three rounds.

2.4.2 Key-Alternating Block Ciphers

Rijndael belongs to a class of block ciphers in which the round key is applied in a particularly simple way: the key-alternating block ciphers. A key-alternating block cipher is an iterative block cipher with the following properties:

1. **Alternation.** The cipher is defined as the alternated application of key-independent round transformations and key additions. The first round key is added before the first round and the last round key is added after the last round.
2. **Simple key addition.** The round keys are added to the state by means of a simple XOR A key addition is denoted by $\sigma[k]$.

We have:

$$B[\mathbf{k}] = \sigma[\mathbf{k}^{(r)}] \circ \rho^{(r)} \circ \sigma[\mathbf{k}^{(r-1)}] \circ \cdots \circ \sigma[\mathbf{k}^{(1)}] \circ \rho^{(1)} \circ \sigma[\mathbf{k}^{(0)}]. \quad (2.49)$$

A graphical illustration is given in Fig. 2.7.

Key-alternating block ciphers are a class of block ciphers that lend themselves to analysis with respect to the resistance against cryptanalysis. This will become clear in Chaps. 7–9. A special class of key-alternating block ciphers are the *key-iterated block ciphers*. In this class, all rounds (except maybe the first or the last) of the cipher use the same round transformation. We have:

$$B[\mathbf{k}] = \sigma[\mathbf{k}^{(r)}] \circ \rho \circ \sigma[\mathbf{k}^{(r-1)}] \circ \cdots \circ \sigma[\mathbf{k}^{(1)}] \circ \rho \circ \sigma[\mathbf{k}^{(0)}]. \quad (2.50)$$

In this case, ρ is called *the* round transformation of the block cipher. The relations between the different classes of block ciphers that we define here are shown in Fig. 2.8.

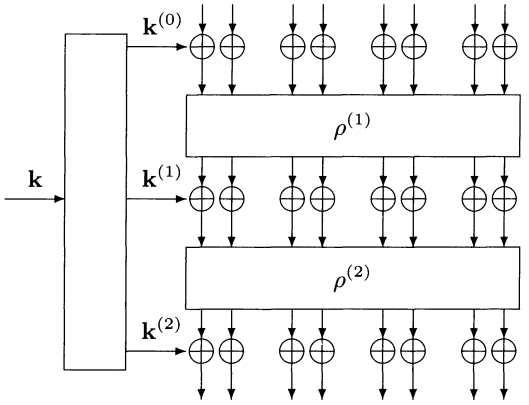

Fig. 2.7. Key-alternating block cipher with two rounds.

Key-iterated block ciphers lend themselves to efficient implementations. In dedicated hardware implementations, one can hard-wire the round transformation and the key addition. The block cipher can be executed by simply iterating the round transformation alternated with the right round keys. In software implementations, the program needs to code only the one round transformation in a loop and the cipher can be executed by executing this loop the required number of times. In practice, for performance reasons, block ciphers will often be implemented by implementing every round separately (so-called *loop unrolling*). In these implementations, it is less important to have identical rounds. Nevertheless, the most-used block ciphers all consist of a number of identical rounds. Some other advantages of the key-iterated structure are discussed in Chap. 5.

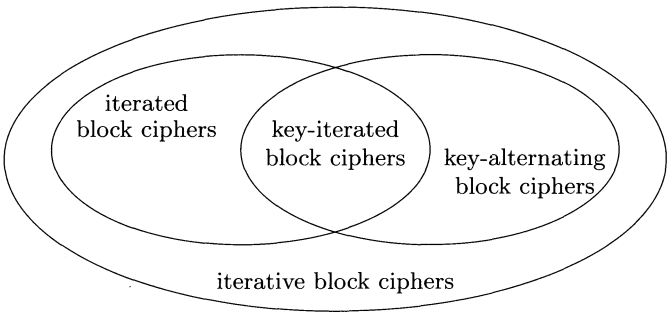

Fig. 2.8. Block cipher taxonomy.

2.5 Block Cipher Modes of Operation

A block cipher is a very simple cryptographic primitive that can convert a plaintext block to a ciphertext block and vice versa under a given cipher key. In order to use a cipher to protect the confidentiality or integrity of long messages, it must be specified how the cipher is used. These specifications are the so-called *modes of operation* of a block cipher. In the following sections, we give an overview of the most-widely applied mode of operation. Modes of encryption are standardized in [43], the use of a block cipher for protecting data integrity is standardized in [42] and cryptographic hashing based on a block cipher is standardized in [44].

2.5.1 Block Encryption Modes

In the block encryption modes, the block cipher is used to transform plaintext blocks into ciphertext blocks and vice versa. The message must be split up into blocks that fit the block length of the cipher. The message can then be encrypted by applying the block cipher to all the blocks independently. The resulting *cryptogram* can be decrypted by applying the inverse of the block cipher to all the blocks independently. This is called the Electronic Code Book mode (ECB).

A disadvantage of the ECB mode is that if the message has two blocks with the same value, so will the cryptogram. For this reason another mode has been proposed: the Cipher Block Chaining (CBC) mode. In this mode, the message blocks are *randomised* before applying the block cipher by performing an XOR with the ciphertext block corresponding with the previous message block. In CBC decryption, a message block is obtained by applying the inverse block cipher followed by an XOR with the previous cryptogram block.

Both ECB and CBC modes have the disadvantage that the length of the message must be an integer multiple of the block length. If this is not the case, the last block must be *padded*, i.e. bits must be appended so that it has the required length. This padding causes the cryptogram to be longer than the message itself, which may be a disadvantage is some applications. For messages that are larger than one block, padding may be avoided by the application of so-called *ciphertext stealing* [70, p. 81], that adds some complexity to the treatment of the last message blocks.

2.5.2 Key-Stream Generation Modes

In so-called *key-stream generation* modes, the cipher is used to generate a *key-stream* that is used for encryption by means of bitwise XOR with a message stream. Decryption corresponds with subtracting (XOR) the key-stream bits from the message. Hence, for correct decryption it suffices to generate the

same key-stream at both ends. It follows that at both ends the same function can be used for the generation of the key-stream and that the inverse cipher is not required to perform decryption. The feedback modes have the additional advantage that there is no need for padding the message and hence that the cryptogram has the same length as the message itself.

In Output Feed Back mode (OFB) and Counter mode, the block cipher is just used as a synchronous key-stream sequence generator. In OFB mode, the key-stream generator is a finite state machine in which the state has the block length of the cipher and the state updating function consists of encryption with the block cipher for some secret value of the key. In Counter mode, the key-stream is the result of applying ECB encryption to a predictable sequence, e.g. an incrementing counter.

In Cipher Feed Back mode (CFB), the key-stream is a key-dependent function of the last n_b bits of the ciphertext. This function consists of encryption with the block cipher for some secret value of the key. Among the key-stream generation modes, the CFB mode has the advantage that decryption is correct from the moment that the last n_b bits of the cryptogram have been correctly received. In other words, it has a self-synchronizing property. In the OFB and Counter modes, synchronization must be assured by external means. For a more thorough treatment of block cipher modes of operation for encryption, we refer to [68, Sect. 7.2.2].

2.5.3 Message Authentication Modes

Many applications do not require the protection of confidentiality of messages but rather the protection of their integrity. As encryption by itself does not provide message integrity, a dedicated algorithm must be used. For this purpose often a cryptographic checksum, requiring a secret key, is computed on a message. Such a cryptographic checksum is called a Message Authentication Code (MAC). In general, the MAC is sent along with the message for the receiving entity to verify that the message has not been modified along the way.

A MAC algorithm can be based on a block cipher. The most widespread way of using a block cipher as a MAC is called the CBC-MAC. in its simplest form it consists of applying a block cipher in CBC mode on a message and taking (part) of the last cryptogram block as the MAC. The generation of a MAC and its verification are very similar processes. The verification consists of reconstructing the MAC from the message using the secret key and comparing it with the MAC received. Hence, similar to the key-stream generation modes of encryption, the CBC-MAC mode of a block cipher does not require decryption with the cipher. For a more thorough treatment of message authentication codes using a block cipher, we refer to [68, Sect. 9.5.1].

2.5.4 Cryptographic Hashing

In some applications, integrity of a message is obtained in two phases: first the message, that may have any length, is compressed to a short, fixed-length message digest with a so-called cryptographic hash function, and subsequently the message digest is authenticated. For some applications this hash function must guarantee that it is infeasible to find two messages that hash to the same message digest (collision resistant). For other applications, it suffices that given a message, no other message can be found so that both hash to the same message digest (second-preimage resistant). For yet other applications it suffices that given a message digest, no message can be found that hashes to that value (one-way or preimage resistant).

A block cipher can be used as the compression function of an iterated hash function by adopting the Davies-Meyer, Matyas-Meyer-Oseas or Miyaguchi-Preneel mode (see [68]). In these modes the length of the hash result (and also the chaining variable) is the block length. In the assumption that the underlying block cipher has no weaknesses, and with the current state of cryptanalysis and technology, a block length of 128 bits is considered sufficient to provide both variants of preimage resistance. If collision resistance is the goal, we advise the adoption of a block length of 256 bits. For a more thorough treatment of cryptographic hashing using a block cipher, we refer to [68, Sect. 9.4.1].

2.6 Conclusions

In this chapter we have given a number of definitions and an introduction to mathematical concepts that are used throughout the book.

3. Specification of Rijndael

In this chapter we specify the cipher structure and the building blocks of Rijndael. After explaining the difference between the Rijndael specifications and the AES standard, we specify the external interface to the ciphers. This is followed by the description of the Rijndael structure and the steps of its round transformation. Subsequently, we specify the number of rounds as a function of the block and key length, and describe the key schedule. We conclude this chapter with a treatment of algorithms for implementing decryption with Rijndael. This chapter is not intended as an implementation guideline. For implementation aspects, we refer to Chap. 4.

3.1 Differences between Rijndael and the AES

The *only* difference between Rijndael and the AES is the range of supported values for the block length and cipher key length.

Rijndael is a block cipher with both a variable block length and a variable key length. The block length and the key length can be independently specified to any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits. It would be possible to define versions of Rijndael with a higher block length or key length, but currently there seems no need for it.

The AES fixes the block length to 128 bits, and supports key lengths of 128, 192 or 256 bits only. The extra block and key lengths in Rijndael were not evaluated in the AES selection process, and consequently they are not adopted in the current FIPS standard.

3.2 Input and Output for Encryption and Decryption

The input and output of Rijndael are considered to be one-dimensional arrays of 8-bit bytes. For encryption the input is a *plaintext block* and a *key*, and the output is a *ciphertext block*. For decryption, the input is a ciphertext block and a key, and the output is a plaintext block. The round transformation of Rijndael, and its steps, operate on an intermediate result, called the *state*.

The state can be pictured as a rectangular array of bytes, with four rows. The number of columns in the state is denoted by $\mathrm{N_b}$ and is equal to the block length divided by 32. Let the plaintext block be denoted by

$$p_0 p_1 p_2 p_3 \ldots p_{4\cdot \mathrm{N_b}-1},$$

where p_0 denotes the first byte,and $p_{4\cdot \mathrm{N_b}-1}$ denotes the last byte of the plaintext block. Similarly, a ciphertext block can be denoted by

$$c_0 c_1 c_2 c_3 \ldots c_{4\cdot \mathrm{N_b}-1}.$$

Let the state be denoted by

$$a_{i,j},\ 0 \le i < 4, 0 \le j < \mathrm{N_b}.$$

where $a_{i,j}$ denotes the byte in row i and column j. The input bytes are mapped onto the state bytes in the order $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$, $a_{1,1}$, $a_{2,1}$, $a_{3,1}$, ... For encryption, the input is a plaintext block and the mapping is

$$a_{i,j} = p_{i+4j},\ 0 \le i < 4, 0 \le j < \mathrm{N_b}. \tag{3.1}$$

For decryption, the input is a ciphertext block and the mapping is

$$a_{i,j} = c_{i+4j},\ 0 \le i < 4, 0 \le j < \mathrm{N_b}. \tag{3.2}$$

At the end of the encryption, the ciphertext is extracted from the state by taking the state bytes in the same order:

$$c_i = a_{i \bmod 4, i/4},\ 0 \le i < 4\mathrm{N_b}. \tag{3.3}$$

At the end of decryption, the plaintext block is extracted from the state according to

$$p_i = a_{i \bmod 4, i/4},\ 0 \le i < 4\mathrm{N_b}. \tag{3.4}$$

Similarly, the key is mapped onto a two-dimensional cipher key. The cipher key is pictured as a rectangular array with four rows similar to the state. The number of columns of the cipher key is denoted by $\mathrm{N_k}$ and is equal to the key length divided by 32. The bytes of the key are mapped onto the bytes of the cipher key in the order: $k_{0,0}$, $k_{1,0}$, $k_{2,0}$, $k_{3,0}$, $k_{0,1}$, $k_{1,1}$, $k_{2,1}$, $k_{3,1}$, $k_{4,1}$ .... If we denote the key by:

$$z_0 z_1 z_2 z_3 \ldots z_{4\cdot \mathrm{N_k}-1},$$

then

$$k_{i,j} = z_{i+4j},\ 0 \le i < 4, 0 \le j < \mathrm{N_k}. \tag{3.5}$$

The representation of the state and cipher key and the mappings plaintext–state and key–cipher key are illustrated in Fig. 3.1.

p_0	p_4	p_8	p_{12}
p_1	p_5	p_9	p_{13}
p_2	p_6	p_{10}	p_{14}
p_3	p_7	p_{11}	p_{15}

k_0	k_4	k_8	k_{12}	k_{16}	k_{20}
k_1	k_5	k_9	k_{13}	k_{17}	k_{21}
k_2	k_6	k_{10}	k_{14}	k_{18}	k_{22}
k_3	k_7	k_{11}	k_{15}	k_{19}	k_{23}

Fig. 3.1. State and cipher key layout for the case $N_b = 4$ and $N_k = 6$.

3.3 Structure of Rijndael

Rijndael is a key-iterated block cipher: it consists of the repeated application of a round transformation on the state. The number of rounds is denoted by N_r and depends on the block length and the key length.

Note that in this chapter, contrary to the definitions (2.47)–(2.50), the key addition is included in the *round* transformation. This is done in order to make the description in this chapter consistent with the description in the FIPS standard.

Following a suggestion of B. Gladman, we changed the names of some steps with respect to the description given in our original AES submission. The new names are more consistent, and are also adopted in the FIPS standard. We made some further changes, all in order to make the description more clear and complete. No changes have been made to the block cipher itself.

An encryption with Rijndael consists of an initial key addition, denoted by `AddRoundKey`, followed by $N_r - 1$ applications of the transformation `Round`, and finally one application of `FinalRound`. The initial key addition and every round take as input the `State` and a round key. The round key for round i is denoted by `ExpandedKey`[i], and `ExpandedKey`[0] denotes the input of the initial key addition. The derivation of `ExpandedKey` from the `CipherKey` is denoted by `KeyExpansion`. A high-level description of Rijndael in pseudo-C notation is shown in List. 3.1.

3.4 The Round Transformation

The round transformation is denoted `Round`, and is a sequence of four transformations, called *steps*. This is shown in List. 3.2. The final round of the cipher is slightly different. It is denoted `FinalRound` and also shown in List. 3.2. In the listings, the *transformations* (`Round`, `SubBytes`, `ShiftRows`, ...) operate on arrays to which pointers (`State`, `ExpandedKey`[i]) are provided. It is

```
Rijndael(State,CipherKey)
{
KeyExpansion(CipherKey,ExpandedKey);
AddRoundKey(State,ExpandedKey[0]);
for(i = 1; i < Nr ; i++) Round(State,ExpandedKey[i]);
FinalRound(State,ExpandedKey[Nr]);
}
```

List. 3.1. High-level algorithm for encryption with Rijndael.

easy to verify that the transformation `FinalRound` is equal to the transformation `Round`, but with the `MixColumns` step removed. The steps are specified in the following subsections, together with the design criteria we used for each step. Besides the step-specific criteria, we also applied the following two general design criteria:

1. **Invertibility.** The structure of the Rijndael round transformation requires that all steps be invertible.
2. **Simplicity.** As explained in Chap. 5, we prefer simple components over complex ones.

```
Round(State,ExpandedKey[i])
{
SubBytes(State);
ShiftRows(State);
MixColumns(State);
AddRoundKey(State,ExpandedKey[i]);
}

FinalRound(State,ExpandedKey[Nr])
{
SubBytes(State);
ShiftRows(State);
AddRoundKey(State,ExpandedKey[Nr]);
}
```

List. 3.2. The Rijndael round transformation.

3.4.1 The SubBytes Step

The `SubBytes` step is the only non-linear transformation of the cipher. `SubBytes` is a bricklayer permutation consisting of an S-box applied to the

bytes of the state. We denote the particular S-box being used in Rijndael by S_{RD}. Figure 3.2 illustrates the effect of the `SubBytes` step on the state. Figure 3.3 shows the pictograms that we will use to represent `SubBytes` and its inverse.

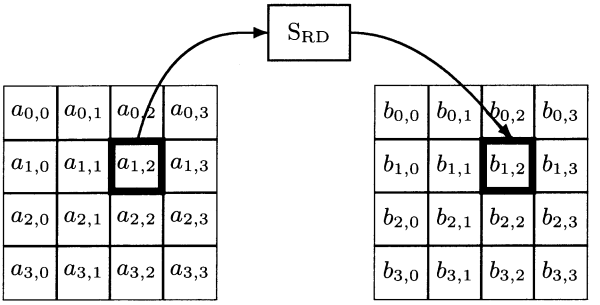

Fig. 3.2. `SubBytes` acts on the individual bytes of the state.

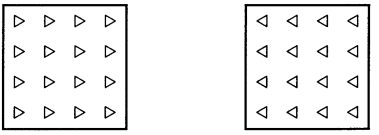

Fig. 3.3. The pictograms for `SubBytes` *(left)* and `InvSubBytes` *(right)*.

Design criteria for S_{RD}. We have applied the following design criteria for S_{RD}, appearing in order of importance:

1. **Non-linearity.**
   a) **Correlation.** The maximum input-output correlation amplitude must be as small as possible.
   b) **Difference propagation probability.** The maximum difference propagation probability must be as small as possible.
2. **Algebraic complexity.** The algebraic expression of S_{RD} in $GF(2^8)$ has to be complex.

Only one S-box is used for all byte positions. This is certainly not a necessity: `SubBytes` could as easily be defined with different S-boxes for every byte. This issue is discussed in Chap. 5. The non-linearity criteria are inspired by linear and differential cryptanalysis. Chap. 9 discusses this in depth.

Selection of S_{RD}. In [74], K. Nyberg gives several construction methods for S-boxes with good non-linearity. For invertible S-boxes operating on bytes,

the maximum correlation amplitude can be made as low as 2^{-3}, and the maximum difference propagation probability can be as low as 2^{-6}. We decided to choose — from the alternatives described in [74] — the S-box that is defined by the following function in GF(2^8):

$$g : a \rightarrow b = a^{-1}. \tag{3.6}$$

We use the polynomial representation of GF(2^8) defined in Sect. 2.1.6: the elements of GF(2^8) are considered as polynomials having a degree smaller than eight, with coefficients in the finite field GF(2). Multiplication is done modulo the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$, and the multiplicative inverse a^{-1} is defined accordingly. The value 00 is mapped onto itself. By definition, g has a very simple algebraic expression. This could allow algebraic manipulations that can be used to mount attacks such as interpolation attacks. Therefore, we built the S-box as the sequence of g and an invertible affine transformation f. This affine transformation has no impact on the non-linearity properties, but if properly chosen, allows S_{RD} to have a complex algebraic expression. We have chosen an affine transformation that has a very simple description per se, but a complicated algebraic expression if combined with the transformation g. Because this still leaves many possibilities for the choice of f, we additionally imposed the restriction that S_{RD} should have no fixed points and no opposite fixed points:

$$S_{RD}[a] \oplus a \neq \texttt{00}, \quad \forall a \tag{3.7}$$

$$S_{RD}[a] \oplus a \neq \texttt{FF}, \quad \forall a. \tag{3.8}$$

Note that we are not aware of any attacks that would exploit the existence of (opposite) fixed points.

The affine transformation f is defined by:

$$b = f(a)$$

$$\Updownarrow$$

$$\begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 1\,1\,1\,1\,1\,0\,0\,0 \\ 0\,1\,1\,1\,1\,1\,0\,0 \\ 0\,0\,1\,1\,1\,1\,1\,0 \\ 0\,0\,0\,1\,1\,1\,1\,1 \\ 1\,0\,0\,0\,1\,1\,1\,1 \\ 1\,1\,0\,0\,0\,1\,1\,1 \\ 1\,1\,1\,0\,0\,0\,1\,1 \\ 1\,1\,1\,1\,0\,0\,0\,1 \end{bmatrix} \times \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}. \tag{3.9}$$

The affine transformation f can also be described as a polynomial multiplication, followed by the XOR with a constant. This is explained in Appendix C, where also a tabular description of S_{RD} is given.

Inverse operation. The inverse operation of `SubBytes` is called `InvSubBytes`. It is a bricklayer permutation consisting of the inverse S-box $S_{RD}{}^{-1}$ applied to the bytes of the state. The inverse S-box $S_{RD}{}^{-1}$ is obtained by applying the inverse of the affine transformation (3.9) followed by taking the multiplicative inverse in GF(2^8). The inverse of (3.9) is specified by:

$$\begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} = \begin{bmatrix} 0\,1\,0\,1\,0\,0\,1\,0 \\ 0\,0\,1\,0\,1\,0\,0\,1 \\ 1\,0\,0\,1\,0\,1\,0\,0 \\ 0\,1\,0\,0\,1\,0\,1\,0 \\ 0\,0\,1\,0\,0\,1\,0\,1 \\ 1\,0\,0\,1\,0\,0\,1\,0 \\ 0\,1\,0\,0\,1\,0\,0\,1 \\ 1\,0\,1\,0\,0\,1\,0\,0 \end{bmatrix} \times \begin{bmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}. \tag{3.10}$$

Tabular descriptions of $S_{RD}{}^{-1}$ and f^{-1} are given in Appendix C.

3.4.2 The `ShiftRows` Step

The `ShiftRows` step is a byte transposition that cyclically shifts the rows of the state over different offsets. Row 0 is shifted over C_0 bytes, row 1 over C_1 bytes, row 2 over C_2 bytes and row 3 over C_3 bytes, so that the byte at position j in row i moves to position $(j - C_i)$ mod N_b. The shift offsets C_0, C_1, C_2 and C_3 depend on the value of N_b.

Design criteria for the offsets. The design criteria for the offsets are the following:

1. **Diffusion optimal.** The four offsets have to be different (see Definition 9.4.1).
2. **Other diffusion effects.** The resistance against truncated differential attacks (see Chap. 10) and saturation attacks has to be maximized.

Diffusion optimality is important in providing resistance against differential and linear cryptanalysis. The other diffusion effects are only relevant when the block length is larger than 128 bits.

Selection of the offsets. The simplicity criterion dictates that one offset is taken equal to 0. In fact, for a block length of 128 bits, the offsets have to be 0, 1, 2 and 3. The assignment of offsets to rows is arbitrary. For block lengths larger than 128 bit, there are more possibilities. Detailed studies of truncated differential attacks and saturation attacks on reduced versions of Rijndael show that not all choices are equivalent. For certain choices, the attacks can be extended with one round. Among the choices that are best with respect to saturation and truncated differential attacks, we picked the simplest ones. The different values are specified in Table 3.1. Figure 3.4 illustrates the effect

of the `ShiftRows` step on the state. Figure 3.5 shows the pictograms for `ShiftRows` and its inverse.

Table 3.1. `ShiftRows`: shift offsets for different block lengths.

N_b	C_0	C_1	C_2	C_3
4	0	1	2	3
5	0	1	2	3
6	0	1	2	3
7	0	1	2	4
8	0	1	3	4

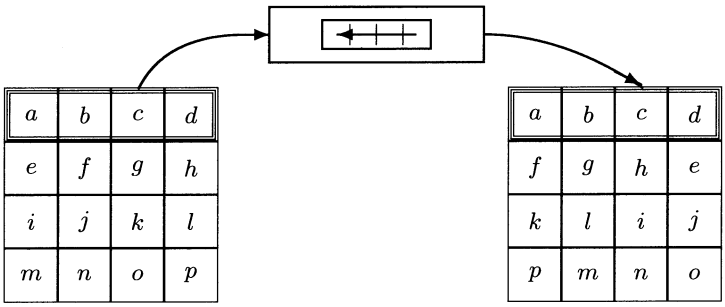

Fig. 3.4. `ShiftRows` operates on the rows of the state.

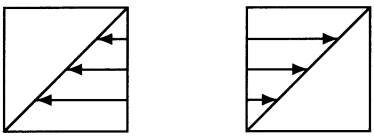

Fig. 3.5. Pictograms for `ShiftRows` *(left)* and `InvShiftRows` *(right)*.

Inverse operation. The inverse operation of `ShiftRows` is called `InvShiftRows`. It is a cyclic shift of the 3 bottom rows over $N_b - C_1$, $N_b - C_2$ and $N_b - C_3$ bytes respectively so that the byte at position j in row i moves to position $(j + C_i) \bmod N_b$.

3.4.3 The `MixColumns` Step

The `MixColumns` step is a bricklayer permutation operating on the state column by column.

Design criteria. The design criteria for the `MixColumns` step are the following:

1. **Dimensions.** The transformation is a bricklayer transformation operating on 4-byte columns.
2. **Linearity.** The transformation is preferably linear over GF(2).
3. **Diffusion.** The transformation has to have *relevant* diffusion power.
4. **Performance on 8-bit processors.** The performance of the transformation on 8-bit processors has to be high.

The criteria about linearity and diffusion are requirements imposed by the wide trail strategy (see Chap. 9). The dimensions criterion of having columns consisting of 4 bytes is to make optimal use of 32-bit architectures in look-up table implementations (see Sect. 4.2). The performance on 8-bit processors is mentioned because `MixColumns` is the only step that good performance on 8-bit processors is not trivial to obtain for.

Selection. The diffusion and performance criteria have lead us to the following choice for the definition of the D-box in `MixColumns`. The columns of the state are considered as polynomials over GF(2^8) and multiplied modulo x^4+1 with a fixed polynomial $c(x)$. The criteria about invertibility, diffusion and performance impose conditions on the coefficients of $c(x)$. The performance criterion can be satisfied if the coefficients have simple values, such as 00, 01, 02, 03, ... Multiplication with the value 00 or 01 implies no processing at all, multiplication with 02 can be implemented efficiently with a dedicated routine (see Sect. 4.1.1) and multiplication with 03 can be implemented as a multiplication with 02 plus an additional XOR operation with the operand. The diffusion criterion induces a more complicated condition on the coefficients of $c(x)$. We determined the coefficients in such a way that the branch number of `MixColumns` is five, i.e. the maximum possible for a transformation with these dimensions. Further explanation of the branch number of a function and the relation to the diffusion power can be found in Sect. 9.3.

The polynomial $c(x)$ is given by

$$c(x) = \texttt{03} \cdot x^3 + \texttt{01} \cdot x^2 + \texttt{01} \cdot x + \texttt{02}. \tag{3.11}$$

This polynomial is coprime to x^4+1 and therefore invertible. As described in Sect. 2.1.7, the modular multiplication with a fixed polynomial can be written as a matrix multiplication. Let $b(x) = c(x) \cdot a(x) \pmod{x^4+1}$. Then

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}. \tag{3.12}$$

Figure 3.6 illustrates the effect of the `MixColumns` step on the state. Figure 3.7 shows the pictograms for `MixColumns` and its inverse.

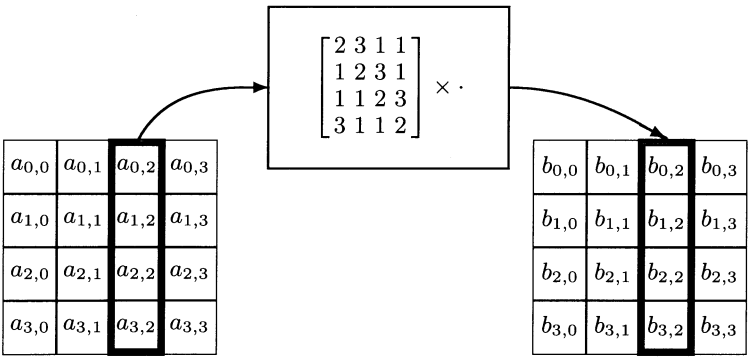

Fig. 3.6. `MixColumns` operates on the columns of the state.

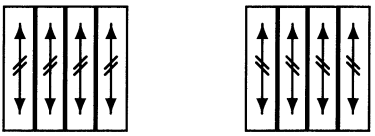

Fig. 3.7. Pictograms for `MixColumns` *(left)* and `InvMixColumns` *(right)*.

Inverse operation. The inverse operation of `MixColumns` is called `InvMixColumns`. It is similar to `MixColumns`. Every column is transformed by multiplying it with a fixed multiplication polynomial $d(x)$, defined by

$$(\texttt{03} \cdot x^3 + \texttt{01} \cdot x^2 + \texttt{01} \cdot x + \texttt{02}) \cdot d(x) \equiv \texttt{01} \pmod{x^4 + 1}. \tag{3.13}$$

It is given by

$$d(x) = \texttt{0B} \cdot x^3 + \texttt{0D} \cdot x^2 + \texttt{09} \cdot x + \texttt{0E}. \tag{3.14}$$

Written as a matrix multiplication, `InvMixColumns` transforms the columns in the following way:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} \texttt{0E 0B 0D 09} \\ \texttt{09 0E 0B 0D} \\ \texttt{0D 09 0E 0B} \\ \texttt{0B 0D 09 0E} \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}. \tag{3.15}$$

3.4.4 The Key Addition

The key addition is denoted `AddRoundKey`. In this transformation, the state is modified by combining it with a round key with the bitwise XOR operation. A round key is denoted by `ExpandedKey`$[i]$, $0 \le i \le \mathrm{N_r}$. The array of round keys `ExpandedKey` is derived from the cipher key by means of the key schedule (see Sect. 3.6). The round key length is equal to the block length. The `AddRoundKey` transformation is illustrated in Fig. 3.8. `AddRoundKey` is its own inverse. Figure 3.9 shows the pictogram for `AddRoundKey`.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

$\oplus$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

$=$

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$
$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$

Fig. 3.8. In `AddRoundKey`, the round key is added to the state with a bitwise XOR.

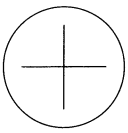

Fig. 3.9. Pictogram for `AddRoundKey`.

3.5 The Number of Rounds

The current state-of-the-art in cryptanalysis indicates that the resistance of iterative block ciphers against cryptanalytic attacks increases with the number of rounds.

We have determined the number of rounds by considering the maximum number of rounds for which shortcut attacks (see Sect. 5.5.1) have been found that are significantly more efficient than an exhaustive key search. Subsequently, we added a considerable security margin. For Rijndael with a block length and key length of 128 bits, no shortcut attacks had been found for reduced versions with more than six rounds. We added four rounds as a security margin. This is a conservative approach, because:

1. Two rounds of Rijndael provide 'full diffusion' in the following sense: every state bit depends on all state bits two rounds ago, or a change in one state bit is likely to affect half of the state bits after two rounds. Adding four rounds can be seen as adding a 'full diffusion step' at the beginning and at the end of the cipher. The high diffusion of the Rijndael round transformation is thanks to its uniform structure that operates on all state bits. For so-called Feistel ciphers, a round only operates on half of the state bits and full diffusion can at best be obtained after three rounds and in practice it typically takes four rounds or more.
2. Generally, linear cryptanalysis, differential cryptanalysis and truncated differential attacks exploit a propagation trail through n rounds in order to attack $n + 1$ or $n + 2$ rounds. This is also the case for the saturation attack (see Sect. 10.2) that uses a four-round propagation structure to

attack six rounds. In this respect, adding four rounds actually doubles the number of rounds through which a propagation trail has to be found.

For Rijndael versions with a longer key, the number of rounds was raised by one for every additional 32 bits in the cipher key. This was done for the following reasons:

1. One of the main objectives is the absence of shortcut attacks, i.e. attacks that are more efficient than an exhaustive key search. Since the workload of an exhaustive key search grows with the key length, shortcut attacks can afford to be less efficient for longer keys.
2. (Partially) known-key and related-key attacks exploit the knowledge of cipher key bits or the ability to apply different cipher keys. If the cipher key grows, the range of possibilities available to the cryptanalyst increases.

The publications on the security of Rijndael with longer keys have shown that this strategy leads to an adequate security margin [31, 36, 62]. For Rijndael versions with a higher block length, the number of rounds is raised by one for every additional 32 bits in the block length, for the following reasons:

1. For a block length above 128 bits, it takes three rounds to realize that full diffusion, i.e. the diffusion power of the round transformation, relative to the block length, diminishes with the block length.
2. The larger block length causes the range of possible patterns that can be applied at the input/output of a sequence of rounds to increase. This additional flexibility may allow the extension of attacks by one or more rounds.

We have found that extensions of attacks by a single round are even hard to realize for the maximum block length of 256 bits. Therefore, this is a conservative margin.

Table 3.2 lists the value of N_r as a function of N_b and N_k. For the AES, N_b is fixed to the value 4; $N_r = 10$ for 128-bit keys ($N_k = 4$), $N_r = 12$ for 192-bit keys ($N_k = 6$) and $N_r = 14$ for 256-bit keys ($N_k = 8$).

Table 3.2. Number of rounds (N_r) as a function of N_b (N_b = block length/32) and N_k (key length/32).

	N_b				
N_k	4	5	6	7	8
4	10	11	12	13	14
5	11	11	12	13	14
6	12	12	12	13	14
7	13	13	13	13	14
8	14	14	14	14	14

3.6 Key Schedule

The key schedule consists of two components: the key expansion and the round key selection. The key expansion specifies how `ExpandedKey` is derived from the cipher key. The total number of bits in `ExpandedKey` is equal to the block length multiplied by the number of rounds plus 1, since the cipher requires one round key for the initial key addition, and one for each of the rounds. Please note that the `ExpandedKey` is always derived from the cipher key; it should never be specified directly.

3.6.1 Design Criteria

The key expansion has been chosen according to the following criteria:

1. **Efficiency.**
   a) **Working memory.** It should be possible to execute the key schedule using a small amount of working memory.
   b) **Performance.** It should have a high performance on a wide range of processors.
2. **Symmetry elimination.** It should use round constants to eliminate symmetries.
3. **Diffusion.** It should have an efficient diffusion of cipher key differences into the expanded key,
4. **Non-linearity.** It should exhibit enough non-linearity to prohibit the full determination of differences in the expanded key from cipher key differences only.

For a more thorough treatment of the criteria underlying the design of the key schedule, we refer to Sect. 5.8.

3.6.2 Selection

In order to be efficient on 8-bit processors, a lightweight, byte-oriented expansion scheme has been adopted. The application of the non-linear S_{RD} ensures the non-linearity of the scheme, without adding much in the way of temporary storage requirements on an 8-bit processor.

During the key expansion the cipher key is expanded into an expanded key array, consisting of 4 rows and $N_b(N_r + 1)$ columns. This array is here denoted by $W[4][N_b(N_r+1)]$. The round key of the ith round, `ExpandedKey`$[i]$, is given by the columns $N_b \cdot i$ to $N_b \cdot (i + 1) - 1$ of W:

$$\begin{aligned}&\texttt{ExpandedKey}[i] = \\ &\mathsf{W}[\cdot][\mathrm{N_b} \cdot i] \parallel \mathsf{W}[\cdot][\mathrm{N_b} \cdot i + 1] \parallel \cdots \parallel \mathsf{W}[\cdot][\mathrm{N_b} \cdot (i+1) - 1], \\ &0 \leq i \leq \mathrm{N_r}.\end{aligned} \tag{3.16}$$

The key expansion function depends on the value of $\mathrm{N_k}$: there is a version for $\mathrm{N_k}$ equal to or below 6, shown in List. 3.3, and a version for $\mathrm{N_k}$ above 6, shown in List. 3.4. In both versions of the key expansion, the first $\mathrm{N_k}$ columns of W are filled with the cipher key. The following columns are defined recursively in terms of previously defined columns. The recursion uses the bytes of the previous column, the bytes of the column $\mathrm{N_k}$ positions earlier, and round constants $\mathrm{RC}[j]$.

The recursion function depends on the position of the column. If i is not a multiple of $\mathrm{N_k}$, column i is the bitwise XOR of columns $i - \mathrm{N_k}$ and column $i - 1$. Otherwise, column i is the bitwise XOR of column $i - \mathrm{N_k}$ and a non-linear function of column $i - 1$. For cipher key length values $\mathrm{N_k} > 6$, this is also the case if $i \bmod \mathrm{N_k} = 4$. The non-linear function is realized by means of the application of $\mathrm{S_{RD}}$ to the four bytes of the column, an additional cyclic rotation of the bytes within the column and the addition of a round constant (for elimination of symmetry). The round constants are independent of $\mathrm{N_k}$, and defined by a recursion rule in $\mathrm{GF}(2^8)$:

$$\mathrm{RC}[1] = x^0 \text{ (i.e. 01)} \tag{3.17}$$

$$\mathrm{RC}[2] = x \text{ (i.e. 02)} \tag{3.18}$$

$$\mathrm{RC}[j] = x \cdot \mathrm{RC}[j-1] = x^{j-1}, \quad j > 2. \tag{3.19}$$

The key expansion process and the round key selection are illustrated in Fig. 3.10.

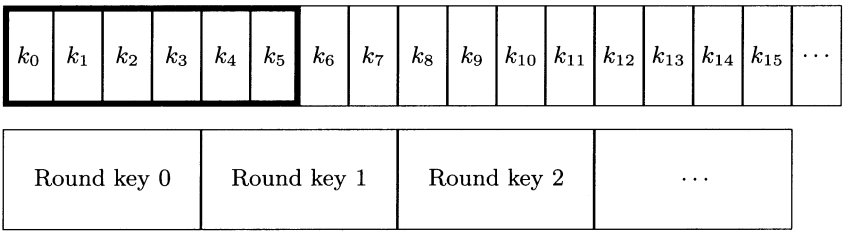

Fig. 3.10. Key expansion and round key selection for $\mathrm{N_b} = 4$ and $\mathrm{N_k} = 6$.

```
KeyExpansion(byte K[4][Nk], byte W[4][Nb(Nr + 1)])
                                                    /* for Nk ≤ 6 */
{
for(j = 0; j < Nk; j++)
     for(i = 0; i < 4; i++) W[i][j] = K[i][j];
for(j = Nk; j < Nb(Nr + 1); j++)
     {
     if (j mod Nk == 0)
          {
          W[0][j] = W[0][j − Nk] ⊕ S[W[1][j − 1]] ⊕ RC[j/Nk];
          for(i = 1; i < 4; i++)
               W[i][j] = W[i][j − Nk] ⊕ S[W[i + 1 mod 4][j − 1]];
          }
     else
          {
          for(i = 0; i < 4; i++)
               W[i][j] = W[i][j − Nk] ⊕ W[i][j − 1];
          }
     }
}
```

List. 3.3. The key expansion for $N_k \leq 6$.

3.7 Decryption

The algorithm for decryption can be found in a straightforward way by using the inverses of the steps `InvSubBytes`, `InvShiftRows`, `InvMixColumns` and `AddRoundKey`, and reversing their order. We call the resulting algorithm the *straightforward decryption algorithm.* In this algorithm, not only so the steps themselves differ from those used in encryption, but also the sequence in which the steps occur is different. For implementation reasons, it is often convenient that the only non-linear step (`SubBytes`) is the first step of the round transformation (see Chap. 4). This aspect has been anticipated in the design. The structure of Rijndael is such that it is possible to define an *equivalent algorithm for decryption* in which the sequence of steps is equal to that for encryption, with the steps replaced by their inverses and a change in the key schedule. We illustrate this in Sect. 3.7.1–3.7.3 for a reduced version of Rijndael, that consists of only one round followed by the final round. Note that this identity in *structure* differs from the identity of *components* and *structure* (cf. Sect. 5.3.5) that is found in most ciphers with the Feistel structure, but also in IDEA [56].

3.7.1 Decryption for a Two-Round Rijndael Variant

The straightforward decryption algorithm with a two-round Rijndael variant consists of the inverse of `FinalRound`, followed by the inverse of `Round`,

```
KeyExpansion(byte K[4][Nk], byte W[4][Nb(Nr + 1)])
                                                    /* for Nk > 6 */
{
for(j = 0; j < Nk; j++)
     for(i = 0; i < 4; ++) W[i][j] = K[i][j];
for(j = Nk; j < Nb(Nr + 1); j++)
     {
     if (j mod Nk == 0)
          {
          W[0][j] = W[0][j − Nk] ⊕ S[W[1][j − 1]] ⊕ RC[j/Nk];
          for(i = 1; i < 4; i++)
               W[i][j] = W[i][j − Nk] ⊕ S[W[i + 1 mod 4][j − 1]];
          }
     else if (j mod Nk == 4)
          {
          for(i = 0; i < 4; i++)
               W[i][j] = W[i][j − Nk] ⊕ S[W[i][j − 1]];
          }
     else
          {
          for(i = 0; i < 4; i++)
               W[i][j] = W[i][j − Nk] ⊕ W[i][j − 1];
          }
     }
}
```

List. 3.4. The key expansion for $N_k > 6$.

followed by a key addition. The inverse transformation of `Round` is denoted `InvRound`. The inverse of `FinalRound` is denoted `InvFinalRound`. Both transformations are described in List. 3.5. Listing 3.6 gives the straightforward decryption algorithm for the two-round Rijndael variant.

3.7.2 Algebraic Properties

In order to derive the equivalent decryption algorithm, we use two properties of the steps:

1. The order of `InvShiftRows` and `InvSubBytes` is indifferent.
2. The order of `AddRoundKey` and `InvMixColumns` can be inverted if the round key is adapted accordingly.

The first property can be explained as follows. `InvShiftRows` simply transposes the bytes and has no effect on the byte values. `InvSubBytes` operates on individual bytes, independent of their position. Therefore, the two steps commute.

```
InvRound(State,ExpandedKey[i])
{
AddRoundKey(State,ExpandedKey[i]);
InvMixColumns(State);
InvShiftRows(State);
InvSubBytes(State);
}

InvFinalRound(State,ExpandedKey[Nr])
{
AddRoundKey(State,ExpandedKey[Nr]);
InvShiftRows(State);
InvSubBytes(State);
}
```

List. 3.5. Round transformations of the straightforward decryption algorithm.

```
AddRoundKey(State,ExpandedKey[2]);
InvShiftRows(State);
InvSubBytes(State);
AddRoundKey(State,ExpandedKey[1]);
InvMixColumns(State);
InvShiftRows(State);
InvSubBytes(State);
AddRoundKey(State,ExpandedKey[0]);
```

List. 3.6. Straightforward decryption algorithm for a two-round variant.

The explanation of the second property is somewhat more sophisticated. For any linear transformation $A : \mathbf{x} \rightarrow \mathbf{y} = A(\mathbf{x})$, it holds by definition that

$$A(\mathbf{x} \oplus \mathbf{k}) = A(\mathbf{x}) \oplus A(\mathbf{k}). \tag{3.20}$$

Since `AddRoundKey` simply adds the constant `ExpandedKey`$[i]$ to its input, and `InvMixColumns` is a linear operation, the sequence of steps

```
AddRoundKey(State,ExpandedKey[i]);
InvMixColumns(State);
```

can be replaced by the following equivalent sequence of steps:

```
InvMixColumns(State);
AddRoundKey(State,EqExpandedKey[i]);
```

where `EqExpandedKey`$[i]$ is obtained by applying `InvMixColumns` to `ExpandedKey`$[i]$. This is illustrated graphically in Fig. 3.11.

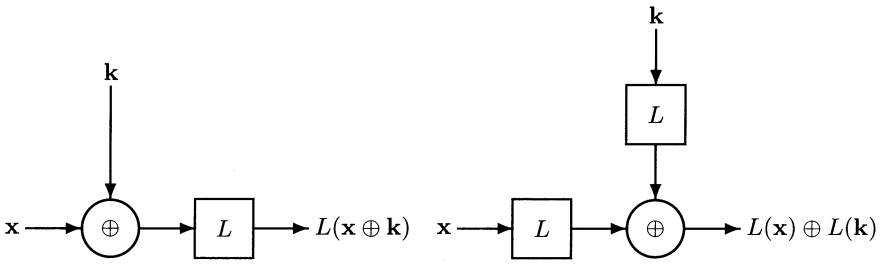

Fig. 3.11. A linear transformation L can be 'pushed through' an XOR.

3.7.3 The Equivalent Decryption Algorithm

Using the properties described above, we can transform the straightforward decryption algorithm given in List. 3.6 into the algorithm given in List. 3.7. Comparing List. 3.7 with the definition of the original round transformations `Round` and `FinalRound` (List. 3.2), we see that we can regroup the operations of List. 3.7 into an initial key addition, a `Round`-like transformation and a `FinalRound`-like transformation. The `Round`-like transformation and the `FinalRound`-like transformation have the same structure as `Round` and `FinalRound`, but they use the inverse transformations. We can generalize this regrouping to any number of rounds.

```
AddRoundKey(State,ExpandedKey[2]);
InvSubBytes(State);
InvShiftRows(State);
InvMixColumns(State);
AddRoundKey(State,EqExpandedKey[1]);
InvSubBytes(State);
InvShiftRows(State);
AddRoundKey(State,ExpandedKey[0]);
```

List. 3.7. Equivalent decryption algorithm for a two-round variant.

We define the equivalent round transformation `EqRound` and the equivalent final round transformation `EqFinalRound` to use in the equivalent decryption algorithm. The transformations are described in List. 3.8. Listing 3.9 gives the equivalent decryption algorithm. Figure 3.12 shows a graphical illustration for encryption with the two-round Rijndael variant, decryption according to the straightforward algorithm and decryption according to the equivalent algorithm. The dashed box encloses the steps that can be implemented together efficiently. In the straightforward decryption algorithm, the (inverse) steps appear in the wrong order and cannot be implemented as efficiently. By changing the order of `InvShiftRows` and `InvSubBytes`, and by pushing `MixColumns` through the XOR of `AddRoundKey`, the equivalent decryption algorithm is obtained. This structure has again the operations in a good order for efficient implementation.

```
EqRound(State,EqExpandedKey[i])
{
InvSubBytes(State);
InvShiftRows(State);
InvMixColumns(State);
AddRoundKey(State,EqExpandedKey[i]);
}

EqFinalRound(State,EqExpandedKey[0])
{
InvSubBytes(State);
InvShiftRows(State);
AddRoundKey(State,EqExpandedKey[0]);
}
```

List. 3.8. Round transformations for the equivalent decryption algorithm.

`EqKeyExpansion`, the key expansion to be used in conjunction with the equivalent decryption algorithm is defined as follows:

```
InvRijndael(State,CipherKey)
{
EqKeyExpansion(CipherKey,EqExpandedKey);
AddRoundKey(State,EqExpandedKey[Nr]);
for(i = Nr − 1; i > 0; i−−) EqRound(State,EqExpandedKey[i]);
EqFinalRound(State,EqExpandedKey[0]);
}
```

List. 3.9. Equivalent decryption algorithm.

1. Apply the key expansion `KeyExpansion`.
2. Apply `InvMixColumns` to all round keys except the first one and the last one.

Listing 3.10 gives a description in pseudo-C notation.

```
EqKeyExpansion(CipherKey,EqExpandedKey)
{
KeyExpansion(CipherKey,EqExpandedKey);
for(i = 1; i < Nr ; i++)
     InvMixColumns(EqExpandedKey[i]);
}
```

List. 3.10. Key expansion for the equivalent decryption algorithm.

3.8 Conclusions

In this chapter we have given the specification of Rijndael encryption and decryption, and the motivation for some of the design choices.

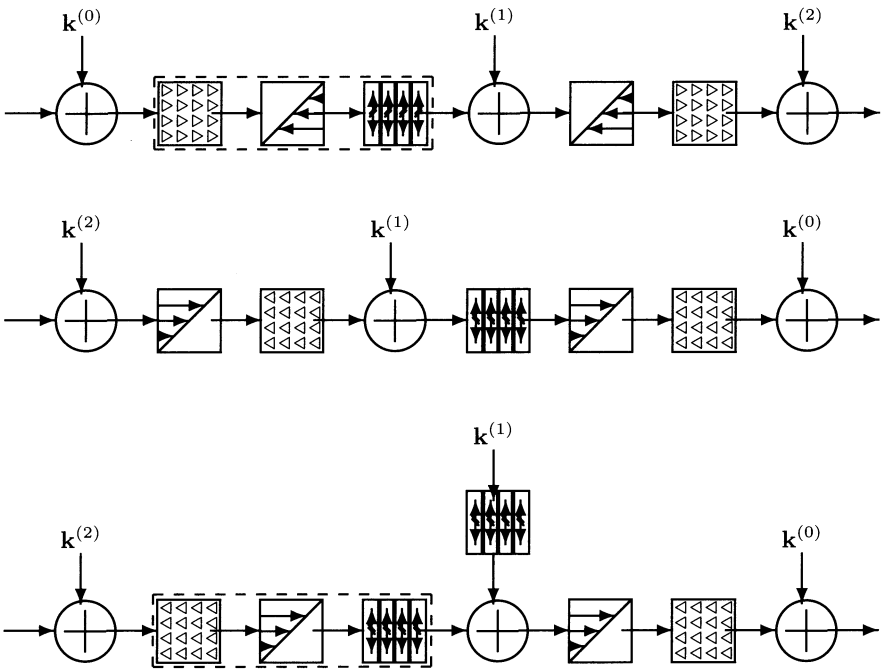

Fig. 3.12. Graphical representation of the algorithm for a two-round Rijndael variant: encryption *(top)*, decryption in the straightforward way *(middle)* and decryption in the equivalent way *(bottom)*. The *dashed box* encloses the operations that can be implemented together efficiently.

4. Implementation Aspects

In this chapter we discuss issues related to the implementation of Rijndael on different platforms. Most topics apply also to related ciphers such as Square, Anubis and Crypton that are discussed in Chap. 11. We have grouped the material of this chapter into sections that deal with the most typical issues for one specific platform each. However, several of the discussed issues are relevant to more than one platform. If you want to squeeze out the best possible performance, we advise reading the whole chapter, with a critical mindset.

4.1 8-Bit Platforms

The performance on 8-bit processors is an important issue, since most smart cards have such a processor and many cryptographic applications run on smart cards.

4.1.1 Finite Field Multiplication

In the algorithm of Rijndael there are no multiplications of two variables in GF(2^8), but only the multiplication of a variable with a constant. The latter is easier to implement than the former.

We describe here how multiplication by the value 02 can be implemented. The polynomial associated with 02 is x. Therefore, if we multiply an element b with 02, we get:

$$\begin{aligned} b \cdot x &= b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 \\ &\quad + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \pmod{m(x)} \qquad (4.1) \\ &= b_6x^7 + b_5x^6 + b_4x^5 + (b_3 \oplus b_7)x^4 \\ &\quad + (b_2 \oplus b_7)x^3 + b_1x^2 + (b_0 \oplus b_7)x + b_7. \qquad (4.2) \end{aligned}$$

The multiplication by 02 is denoted $\texttt{xtime}(x)$. `xtime` can be implemented with a shift operation and a conditional XOR operation. To prevent timing attacks, attention must be paid so that `xtime` is implemented in such a

way that it takes a fixed number of cycles, independently of the value of its argument. This can be achieved by inserting dummy instructions at the right places. However, this approach is likely to introduce weaknesses against power analysis attacks (see Sect. 10.8.2). A better approach seems to define a table M, where $M[a] = \texttt{02} \cdot a$. The routine `xtime` is then implemented as a table look-up into M.

Since all elements of $\mathrm{GF}(2^8)$ can be written as a sum of powers of `02`, multiplication by any constant value can be implemented by a repeated use of `xtime`.

Example 4.1.1. The multiplication of an input b by the constant value `15` can be implemented as follows:

$$\begin{aligned} b \cdot \texttt{15} &= b \cdot (\texttt{01} \oplus \texttt{04} \oplus \texttt{10}) \\ &= b \cdot (\texttt{01} \oplus \texttt{02}^2 \oplus \texttt{02}^4) \\ &= b \oplus \texttt{xtime}(\texttt{xtime}(b)) \oplus \texttt{xtime}(\texttt{xtime}(\texttt{xtime}(\texttt{xtime}(b)))) \\ &= b \oplus \texttt{xtime}(\texttt{xtime}(b \oplus \texttt{xtime}(\texttt{xtime}(b)))). \end{aligned}$$

4.1.2 Encryption

On an 8-bit processor, encryption with Rijndael can be programmed by simply implementing the different steps. The implementation of `ShiftRows` and `AddRoundKey` is straightforward from the description. The implementation of `SubBytes` requires a table of 256 bytes to store $\mathrm{S_{RD}}$.

`AddRoundKey`, `SubBytes` and `ShiftRows` can be efficiently combined and executed serially per state byte. Indexing overhead is minimized by explicitly coding the operation for every state byte.

`MixColumns`. In choosing the `MixColumns` polynomial, we took into account the efficiency on 8-bit processors. We illustrate in List. 4.1 how `MixColumns` can be realized in a small series of instructions. (The listing gives the algorithm to process one column.) The only finite field multiplication used in this algorithm is multiplication with the element `02`, denoted by '`xtime`'.

$t = a[0] \oplus a[1] \oplus a[2] \oplus a[3]$; `/* a is a column */`
$u = a[0]$;
$v = a[0] \oplus a[1]$; $v = \texttt{xtime}(v)$; $a[0] = a[0] \oplus v \oplus t$;
$v = a[1] \oplus a[2]$; $v = \texttt{xtime}(v)$; $a[1] = a[1] \oplus v \oplus t$;
$v = a[2] \oplus a[3]$; $v = \texttt{xtime}(v)$; $a[2] = a[2] \oplus v \oplus t$;
$v = a[3] \oplus u$; $v = \texttt{xtime}(v)$; $a[3] = a[3] \oplus v \oplus t$;

List. 4.1. Efficient implementation of `MixColumns`.

The key expansion. Implementing the key expansion in a single-shot operation is likely to occupy too much RAM in a smart card. Moreover, in most smart card applications, such as debit cards or electronic purses, the amount of data to be encrypted, decrypted or which is subject to a MAC is typically only a few blocks per session. Hence, not much performance can be gained by storing the expanded key instead of regenerating it for every application of the block cipher.

In the design of the key schedule, we took into account the restrictions imposed by a smart card. The key expansion can be implemented using a cyclic buffer of $4N_k$ bytes. When all bytes of the buffer have been used, the buffer content is updated. All operations in this key update can be implemented efficiently with byte-level operations.

4.1.3 Decryption

For implementations on 8-bit platforms, there is no benefit in following the equivalent decryption algorithm. Instead, the straightforward decryption algorithm is followed.

`InvMixColumns`. Decryption is similar in structure to encryption, but uses the `InvMixColumns` step instead of `MixColumns`. Where the `MixColumns` coefficients are limited to `01`, `02` and `03`, the coefficients of `InvMixColumns` are `09`, `0E`, `0B` and `0D`. In our 8-bit implementation, these multiplications take significantly more time and this results in a small performance degradation of the 8-bit implementation. A considerable speed-up can be obtained by using look-up tables at the cost of additional tables.

P. Barreto observes the following relation between the `MixColumns` polynomial $c(x)$ and the `InvMixColumns` polynomial $d(x)$:

$$d(x) = (\texttt{04}x^2 + \texttt{05})c(x) \pmod{x^4 + \texttt{01}}. \tag{4.3}$$

In matrix notation, this relation becomes:

$$\begin{bmatrix} \texttt{0E 0B 0D 09} \\ \texttt{09 0E 0B 0D} \\ \texttt{0D 09 0E 0B} \\ \texttt{0B 0D 09 0E} \end{bmatrix} = \begin{bmatrix} \texttt{02 03 01 01} \\ \texttt{01 02 03 01} \\ \texttt{01 01 02 03} \\ \texttt{03 01 01 02} \end{bmatrix} \times \begin{bmatrix} \texttt{05 00 04 00} \\ \texttt{00 05 00 04} \\ \texttt{04 00 05 00} \\ \texttt{00 04 00 05} \end{bmatrix}. \tag{4.4}$$

The consequence is that `InvMixColumns` can be implemented as a simple preprocessing step, followed by a `MixColumns` step. An algorithm for the preprocessing step is given in List. 4.2. If the small performance drop caused by this implementation of the preprocessing step is acceptable, no extra tables have to be defined.

```
u = xtime(xtime(a[0] ⊕ a[2])); /* a is a column */
v = xtime(xtime(a[1] ⊕ a[3]));
a[0] = a[0] ⊕ u;
a[1] = a[1] ⊕ v;
a[2] = a[2] ⊕ u;
a[3] = a[3] ⊕ v;
```

List. 4.2. Preprocessing step for implementation of the decryption.

The key expansion. The key expansion operation that generates W is defined in such a way that we can also start with the last N_k words of round key information and roll back to the original cipher key. When applications need to calculate frequently the decryption round keys 'on-the-fly', it is preferable to calculate the last N_k words of round key information once and store them for later reuse. The decryption round key calculation can then be implemented in such a way that it outputs the round keys in the order they that are needed for the decryption process. Listings 4.3 and 4.4 give a description of `InvKeyExpansion` in pseudo C notation. First note that K_i, the first input of the routine, is *not* the cipher key. Instead, K_i consists of the last N_k columns of expanded key, generated from the cipher key by means of `KeyExpansion` (see Sect. 3.6). After running `InvKeyExpansion`, W_i contains the decryption round keys in the order they are used for decryption, i.e. columns with lower indices are used first. Secondly, note that this is the key expansion for use in conjunction with the straightforward decryption algorithm. If the equivalent decryption algorithm is implemented, all but two of the round keys have additionally to be transformed by `InvMixColumns` (see Sect. 3.7.3).

4.2 32-Bit Platforms

The different steps of the round transformation can be combined in a single set of look-up tables, allowing for very fast implementations on processors with word lengths 32 or greater. In this section, we explain how this can be done.

```
InvKeyExpansion(byte K_i[4][N_k], byte W_i[4][N_b(N_r + 1)])
                                                    /* for N_k ≤ 6 */
{
for(j = 0; j < N_k; j++)
  for(i = 0; i < 4; i++) W_i[i][j] = K_i[i][j];
for(j = N_k; j < N_b(N_r + 1); j++)
  {
  if (j mod N_k == 0)
    {
    W_i[0][j] = W_i[0][j − N_k] ⊕ S[W_i[1][j − 1] ⊕ W_i[1][j − 2]] ⊕ RC[N_r + 1 − j/N_k];
    for(i = 1; i < 4; i++)
      W_i[i][j] = W_i[i][j − N_k] ⊕ S[W_i[i + 1 mod 4][j − 1] ⊕ W_i[i + 1 mod 4][j − 2]];
    }
  else
    {
    for(i = 0; i < 4; i++)
      W_i[i][j] = W_i[i][j − N_k] ⊕ W_i[i][j − N_k − 1];
    }
  }
  }
```

List. 4.3. Algorithm for the inverse key expansion for $N_k \leq 6$.

```
InvKeyExpansion(byte K_i[4][N_k], byte W_i[4][N_b(N_r + 1)])
                                                    /* for N_k > 6 */
{
for(j = 0; j < N_k; j++)
  for(i = 0; i < 4; i++) W_i[i][j] = K_i[i][j];
for(j = N_k; j < N_b(N_r + 1); j++)
  {
  if (j mod N_k == 0)
    {
    W_i[0][j] = W_i[0][j − N_k] ⊕ S[W_i[1][j − 1] ⊕ W_i[1][j − 2]] ⊕ RC[N_r + 1 − j/N_k];
    for(i = 1; i < 4; i++)
      W_i[i][j] = W_i[i][j − N_k] ⊕ S[W_i[i + 1 mod 4][j − 1] ⊕ W_i[i + 1 mod 4][j − 2]];
    }
  else if (j mod N_k == 4)
    {
    for(i = 0; i < 4; i++)
      W_i[i][j] = W_i[i][j − N_k] ⊕ S[W_i[i][j − N_k − 1]];
    }
  else
    {
    for(i = 0; i < 4; i++)
      W_i[i][j] = W_i[i][j − N_k] ⊕ W_i[i][j − N_k − 1];
    }
  }
  }
```

List. 4.4. Algorithm for the inverse key expansion for $N_k > 6$.

Let the input of the round transformation be denoted by **a**, and the output of `SubBytes` by **b**:

$$b_{i,j} = \mathrm{S_{RD}}[a_{i,j}], \quad 0 \le i < 4; 0 \le j < \mathrm{N_b}. \tag{4.5}$$

Let the output of `ShiftRows` be denoted by **c** and the output of `MixColumns` by **d**:

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j+C_0} \\ b_{1,j+C_1} \\ b_{2,j+C_2} \\ b_{3,j+C_3} \end{bmatrix}, \quad 0 \le j < \mathrm{N_b} \tag{4.6}$$

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} \mathtt{02\ 03\ 01\ 01} \\ \mathtt{01\ 02\ 03\ 01} \\ \mathtt{01\ 01\ 02\ 03} \\ \mathtt{03\ 01\ 01\ 02} \end{bmatrix} \cdot \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}, \quad 0 \le j < \mathrm{N_b}. \tag{4.7}$$

The addition in the indices of (4.6) must be done in modulo $\mathrm{N_b}$. Equations (4.5)–(4.7) can be combined into:

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} \mathtt{02\ 03\ 01\ 01} \\ \mathtt{01\ 02\ 03\ 01} \\ \mathtt{01\ 01\ 02\ 03} \\ \mathtt{03\ 01\ 01\ 02} \end{bmatrix} \cdot \begin{bmatrix} \mathrm{S_{RD}}[a_{0,j+C_0}] \\ \mathrm{S_{RD}}[a_{1,j+C_1}] \\ \mathrm{S_{RD}}[a_{2,j+C_2}] \\ \mathrm{S_{RD}}[a_{3,j+C_3}] \end{bmatrix}, \quad 0 \le j < \mathrm{N_b}. \tag{4.8}$$

The matrix multiplication can be interpreted as a linear combination of four column vectors:

$$\begin{aligned} \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = {} & \begin{bmatrix} \mathtt{02} \\ \mathtt{01} \\ \mathtt{01} \\ \mathtt{03} \end{bmatrix} \mathrm{S_{RD}}[a_{0,j+C_0}] \oplus \begin{bmatrix} \mathtt{03} \\ \mathtt{02} \\ \mathtt{01} \\ \mathtt{01} \end{bmatrix} \mathrm{S_{RD}}[a_{1,j+C_1}] \oplus \\ & \begin{bmatrix} \mathtt{01} \\ \mathtt{03} \\ \mathtt{02} \\ \mathtt{01} \end{bmatrix} \mathrm{S_{RD}}[a_{2,j+C_2}] \oplus \begin{bmatrix} \mathtt{01} \\ \mathtt{01} \\ \mathtt{03} \\ \mathtt{02} \end{bmatrix} \mathrm{S_{RD}}[a_{3,j+C_3}], \quad 0 \le j < \mathrm{N_b}. \end{aligned} \tag{4.9}$$

We define now the four T-tables: T_0, T_1, T_2 and T_3:

$$T_0[a] = \begin{bmatrix} \mathtt{02} \cdot \mathrm{S_{RD}}[a] \\ \mathtt{01} \cdot \mathrm{S_{RD}}[a] \\ \mathtt{01} \cdot \mathrm{S_{RD}}[a] \\ \mathtt{03} \cdot \mathrm{S_{RD}}[a] \end{bmatrix}, \quad T_1[a] = \begin{bmatrix} \mathtt{03} \cdot \mathrm{S_{RD}}[a] \\ \mathtt{02} \cdot \mathrm{S_{RD}}[a] \\ \mathtt{01} \cdot \mathrm{S_{RD}}[a] \\ \mathtt{01} \cdot \mathrm{S_{RD}}[a] \end{bmatrix}, \tag{4.10}$$

$$T_2[a] = \begin{bmatrix} \mathtt{01} \cdot \mathrm{S_{RD}}[a] \\ \mathtt{03} \cdot \mathrm{S_{RD}}[a] \\ \mathtt{02} \cdot \mathrm{S_{RD}}[a] \\ \mathtt{01} \cdot \mathrm{S_{RD}}[a] \end{bmatrix}, \quad T_3[a] = \begin{bmatrix} \mathtt{01} \cdot \mathrm{S_{RD}}[a] \\ \mathtt{01} \cdot \mathrm{S_{RD}}[a] \\ \mathtt{03} \cdot \mathrm{S_{RD}}[a] \\ \mathtt{02} \cdot \mathrm{S_{RD}}[a] \end{bmatrix}. \tag{4.11}$$

These tables have each 256 4-byte word entries and require 4 kB of storage space. Using these tables, (4.9) translates into:

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = T_0[a_{0,j+C_0}] \oplus T_1[a_{1,j+C_1}] \oplus T_2[a_{2,j+C_2}] \oplus T_3[a_{3,j+C_3}],$$

$$0 \leq j < \mathrm{N_b}. \tag{4.12}$$

Taking into account that `AddRoundKey` can be implemented with an additional 32-bit XOR operation per column, we get a look-up table implementation with 4 kB of tables that takes only four table look-ups and four XOR operations per column per round.

Furthermore, the entries $T_0[a], T_1[a], T_2[a]$ and $T_3[a]$ are rotated versions of one another, for all values a. Consequently, at the cost of three additional rotations per round per column, the look-up table implementation can be realized with only one table, i.e. with a total table size of 1 kB. The size of the encryption routine (relevant in applets) can be kept small by including a program to generate the tables instead of the tables themselves.

In the final round, there is no `MixColumns` operation. This boils down to the fact that $\mathrm{S_{RD}}$ must be used instead of the T-tables. The need for additional tables can be suppressed by extracting the $\mathrm{S_{RD}}$-table from a T-table by masking while executing the final round.

Most operations in the key expansion are 32-bit XOR operations. The additional transformations are the application $\mathrm{S_{RD}}$ and a cyclic shift over 8 bits. This can be implemented very efficiently.

Decryption can be described in terms of the transformations `EqRound` and `EqFinalRound` used in the equivalent decryption algorithm. These can be implemented with look-up tables in exactly the same way as the transformations `Round` and `FinalRound`. There is no performance degradation compared to encryption. The look-up tables for the decryption are however different. The key expansion to be used in conjunction with the equivalent decryption algorithm is slower, because after the key expansion all but two of the round keys are subject to `InvMixColumns` (cf. Sect. 3.7).

4.3 Dedicated Hardware

Rijndael is suited to be implemented in dedicated hardware. There are several trade-offs between chip area and speed possible. Because the implementation in software on general-purpose processors is already very fast, the need for hardware implementations will very probably be limited to two specific cases:

1. Extremely high-speed chip with no area restrictions: the T-tables can be hardwired and the XOR operations can be conducted in parallel.
2. Compact coprocessor on a smart card to speed up Rijndael execution: for this platform typically S_{RD} and the `xtime` (or the complete `MixColumns`) operation can be hard-wired.

In dedicated hardware, `xtime` can be implemented with the combination of a hard-wired bit transposition and four XOR gates. The `SubBytes` step is the most critical part for a hardware implementation, for two reasons:

1. In order to achieve the highest performance, S_{RD} needs to be instantiated 16 times (disregarding the key schedule). A straightforward implementation with 16 256-byte tables is likely to dominate the chip area requirements or the consumption of logic blocks.
2. Since Rijndael encryption and decryption use different transformations, a circuit that implements Rijndael encryption does not automatically support decryption.

However, when building dedicated hardware for supporting both encryption and decryption, we can limit the required chip area by using parts of the circuit for both transformations. In the following, we explain how S_{RD} and ${S_{RD}}^{-1}$ can be implemented efficiently.

4.3.1 Decomposition of S_{RD}

The Rijndael S-box S_{RD} is constructed from two transformations:

$$S_{RD}[a] = f(g(a)) \ , \tag{4.13}$$

where $g(a)$ is the transformation

$$a \rightarrow a^{-1} \text{ in } \mathrm{GF}(2^8), \tag{4.14}$$

and $f(a)$ is an affine transformation. The transformation $g(a)$ is a self-inverse and hence

$${S_{RD}}^{-1}[a] = g^{-1}(f^{-1}(a)) = g(f^{-1}(a)) \ . \tag{4.15}$$

Therefore, when we want both S_{RD} and ${S_{RD}}^{-1}$, we need to implement only g, f and f^{-1}. Since both f and f^{-1} can be implemented with a limited number of XOR gates, the extra hardware can be reduced significantly compared to having to hardwire both S_{RD} and ${S_{RD}}^{-1}$.

The affine transformations f and f^{-1} are defined in Sect. 3.4.1. For ease of reference, we give a tabular description of the functions f, f^{-1} and g in Appendix C.

4.3.2 Efficient Inversion in GF(2^8)

The problem of designing efficient circuits for inversion in finite fields has been studied extensively before; e.g. by C. Paar and M. Rosner in [78]. We summarize here a possible approach.

Every element of GF(2^8) can be mapped by a linear transformation to an element of GF$(2^4)^2$, i.e. a polynomial of degree one with coefficients in GF(2^4). In order to define multiplication in GF$(2^4)^2$, we need a polynomial of degree two that is irreducible over GF(2^4). There exist irreducible polynomials of the form

$$P(x) = x^2 + x + A. \tag{4.16}$$

Here 'A' is a constant element of GF(2^4) that can be chosen to optimize the hardware, as long as $P(x)$ stays irreducible. The inverse of an arbitrary element $(bx + c)$ is then given by the polynomial $(px + q)$ iff

$$1 = (bx + c) \cdot (px + q) \bmod P(x) \tag{4.17}$$

$$= (cp \oplus bq \oplus bp)x + (cq \oplus bpA). \tag{4.18}$$

This gives a set of linear equations in p and q, with the following solution:

$$\begin{cases} p = b(Ab^2 \oplus bc \oplus c^2)^{-1} \\ q = (c \oplus b)(Ab^2 \oplus bc \oplus c^2)^{-1}. \end{cases} \tag{4.19}$$

The problem of generating an inverse in GF(2^8) has been translated into the calculation of an inverse and some operations in GF(2^4). The calculation of an inverse in GF(2^4) can be done with a small table.

4.4 Multiprocessor Platforms

There is considerable parallelism in the round transformation. All four steps of the round act in a parallel way on bytes, rows or columns of the state. In the look-up table implementation, all table look-ups can in principle be done in parallel. The XOR operations can be done mostly in parallel as well.

The key expansion is clearly of a more sequential nature: the value of $\mathsf{W}[i-1]$ is needed for the computation of $\mathsf{W}[i]$. However, in most applications where speed is critical, the key expansion has to be done only once for a large number of cipher executions. In applications where the cipher key changes often (in extremis, once per application of the block cipher), the key expansion and the cipher rounds can be done in parallel.

A study by C. Clapp [17] demonstrates that the performance of Rijndael on parallel processors is not constrained by the critical path length. Instead, the limiting factor for Rijndael implementations is the number of memory references that can be done per cycle.

4.5 Performance Figures

We conclude this chapter with the performance results that other people have obtained. Performance figures for a popular cryptographic primitive have a tendency to be outdated after a very small time period. The figures given here represent only a snap shot, taken at the time of writing of this book (near the end of 2001):

8-bit processor: G. Keating reports a performance of 54 kbit/s on a Motorola 6805 processor, with a 4 MHz clock [48]. This timing includes a new key setup with every encryption.

32-bit processor. H. Lipmaa reports a performance of 426 Mbit/s for manually optimized assembly implementation, running on an 800 MHz Pentium III [57].

FPGA. The fastest implementation being reported for feedback modes is by K. Gaj and P. Chodowiec. It runs at 414 Mbit/s on a Xilinx Virtex XCV 1000 [34]. For non-feedback modes, A. Elbirt et al. report a performance of 1938 Mbit/s on the same FPGA [29].

ASIC. H. Kuo and I. Verbauwhede report a throughput of 6.1 Gbit/s, using 0.18 μm standard cell technology [55] for an implementation without pipelining. Their design uses 19 000 gates. B. Weeks et al. report a throughput of 5 Gbit/s [91] for a fully pipelined version. They use a 0.5 μm standard cell library that is not available outside NSA.

A note about pipelining. In hardware implementations, pipelining often results in a significant performance increase. However, pipelining is only possible in non-feedback modes (e.g. ECB and counter modes), and therefore it is not always applicable.

4.6 Conclusions

In this chapter we have shown how Rijndael can be efficiently implemented in dedicated hardware and in software on a wide variety of processors.

5. Design Philosophy

In this chapter we motivate the choices we have made in the process of designing Rijndael and its predecessors. We start with discussing the criteria that are widely considered important for block ciphers such as security and efficiency. After that, we introduce the criterion of simplicity that plays such an important role in our design approach. We explain what we mean by it and why it is so important. A very effective way to keep things simple is by the introduction of symmetry. After discussing different ways of introducing symmetry, we motivate the choice of operations in Rijndael and its predecessors and our approach to security. This is followed by a discussion of what we think it takes to design a block cipher that satisfies our criteria. We conclude this chapter with a discussion on the generation and usage of round keys.

5.1 Generic Criteria in Cipher Design

In this section we describe a number of design criteria that are adopted by most cryptographers.

5.1.1 Security

The most important criterion for a block cipher is *security*, meaning the absence of cryptanalytic attacks that exploit its internal structure. Among other things, this implies the absence of attacks that have a workload smaller than that of an exhaustive search of the key. In the context of security one refers often to the *security margin* of a cipher. If, for a cipher with n rounds, there exists a cryptanalytic attack against a reduced-round version with $n-k$ rounds, the cipher has an absolute security margin of k rounds or a relative security margin of k/n. As advances in cryptanalysis of a cipher tend to enable the breaking of more and more rounds over time, the security margin indicates the resistance of the cipher against improvements of known types of cryptanalysis. However, it says nothing about the likelihood of these advances in cryptanalysis or about the resistance of the cipher against unknown attacks.

5.1.2 Efficiency

The complementary criterion is that of *efficiency*. Efficiency refers to the amount of resources required to perform an encryption or decryption. In dedicated hardware implementations, encryption and decryption speed and the required chip area are relevant. In software implementations, the encryption/decryption speed and the required amount of working memory and program-storage memory storage are relevant.

5.1.3 Key Agility

When quoting the speed of a cipher, one often makes the silent assumption that a large amount of data is encrypted with the same key. In that case the key schedule can be neglected. However, if a cipher key is used to secure messages consisting of a few blocks only, the amount of cycles taken by the computation of the key schedule becomes important. The ability to efficiently change keys is called *key agility*.

5.1.4 Versatility

Differences in processor word length and instruction sets may cause the efficiency of a cipher to be very dependent on the processor type. As the AES will be implemented on smart cards, palmtops, desktop PCs, workstations, routers, set-top boxes, hardware security modules and probably some other types of devices, we have attempted to design a cipher that is efficient on the widest range of processors possible. Although just a qualifier for efficiency, we call this requirement *versatility*.

5.1.5 Discussion

The criteria of security and efficiency are applied by all cipher designers. There are cases in which efficiency is sacrificed to obtain a higher security margin. The challenge is to come up with a cipher design that offers a reasonable security margin while optimizing efficiency.

The criteria of key agility and versatility are less universal. In some cases these criteria are irrelevant because the cipher is meant for a particular application and will be implemented on a specific platform. For the AES — the successor of the ubiquitous DES — we expected key agility and versatility to be major issues. Still, a large part of the ciphers submitted to the AES focus on efficiency of bulk data encryption on 32-bit processors without much attention to 8-bit processors, multiprocessors or dedicated hardware, or an efficient key schedule.

5.2 Simplicity

A notion that characterizes our design philosophy is *simplicity*. The design process can be broken down into a number of decisions and choices. In each of these decisions and choices, the simplicity criterion has played an important role.

We distinguish *simplicity of specification* and *simplicity of analysis*. A specification is simple if it makes use of a limited number of operations and if the operations by themselves can be easily explained. An obvious advantage of a simple specification is that it facilitates a correct implementation. Another advantage is that a cipher with a simple specification seems a more interesting object to study than a cipher with a complex specification. Moreover, the simplicity of the specification may lead people to believe that it is easier to find a successful attack. In other words, the simplicity of a cipher contributes to the appeal it has for cryptanalysts, and in the absence of successful cryptanalysis, to its cryptographic credibility.

Simplicity of analysis corresponds to the ability to demonstrate and understand in what way the cipher offers protection against known types of cryptanalysis. In this way, resistance against known attacks can be covered in the design phase, thereby providing a certain level of cryptographic credibility from the start. This contributes again to the appeal to cryptanalysts: successful cryptanalysis of a cipher with some credibility gives more prestige than cryptanalysis of an insignificant cipher.

Simplicity of specification does not necessarily imply simplicity of analysis. It is relatively easy to come up with a cipher with a very simple description for which the analysis with respect to known attacks is very hard.

On top of the advantages cited above, we use the criterion of simplicity to obtain a good trade-off between security on the one hand and efficiency and versatility on the other hand. This is explained in the Sect. 5.3.

Simplicity can be achieved in a number of ways. In the design of Rijndael and its predecessors, we have mostly realized it through the adoption of symmetry and our choice of operations.

5.3 Symmetry

A very powerful tool for introducing simplicity is symmetry. Symmetry can be applied in several ways. We distinguish symmetry across the rounds, symmetry within the round transformation and symmetry in the steps.

5.3.1 Symmetry Across the Rounds

We design a cipher as the repeated iteration of the same keyed round transformation. This approach has the advantage that in specifications only one round transformation needs to be specified, and in software implementations only one round has to be programmed. Moreover, it allows dedicated hardware implementations that only contains a circuit for the round transformation and the key schedule. In Rijndael, the last round is different from the other ones in order to make the algorithms for decryption and encryption have the same structure (see Chap. 4).

One may wonder whether this symmetry cannot be exploited in cryptanalysis. As a matter of fact, the so-called *slide attacks* as described by A. Biryukov and D. Wagner in [12] exploit this kind of symmetry. However, for slide attacks to work, also the key schedule must exhibit a large degree of symmetry. Hence, protection against known slide attacks can already be achieved with a very simple key schedule, e.g. consisting merely of the XOR of well-chosen constants with the cipher key.

5.3.2 Symmetry Within the Round Transformation

Symmetry within the round transformation implies that it treats all bits of the state in a similar way. In the Feistel round structure, as adopted in the DES (see Chap. 6) this is clearly not the case since the two halves of the state are treated quite differently.

A consequence of our design strategy (see Chap. 9) is that the round transformation consists of a sequence of steps, each with its own particular function. For each of these steps, the symmetry requirement translates easily into some concrete restrictions:

1. **Non-linear step.** A bricklayer transformation consisting of non-linear S-boxes operating independently on bundles. The symmetry requirement translates easily in the requirement that the same S-box is used for all bundle positions.
2. **Mixing step.** A bricklayer transformation consisting of linear D-boxes operating independently on columns. The symmetry requirement translates in the requirement that the same D-box is used for all column positions. Additionally, alignment between bundles and columns may be imposed: all bits in the same bundle are also in the same column.
3. **Transposition step.** The transposition step consist of the mere transposition of bundles. Alignment with the non-linear step may be imposed: the transposition step is a bundle transposition rather than a bit transposition.

These symmetry requirements offer a framework in which only the size of the bundles and the columns, the S-box and the D-box, and the bundle transposition need to be specified to fully define the round transformation.

Having a large degree of symmetry in the round transformation may lead to cryptographic weaknesses. An example of such a weakness is the complementation property of the DES [41]. If in Rijndael the key application is not taken into account, there exist a number of byte transpositions π that commute with the round transformation. In other words, we have $\pi \circ \rho = \rho \circ \pi$. If all round keys are 0, this is also valid for the complete cipher. The same property holds if each individual round key is composed of bytes that all have the same value. These symmetry properties can however be eliminated by using even a simple key schedule.

Imposing alignment results in the cipher actually operating on bundles rather than bits. As a matter of fact, this property is exploited in the most powerful attacks against reduced-round versions of Rijndael and its relatives to date, the so-called saturation attacks, which are described in Sect. 10.2. Fortunately, saturation attacks seem only to be feasible up to six or seven rounds, and have an extremely high computation cost beyond that point. Still, the existence of the saturation attack is one of the main motivations behind the number of rounds in Rijndael and its relatives.

Note. Instead of translating symmetry into the requirement for bundle alignment, as is done for Rijndael and its relatives, one may choose for the opposite: non-alignment. In this case the transposition step moves bits belonging to the same bundle to bits in different bundles. This is the approach followed for the bit-slice ciphers 3-Way[20], BaseKing [23] and Noekeon [24]. Because of the small size of their S-box, these ciphers are very compact in dedicated hardware. In software they are in general slower than Rijndael and its relatives. Perhaps the best known bit-slice cipher is Serpent, which is the AES candidate submitted by E. Biham et al. [3]. The designers of Serpent have not followed the same simplicity strategy: it has 8 different S-boxes giving rise to 8 different round transformations, and the mixing step has a substantial amount of asymmetry. These factors make it harder to prove bounds for Serpent than for Rijndael and its relatives and the more symmetric bit-slice ciphers mentioned above.

5.3.3 Symmetry in the D-box

Specifying a D-box with the same size as the one used in the mixing step of Rijndael can in general be done with a binary 32×32 matrix, taking 128 bytes. By interpreting bytes as elements in a finite field, and restricting ourselves to a matrix multiplication over $\mathrm{GF}(2^8)$, the D-box can be specified with 16 byte values. We have imposed that the matrix is a circulant matrix, imposing on the matrix elements $a_{i,j} = a_{0,i-j \bmod n}$ for all i, j. This reduces the number

of bytes required to specify the D-box to 4. Other types of symmetry may be imposed. For example, the mixing step of Anubis [4] makes use of a matrix where the matrix elements satisfy $a_{i,j} = a_{0,i \oplus j}$ for all i, j.

5.3.4 Symmetry and Simplicity in the S-box

For a discussion on the design underlying the S-box and its predecessors used in Rijndael, we refer to Sect. 3.4.1.

5.3.5 Symmetry between Encryption and Decryption

In general it is an advantage for a block cipher that encryption and decryption can be performed with the same software program or make use of the same hardware circuit. In Feistel ciphers such as the DES (see Chap. 6) this can easily be achieved by omitting the switching of the two halves in the last round. It suffices to execute the rounds taking the round keys in reverse order.

In ciphers that have a round structure such as Rijndael, this is less trivial to achieve. For Rijndael and its predecessors, encryption and decryption are different algorithms. Still, in Sect. 3.7 we derive an equivalent decryption algorithm that has the same structure as the encryption algorithm. By selecting the steps of the round transformation in a careful way, it is possible to design a Rijndael-like block cipher that has encryption and decryption algorithms that are identical with the exception of the key schedule. This is illustrated by the design of Anubis (see Sect. 11.5.3).

5.3.6 Additional Benefits of Symmetry

In this section we describe a number of benefits that result from the application of symmetry.

Parallelism. A consequence of the symmetry in the different steps is that they all exhibit a large degree of parallelism. The order in which the S-boxes of the non-linear step are computed is unimportant, and so they may be all computed in parallel. The same argument is valid for the different D-boxes of the mixing step and for the key application. In dedicated hardware implementations of the round transformation, this gives rise to a critical path consisting only of the S-box, the D-box and the XOR of the key addition. In software implementations, this gives the programmer a lot of flexibility in the order in which the computations are executed. Moreover, it allows the efficient exploitation of parallelism supported by multiprocessors, as C. Clapp demonstrated in [17].

Flexibility in the order of steps. The linearity of three of the four steps and the symmetry of the non-linear step allow even more freedom in the order in which the steps of the round transformation are executed. The transposition step and the mixing step both commute with the key addition under the condition that the key value is adapted to this changed order. On the other hand, thanks to the fact that the non-linear step has the same effect on all bundles, it commutes with the transposition step. This gives software implementers even more freedom and in fact allows construction of an equivalent algorithm for decryption that has the same structure as the algorithm for encryption (see Sect. 3.7).

Variable block length. Rijndael shares with the AES candidate RC6 [82] the property that it supports different block lengths. In RC6, the state consists of four 32-bit words and these words appear as arguments in multiplication modulo 2^{32}, XOR and cyclic shift. By adopting another word length, the block length can be varied in steps of 4 bits. For example, adopting a word length of 40 leads to a block length of 160 bits.

The symmetry in the steps of Rijndael similarly facilitates the definition of the round transformation for multiple block lengths. The non-linear step only requires the block length to be a multiple of the bundle size. The mixing step requires the block length to be a multiple of the column size. The key addition does not impose any condition at all. The only step that must be specified explicitly for each block length supported is the byte transposition.

Changing the block length in RC6 may have a dramatic impact on the efficiency of implementations. For example, implementing 40-bit multiplications and cyclic shifts on a 32-bit processor is not trivial. Changing the block length is easy in the specifications, but costly in implementations. In Rijndael, the basic operations and the components of the state, the bytes and columns keep their length if the block length changes. This gives the Rijndael round transformation the unique property that the block length can be varied without affecting its computational cost per byte, on any platform.

5.4 Choice of Operations

In the specification of Rijndael and its predecessors, we have limited ourselves to relatively simple operations such as XOR and multiplication with constants in GF(2^8). The S-box makes use of the multiplicative inverse in GF(2^8) and an affine transformation.

With this limitation we have excluded a number of simple and efficient operations that are widely used as components in block ciphers and that appear to have excellent non-linearity and/or diffusion properties. The first class are arithmetic operations such as addition, subtraction and multiplication, most often performed in modulo a number of the form 2^n. The second

class are cyclic shifts over an offset that depends on state or key bits. We explain our objections against these operations in the following subsections.

5.4.1 Arithmetic Operations

Addition, subtraction and multiplication seem to be simple operations to describe. Moreover, Boolean transformations based on multiplication seem to perform very well with respect to most common non-linearity and diffusion criteria. Most processors support arithmetic instructions that execute in as few cycles as a simple bitwise XOR.

Unfortunately, if the word length of the processor does not match the length of the operands, either it becomes hard to take full advantage of the processor power due to carry propagation, or limitations in the processing power become apparent. For example, implementing a 32-bit multiplication modulo 2^{32} on an 8-bit processor smart card requires about 10 multiply instructions and 16 addition instructions.

In dedicated hardware, the number of gates required for addition (or subtraction) is about three times that of a bitwise XOR, and due to the carry propagation the gate delay is much larger and even depends on the word length. Implementing multiplication in dedicated hardware appears to give rise to circuits with a large amount of gates and a large gate delay.

Another cost that appears is the protection against power analysis attacks. The carry propagation complicates the implementation of certain protection measures against differential power analysis (DPA) that are trivial for XOR, such as balancing (cf. Sect. 10.8.2).

If arithmetic operations are used that operate on numbers that are represented by more than a single byte, one needs to define in what order these bytes must be interpreted as an integer. In processors there are two architectures: *big endian* and *little endian* [87]. Depending on how the order is defined in the cipher specification, one of the two architectures is typically favoured. By not using arithmetic operations, an *endian neutral* cipher can be obtained.

5.4.2 Data-Dependent Shifts

Data-dependent shift operations seem to be simple operations to describe. Moreover, Boolean transformations based on data-dependent shifts seem to perform well with respect to most common non-linearity and diffusion criteria. Many processors support data-dependent (cyclic) shift instructions that execute in a small fixed number of cycles. Unfortunately, if the word length of the processor does not match the length of the operand that is shifted, it takes several instructions to realize the shift operation.

Protection against implementation attacks (see Sect. 10.8) may be very cumbersome on certain platforms. For example, on a typical smart-card processor the only shift instructions available are those which shift the content of an 8-bit register over 1 bit. A straightforward implementation of a data-dependent shift would execute in a variable number of cycles, depending on the value of the offset. Providing protection against timing attacks can be achieved by inserting dummy instructions, resulting in a constant number of cycles given by the worst-case offset value. Protecting against DPA on top of that seems a non-trivial exercise and may result in a multiplication of the number of cycles by at least a factor of two.

5.5 Approach to Security

5.5.1 Security Goals

In this section, we present the goals we have set for the security of Rijndael. We introduce two security criteria in order to define the meaning of a successful cryptanalytic attack. Note that we cannot prove that Rijndael satisfies these criteria.

In order to formulate our goals, some security-related concepts need to be defined. A block cipher of block length n_b has 2^{n_b} possible inputs. If the key length is n_k, it defines a set of 2^{n_k} permutations. For a block length of n_b, the number of possible permutations is $2^{n_\mathrm{b}}!$. Hence the number of all possible block ciphers of dimensions n_b and n_k is

$$2^{n_\mathrm{b}}!^{2^{n_\mathrm{k}}} \quad . \tag{5.1}$$

For practical values of the dimensions (e.g. n_b and n_k above 40), the subset of block ciphers with exploitable weaknesses form a negligible minority in this set. We define two security criteria *K-secure* and *hermetic* as criteria that are satisfied by the majority of block ciphers for the given dimensions.

Definition 5.5.1. *A block cipher is K-secure if all possible attack strategies for it have the same expected work factor and storage requirements as for the majority of possible block ciphers with the same dimensions. This must be the case for all possible modes of access for the adversary (known/chosen/adaptively chosen plaintext/ciphertext, known/chosen/adaptively chosen key relations...) and for any a priori key distribution.*

K-security is a very strong notion of security. If one of the following weaknesses apply to a cipher, it cannot be called K-secure:

1. Existence of key-recovering attacks faster than exhaustive search. These are usually called *shortcut attacks.*

2. Certain symmetry properties in the block cipher (e.g. complementation property).
3. Occurrence of non-negligible classes of weak keys (as in IDEA).
4. Related-key attacks.

K-security is essentially a relative measure. It is quite possible to build a K-secure block cipher with a 5-bit block and key length. The lack of security offered by such a scheme is due to its small dimensions, not to the fact that the scheme fails to meet the requirements imposed by these dimensions. Clearly, the longer the key, the higher the security requirements.

It is possible to imagine ciphers that have certain weaknesses and still are K-secure. An example of such a weakness would be a block cipher with a block length larger than the key length and a single weak key, for which the permutation is linear. The detection of the usage of the key would take at least a few encryptions, whereas checking whether the key is used would only take a single encryption. If this cipher would be used for encryption, this single weak key would pose no problem. However, used as a component in a larger scheme, for instance as the compression function of a hash function, this property could introduce a way efficiently generating collisions. For these reasons we introduce yet another security concept, denoted by the term hermetic.

Definition 5.5.2. *A block cipher is hermetic if it does not have weaknesses that are not present for the majority of block ciphers with the same block and key length.*

Informally, a block cipher is hermetic if its internal structure cannot be exploited in any application. For all key and block lengths defined, the security goals are that the Rijndael cipher is K-secure and hermetic. If Rijndael lives up to its goals, the strength against any known or unknown attacks is as good as can be expected from a block cipher with the given dimensions.

5.5.2 Unknown Attacks Versus Known Attacks

'Prediction is very difficult, especially about the future.' (*Niels Bohr*)

Sometimes in cipher design, so-called *resistance against future, as yet unknown, types of cryptanalysis* is used as a rationale to introduce complexity. We prefer to base our ciphers on well-understood components that interact in well-understood ways allowing us to provide bounds that give evidence that the cipher is secure with respect to all known attacks. For ciphers making use of many different operations that interact in hard-to-analyse ways, it is much harder to provide such bounds.

5.5.3 Provable Security Versus Provable Bounds

Often claims are made that a cipher would be *provably secure*. Designing a block cipher that is provably secure in an absolute sense seems for now an unattainable goal. Reasonings that have been presented as proofs of security have been shown to be based on (often implicit) assumptions that make these 'proofs of security' irrelevant in the real world. Still, we consider having provable bounds for the workload of known types of cryptanalysis for a block cipher an important feature of the design.

5.6 Approaches to Design

5.6.1 Non-Linearity and Diffusion Criteria

Many papers are devoted to describing non-linearity and diffusion criteria and counting or characterizing classes of Boolean functions that satisfy them. In most of these papers the Boolean functions are (tacitly) assumed to be (components of) S-boxes located in the F-function of a Feistel structure or in an academic round transformation model such as *so-called* substitution-permutation networks [1, 77]. These networks consist of the alternation of parallel S-boxes and bit permutations, and were proposed in [30, 47]. The S-boxes are considered to be *the* elements in the round transformation that give the cipher its strength against cryptanalysis. Maybe the most important contribution of the wide trail strategy is the demonstration of the importance of the *linear* steps in the round transformation, and quantitative measures for the quality of the linear steps (cf. branch numbers, Sect. 9.3).

Many of the diffusion and non-linearity criteria described in cryptology literature are just criteria a block cipher must satisfy in order to be secure. They are necessary conditions for security, but not sufficient. To be of some use in cryptographic design, criteria for the *components* of a cipher are needed rather than criteria for the target *cipher*. Imposing criteria on components in a cipher only makes sense if first a structure of the cipher is defined in which the components have a specific function.

5.6.2 Resistance against Differential and Linear Cryptanalysis

The discovery of differential and linear cryptanalysis (see Chaps. 6–8) has given rise to a theoretical basis for the design of iterative block ciphers. Nowadays, a new block cipher is only taken seriously if it is accompanied with evidence that it resists differential and linear cryptanalysis. Naturally, differential and linear cryptanalysis are not the only attacks that can be mounted against block ciphers. In Chap. 10 we consider a number of generic types

of cryptanalysis and attacks that are specific for the structure of Rijndael and its related ciphers. A block cipher should resist all types of cryptanalysis imaginable. Still, we see that nowadays in most cases resistance against differential and linear cryptanalysis are the criteria that shape a block cipher and the other known attacks are only considered later and resistance against them can be obtained with small modifications in the original design (e.g. the affine transformation in the S_{RD} to thwart interpolation attacks, cf. Sect. 10.4).

Almost always, an iterative block cipher can be made resistant against differential and linear cryptanalysis by taking enough rounds. Even if a round transformation is used that offers very little non-linearity or diffusion, repeating it often enough will result in a block cipher that is not breakable by differential or linear cryptanalysis. For an iterated cipher, the workload of an encryption is the workload of the round transformation multiplied by the number of rounds. The engineering challenge is to design a round transformation in such a way that this product is minimized while providing lower bounds for the complexity of differential and linear cryptanalysis that are higher than exhaustive key search.

5.6.3 Local Versus Global Optimization

The engineering challenge can be taken in different ways. We distinguish two approaches:

1. **local optimization.** The round transformation is designed in such a way that the worst-case behaviour of one round is optimized.
2. **global optimization.** The round transformation is designed in such a way that the worst-case behaviour of a sequence of rounds is optimized.

In both cases, the worst-case behaviour is then used to determine the required number of rounds to offer resistance against differential and linear cryptanalysis. For the actual block cipher, usually some more rounds are taken to have a security margin (see Sect. 5.1.1).

In the context of linear cryptanalysis, this worst-case behaviour corresponds with the maximum input-output correlation (see Chap. 7) and in the case of differential cryptanalysis it corresponds to the maximum difference propagation probability (see Chap. 8).

In the case of local optimization, the maximum input-output correlation and the maximum difference propagation probability of the round transformation determine the number of rounds required. In Feistel ciphers (see Chap. 6) it does not make sense to evaluate these criteria over a single round, since part of the state is merely transposed and undergoes no non-linear operation. Therefore, for Feistel ciphers local optimization is done on a sequence of two rounds.

In Chaps. 7 and 8 we show that to obtain low maximum correlations and difference propagation probabilities, a Boolean transformation must have many input bits. In the local optimization approach the round must thus make use of expensive non-linear functions such as large S-boxes or modular multiplication. This can be considered to be a *greedy* approach: good non-linearity is obtained with only few rounds but at a high implementation cost.

The tendency to do local optimization can be found in many ciphers. For example, in [56] X. Lai et al. claim that the maximum difference propagation probability over a single round is an important measure of the resistance that a round transformation offers against differential cryptanalysis. Another example of local optimization is [76] by K. Nyberg and L. Knudsen. All results are obtained in terms of the maximum difference propagation probability of the F-function (see Chap. 6) of a Feistel cipher.

In global optimization, the maximum input-output correlation and difference propagation probability of the round transformation do not play such an important role. Here several approaches are possible. One of the approaches is the *wide trail strategy* that we have adopted for the design of Rijndael and its predecessors. To fully understand the wide trail strategy, we advise reading Chaps. 6–9.

As opposed to local optimization, global optimization allows cheap non-linear Boolean transformations such as small S-boxes. Global optimization introduces new diffusion criteria. These diffusion criteria no longer specify what the block cipher should satisfy, but give concrete criteria for the design of components of the round transformation. In most cases, the round transformation contains components that realize non-linearity and components that realize diffusion. The function of the diffusion components is to make sure that the input-output correlation (difference propagation probability) over r rounds is much less than the nth power of the maximum input-output correlation (difference propagation probability) of the round transformation.

For most round transformations, finding the maximum difference propagation probability and the maximum input-output correlation is computationally feasible. Computing the maximum difference propagation probability and the maximum input-output correlation over multiple rounds can, however, become very difficult. In the original differential cryptanalysis and linear cryptanalysis attacks on the DES, finding high difference propagation probabilities and input-output correlations over all but a few rounds of the cipher turned out to be one of the major challenges. In the linear cryptanalysis attack (cf. Chap. 6 and [65, 66]), M. Matsui had to write a sophisticated program that searched for the best linear expression.

In Rijndael and its predecessors, we have made use of symmetry and alignment to easily prove lower bounds for sequences of four rounds (two rounds in SHARK, see Chap. 11). If alignment is not applied, proving bounds becomes more difficult. An illustration of this is the AES finalist Serpent [3],

which also applied the principle of global optimization. The Serpent submission contains a table giving the maximum difference propagation probabilities and input-output correlations for 1–7 rounds, clearly illustrating this. Especially the bounds for 5–7 rounds were excellent. Unfortunately, the paper did not give a proof of these bounds nor a description of how they were obtained. Moreover, recently the designers of Serpent had to weaken the bounds due to new insights [8]. In our recent bit-slice cipher Noekeon [24], we have provided bounds for four rounds using an exhaustive search program with a relatively simple structure. By exploiting the high level of symmetry in Noekeon, many optimizations were possible in this program, enabling us to *demonstrate* surprisingly good bounds.

5.7 Key-Alternating Cipher Structure

By applying the key with a simple XOR, we simplify the analysis of the cipher and hence make it easier to prove lower bounds in the resistance against particular attacks such as differential and linear cryptanalysis (see Sect. 9.1).

The advantage of the key-alternating structure is that the *quality* of the round transformations in the context of linear or differential cryptanalysis is independent of the round key. By adopting a key-alternating structure, the analysis of linear and differential trails can be conducted without even considering the influence of the key. An example of a radically different approach is the block cipher IDEA [56].

Example 5.7.1. In IDEA the subkeys are applied by means of modular multiplication and addition. The difference propagation probability of the round transformation depend heavily on the value of these subkeys. However, the designers of IDEA have proposed considering alternative notions of *difference* to come to difference propagation probabilities that are independent of value of the round key. Unfortunately, attacks based on XOR as the difference appear to be more powerful than attacks making use of the alternative notion of difference. Moreover, the existence of weak subkeys and an unfortunate choice in the key schedule give rise to large classes of weak keys for which IDEA exhibits difference propagations probabilities equal to 1. Similar arguments apply for the resistance of IDEA against linear cryptanalysis.

5.8 The Key Schedule

5.8.1 The Function of a Key Schedule

The function of a key schedule is the generation of the round keys from the cipher key. For a key-alternating cipher with a block length of n_b and r rounds,

this means $n_b(r+1)$ bits. There is no consensus on the criteria that a key schedule must satisfy. In some design approaches, the key schedule must generate round keys in such a way that they appear to be *mutually independent* and can be considered *random* (see Sect.7.10.2 and 8.7.2). Moreover, for some ciphers the key schedule is so strong that the knowledge of one round key does not help in finding the cipher key or other round keys. In these ciphers, the key schedule appears to make use of components that can be considered as cryptographic primitives in their own right.

For the key schedule in Rijndael the criteria are less ambitious. Basically, the key schedule is there for three purposes.

1. The first one is the introduction of asymmetry. Asymmetry in the key schedule prevents symmetry in the round transformation and between the rounds leading to weaknesses or allows attacks. Examples of such weaknesses are the complementation property of the DES or weak keys such as in the DES [28]. Examples of attacks that exploit symmetry are slide attacks.
2. The second purpose is the resistance against related-key attacks (cf. Sect. 10.7).
3. The third purpose is the resistance against attacks in which the cipher key is (partially) known by or can be chosen by the cryptanalyst. This is the case if the cipher is used as the compression function of a hash function [52].

All other attacks are supposed to be prevented by the rounds of the block cipher. The modest criteria we impose can be met by a key schedule that is relatively simple and uses only a small amount of resources. This gives Rijndael its high key agility.

5.8.2 Key Expansion and Key Selection

In Rijndael, the key schedule consists of two parts: the *key expansion* that maps the n_k-bit cipher key to a so-called *expanded key*, and the *round key selection* that selects the n_b-bit round keys from the expanded key. This modularity facilitates the definition of a key expansion that is independent of the block length, and a round key selection that is independent of the cipher key length. For Rijndael, round key selection is very simple: the expanded key is seen as the concatenation of n_b-bit round keys starting with the first round key.

5.8.3 The Cost of the Key Expansion

In general, the key schedule is independent of the value of the plaintext or the ciphertext. If a cipher key is used for encrypting (or decrypting) multiple blocks, one may limit the computational cost of the key schedule by

performing the key expansion only once and keeping the expanded key in working memory for the complete sequence of encryptions. If cipher keys are used to encrypt large amounts of data, the computational cost of the key schedule can be neglected. Still, in many applications a cipher key is used for the encryption of only a small number of blocks. If the block cipher is used as a one-way function (e.g. in key derivation), or as the compression function in a hash function, each encryption is accompanied by the execution of the key schedule. In these cases, the computational cost of the key schedule is very important. Keeping the expanded key in working memory consumes $n_b(r+1)$ bits. In the case of Rijndael, a block length and key length of 128 bits require 176 bytes of key storage, whereas a block length and key length of 256 bits require 480 bytes of key storage. On some resource-limited platforms such as smart cards there may be not enough working memory available for the storage of the expanded key. To allow efficient implementations on these platforms, the key schedule must allow implementations using a limited amount of working memory in a limited number of processor cycles, in a small program.

5.8.4 A Recursive Key Expansion

We addressed the requirements discussed above by adopting a recursive structure. After the first n_k bits of the expanded key have been initialized with the cipher key, each subsequent bit is computed in terms of bits that have previously been generated. More specifically, if we picture the expanded key as a sequence of 4-byte columns, the column at position i in the expanded key can be computed using the columns at positions from $i - N_k$ to $i - 1$ only. Let us now consider a *block* consisting of the columns with indices from j to $j + N_k - 1$. By working out the dependencies, we can show that this block can be computed using columns $j - N_k$ to $j - 1$. In other words, each N_k-column block is completely determined by the previous N_k-column block. As N_k columns of the expanded key are sufficient for the computation of all following columns, the key schedule can be implemented taking only a working memory that is the size of the cipher key. The round keys are generated on-the-fly and the key expansion is executed whenever round key bits are required. In the case where the block length is equal to the key length, the blocks described above coincide with round keys, and round key i can be computed from round key $i + 1$ by what can be considered to be one *round* of the key schedule. Additionally, the recursion can be inverted, i.e. column i can be expressed in terms of columns $i+1$ to $i+N_k$. This implies that the expanded key can be computed backwards, starting from the last N_k columns. This allows on-the-fly round key generation for decryption.

The recursion function must have a low implementation cost while providing sufficient diffusion and asymmetry to thwart the attacks mentioned above. To protect against related key attacks, non-linearity can be introduced. More

specifically, the non-linearity should prohibit the full determination of differences in the expanded key from cipher key differences only.

5.9 Conclusions

In this chapter we have tried to make explicit the mindset with which we have designed Rijndael and its predecessors. A large part of it is the formulation of criteria that the result must satisfy. Cipher design is still more engineering than science. In many cases compromises have to be made between conflicting requirements. We are aware that Rijndael is just an attempt to achieve a cipher that satisfies our criteria and that it shows such compromises.

6. The Data Encryption Standard

In this chapter we given a brief description of the block cipher DES [33]. Both differential cryptanalysis and linear cryptanalysis were successfully applied to the DES: differential cryptanalysis was the first chosen-plaintext attack, and linear cryptanalysis was the first known-plaintext attack that was theoretically more efficient than an exhaustive key search for the DES. Resistance against these two attacks is the most important criterion in the design of Rijndael.

We give a summary of the original differential cryptanalysis and linear cryptanalysis attacks using the terminology of their inventors. For a more detailed treatment of the attacks, we refer to the original publications [9, 65]. The only aim of our description is to indicate the aspects of the attacks that determine their expected work factor. For differential cryptanalysis the critical aspect is the maximum *probability* for difference propagations, for linear cryptanalysis it is the maximum *deviation* from 0.5 of the probability that linear expressions hold.

6.1 The DES

The cipher that was the most important object of the attacks to be discussed is the DES [33]. Therefore, we start with a brief description of its structure.

The DES is an iterated block cipher with a block length of 64 bits and a key length of 56 bits. Its main body consists of 16 iterations of a *keyed round function*. The computational graph of the round function is depicted in Fig. 6.1. The state is split into a 32-bit left part L_i and a 32-bit right part R_i. The latter is the argument of the keyed F-function. L_i is modified by combining it with the output of the F-function by means of an XOR operation. Subsequently, the left and the right parts are interchanged. This round function has the so-called Feistel structure: the result of applying a key-dependent function to part of the state is added (using a bitwise XOR operation) to another part of the state, followed by a transposition of parts of the state. A block cipher that has rounds with this structure is called a Feistel cipher.

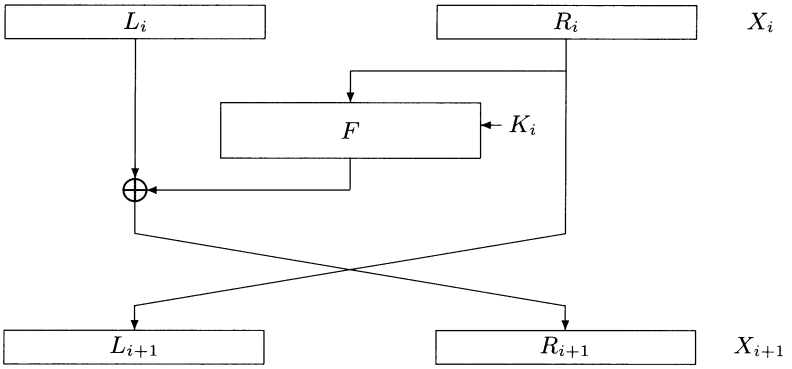

Fig. 6.1. Computational graph of the DES round function.

The computational graph of the F-function is depicted in Fig. 6.2. It consists of the succession of four steps:

1. **Expansion** E**.** the 32 input bits are expanded to a 48-bit vector. In this expansion, the 32-bit vector is split up in 4-bit nibbles, and the first and last bit of each nibble is duplicated.
2. **Key addition.** the 48-bit vector is modified by combining it with a 48-bit round key using the bitwise XOR operation.
3. **S-boxes.** the resulting 48-bit vector is mapped onto a 32-bit vector by non-linear S-boxes. The 48-bit vector is split up into eight 6-bit tuples that are converted into eight 4-bit nibbles by eight different non-linear S-boxes that each convert 6 input bits into 4 output bits. As an example, Table 6.1 gives the specification of the second S-box. This table must be read as follows. If the 6-bit input is denoted by $a_1a_2a_3a_4a_5a_6$, the output is given by the entry in row $a_1 + 2a_6$ and column $a_2 + 2a_3 + 4a_4 + 8a_5$. The 4-bit values are given in hexadecimal notation, e.g. `D` denotes 1101.
4. **Bit permutation** P**.** The bits of the 32-bit vector are transposed.

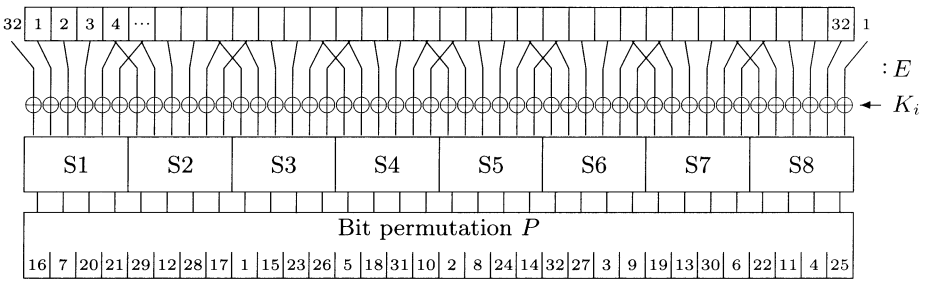

Fig. 6.2. Computational graph of the DES F-function.

Observe that the only non-linear step in the F-function (and also in the round transformation) consists of the S-boxes. The 48-bit round keys are extracted from the 56-bit cipher key by means of a linear key schedule.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 :	F	1	8	E	6	B	3	4	9	7	2	D	C	0	5	A
1 :	3	D	4	7	F	2	8	E	C	0	1	A	6	9	B	5
2 :	0	E	7	B	A	4	D	1	5	8	C	6	9	3	2	F
3 :	D	8	A	1	3	F	4	2	B	6	7	C	0	5	E	9

Table 6.1. Specification of the DES S-box S_2.

6.2 Differential Cryptanalysis

In this section we summarize the most important elements of differential cryptanalysis as E. Biham and A. Shamir described it in [9].

Differential cryptanalysis is a chosen-plaintext (difference) attack in which a large number of plaintext-ciphertext pairs are used to determine the value of key bits. Statistical key information is deduced from ciphertext blocks obtained by encrypting pairs of plaintext blocks with a specific bitwise difference A' under the target key. The work factor of the attack depends critically on the largest probability $\text{Prob}(B'|A')$ with B' being a difference at some fixed intermediate stage of the block cipher, e.g. at the input of the last round.

In a first approximation, the probabilities $\text{Prob}(B'|A')$ for the DES are assumed to be independent of the specific value of the key.

In the basic form of the attack, key information is extracted from the output pairs in the following way. For each pair it is assumed that the intermediate difference is equal to B'. The absolute values of the output pair and the (assumed) intermediate difference B' impose restrictions upon a number ℓ of key bits of the last round key. A pair is said to *suggest* the subkey values that are compatible with these restrictions. While for some pairs many keys are suggested, no keys are found for other pairs, implying that the output values are incompatible with B'. For each suggested subkey value, a corresponding entry in a frequency table is incremented.

The attack is successful if the correct value of the subkey is suggested significantly more often than any other value. Pairs with an intermediate difference not equal to B' are called *wrong pairs*. Sub-key values suggested by these pairs are in general wrong. Right pairs, with an intermediate difference equal to B', do not only suggest the right subkey value but often also a number of wrong subkey values. For the DES, the wrong suggestions may be

considered uniformly distributed among the possible key values if the value $\text{Prob}(B'|A')$ is significantly larger than $\text{Prob}(C'|A')$ for any $C' \neq B'$.

Under these conditions it makes sense to calculate the ratio between the number of times the right value is suggested and the average number of suggestions per entry, the so-called *signal-to-noise* (S/N) *ratio*.

The size of the table of possible values of the ℓ-bit subkey is 2^ℓ. If we denote the average number of suggested subkeys per pair by γ, the S/N ratio is given by:

$$\text{S/N} = \text{Prob}(B'|A')2^\ell/\gamma. \tag{6.1}$$

The S/N ratio strongly affects the number of right pairs needed to uniquely identify the correct subkey value. Experimental results [9] showed that for a ratio of 1–2 about 20–40 right pairs are enough. For larger ratios only a few right pairs are needed and for ratios that are much smaller than 1 the required amount of right pairs makes a practical attack infeasible.

Pairs of differences A' and B' with a large probability $\text{Prob}(B'|A')$ are found by the construction of so-called *characteristics*. An r-round characteristic constitutes an $(r+1)$-tuple of difference patterns: $(X'_0, X'_1, \ldots, X'_r)$. The probability of this characteristic is the probability that an initial difference pattern X'_0 propagates to difference patterns $X'_1, X'_2, \ldots, X'_r$ after 1, 2, ..., r rounds, respectively. Under the so-called *Markov assumption* (cf. also Sect. 8.7.2), i.e. that the propagation probability from X'_{i-1} to X'_i is independent of the propagation from X'_0 to X'_{i-1}, this probability is given by

$$\prod_i \text{Prob}(X'_i|X'_{i-1}), \tag{6.2}$$

where $\text{Prob}(X'_i|X'_{i-1})$ is the probability that the difference pattern X'_{i-1} at the input of the round transformation gives rise to X'_i at its output. Hence, the multiple-round characteristic is a sequence of single-round characteristics (X'_{i-1}, X'_i) with probability $\text{Prob}(X'_i|X'_{i-1})$.

In the construction of high-probability characteristics for the DES, advantage is taken from the linearity in the round transformation. Single-round characteristics of the form $(L'_{i-1}\|R'_{i-1}, L'_i\|R'_i)$, where $R'_i = L'_{i-1}$ and $L'_i = R'_{i-1} = 0$ have probability 1 and are called *trivial*. The most probable non-trivial single-round characteristics have an input difference pattern that only affects a small number of the eight S-boxes.

Trivial characteristics have been exploited to construct high-probability *iterative characteristics*. These are characteristics with a periodic sequence of differences. The iterative characteristic with highest probability has a period of two. Of the two involved single-round characteristics, one is trivial. In the other one there is a non-zero difference pattern at the input of three neighbouring S-boxes, which propagates to a zero difference pattern at the

output of the S-boxes with probability 1/234. Hence, the resulting iterative characteristics have a probability of 1/234 per two rounds.

In the actual differential attacks on the DES, some techniques are used to make the attack more efficient. This involves a special treatment in the first and last rounds. For these techniques we refer to [9].

6.3 Linear Cryptanalysis

In this section we summarize the most important elements of linear cryptanalysis as M. Matsui presented them in [65]. Linear cryptanalysis is a known-plaintext attack in which a large number of plaintext-ciphertext pairs are used to determine the value of key bits.

A condition for applying linear cryptanalysis to a block cipher is to find 'effective' linear expressions. Let $\mathbf{A}[i_1, i_2, \ldots, i_a]$ be the bitwise sum of the bits of $\mathbf{A}$ with indices in a *selection pattern* $\{i_1, i_2, \ldots, i_a\}$; i.e.

$$\mathbf{A}[i_1, i_2, \ldots, i_a] = \mathbf{A}[i_1] \oplus \mathbf{A}[i_2] \oplus \cdots \oplus \mathbf{A}[i_a]. \tag{6.3}$$

Let $\mathbf{P}, \mathbf{C}$ and $\mathbf{K}$ denote the plaintext, the ciphertext and the key, respectively. A linear expression is an expression of the following type:

$$\mathbf{P}[i_1, i_2, \ldots, i_a] \oplus \mathbf{C}[j_1, j_2, \ldots, j_b] = \mathbf{K}[k_1, k_2, \ldots, k_c], \tag{6.4}$$

with $i_1, i_2, \ldots, i_a$, $j_1, j_2, \ldots, j_b$ and $k_1, k_2, \ldots, k_c$ being fixed bit locations. The effectiveness, or deviation, of such a linear expression in linear cryptanalysis is given by $|p - 1/2|$ where p is the probability that the expression holds. By checking the value of the left-hand side of (6.4) for a large number of plaintext-ciphertext pairs, the right-hand side can be guessed by taking the value that occurs most often. In principle, this gives a single bit of information about the key. In [65] it is shown that the probability of making a wrong guess is very small if the number of plaintext-ciphertext pairs is larger than $|p - 1/2|^{-2}$.

In [65] another algorithm is given that determines more than a single bit of key information using a similar linear expression. Instead of (6.4), an expression is used that contains no plaintext or ciphertext bits, but instead contains bits of the intermediate encryption values I_1 and I_{15}, respectively, after exactly one round and after all rounds but one:

$$\mathbf{I}_1[i_1, i_2, \ldots, i_a] \oplus \mathbf{I}_{15}[j_1, j_2, \ldots, j_b] = \mathbf{K}[\ell_1, \ell_2, \ldots, \ell_c]. \tag{6.5}$$

By assuming values for a subset ν_k of the subkey bits of the first and last round, the bits of $\mathbf{I}_1$ and $\mathbf{I}_{15}$ that occur in (6.5) can be calculated. These bits are correct if the values assumed for the key bits with indices in ν_k are correct. Given a large number ℓ of plaintext-ciphertext pairs, the correct

values of all bits in ν_k and the value of the right-hand side of (6.5) can be determined in the following way. For all values of the key bits with indices in ν_k, the number of plaintext-ciphertext pairs are counted for which (6.5) holds. For the correct assumption the expected value of this sum is $p\ell$ or $(1-p)\ell$. Thanks to the non-linear behavior of the round transformation this sum is expected to have significantly less bias for all wrongly assumed subkey values. Given a linear expression (6.5) that holds with probability p, the probability that this algorithm leads to a wrong guess is very small if the number of plaintext-ciphertext pairs is significantly (say more than a factor 8) larger than $|p-1/2|^{-2}$. In an improved version of this attack, this factor 8 is reduced to 1 [66]. Hence, in both variants the value of $|p-1/2|$ is critical for the work factor of the attack.

Effective linear expressions (6.4) and (6.5) are constructed by 'chaining' single-round linear expressions. An $(r-1)$-round linear expression can be turned into an r-round linear expression by appending a single-round linear expression such that all the intermediate bits cancel:

$$\begin{aligned} \mathbf{P}[i_1, i_2, \ldots, i_a] \oplus \mathbf{I}_{r-1}[j_1, j_2, \ldots, j_b] &= \mathbf{K}[k_1, k_2, \ldots, k_c] \\ \oplus & \\ \mathbf{I}_{r-1}[j_1, j_2, \ldots, j_b] \oplus \mathbf{I}_r[m_1, m_2, \ldots, m_a] &= \mathbf{K}[k_2, k_5, \ldots, k_d] \\ = & \\ \mathbf{P}[i_1, i_2, \ldots, i_a] \oplus \mathbf{I}_r[m_1, m_2, \ldots, m_a] &= \mathbf{K}[k_1, k_3, \ldots, k_d] \end{aligned} \tag{6.6}$$

In [65] it is shown that the probability that the resulting linear expression holds can be approximated by $1/2 + 2(p_1 - 1/2)(p_2 - 1/2)$, given that the component linear expressions hold with probabilities p_1 and p_2, respectively.

The DES single-round linear expressions and their probabilities can be studied by observing the dependencies in the computational graph of the round transformation. The selected round output bits completely specify a selection pattern at the output of the S-boxes. If only round output bits are selected from the left half, this involves no S-box output bits at all, resulting in linear expressions that hold with a probability of 1. These are of the following type:

$$\mathbf{I}_{\ell-1}[j_1 + 32, j_2 + 32, \ldots, j_a + 32] = \mathbf{I}_\ell[j_1, j_2, \ldots, j_a]. \tag{6.7}$$

This is called a *trivial* expression. Apparently, the most useful non-trivial single-round linear expressions only select bits coming from a single S-box. For a given S-box, all possible linear expressions and their probabilities can be exhaustively calculated. Together with the key application before the S-boxes, each of these linear expressions can be converted into a single-round linear expression. The most effective multiple-round linear expressions for the DES are constructed by combining single-round trivial expressions with linear expressions involving output bits of only a single S-box. The resulting most effective 14-round linear expression has a probability of $1/2 \pm 1.19 \times 2^{-21}$.

6.4 Conclusions

In this section we have explained the round structure of the DES and have given a summary of the two most important cryptanalytic attacks on the DES using the terminology and formalism of the original publications.

7. Correlation Matrices

In this chapter we consider correlations over Boolean functions and iterated Boolean transformations. Correlations play an important role in cryptanalysis in general and linear cryptanalysis in particular.

We introduce algebraic tools such as *correlation matrices* to adequately describe the properties that make linear cryptanalysis possible. We derive a number of interesting relations and equalities and apply these to iterated Boolean transformations.

7.1 The Walsh-Hadamard Transform

7.1.1 Parities and Selection Patterns

A *parity* of a Boolean vector is a binary Boolean function that consists of the XOR of a number of bits. A parity is determined by the positions of the bits of the Boolean vector that are included in the XOR.

The *selection pattern* $\mathbf{w}$ of a parity is a Boolean vector value that has a 1 in the components that are included in the parity and a 0 in all other components. Analogous to the inner product of vectors in linear algebra, we express the parity of vector $\mathbf{a}$ corresponding with selection pattern $\mathbf{w}$ as $\mathbf{w}^{\mathrm{T}}\mathbf{a}$. The concepts of selection vector and parity are illustrated with an example in Fig. 7.1.

Note that for a vector $\mathbf{a}$ with n bits, there are 2^n different parities. The set of parities of a Boolean vector is in fact the set of all linear binary Boolean functions of that vector.

7.1.2 Correlation

Linear cryptanalysis exploits large correlations over all but a few rounds of a block cipher.

Definition 7.1.1. *The correlation* $\mathrm{C}(f, g)$ *between two binary Boolean functions* $f(\mathbf{a})$ *and* $g(\mathbf{a})$ *is defined as*

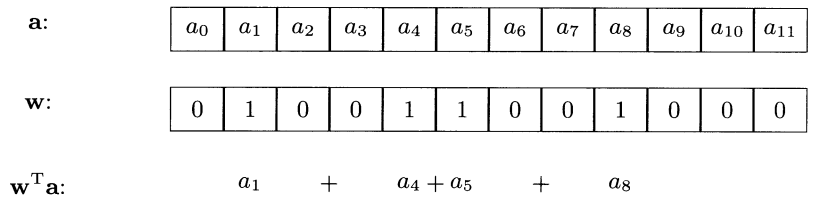

Fig. 7.1. Example of state $\mathbf{a}$, selection pattern $\mathbf{w}$ and parity $\mathbf{w}^T\mathbf{a}$.

$$C(f,g) = 2 \cdot \text{Prob}(f(\mathbf{a}) = g(\mathbf{a})) - 1. \tag{7.1}$$

From this definition it follows that $C(f,g) = C(g,f)$. The correlation between two binary Boolean functions ranges between -1 and 1. If the correlation is different from zero, the binary Boolean functions are said to be *correlated*. If the correlation is 1, the binary Boolean functions are equal; if it is -1, the binary Boolean functions are each other's complement.

7.1.3 Real-valued Counterpart of a Binary Boolean Function

Let $\hat{f}(\mathbf{a})$ be a real-valued function that is -1 for $f(\mathbf{a}) = 1$ and $+1$ for $f(\mathbf{a}) = 0$. This can be expressed by

$$\hat{f}(\mathbf{a}) = (-1)^{f(\mathbf{a})} \tag{7.2}$$

In this notation the real-valued function corresponding to a parity $\mathbf{w}^T\mathbf{a}$ becomes $(-1)^{\mathbf{w}^T\mathbf{a}}$. The real-valued counterpart of the XOR of two binary Boolean functions is the product of their real-valued counterparts, i.e.

$$\widehat{f(\mathbf{a}) \oplus g(\mathbf{a})} = \hat{f}(\mathbf{a})\hat{g}(\mathbf{a}). \tag{7.3}$$

7.1.4 Orthogonality and Correlation

We define the *inner product* of two binary Boolean functions f and g as

$$< \hat{f}, \hat{g} >= \sum_{\mathbf{a}} \hat{f}(\mathbf{a})\hat{g}(\mathbf{a}). \tag{7.4}$$

This inner product defines the following *norm*:

$$\|\hat{f}\| = \sqrt{< \hat{f}, \hat{f} >}. \tag{7.5}$$

The norm of a binary Boolean function $f(\mathbf{a})$ is equal to the square root of its domain size, i.e. $2^{n/2}$.

From the definition of correlation it follows that

$$\mathrm{C}(f,g) = \frac{<\hat{f},\hat{g}>}{\|\hat{f}\| \cdot \|\hat{g}\|}, \tag{7.6}$$

or in words, the correlation between two binary Boolean functions is equal to their inner product divided by their norms. In Fig. 7.2 this is illustrated in a geometrical way.

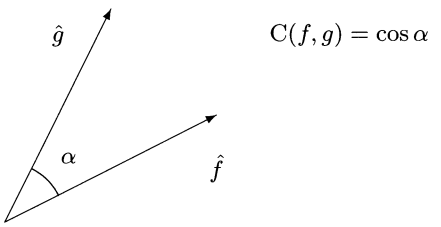

Fig. 7.2. Geometric representation of binary Boolean functions and their correlation.

7.1.5 Spectrum of a Binary Boolean Function

The set of binary Boolean functions of an n-bit vector can be seen as elements of a vector space of dimension 2^n. A vector f has 2^n components given by $(-1)^{f(\mathbf{a})}$ for the 2^n values of $\mathbf{a}$. Vector addition corresponds with addition of the components in $\mathbb{R}$, scalar multiplication as multiplication of components with elements of $\mathbb{R}$. This is the vector space $<\mathbb{R}^{2^n},+,.>$.

In $<\mathbb{R}^{2^n},+,.>$ the parities form an orthogonal basis with respect to the inner product defined by (7.4):

$$\begin{aligned}
<(-1)^{\mathbf{u}^{\mathrm{T}}\mathbf{a}},(-1)^{\mathbf{v}^{\mathrm{T}}\mathbf{a}}> &= \sum_{\mathbf{a}}(-1)^{\mathbf{u}^{\mathrm{T}}\mathbf{a}}(-1)^{\mathbf{v}^{\mathrm{T}}\mathbf{a}} \\
&= \sum_{\mathbf{a}}(-1)^{\mathbf{u}^{\mathrm{T}}\mathbf{a}\oplus\mathbf{v}^{\mathrm{T}}\mathbf{a}} \\
&= \sum_{\mathbf{a}}(-1)^{(\mathbf{u}\oplus\mathbf{v})^{\mathrm{T}}\mathbf{a}} \\
&= 2^n\delta(\mathbf{u}\oplus\mathbf{v}).
\end{aligned}$$

Here $\delta(w)$ is the Kronecker delta function that is equal to 1 if w is the zero vector and 0 otherwise. The representation of a binary Boolean function with respect to the parity basis is called its Walsh-Hadamard spectrum, or just its *spectrum* [38, 79].

Consider now $\mathrm{C}(f(\mathbf{a}), \mathbf{w}^{\mathrm{T}}\mathbf{a})$, which is the correlation between a binary Boolean function $f(\mathbf{a})$ and the parity $\mathbf{w}^{\mathrm{T}}\mathbf{a}$. If we denote this by $F(\mathbf{w})$, we have

$$\hat{f}(\mathbf{a}) = \sum_{\mathbf{w}} F(\mathbf{w})(-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}}, \tag{7.7}$$

where $\mathbf{w}$ ranges over all possible 2^n values. In words, the coordinates of a binary Boolean function in the parity basis are the correlations between the binary Boolean function and the parities. It follows that a Boolean function is completely specified by the set of correlations with all parities.

Dually, we have:

$$F(\mathbf{w}) = \mathrm{C}(f(\mathbf{a}), \mathbf{w}^{\mathrm{T}}\mathbf{a}) = 2^{-n} \sum_{\mathbf{a}} \hat{f}(\mathbf{a})(-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}}, \tag{7.8}$$

We denote the Walsh-Hadamard transform by the symbol $\mathcal{W}$. We have

$$\mathcal{W} : f(\mathbf{a}) \mapsto F(\mathbf{w}) : \quad F(\mathbf{w}) = \mathcal{W}(f(\mathbf{a})). \tag{7.9}$$

If we take the square of the norm of both sides of (7.7), we obtain:

$$< \hat{f}(\mathbf{a}), \hat{f}(\mathbf{a}) > = < \sum_{\mathbf{w}} F(\mathbf{w})(-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}}, \sum_{\mathbf{v}} F(\mathbf{v})(-1)^{\mathbf{v}^{\mathrm{T}}\mathbf{a}} > . \tag{7.10}$$

Working out both sides gives:

$$2^n = \sum_{\mathbf{w}} F(\mathbf{w}) < (-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}}, \sum_{\mathbf{v}} F(\mathbf{v})(-1)^{\mathbf{v}^{\mathrm{T}}\mathbf{a}} > \tag{7.11}$$

$$= \sum_{\mathbf{w}} F(\mathbf{w}) \sum_{\mathbf{v}} F(\mathbf{v}) < (-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}}, (-1)^{\mathbf{v}^{\mathrm{T}}\mathbf{a}} > \tag{7.12}$$

$$= \sum_{\mathbf{w}} \sum_{\mathbf{v}} F(\mathbf{w}) F(\mathbf{v}) 2^n \delta(\mathbf{w} \oplus \mathbf{v}) \tag{7.13}$$

$$= 2^n \sum_{\mathbf{w}} F^2(\mathbf{w}). \tag{7.14}$$

Dividing this by 2^n yields the theorem of Parseval [63, p. 416]:

$$\sum_{\mathbf{w}} F^2(\mathbf{w}) = 1. \tag{7.15}$$

This theorem expresses a relation between the number of parities that a given binary Boolean function is correlated with and the amplitude of the correlations. If we denote the square of a correlation by *correlation potential*, it states that the correlation potentials corresponding to all input parities sum to 1.

7.2 Composing Binary Boolean Functions

7.2.1 XOR

The spectrum of the XOR of two binary Boolean functions $f(\mathbf{a}) \oplus g(\mathbf{a})$ can be derived using (7.7):

$$\begin{aligned}\hat{f}(\mathbf{a})\hat{g}(\mathbf{a}) &= \sum_{\mathbf{u}} F(\mathbf{u})(-1)^{\mathbf{u}^{\mathrm{T}}\mathbf{a}} \sum_{\mathbf{v}} G(\mathbf{v})(-1)^{\mathbf{v}^{\mathrm{T}}\mathbf{a}} \\ &= \sum_{\mathbf{u}} \sum_{\mathbf{v}} F(\mathbf{u})G(\mathbf{v})(-1)^{(\mathbf{u}\oplus\mathbf{v})^{\mathrm{T}}\mathbf{a}} \\ &= \sum_{\mathbf{w}} \left(\sum_{\mathbf{v}} F(\mathbf{v} \oplus \mathbf{w})G(\mathbf{v}) \right) (-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}}. \end{aligned} \tag{7.16}$$

The values of of the spectrum $H(\mathbf{w}) = \mathcal{W}(f \oplus g)$ are therefore given by

$$H(\mathbf{w}) = \sum_{\mathbf{v}} F(\mathbf{v} \oplus \mathbf{w})G(\mathbf{v}). \tag{7.17}$$

Hence, the spectrum of the XOR of binary Boolean functions is equal to the convolution of the corresponding spectra. We express this as

$$\mathcal{W}(f \oplus g) = \mathcal{W}(f) \otimes \mathcal{W}(g), \tag{7.18}$$

where $\otimes$ denotes the convolution operation. Given this convolution property it is easy to demonstrate some composition properties that are useful in the study of linear cryptanalysis:

1. The spectrum of the complement of a binary Boolean function $g(\mathbf{a}) = f(\mathbf{a}) \oplus 1$ is the negative of the spectrum of $f(\mathbf{a})$: $G(\mathbf{w}) = -F(\mathbf{w})$.
2. The spectrum of the sum of a binary Boolean function and a parity $g(\mathbf{a}) = f(\mathbf{a}) \oplus \mathbf{u}^{\mathrm{T}}\mathbf{a}$ is equal to the spectrum of $f(\mathbf{a})$ transformed by a so-called *dyadic* shift: $G(\mathbf{w}) = F(\mathbf{w} \oplus \mathbf{u})$.

7.2.2 AND

For the AND of two binary Boolean functions we have

$$\widehat{f(\mathbf{a})g(\mathbf{a})} = \frac{1}{2}(1 + \hat{f}(\mathbf{a}) + \hat{g}(\mathbf{a}) - \hat{f}(\mathbf{a})\hat{g}(\mathbf{a})). \tag{7.19}$$

It follows that

$$\mathcal{W}(fg) = \frac{1}{2}(\delta(\mathbf{w}) + \mathcal{W}(f) + \mathcal{W}(g) - \mathcal{W}(f \oplus g)). \tag{7.20}$$

7.2.3 Disjunct Boolean Functions

The subspace of $\mathrm{GF}(2)^n$ generated by the selection patterns $\mathbf{w}$ for which $F(\mathbf{w}) \neq 0$ is called the *support space* of f and is denoted by $\mathcal{V}_f$. The support space of the XOR of two binary Boolean functions is a subspace of the (vector) sum of their corresponding support spaces:

$$\mathcal{V}_{f \oplus g} \subseteq \mathcal{V}_f + \mathcal{V}_g \tag{7.21}$$

This follows directly from the convolution property. Two binary Boolean functions are called *disjunct* if their support spaces are disjunct, i.e., if the intersection of their support spaces only contains the origin. A vector $\mathbf{v} \in \mathcal{V}_{f \oplus g}$ with f and g disjunct has a unique decomposition into a component $\mathbf{u} \in \mathcal{V}_f$ and a component $\mathbf{w} \in \mathcal{V}_g$. In this case the spectrum of $h = f \oplus g$ satisfies

$$H(\mathbf{v}) = F(\mathbf{u})G(\mathbf{w}) \text{ where } \mathbf{v} = \mathbf{u} \oplus \mathbf{w} \text{ and } \mathbf{u} \in \mathcal{V}_f, \mathbf{w} \in \mathcal{V}_g. \tag{7.22}$$

A pair of binary Boolean functions that depend on non-overlapping sets of input bits is a special case of disjunct functions.

7.3 Correlation Matrices

Almost all components in block ciphers are Boolean functions mapping n-bit vectors to m-bit vectors. Examples are S-boxes, round transformations and their steps, and block ciphers themselves. In many cases $m = n$. These functions can be represented by their *correlation matrix*.

A Boolean function $h : \mathrm{GF}(2)^n \to \mathrm{GF}(2)^m$ can be decomposed into m *component* binary Boolean functions:

$$(h_0, h_1, \ldots, h_{m-1}).$$

Each of these component binary Boolean functions h_i has a spectrum H_i. The vector function H with components H_i can be considered the spectrum of the Boolean function h. As in the case of binary Boolean functions, H completely determines the function h.

The spectrum of any parity of components of h is specified by a simple extension of (7.18):

$$\mathcal{W}(\mathbf{u}^{\mathrm{T}} h) = \bigotimes_{u_i=1} H_i. \tag{7.23}$$

The correlations between input parities and output parities of a Boolean function h can be arranged in a $2^m \times 2^n$ *correlation matrix* $\mathsf{C}^{(h)}$. The element $\mathsf{C}^{(h)}_{\mathbf{u},\mathbf{w}}$ in row $\mathbf{u}$ and column $\mathbf{w}$ is equal to $\mathrm{C}(\mathbf{u}^{\mathrm{T}} h(\mathbf{a}), \mathbf{w}^{\mathrm{T}} \mathbf{a})$.

Row $\mathbf{u}$ of a correlation matrix can be interpreted as

$$(-1)^{\mathbf{u}^{\mathrm{T}}h(\mathbf{a})} = \sum_{\mathbf{w}} \mathsf{C}^{(h)}_{\mathbf{u},\mathbf{w}} (-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}}. \tag{7.24}$$

This expresses an output parity with respect to the basis of input parities.

A binary Boolean function $f(\mathbf{a})$ is a special case of a Boolean function: it has $m = 1$. Its correlation matrix has two rows: row 0 and row 1. Row 1 contains the spectrum of $f(\mathbf{a})$. Row 0 contains the spectrum of the *empty parity*: the binary Boolean function that is equal to 0. This row has a 1 in column 0 and zeroes in all other columns.

7.3.1 Equivalence of a Boolean Function and its Correlation Matrix

A correlation matrix $\mathsf{C}^{(h)}$ defines a linear map with domain $\mathbb{R}^{2^n}$ and range $\mathbb{R}^{2^m}$. Let $\mathcal{L}$ be a transformation from the space of binary vectors to the space of real-valued functions, which transforms a binary vector of dimension n to a real-valued function of dimension 2^n. $\mathcal{L}$ is defined by

$$\mathcal{L} : \mathrm{GF}(2)^n \to \mathbb{R}^{2^n} : \mathbf{a} \mapsto \mathcal{L}(\mathbf{a}) = \alpha \Leftrightarrow \alpha_u = (-1)^{\mathbf{u}^{\mathrm{T}}\mathbf{a}}. \tag{7.25}$$

Since $\mathcal{L}(\mathbf{a} \oplus \mathbf{b}) = \mathcal{L}(\mathbf{a}) \cdot \mathcal{L}(\mathbf{b})$, $\mathcal{L}$ is a group homomorphism from $< \mathrm{GF}(2)^n, \oplus >$ to $< (\mathbb{R} \backslash \{0\})^{2^n}, \cdot >$, where ' $\cdot$ ' denotes the component-wise product. From (7.24) it follows that

$$\mathsf{C}^{(h)} \mathcal{L}(\mathbf{a}) = \mathcal{L}(h(\mathbf{a})). \tag{7.26}$$

In words, applying a Boolean function h to a Boolean vector $\mathbf{a}$ and transforming the corresponding function $(-1)^{\mathbf{u}^{\mathrm{T}}\mathbf{a}}$ with the correlation matrix $\mathsf{C}^{(h)}$ are just different representations of the same operation. This is illustrated in Fig. 7.3.

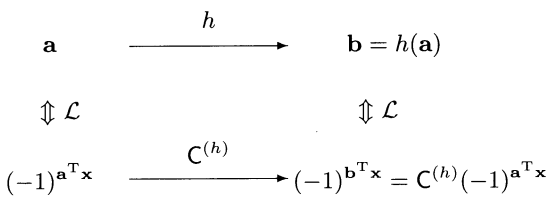

Fig. 7.3. The equivalence of a Boolean function and its correlation matrix.

7.3.2 Iterative Boolean Functions

Consider an iterated Boolean function h that is the composition of two Boolean functions $h = h^{(2)} \circ h^{(1)}$ or $h(\mathbf{a}) = h^{(2)}(h^{(1)}(\mathbf{a}))$, where the function $h^{(1)}$ transforms n-bit vectors to p-bit vectors and where the function $h^{(2)}$ transforms p-bit vectors to m-bit vectors. The correlation matrix of h is determined by the correlation matrices of the component functions. We have

$$\begin{aligned}(-1)^{\mathbf{u}^{\mathrm{T}}h(\mathbf{a})} &= \sum_{\mathbf{v}} \mathsf{C}^{(h^{(2)})}_{\mathbf{u},\mathbf{v}}(-1)^{\mathbf{v}^{\mathrm{T}}h^{(1)}(\mathbf{a})} \\ &= \sum_{\mathbf{v}} \mathsf{C}^{(h^{(2)})}_{\mathbf{u},\mathbf{v}} \sum_{\mathbf{w}} \mathsf{C}^{(h^{(1)})}_{\mathbf{v},\mathbf{w}}(-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}} \\ &= \sum_{\mathbf{w}} (\sum_{\mathbf{v}} \mathsf{C}^{(h^{(2)})}_{u,v} \mathsf{C}^{(h^{(1)})}_{\mathbf{v},\mathbf{w}})(-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}}.\end{aligned}$$

Hence, we have

$$\mathsf{C}^{(h^{(2)} \circ h^{(1)})} = \mathsf{C}^{(h^{(2)})} \times \mathsf{C}^{(h^{(1)})}, \tag{7.27}$$

where $\times$ denotes the matrix product, $\mathsf{C}^{(h^{(1)})}$ is a $2^p \times 2^n$ matrix and $\mathsf{C}^{(h^{(2)})}$ is a $2^m \times 2^p$ matrix. Hence the correlation matrix of the composition of two Boolean functions is the product of their correlation matrices. This is illustrated in Fig. 7.4.

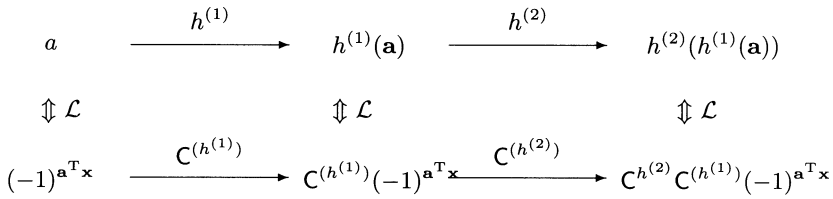

Fig. 7.4. Composition Boolean functions and multiplication of correlation matrices.

The correlations over $h = h^{(2)} \circ h^{(1)}$ are given by

$$\mathrm{C}(\mathbf{u}^{\mathrm{T}}h(\mathbf{a}), \mathbf{w}^{\mathrm{T}}\mathbf{a}) = \sum_{\mathbf{v}} \mathrm{C}(\mathbf{u}^{\mathrm{T}}h^{(1)}(\mathbf{a}), \mathbf{v}^{\mathrm{T}}\mathbf{a})\mathrm{C}(\mathbf{v}^{\mathrm{T}}h^{(2)}(\mathbf{a}), \mathbf{w}^{\mathrm{T}}\mathbf{a}). \tag{7.28}$$

7.3.3 Boolean Permutations

If h is a permutation in $\mathrm{GF}(2)^n$, we have

$$\mathrm{C}(\mathbf{u}^{\mathrm{T}}h^{-1}(\mathbf{a}), \mathbf{w}^{\mathrm{T}}\mathbf{a}) = \mathrm{C}(\mathbf{u}^{\mathrm{T}}\mathbf{b}, \mathbf{w}^{\mathrm{T}}h(\mathbf{b})) = \mathrm{C}(\mathbf{w}^{\mathrm{T}}h(\mathbf{b}), \mathbf{u}^{\mathrm{T}}\mathbf{b}). \tag{7.29}$$

It follows that the correlation matrix of h^{-1} is the transpose of the correlation matrix of h:

$$\mathsf{C}^{(h^{-1})} = (\mathsf{C}^{(h)})^{\mathrm{T}}, \tag{7.30}$$

Moreover, from the fact that the composition of a Boolean transformation and its inverse gives the identity function, the product of the corresponding correlation matrices must result in the identity matrix:

$$\mathsf{C}^{(h^{-1})} \times \mathsf{C}^{(h)} = \mathsf{I} = \mathsf{C}^{(h)} \times \mathsf{C}^{(h^{-1})}. \tag{7.31}$$

It follows that:

$$\mathsf{C}^{(h^{-1})} = (\mathsf{C}^{(h)})^{-1}. \tag{7.32}$$

We can now prove the following theorem:

Theorem 7.3.1. *A Boolean transformation is invertible iff it has an invertible correlation matrix.*

Proof.

$\Rightarrow$ For an invertible Boolean transformation, both (7.30) and (7.32) are valid. Combining these two equations, we have

$$(\mathsf{C}^{(h)})^{-1} = (\mathsf{C}^{(h)})^{\mathrm{T}}.$$

$\Leftarrow$ Interpreting the rows of the correlation matrix according to (7.24) yields a set of n equations, one for each value of $\mathbf{u}$:

$$(-1)^{\mathbf{u}^{\mathrm{T}} h(\mathbf{a})} = \sum_{\mathbf{w}} \mathsf{C}^{(h)}_{\mathbf{u},\mathbf{w}} (-1)^{\mathbf{w}^{\mathrm{T}} \mathbf{a}}.$$

If we assume that the inverse of $C^{(h)}$ exists, we can convert this set of n equations, one for each value of $\mathbf{w}$:

$$(-1)^{\mathbf{w}^{\mathrm{T}} a} = \sum_{\mathbf{u}} (C^{(h)})^{-1}_{\mathbf{w},\mathbf{u}} (-1)^{\mathbf{u}^{\mathrm{T}} h(\mathbf{a})}. \tag{7.33}$$

Assume that we have two Boolean vectors $\mathbf{x}$ and $\mathbf{y}$ for which $h(\mathbf{x}) = h(\mathbf{y})$. By substituting $\mathbf{a}$ in (7.33) by $\mathbf{x}$ and $\mathbf{y}$ respectively, we obtain n equations, one for each value of $\mathbf{w}$:

$$(-1)^{\mathbf{w}^{\mathrm{T}} \mathbf{x}} = (-1)^{\mathbf{w}^{\mathrm{T}} \mathbf{y}}.$$

From this it follows that $\mathbf{x} = \mathbf{y}$ and hence that h is injective. It follows that h is invertible. □

7.4 Special Boolean Functions

7.4.1 XOR with a Constant

In the following, the superscript (h) in $\mathsf{C}^{(h)}$ will be omitted. Consider the function that consists of the bitwise XOR with a constant vector $\mathbf{k}$: $h(\mathbf{a}) = \mathbf{a} \oplus \mathbf{k}$. Since $\mathbf{u}^{\mathrm{T}} h(\mathbf{a}) = \mathbf{u}^{\mathrm{T}}\mathbf{a} \oplus \mathbf{u}^{\mathrm{T}}\mathbf{k}$, the correlation matrix is a diagonal matrix with

$$\mathsf{C}_{\mathbf{u},\mathbf{u}} = (-1)^{\mathbf{u}^{\mathrm{T}}\mathbf{k}}. \tag{7.34}$$

Therefore the effect of the bitwise XOR with a constant vector before (or after) a function h on its correlation matrix is a multiplication of some columns (or rows) by -1.

7.4.2 Linear Functions

Consider a linear function $h(\mathbf{a}) = \mathsf{M}\mathbf{a}$, with M an $m \times n$ binary matrix. We have

$$\mathbf{u}^{\mathrm{T}} h(\mathbf{a}) = \mathbf{u}^{\mathrm{T}}\mathsf{M}\mathbf{a} = (\mathsf{M}^{\mathrm{T}}\mathbf{u})^{\mathrm{T}}\mathbf{a}. \tag{7.35}$$

The elements of the corresponding correlation matrix are given by

$$\mathsf{C}_{\mathbf{u},\mathbf{w}} = \delta(\mathsf{M}^{\mathrm{T}}\mathbf{u} \oplus \mathbf{w}). \tag{7.36}$$

If M is an invertible matrix, the correlation matrix is a permutation matrix. The single non-zero element in row $\mathbf{u}$ is in column $\mathsf{M}^{\mathrm{T}}\mathbf{u}$. The effect of applying an invertible linear function before (or after) a function h on the correlation matrix is only a permutation of its columns (or rows).

7.4.3 Bricklayer Functions

Consider a bricklayer function $\mathbf{b} = h(\mathbf{a})$ that is defined by the following component functions:

$$\mathbf{b}_{(i)} = h_{(i)}(\mathbf{a}_{(i)})$$

for $1 \leq i \leq \ell$. For every component function $h_{(i)}$, there is a corresponding correlation matrix denoted by $\mathsf{C}^{(i)}$.

From the fact that the different component functions $h_{(i)}$ operate on non-overlapping sets of input bits and are therefore disjunct, (7.22) can be applied. The elements of the correlation matrix of h are given by

$$\mathsf{C}_{\mathbf{u},\mathbf{w}} = \prod_i \mathsf{C}^{(i)}_{\mathbf{u}_{(i)},\mathbf{w}_{(i)}}, \tag{7.37}$$

where

$$\mathbf{u} = (\mathbf{u}_{(1)}, \mathbf{u}_{(2)}, \ldots, \mathbf{u}_{(\ell)})$$

and

$$\mathbf{w} = (\mathbf{w}_{(1)}, \mathbf{w}_{(2)}, \ldots, \mathbf{w}_{(\ell)}).$$

In words, the correlation between an input parity and an output parity is the product of the correlations of the corresponding input and output parities of the component functions $\mathsf{C}^{(i)}_{\mathbf{u}_{(i)},\mathbf{w}_{(i)}}$.

7.5 Derived Properties

The concept of the correlation matrix is a valuable tool for demonstrating properties of Boolean functions and their spectra. We will illustrate this with some examples.

Lemma 7.5.1. *The elements of the correlation matrix of a Boolean function satisfy*

$$\mathsf{C}_{(\mathbf{u}\oplus\mathbf{v}),\mathbf{x}} = \sum_{\mathbf{w}} \mathsf{C}_{\mathbf{u},(\mathbf{w}\oplus\mathbf{x})}\mathsf{C}_{\mathbf{v},\mathbf{w}}, \tag{7.38}$$

for all $\mathbf{u}, \mathbf{v}, \mathbf{x} \in \mathrm{GF}(2)^n$.

Proof. Using the convolution property, we have

$$\begin{aligned} \mathcal{W}((\mathbf{u}\oplus\mathbf{v})^{\mathrm{T}}h(\mathbf{a})) &= \mathcal{W}(\mathbf{u}^{\mathrm{T}}h(\mathbf{a}) \oplus \mathbf{v}^{\mathrm{T}}h(\mathbf{a})) && (7.39)\\ &= \mathcal{W}(\mathbf{u}^{\mathrm{T}}h(\mathbf{a})) \otimes \mathcal{W}(\mathbf{v}^{\mathrm{T}}h(\mathbf{a})). && (7.40) \end{aligned}$$

Since the components of $\mathcal{W}(\mathbf{u}^{\mathrm{T}}h(\mathbf{a}))$ are given by $\mathsf{C}_{\mathbf{u},\mathbf{w}}$, the projection of (7.40) onto the component with index $\mathbf{x}$ gives rise to (7.38). □

From this lemma it follows:

Corollary 7.5.1. *The correlation between two output parities defined by* $\mathbf{u}$ *and* $\mathbf{v}$ *is equal to the convolution of columns* $\mathbf{u}$ *and* $\mathbf{v}$ *of the correlation matrix.*

$$\mathsf{C}_{(\mathbf{u}\oplus\mathbf{v}),0} = \sum_{\mathbf{w}} \mathsf{C}_{\mathbf{u},\mathbf{w}}\mathsf{C}_{\mathbf{v},\mathbf{w}}. \tag{7.41}$$

A binary Boolean function is *balanced* if it is 1 (or 0) for exactly half of the elements in the domain. Clearly, being balanced is equivalent to being uncorrelated to the binary Boolean function equal to 0 (or 1). Using the properties of correlation matrices we can now give an elegant proof of the following well-known theorem [1]:

Theorem 7.5.1. *A Boolean transformation is invertible iff every output parity is a balanced binary Boolean function of input bits.*

Proof.

$\Rightarrow$ If h is an invertible transformation, its correlation matrix C is orthogonal. Since $\mathsf{C}_{0,0} = 1$ and all rows and columns have norm 1, it follows that there are no other elements in row 0 or column 0 different from 0. Hence, $\mathrm{C}(\mathbf{u}^{\mathrm{T}}h(\mathbf{a}), 0) = \delta(\mathbf{u})$ or $\mathbf{u}^{\mathrm{T}}h(\mathbf{a})$ is balanced for all $\mathbf{u} \neq 0$.

$\Leftarrow$ The condition that all output parities are balanced binary Boolean functions of input bits corresponds to $\mathsf{C}_{\mathbf{u},0} = 0$ for $\mathbf{u} \neq 0$. If this is the case, we can show that the correlation matrix is orthogonal. The expression $\mathsf{C}^{\mathrm{T}} \times \mathsf{C} = \mathsf{I}$ is equivalent to the following set of conditions:

$$\sum_{\mathbf{w}} \mathsf{C}_{\mathbf{u},\mathbf{w}}\mathsf{C}_{\mathbf{v},\mathbf{w}} = \delta(\mathbf{u} \oplus \mathbf{v}) \text{ for all } \mathbf{u}, \mathbf{v} \in \mathrm{GF}(2)^n. \tag{7.42}$$

Using (7.41), we have

$$\sum_{\mathbf{w}} \mathsf{C}_{\mathbf{u},\mathbf{w}}\mathsf{C}_{\mathbf{v},\mathbf{w}} = \mathsf{C}_{(\mathbf{u}\oplus\mathbf{v}),0}. \tag{7.43}$$

Since $\mathsf{C}_{\mathbf{u},0} = 0$ for all $\mathbf{u} \neq 0$, and $\mathsf{C}_{0,0} = 1$, (7.42) holds for all possible pairs $\mathbf{u}, \mathbf{v}$. It follows that C is an orthogonal matrix, hence h^{-1} exists and is defined by C^{T}. □

Lemma 7.5.2. *The elements of the correlation matrix of a Boolean function with domain* $\mathrm{GF}(2)^n$ *and the spectrum values of a binary Boolean function with domain* $\mathrm{GF}(2)^n$ *are integer multiples of* 2^{1-n}.

Proof. The sum in the right-hand side of (7.8) is always even since its value is of the form $k \cdot (1) + (2^n - k) \cdot (-1) = 2k - 2^n$. It follows that the spectrum values must be integer multiples of 2^{1-n}. □

7.6 Truncating Functions

A function from $\mathrm{GF}(2)^n$ to $\mathrm{GF}(2)^m$ can be converted into a function from $\mathrm{GF}(2)^{n-1}$ to $\mathrm{GF}(2)^m$ by fixing a single bit of the input. More generally, a bit of the input can be set equal to a parity of other input components, possibly complemented. Such a restriction is of the type

$$\mathbf{v}^{\mathrm{T}}\mathbf{a} = \epsilon, \tag{7.44}$$

where $\epsilon \in \mathrm{GF}(2)$. Assume that $v_s \neq 0$.

The restriction can be modelled by a Boolean function $\mathbf{a}' = h^r(\mathbf{a})$ that is applied before h. It maps $\mathrm{GF}(2)^{n-1}$ to $\mathrm{GF}(2)^n$, and is specified by $a'_i = a_i$ for $i \neq s$ and $a'_s = \epsilon \oplus \mathbf{v}^{\mathrm{T}}\mathbf{a} \oplus a_s$. The non-zero elements of the correlation matrix of h^r are

$$\mathsf{C}^{(h^r)}_{\mathbf{w},\mathbf{w}} = 1 \text{ and } \mathsf{C}^{(h^r)}_{(\mathbf{v}\oplus\mathbf{w}),\mathbf{w}} = (-1)^{\epsilon} \text{ for all } \mathbf{w} \text{ where } \mathrm{w_s} = 0. \tag{7.45}$$

All columns of this matrix have exactly two non-zero entries with amplitude 1.

The function restricted to the specified subset of inputs is the consecutive application of h^r and the function itself. Hence, its correlation matrix C' is $\mathsf{C} \times \mathsf{C}^{(h^r)}$. The elements of this matrix are

$$\mathsf{C}'_{\mathbf{u},\mathbf{w}} = \mathsf{C}_{\mathbf{u},\mathbf{w}} + (-1)^{\epsilon}\mathsf{C}_{\mathbf{u},(\mathbf{w}\oplus\mathbf{v})} \tag{7.46}$$

if $\mathrm{w_s} = 0$, and 0 if $\mathrm{w_s} = 1$. The elements in C' are spectrum values of Boolean functions of $(n-1)$-dimensional vectors. Hence, from Lemma 7.5.2 they must be integer multiples of 2^{2-n}.

Applying (7.15) to the rows of the restricted correlation matrices gives additional laws for the spectrum values of Boolean functions. For the single restrictions of the type $\mathbf{v}^{\mathrm{T}}\mathbf{a} = \epsilon$ we have

$$\sum_{\mathbf{w}}(F(\mathbf{w}) + F(\mathbf{w} \oplus \mathbf{v}))^2 = \sum_{\mathbf{w}}(F(\mathbf{w}) - F(\mathbf{w} \oplus \mathbf{v}))^2 = 2. \tag{7.47}$$

Lemma 7.6.1. *The elements of a correlation matrix corresponding to an invertible transformation of n-bit vectors are integer multiples of 2^{2-n}.*

Proof. Let g be the Boolean function from $\mathrm{GF}(2)^{n-1}$ to $\mathrm{GF}(2)^m$ that is obtained by restricting the input of function h. Let the input restriction be specified by the vector $\mathbf{w}$: $\mathbf{w}^{\mathrm{T}}\mathbf{a} = 0$. Then $\mathsf{C}^{(g)}_{\mathbf{u},\mathbf{v}} = \mathsf{C}^{(h)}_{\mathbf{u},\mathbf{v}} + \mathsf{C}^{(h)}_{\mathbf{u},(\mathbf{v}\oplus\mathbf{w})}$ or $\mathsf{C}^{(g)}_{\mathbf{u},\mathbf{v}} = 0$. By filling in 0 for $\mathbf{v}$, this yields: $\mathsf{C}^{(g)}_{\mathbf{u},0} = \mathsf{C}^{(h)}_{\mathbf{u},0} + \mathsf{C}^{(h)}_{\mathbf{u},\mathbf{w}}$. Now, $\mathsf{C}^{(g)}_{\mathbf{u},0}$ must be an integer multiple of 2^{2-n}, and since according to Theorem 7.5.1 $\mathsf{C}^{(h)}_{\mathbf{u},0} = 0$, it follows that $\mathsf{C}^{(h)}_{\mathbf{u},\mathbf{w}}$ is also an integer multiple of 2^{2-n}. □

7.7 Cross-correlation and Autocorrelation

The cross-correlation function [67, p. 117] of two Boolean functions $f(\mathbf{a})$ and $g(\mathbf{a})$ is denoted by $\hat{c}_{fg}(\mathbf{b})$, and given by

$$\hat{c}_{fg}(\mathbf{b}) = \mathrm{C}(f(\mathbf{a}), g(\mathbf{a} \oplus \mathbf{b})) \tag{7.48}$$

$$= 2^{-n}\sum_{\mathbf{a}} \hat{f}(\mathbf{a})\hat{g}(\mathbf{a} \oplus \mathbf{b}) = 2^{-n}\sum_{\mathbf{a}}(-1)^{f(\mathbf{a})\oplus g(\mathbf{a}\oplus\mathbf{b})}. \tag{7.49}$$

Now consider FG, the product of the spectra of two binary Boolean functions f and g:

$$F(\mathbf{w})G(\mathbf{w}) = (2^{-n}\sum_{\mathbf{a}} \hat{f}(\mathbf{a})(-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}})(2^{-n}\sum_{\mathbf{b}} \hat{g}(\mathbf{b})(-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{b}}) \tag{7.50}$$

$$= 2^{-n}\sum_{\mathbf{a}}(2^{-n}\sum_{\mathbf{b}} \hat{f}(\mathbf{a})\hat{g}(\mathbf{b})(-1)^{\mathbf{w}^{\mathrm{T}}(\mathbf{a}\oplus\mathbf{b})}) \tag{7.51}$$

$$= 2^{-n}\sum_{\mathbf{a}}((2^{-n}\sum_{\mathbf{c}} \hat{f}(\mathbf{a})\hat{g}(\mathbf{a}\oplus\mathbf{c})(-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{c}})) \tag{7.52}$$

$$= \mathcal{W}((2^{-n}\sum_{\mathbf{c}} \hat{f}(\mathbf{a})\hat{g}(\mathbf{a}\oplus\mathbf{c})) \tag{7.53}$$

$$= \mathcal{W}(\hat{c}_{fg}(\mathbf{b})). \tag{7.54}$$

Hence the spectrum of the cross-correlation function of two binary Boolean functions equals the product of the spectra of the binary Boolean functions: $\hat{c}_{fg} = \mathcal{W}^{-1}(FG)$.

The cross-correlation function of a binary Boolean function with itself, $\hat{c}_{ff}$, is called the *autocorrelation function* of f and is denoted by $\hat{r}_f$. It follows that the components of the spectrum of the autocorrelation function are the squares of the components of the spectrum of f, i.e.

$$F^2 = \mathcal{W}(\hat{r}_f). \tag{7.55}$$

This equation is generally referred to as the Wiener-Khintchine theorem [79].

7.8 Linear Trails

Let β be an iterative Boolean transformation operating on n-bit vectors:

$$\beta = \rho^{(r)} \circ \rho^{(r-1)} \circ \ldots \circ \rho^{(2)} \circ \rho^{(1)}. \tag{7.56}$$

The correlation matrix of β is the product of the correlation matrices corresponding to the respective Boolean transformations:

$$\mathsf{C}^{(\beta)} = \mathsf{C}^{(\rho^{(r)})} \times \ldots \times \mathsf{C}^{(\rho^{(2)})} \times \mathsf{C}^{(\rho^{(1)})}. \tag{7.57}$$

A *linear trail* U over an iterative Boolean transformation consists of a sequence of $r+1$ selection patterns:

$$\mathsf{U} = \left(\mathbf{u}^{(0)}, \mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \ldots, \mathbf{u}^{(r-1)}, \mathbf{u}^{(r)}\right). \tag{7.58}$$

This linear trail is a sequence of r linear steps $\left(\mathbf{u}^{(i-1)}, \mathbf{u}^{(i)}\right)$ that have a correlation

$$\mathrm{C}\left(\mathbf{u}^{(i)\mathrm{T}} \rho^{(i)}(\mathbf{a}), \mathbf{u}^{(i-1)\mathrm{T}} \mathbf{a}\right).$$

The *correlation contribution* $\mathrm{C_p}$ of a linear trail is the product of the correlation of all its steps:

$$\mathrm{C_p}(\mathsf{U}) = \prod_i \mathrm{C}^{\rho^{(i)}}_{\mathbf{u}^{(i)}\mathbf{u}^{(i-1)}}. \tag{7.59}$$

As the correlations range between -1 and $+1$, so does the correlation contribution.

From this definition and (7.57), we can derive the following theorem:

Theorem 7.8.1 (Theorem of Linear Trail Composition). *The correlation between output parity* $\mathbf{u}^{\mathrm{T}}\beta(\mathbf{a})$ *and input parity* $\mathbf{w}^{\mathrm{T}}\mathbf{a}$ *of an iterated Boolean transformation with* r *rounds is the sum of the correlation contributions of all* r*-round linear trails* U *with initial selection pattern* $\mathbf{w}$ *and final selection pattern* $\mathbf{u}$*:*

$$\mathrm{C}(\mathbf{u}^{\mathrm{T}}\beta(\mathbf{a}), \mathbf{w}^{\mathrm{T}}\mathbf{a}) = \sum_{\mathbf{u}^{(0)}=\mathbf{w}, \mathbf{u}^{(r)}=\mathbf{u}} \mathrm{C_p}(U). \tag{7.60}$$

Both the correlation and the correlation contributions are signed. Some of the correlation contributions will have the same sign as the resulting correlation and contribute positively to its amplitude; the others contribute negatively to its amplitude. We speak of *constructive interference* in the case of two linear trails that have a correlation contribution with the same sign and of *destructive interference* if their correlation contributions have a different sign.

7.9 Ciphers

The described formalism and tools can be applied to the calculation of correlations in iterated block ciphers such as the DES and Rijndael.

7.9.1 General Case

In general, an iterative cipher consists of a sequence of keyed rounds, where each round $\rho^{(i)}$ depends on its round key $\mathbf{k}^{(i)}$. In a typical cryptanalytic setting, the round keys are fixed and we can model the cipher as an iterative Boolean transformation.

Linear cryptanalysis requires the knowledge of an output parity and an input parity that have a high correlation over all but a few rounds of the cipher. These correlations are the sum of the correlation contributions of all linear trails that connect the output parity with the input parity.

In general, the correlations over a round depend on the key value, and hence computing the correlation contribution of linear trails requires making assumptions about the round key values. However, in many cases the cipher structure allows the analysis of linear trails without having to make assumptions about the value of the round keys. In Sect. 7.9.2 we show that for a key-alternating cipher the amplitude of the correlation contribution is independent of the round key.

7.9.2 Key-Alternating Cipher

We have shown that the Boolean transformation corresponding to a key addition consisting of the XOR with a round key $\mathbf{k}^{(i)}$ has a correlation matrix with only non-zero elements on its diagonal. The element is -1 if $\mathbf{u}^{\mathrm{T}}\mathbf{k}^{(i)} = 1$, and 1 otherwise. If we denote the correlation matrix of ρ by C, the correlation contribution of the linear trail U then becomes:

$$\mathrm{C_p}(\mathsf{U}) = \prod_i (-1)^{\mathbf{u}^{(i)\mathrm{T}}\mathbf{k}^{(i)}} \mathsf{C}_{\mathbf{u}^{(i)},\mathbf{u}^{(i-1)}} \tag{7.61}$$

$$= (-1)^{d_\mathsf{U} \oplus \bigoplus_i \mathbf{u}^{(i)\mathrm{T}}\mathbf{k}^{(i)}} |\mathrm{C_p}(\mathsf{U})|, \tag{7.62}$$

where $d_\mathsf{U} = 1$ if $\prod_i \mathsf{C}_{\mathbf{u}^{(i)}\mathbf{u}^{(i-1)}}$ is negative, and $d_\mathsf{U} = 0$ otherwise. $|\mathrm{C_p}(\mathsf{U})|$ is independent of the round keys, and hence only the sign of the correlation contribution is key-dependent. The sign of the correlation contribution can be expressed as a parity of the expanded key K plus a key-independent constant:

$$s = \mathsf{U}^{\mathrm{T}}\mathsf{K} \oplus d_\mathsf{U}, \tag{7.63}$$

where K denotes the expanded key and U denotes the concatenation of the selection patterns $\mathbf{u}^{(i)}$.

The correlation between output parity $\mathbf{u}^{\mathrm{T}}\beta(\mathbf{a})$ and input parity $\mathbf{w}^{\mathrm{T}}\mathbf{a}$ expressed in terms of the correlation contributions of linear trails now becomes

$$\mathrm{C}(\mathbf{v}^{\mathrm{T}}\beta(\mathbf{a}), \mathbf{w}^{\mathrm{T}}\mathbf{a}) = \sum_{\mathbf{u}^{(0)}=\mathbf{w}, \mathbf{u}^{(r)}=\mathbf{v}} (-1)^{d_\mathsf{U} \oplus \mathsf{U}^{\mathrm{T}}\mathsf{K}} |\mathrm{C_p}(\mathsf{U})|. \tag{7.64}$$

Even though for a key-alternating cipher the amplitudes of the correlation contribution of the individual linear trails are independent of the round keys, this is not the case for the amplitude of the resulting correlation at the left-hand side of the equation. The terms in the right-hand side of the equation are added or subtracted depending on the value of the round keys. It depends on the value of the round keys whether interference between a pair of linear trails is constructive or destructive.

7.9.3 Averaging over all Round Keys

In Sect. 7.9.2 we have only discussed correlations for cases in which the value of the key is fixed. For key-alternating ciphers we can provide an expression for the expected value of correlation potentials, taken over all possible values of the expanded key (i.e. the concatenation of all round keys).

Assume that we are studying the correlation $\mathrm{C_t}$ between a particular input selection pattern and a particular output selection pattern, and that there are n linear trails U_i connecting them. In the case of a key-alternating cipher, the correlation contribution amplitudes $|\mathrm{C_p}(\mathsf{U}_i)|$ of these trails are independent of the key. Let us now express the correlation contribution of trail U_i as

$$\mathrm{C_p}(\mathsf{U}_i) = (-1)^{s_i} C_i,$$

where C_i is the amplitude of the correlation contribution and s_i is a bit determining the sign. The sign for a trail U_i is equal to a parity of the expanded key plus a trail-specific bit: $s_i = \mathsf{U}_i{}^{\mathrm{T}}\mathsf{K} \oplus d_{\mathsf{U}_i}$. The expected value of the correlation potential is given by

$$\mathrm{E}(\mathrm{C_t}^2) = 2^{-n_\mathsf{K}} \sum_{\mathsf{K}} \Big(\sum_i (-1)^{s_i} C_i\Big)^2 \tag{7.65}$$

$$= 2^{-n_\mathsf{K}} \sum_{\mathsf{K}} \Big(\sum_i (-1)^{\mathsf{U}_i{}^{\mathrm{T}}\mathsf{K} \oplus d_{\mathsf{U}_i}} C_i\Big)^2. \tag{7.66}$$

We can now prove the following theorem:

Theorem 7.9.1. *The average correlation potential between an input and an output selection pattern is the sum of the correlation potentials of all linear trails between the input and output selection patterns:*

$$\mathrm{E}(\mathrm{C_t}^2) = \sum_i {C_i}^2. \tag{7.67}$$

Proof.

$$\begin{aligned}
\mathrm{E}(\mathrm{C_t}^2) &= 2^{-n_\mathsf{K}} \sum_{\mathsf{K}} \Big(\sum_i (-1)^{\mathsf{U}_i{}^{\mathrm{T}}\mathsf{K} \oplus d_{\mathsf{U}_i}} C_i\Big)^2 \\
&= 2^{-n_\mathsf{K}} \sum_{\mathsf{K}} \Big(\sum_i (-1)^{\mathsf{U}_i{}^{\mathrm{T}}\mathsf{K} \oplus d_{\mathsf{U}_i}} C_i\Big)\Big(\sum_j (-1)^{\mathsf{U}_j{}^{\mathrm{T}}\mathsf{K} \oplus d_{\mathsf{U}_j}} C_j\Big) \\
&= 2^{-n_\mathsf{K}} \sum_{K} \sum_i \sum_j \Big((-1)^{\mathsf{U}_i{}^{\mathrm{T}}\mathsf{K} \oplus d_{\mathsf{U}_i}} C_i\Big)\Big((-1)^{\mathsf{U}_j{}^{\mathrm{T}}\mathsf{K} \oplus d_{\mathsf{U}_j}} C_j\Big) \\
&= 2^{-n_\mathsf{K}} \sum_{\mathsf{K}} \sum_i \sum_j (-1)^{(\mathsf{U}_i \oplus \mathsf{U}_j)^{\mathrm{T}}\mathsf{K} \oplus d_{\mathsf{U}_i} \oplus d_{\mathsf{U}_j}} C_i C_j \\
&= 2^{-n_\mathsf{K}} \sum_i \sum_j \Big(\sum_{\mathsf{K}} (-1)^{(\mathsf{U}_i \oplus \mathsf{U}_j)^{\mathrm{T}}\mathsf{K} \oplus d_{\mathsf{U}_i} \oplus d_{\mathsf{U}_j}}\Big) C_i C_j.
\end{aligned} \tag{7.68}$$

For the factor of C_iC_j in (7.68), we have:

$$\sum_{\mathsf{K}}(-1)^{(\mathsf{U}_i\oplus\mathsf{U}_j)^{\mathrm{T}}\mathsf{K}\oplus d_{\mathsf{U}_i}\oplus d_{\mathsf{U}_j}} = 2^{n_\mathsf{K}}\delta(i\oplus j). \tag{7.69}$$

Clearly, the expression is equal to 0 if $\mathsf{U}_i \oplus \mathsf{U}_j \neq 0$: if we sum over all values of K, the exponent of (-1) is 1 for half the terms and 0 for the other half. For any pair $i \neq j$, U_i and U_j are also different as we sum over all different linear trails. On the other hand, if $i = j$ the exponent of (-1) becomes 0 and the expression is equal to 2^{n_K}. Substitution in (7.68) yields

$$\begin{aligned}\mathrm{E}(\mathrm{C_t}^2) &= 2^{-n_\mathsf{K}}\sum_i\sum_j 2^{n_\mathsf{K}}\delta(i\oplus j)C_iC_j\\ &= \sum_i {C_i}^2,\end{aligned}$$

proving our theorem. □

7.9.4 The Effect of the Key Schedule

In the previous section we have taken the average of the correlation potentials over all possible values of the expanded key, implying independent round keys. In practice, the values of the round keys are restricted by the key schedule that computes the round keys from the cipher key. In this section we investigate the effect that the key schedule has on expected values of correlation potentials.

First assume that we have a linear or affine key schedule. For the sake of simplicity, we limit ourselves to the linear case, but the conclusions are also valid for the affine case. If the key schedule is linear, the relation between the expanded key K and the cipher key $\mathbf{k}$ can be expressed as the multiplication with a binary matrix:

$$\mathsf{K} = \mathsf{M}_\kappa\mathbf{k}. \tag{7.70}$$

If we substitute this in (7.66), we obtain

$$\mathrm{E}(\mathrm{C_t}^2) = 2^{-n_\mathrm{k}}\sum_{\mathbf{k}}(\sum_i(-1)^{{\mathsf{U}_i}^{\mathrm{T}}\mathsf{M}_\kappa\mathbf{k}\oplus d_{\mathsf{U}_i}}C_i)^2. \tag{7.71}$$

Working out the squares in this equation yields:

$$\mathrm{E}(\mathrm{C_t}^2) = 2^{-n_\mathrm{k}}\sum_i\sum_j(\sum_{\mathbf{k}}(-1)^{(\mathsf{U}_i\oplus\mathsf{U}_j)^{\mathrm{T}}\mathsf{M}_\kappa k\oplus d_{\mathsf{U}_i}\oplus d_{\mathsf{U}_j}})C_iC_j. \tag{7.72}$$

For the factor of C_iC_j in (7.68), we have

$$\sum_{\mathbf{k}}(-1)^{\mathsf{U}_i\oplus\mathsf{U}_j{}^{\mathrm{T}}\mathsf{M}_\kappa\mathbf{k}\oplus d_{\mathsf{U}_i}\oplus d_{\mathsf{U}_j}} = (-1)^{(d_{\mathsf{U}_i}\oplus d_{\mathsf{U}_j})}2^{n_\mathbf{k}}\delta(\mathsf{M}_\kappa{}^{\mathrm{T}}(\mathsf{U}_i\oplus\mathsf{U}_j)). \tag{7.73}$$

As above, the expression is equal to 0 if $(\mathsf{U}_i\oplus\mathsf{U}_j)^{\mathrm{T}}\mathsf{M}_\kappa \neq 0$: if we sum over all values of $\mathbf{k}$, the exponent of (-1) is 1 for half the terms and 0 in the other half. However, if $(\mathsf{U}_i\oplus\mathsf{U}_j)^{\mathrm{T}}\mathsf{M}_\kappa = 0$, all terms have the same sign: $(-1)^{(d_{\mathsf{U}_i}\oplus d_{\mathsf{U}_j})}$. The condition $\mathsf{M}_\kappa{}^{\mathrm{T}}(\mathsf{U}_i\oplus\mathsf{U}_j) = 0$ is equivalent to saying that the bitwise difference of the two trails is mapped to 0 by M_κ, or equivalently, that the two linear trails depend on the same parities of the cipher key for the sign of their correlation contribution. Let us call such a pair of trails a colliding pair. The effect of a colliding pair on the expression of the expected correlation potential in terms of the correlation potentials of the trails is the following. Next to the terms ${C_i}^2$ and ${C_j}^2$ there is twice the term $(-1)^{(d_{\mathsf{U}_i}\oplus d_{\mathsf{U}_j})}C_iC_j$. These four terms can be combined into the expression $(C_i+(-1)^{(d_{\mathsf{U}_i}\oplus d_{\mathsf{U}_j})}C_j)^2$. The effect of a colliding pair is that their correlation contributions cannot be considered to be independent in the computation of the expected correlation potential. They systematically interfere positively if $d_{\mathsf{U}_i} = d_{\mathsf{U}_j}$, and negatively otherwise. Their contribution to the correlation potential is $(C_i+(-1)^{(d_{\mathsf{U}_i}+d_{\mathsf{U}_j})}C_j)^2$.

Example 7.9.1. We illustrate the above reasoning on an example in which the key schedule has a dramatic impact. Consider a block cipher $B2[\mathbf{k}^{(1)},\mathbf{k}^{(2)}](\mathbf{x})$ consisting of two rounds. The first round is encryption with a block cipher B with round key $\mathbf{k}^{(1)}$, and the second round is decryption with that same block cipher B with round key $\mathbf{k}^{(2)}$. Assume we have taken a state-of-the-art block cipher B. In that case, the expected correlation potentials between any pair of input and output selection patterns of $B2$ are the sum of the correlation potentials of many linear trails over the composition of B and B^{-1}. The expected value of correlation potentials corresponding to any pair of input and output selection patterns is of the order 2^{-n_b}.

Let us now consider a block cipher $C[\mathbf{k}]$ defined by

$$C[\mathbf{k}](\mathbf{x}) = B2[\mathbf{k},\mathbf{k}](\mathbf{x}).$$

Setting $\mathbf{k}^{(1)} = \mathbf{k}^{(2)} = \mathbf{k}$ can be seen as a very simple key schedule. Clearly, the block cipher $C[\mathbf{k}](\mathbf{x})$ is the identity map, and hence we know that it has correlation potentials equal to 1 if the input and output selection pattern are the same, and 0 otherwise. This is the consequence of the fact that in this particular example, the key schedule is such that the round keys can no longer be considered as being independent. We have:

$$\mathsf{C}^{(B2[\mathbf{k}^{(1)},\mathbf{k}^{(2)}])}_{\mathbf{u},\mathbf{w}} = \sum_{\mathbf{v}}\mathsf{C}^{(B[\mathbf{k}^{(1)}])}_{\mathbf{u},\mathbf{v}}\mathsf{C}^{(B^{-1}[\mathbf{k}^{(2)}])}_{\mathbf{v},\mathbf{w}} = \sum_{\mathbf{v}}\mathsf{C}^{(B[\mathbf{k}^{(1)}])}_{\mathbf{u},\mathbf{v}}\mathsf{C}^{(B[\mathbf{k}^{(2)}])}_{\mathbf{w},\mathbf{v}}. \tag{7.74}$$

If $\mathbf{k}^{(1)} = \mathbf{k}^{(2)} = \mathbf{k}$, this is reduced to

$$\mathsf{C}^{(C[\mathbf{k}])}_{\mathbf{u},\mathbf{w}} = \sum_{\mathbf{v}} \mathsf{C}^{(B[\mathbf{k}])}_{\mathbf{u},\mathbf{v}} \mathsf{C}^{(B[\mathbf{k}])}_{\mathbf{w},\mathbf{v}} = \delta(\mathbf{u} \oplus \mathbf{w}). \tag{7.75}$$

This follows from the fact that for any given value of $\mathbf{k}$, $\mathsf{C}^{(B[\mathbf{k}])}$ is an orthogonal matrix.

As opposed to this extreme example, pairs of linear trails that always interfere constructively or destructively due to the linear key schedule are very rare. The condition is that the two trails depend on the same parity of cipher key bits for their sign. The probability that this is the case for two trails is 2^{-n_k}. If the key schedule is non-linear, linear trails that always interfere constructively or destructively due to the key schedule can even not occur. Instead of $\mathsf{K} = \mathsf{M}_\kappa \mathbf{k}$ we have $\mathsf{K} = f_\kappa(\mathbf{k})$ with f_κ a non-linear function. The coefficient of the mixed terms are of the form

$$\sum_k (-1)^{\mathsf{U}_i \oplus \mathsf{U}_j{}^\mathrm{T} f_\kappa(\mathbf{k}) \oplus d_{\mathsf{U}_i} \oplus d_{\mathsf{U}_j}}. \tag{7.76}$$

It seems hard to come up with a reasonable key schedule for which this expression does not have approximately as many positive as negative terms. If that is the case, the sum of the correlation potentials of the linear trails are a very good approximation of the expected correlation potentials. Still, taking a non-linear key schedule to avoid systematic constructive interference seems unnecessary in the light of the rarity of the phenomenon.

7.10 Correlation Matrices and Linear Cryptanalysis Literature

In this section we make an attempt to position our approach with respect to the formalism and terminology that are mostly used on the subject of linear cryptanalysis in the cryptographic literature.

7.10.1 Linear Cryptanalysis of the DES

For an overview of the original linear cryptanalysis attack on the DES we refer to Sect. 6.3. The multiple-round linear expressions described in [65] correspond to what we call linear trails. The probability p that such an expression holds corresponds to $\frac{1}{2}(1 + \mathrm{C_p}(\mathsf{U}))$, where $\mathrm{C_p}(\mathsf{U})$ is the correlation contribution of the corresponding linear trail. The usage of probabilities in [65] requires the application of the so-called piling-up lemma in the computation of probabilities of composed transformations. When working with

correlations, no such tricks are required: correlations can be simply multiplied.

In [65] the correlation over multiple rounds is approximated by the correlation contribution of a single linear trail. The silent assumption underlying this approximation, that the correlation is dominated by a single linear trail, seems valid because of the large relative amplitude of the described correlation. There are no linear trails with the same initial and final selection patterns that have a correlation contribution that comes close to the dominant trail.

The amplitude of the correlation of the linear trail is independent of the value of the key, and consists of the product of the correlations of its steps. In general, the elements of the correlation matrix of the DES round function are not independent of the round keys, due to the fact that the inputs of neighbouring S-boxes overlap while depending on different key bits. However, in the described linear trails the actual independence is caused by the fact that the steps of the described linear trail only involve bits of a single S-box.

The input-output correlations of the F-function of the DES can be calculated by applying the rules given in Sect. 7.4. The 32-bit selection pattern $\mathbf{b}$ at the output of the bit permutation P is converted into a 32-bit selection pattern $\mathbf{c}$ at the output of the S-boxes by a simple linear function. The 32-bit selection pattern $\mathbf{a}$ at the input of the (linear) expansion E gives rise to a set α of $2^{2\ell}$ 48-bit selection patterns after the expansion, where ℓ is the number of pairwise neighbouring S-box pairs that are addressed by $\mathbf{a}$.

On the assumption that the round key is all-zero, the correlation between $\mathbf{c}$ and $\mathbf{a}$ can now be calculated by simply adding the correlations corresponding to $\mathbf{c}$ and all vectors in α. Since the S-boxes form a bricklayer function, these correlations can be calculated from the correlation matrices of the individual S-boxes. For $\ell > 0$ the calculations can be greatly simplified by recursively reusing intermediate results in computing these correlations. The total number of calculations can be reduced to less than 16ℓ multiplications and additions of S-box correlations.

The effect of a non-zero round key is the multiplication of some of these correlations by -1. Hence, if $\ell > 0$ the correlation depends on the value of 2ℓ different linear combinations of round key bits. If $\ell = 0$, α only contains a single vector and the correlation is independent of the key.

7.10.2 Linear Hulls

A theorem similar to Theorem 7.9.1 has been proved by K. Nyberg in [75], in her treatment of so-called *linear hulls*. The difference is the following. Theorem 7.9.1 expresses the expected correlation potential between an input selection pattern and an output selection pattern, averaged over all values of

the expanded keys, as the sum of the correlation potentials ${C_i}^2$ of the individual trails between these selection patterns. It is valid for key-alternating ciphers. However, the theorem in [75] is proven for DES-like ciphers and does not mention key-alternating ciphers. As the DES is not a key-alternating cipher, the correlation potential of a linear trail is in general not independent of the expanded key. In Theorem 7.9.1, the correlation potentials ${C_i}^2$ of the linear trails must be replaced by the expected correlation potentials of the trails $\mathrm{E}({C_i}^2)$, i.e., averaged over all values of the expanded key. In [75] the set of linear trails connecting the same initial selection pattern and final selection patterns is called an *approximate linear hull.*

Unfortunately, the presentation in [75] does not consider the effect of the key schedule and only considers the case of independent round keys. This is often misunderstood by cipher designers as an incentive to design heavy key schedules, in order to make the relations between round keys very complicated, or 'very random'. As we have shown above, linear cryptanalysis does not suggest complicated key schedules as even in the case of a linear key schedule systematic constructive interference of linear trails is very rare.

Extrapolating Theorem 7.9.1 to ciphers that are not key-alternating can be very misleading. First of all, in actual cryptanalysis it is not so much the maximum average correlation potential that is relevant but the maximum correlation potential corresponding to the given key under attack. We illustrate this with an example.

Example 7.10.1. We consider a cipher B that consists of the multiplication with an invertible binary matrix, where the matrix is the key:

$$B[\mathsf{K}](\mathbf{x}) = \mathsf{K}\mathbf{x}.$$

For each given key K, each input parity has a correlation of amplitude 1 with exactly one output parity and no correlation with all other output parities. Averaged over all possible keys K (i.e. invertible matrices), the expected correlation potential between any pair of input-output parities as predicted by Theorem 7.9.1 is exactly 2^{-n_b}. Unfortunately, despite this excellent property with respect to average correlation amplitudes, the cipher is linear and trivially breakable.

The following physical metaphor summarizes the problem with the extrapolation. At any given time, on average half of the world's populations is asleep. This does not mean that everyone is only half awake all the time.

Even for key-alternating ciphers one must take care in interpreting expected values of the correlation potential. For example, take the case of a large correlation that is the result of one dominant trail with correlation C_1. The expected correlation potential is ${C_1}^2$ and the required number of plaintext-ciphertext pairs for a given success rate of the linear attack is proportional to ${C_1}^{-2}$. Now let us see what happens when another linear trail

is discovered with a correlation C_2 of the same order of magnitude. The expected correlation potential now becomes ${C_1}^2 + {C_2}^2$. One would expect that the required number of plaintext-ciphertext pairs for a given success rate would diminish. In fact, the required number of plaintext-ciphertext pairs for a given success rate is $(C_1 + C_2)^{-2}$ for half of the keys and $(C_1 - C_2)^{-2}$ for the other half of the keys. Hence, although the additional trail makes the expected correlation potential go up, the required number of plaintext-ciphertext pairs for a given success rate increases in half of the cases. Even if the average is taken over the two cases, the expected required number of plaintext-ciphertext pairs increases. It follows that expected correlation potentials should be interpreted with caution. We think expected correlation potentials may have some relevance in the case where they result from many linear trails with correlation potentials that have the same order of magnitude. Still, we think that in design one should focus on worst-case behaviour and consider the possibility of constructive interference of (signed) correlation contributions (see Sect. 9.1.1).

7.11 Conclusions

In this chapter we have provided a number of tools for describing and investigating the correlations in Boolean functions, iterated Boolean functions and block ciphers. This includes the concept of the correlation matrix and its properties and the systematic treatment of the silent assumptions made in linear cryptanalysis. We have compared our approach with the formalism usually adopted in cryptographic literature, and have argued why it is an improvement. An extension of our approach to functions and block ciphers operating on arrays of elements of $\mathrm{GF}(2^n)$ is presented in Appendix A.

8. Difference Propagation

In this chapter we consider difference propagation in Boolean functions. Difference propagation plays an important role in cryptanalysis in general and in differential cryptanalysis in particular.

We describe how differences propagate through several types of Boolean functions. We show that the difference propagation probabilities and the correlation potentials of a Boolean function are related by a simple expression. This is followed by a treatment of difference propagation through iterated Boolean transformations and in key-alternating ciphers. Finally we apply our analysis to the differential cryptanalysis of the DES and compare it with the influential concept of Markov ciphers.

8.1 Difference Propagation

Consider a couple of n-bit vectors $\mathbf{a}$ and $\mathbf{a}^*$ with bitwise difference $\mathbf{a} \oplus \mathbf{a}^* = \mathbf{a}'$. Let $\mathbf{b} = h(\mathbf{a}), \mathbf{b}^* = h(\mathbf{a}^*)$ and $\mathbf{b}' = \mathbf{b} \oplus \mathbf{b}^*$. The difference $\mathbf{a}'$ propagates to the difference $\mathbf{b}'$ through h. In general, $\mathbf{b}'$ is not fully determined by $\mathbf{a}'$ but depends on the value of $\mathbf{a}$ (or $\mathbf{a}^*$).

Definition 8.1.1. *A* difference propagation probability $\mathrm{Prob}^h(\mathbf{a}', \mathbf{b}')$ *is defined as*

$$\mathrm{Prob}^h(\mathbf{a}', \mathbf{b}') = 2^{-n} \sum_{\mathbf{a}} \delta(\mathbf{b}' \oplus h(\mathbf{a} \oplus \mathbf{a}') \oplus h(\mathbf{a})). \tag{8.1}$$

For a pair chosen uniformly from the set of all pairs $(\mathbf{a}, \mathbf{a}^*)$ where $\mathbf{a} \oplus \mathbf{a}^* = \mathbf{a}'$, $\mathrm{Prob}^h(\mathbf{a}', \mathbf{b}')$ is the probability that $h(\mathbf{a}) \oplus h(\mathbf{a}^*) = \mathbf{b}'$. Difference propagation probabilities range between 0 and 1. Since

$$h(\mathbf{a} \oplus \mathbf{a}') \oplus h(\mathbf{a}) = h(\mathbf{a}) \oplus h(\mathbf{a} \oplus \mathbf{a}'), \tag{8.2}$$

their value must be an integer multiple of 2^{1-n}. We have:

$$\sum_{\mathbf{b}'} \mathrm{Prob}^h(\mathbf{a}', \mathbf{b}') = 1. \tag{8.3}$$

The difference propagation from $\mathbf{a}'$ to $\mathbf{b}'$ occurs for a fraction of all possible input values $\mathbf{a}$ (and $\mathbf{a}^*$). This fraction is $\text{Prob}^h(\mathbf{a}', \mathbf{b}')$. If $\text{Prob}^h(\mathbf{a}', \mathbf{b}') = 0$, we say that the input difference $\mathbf{a}'$ and the output difference $\mathbf{b}'$ are *incompatible* through h.

Definition 8.1.2. *The weight of a difference propagation* $(\mathbf{a}', \mathbf{b}')$ *is the negative of the binary logarithm of the difference propagation probability, i.e.,*

$$\text{w}_\text{r}(\mathbf{a}', \mathbf{b}') = -\log_2 \text{Prob}^h(\mathbf{a}', \mathbf{b}'). \tag{8.4}$$

The weight corresponds to the amount of information (expressed in bits) that the difference propagation gives about $\mathbf{a}$. Equivalently, it is the loss in *entropy* [85] of $\mathbf{a}$ due to the restriction that $\mathbf{a}'$ propagates to $\mathbf{b}'$. The weight ranges between 0 and $n-1$: in the worst case the difference propagation gives no information on $\mathbf{a}$, and in the best case it leaves only one bit of uncertainty on $\mathbf{a}$ and $\mathbf{a}^*$.

If h is a linear function, a difference pattern at the input completely determines the difference pattern at the output:

$$\mathbf{b}' = \mathbf{b} \oplus \mathbf{b}^* = h(\mathbf{a}) \oplus h(\mathbf{a}^*) = h(\mathbf{a} \oplus \mathbf{a}^*) = h(\mathbf{a}'). \tag{8.5}$$

From $\text{w}_\text{r}(\mathbf{a}', \mathbf{b}') = 0$ it follows that this difference propagation does not give any information on $\mathbf{a}$.

8.2 Special Functions

8.2.1 Affine Functions

An *affine* function h from $\text{GF}(2)^n$ to $\text{GF}(2)^m$ is specified by

$$\mathbf{b} = \mathsf{M}\mathbf{a} \oplus \mathbf{k}, \tag{8.6}$$

where M is a $m \times n$ matrix and $\mathbf{k}$ is an m-dimensional vector. The difference propagation for this function is determined by

$$\mathbf{b}' = \mathsf{M}\mathbf{a}'. \tag{8.7}$$

8.2.2 Bricklayer Functions

For a bricklayer function h, the difference propagation probability is the product of the difference propagation probabilities of the component functions:

$$\text{Prob}^h(\mathbf{a}', \mathbf{b}') = \prod_i \text{Prob}^{h_{(i)}}(\mathbf{a}'_{(i)}, \mathbf{b}'_{(i)}). \tag{8.8}$$

The weight is the sum of the weights of the difference propagation in the component functions:

$$\mathrm{w_r}(\mathbf{a}', \mathbf{b}') = \sum_i \mathrm{w_r}(\mathbf{a}'_{(i)}, \mathbf{b}'_{(i)}), \tag{8.9}$$

where $\mathbf{a}' = (\mathbf{a}'_{(1)}, \mathbf{a}'_{(2)}, \ldots, \mathbf{a}'_{(\ell)})$ and $\mathbf{b}' = (\mathbf{b}'_{(1)}, \mathbf{b}'_{(2)}, \ldots, \mathbf{b}'_{(\ell)})$.

8.2.3 Truncating Functions

A Boolean function h from $\mathrm{GF}(2)^n$ to $\mathrm{GF}(2)^m$ can be converted into a Boolean function h_s from $\mathrm{GF}(2)^n$ to $\mathrm{GF}(2)^{m-1}$ by discarding a single output bit a_s. The difference propagation probabilities of h_s can be expressed in terms of the difference propagation probabilities of h:

$$\mathrm{Prob}^{h_s}(\mathbf{a}', \mathbf{b}') = \mathrm{Prob}^{h}(\mathbf{a}', \omega^0) + \mathrm{Prob}^{h}(\mathbf{a}', \omega^1), \tag{8.10}$$

where $\mathbf{b}'_i = \omega_i^0 = \omega_i^1$ for $i \neq s$ and $\omega_s^1 = 1$ and $\omega_s^0 = 0$. We generalise this to the situation in which only a number of linear combinations of the output are considered. Let λ be a linear function corresponding to an $m \times \ell$ binary matrix M. The difference propagation probabilities of $\theta \circ h$ are given by

$$\mathrm{Prob}^{\lambda \circ h}(\mathbf{a}', \mathbf{b}') = \sum_{\omega | \mathbf{b}' = \mathsf{M}\omega} \mathrm{Prob}^{h}(\mathbf{a}', \omega). \tag{8.11}$$

8.3 Difference Propagation Probabilities and Correlation

The difference propagation probabilities of Boolean functions can be expressed in terms of their spectrum and their correlation matrix elements. The probability of difference propagation $\mathrm{Prob}^{f}(\mathbf{a}', 0)$ is given by

$$\begin{aligned}
\mathrm{Prob}^{f}(\mathbf{a}', 0) &= 2^{-n} \sum_{\mathbf{a}} \delta(f(\mathbf{a}) \oplus f(\mathbf{a} \oplus \mathbf{a}')) \\
&= 2^{-n} \sum_{\mathbf{a}} \frac{1}{2}(1 + \hat{f}(\mathbf{a})\hat{f}(\mathbf{a} \oplus \mathbf{a}')) \\
&= 2^{-n} \sum_{\mathbf{a}} \frac{1}{2} + 2^{-n} \sum_{\mathbf{a}} \frac{1}{2}\hat{f}(\mathbf{a})\hat{f}(\mathbf{a} \oplus \mathbf{a}') \\
&= \frac{1}{2}(1 + \hat{r}_f(\mathbf{a}')) \\
&= \frac{1}{2}(1 + \sum_{\mathbf{w}} (-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}'} \hat{F}^2(\mathbf{w})).
\end{aligned} \tag{8.12}$$

The component of the autocorrelation function $\hat{r}_f(\mathbf{a}')$ corresponds to the amount that $\text{Prob}^f(\mathbf{a}', 0)$ deviates from 0.5.

For functions from $\text{GF}(2)^n$ to $\text{GF}(2)^m$, we denote the autocorrelation function of $\mathbf{u}^{\mathrm{T}}h(\mathbf{a})$ by $\hat{r}_{\mathbf{u}}(\mathbf{a}')$, i.e.,

$$\hat{r}_{\mathbf{u}}(\mathbf{a}') = 2^{-n}\sum_{\mathbf{a}}(-1)^{\mathbf{u}^{\mathrm{T}}h(\mathbf{a})\oplus\mathbf{u}^{\mathrm{T}}h(\mathbf{a}\oplus\mathbf{a}')}. \tag{8.13}$$

Now we can prove the following theorem that expresses the duality between the difference propagation and the correlation properties of a Boolean function.

Theorem 8.3.1. *The difference propagation probability table and correlation potential table of a Boolean function are linked by a (scaled) Walsh-Hadamard transform. We have*

$$\text{Prob}(\mathbf{a}', \mathbf{b}') = 2^{-m}\sum_{\mathbf{u},\mathbf{w}}(-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}'\oplus\mathbf{u}^{\mathrm{T}}\mathbf{b}'}C^{\mathbf{u},\mathbf{w}2}, \tag{8.14}$$

and dually

$$C_{\mathbf{u},\mathbf{w}}{}^2 = 2^{-n}\sum_{\mathbf{a}',\mathbf{b}'}(-1)^{w^{\mathrm{T}}\mathbf{a}'\oplus u^{\mathrm{T}}\mathbf{b}'}\text{Prob}(\mathbf{a}', \mathbf{b}'). \tag{8.15}$$

Proof.

$$\begin{aligned}
\text{Prob}(\mathbf{a}', \mathbf{b}') &= 2^{-n}\sum_{\mathbf{a}}\delta(h(\mathbf{a}) \oplus h(\mathbf{a}\oplus\mathbf{a}') \oplus \mathbf{b}') \\
&= 2^{-n}\sum_{\mathbf{a}}\prod_i \frac{1}{2}((-1)^{h_i(\mathbf{a})\oplus h_i(\mathbf{a}\oplus\mathbf{a}')\oplus\mathbf{b}'_i} + 1) \\
&= 2^{-n}\sum_{\mathbf{a}}2^{-m}\sum_{\mathbf{u}}(\prod_{u_i=1}(-1)^{h_i(\mathbf{a})\oplus h_i(\mathbf{a}\oplus\mathbf{a}')\oplus\mathbf{b}'_i}) \\
&= 2^{-n}\sum_{\mathbf{a}}2^{-m}\sum_{\mathbf{u}}(-1)^{\mathbf{u}^{\mathrm{T}}(h(\mathbf{a})\oplus h(\mathbf{a}\oplus\mathbf{a}')\oplus\mathbf{b}')} \\
&= 2^{-n}\sum_{\mathbf{a}}2^{-m}\sum_{\mathbf{u}}(-1)^{\mathbf{u}^{\mathrm{T}}h(\mathbf{a})\oplus\mathbf{u}^{\mathrm{T}}h(\mathbf{a}\oplus\mathbf{a}')\oplus\mathbf{u}^{\mathrm{T}}\mathbf{b}'} \\
&= 2^{-m}\sum_{\mathbf{u}}(-1)^{\mathbf{u}^{\mathrm{T}}\mathbf{b}'}2^{-n}\sum_{\mathbf{a}}(-1)^{\mathbf{u}^{\mathrm{T}}h(\mathbf{a})\oplus\mathbf{u}^{\mathrm{T}}h(\mathbf{a}\oplus\mathbf{a}')} \\
&= 2^{-m}\sum_{\mathbf{u}}(-1)^{\mathbf{u}^{\mathrm{T}}\mathbf{b}'}\hat{r}_{\mathbf{u}}(\mathbf{a}') \\
&= 2^{-m}\sum_{\mathbf{u}}(-1)^{\mathbf{u}^{\mathrm{T}}\mathbf{b}'}\sum_{\mathbf{w}}(-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}'}C_{\mathbf{u},\mathbf{w}}{}^2 \\
&= 2^{-m}\sum_{\mathbf{u},\mathbf{w}}(-1)^{\mathbf{w}^{\mathrm{T}}\mathbf{a}'\oplus\mathbf{u}^{\mathrm{T}}\mathbf{b}'}C_{\mathbf{u},\mathbf{w}}{}^2.
\end{aligned}$$

□

8.4 Differential Trails

In this section we apply the described formalism and tools to the propagation of differences in iterative Boolean transformations.

8.4.1 General Case

Let β be an iterative Boolean transformation operating on n-bit vectors that is a sequence of r transformations:

$$\beta = \rho^{(r)} \circ \rho^{(r-1)} \circ \ldots \circ \rho^{(2)} \circ \rho^{(1)}. \tag{8.16}$$

A *differential trail* Q over an iterative transformation consists of a sequence of $r+1$ difference patterns:

$$\mathsf{Q} = (\mathbf{q}^{(0)}, \mathbf{q}^{(1)}, \mathbf{q}^{(2)}, \ldots, \mathbf{q}^{(r-1)}, \mathbf{q}^{(r)}). \tag{8.17}$$

A differential trail has a *probability*. The probability of a differential trail is the number of values $\mathbf{a}^{(0)}$ for which the difference patterns follow the differential trail divided by the number of possible values for $\mathbf{a}^{(0)}$. This differential trail is a sequence of r differential steps $\left(\mathbf{q}^{(i-1)}, \mathbf{q}^{(i)}\right)$, which have a weight:

$$\mathrm{w_r}^{\rho^{(i)}}\left(\mathbf{q}^{(i-1)}, \mathbf{q}^{(i)}\right), \tag{8.18}$$

or $\mathrm{w_r}^{(i)}$ for short.

Definition 8.4.1. *The weight of a differential trail* Q *is the sum of the weights of its differential steps, i.e.*

$$\mathrm{w_r}(\mathsf{Q}) = \sum_i \mathrm{w_r}^{\rho^{(i)}}(\mathbf{q}^{(i-1)}, \mathbf{q}^{(i)}). \tag{8.19}$$

The significance of the weight of a differential trail is explained in the following section.

8.4.2 Independence of Restrictions

A differential step $\left(\mathbf{q}^{(i-1)}, \mathbf{q}^{(i)}\right)$ imposes restrictions on the intermediate state $\mathbf{a}^{(i-1)}$ in the following way. The differential step imposes that the value of $\mathbf{a}^{(i-1)}$ is in a set that contains a fraction $2^{-\mathrm{w_r}^{(i)}}$ of all possible values. We denote this set as $\alpha_{i-1}(i)$: the set of possible values of $\mathbf{a}^{(i-1)}$ with the restrictions imposed by the ith step $\left(\mathbf{q}^{(i-1)}, \mathbf{q}^{(i)}\right)$. As $\mathbf{a}^{(i-1)}$ is completely determined by $\mathbf{a}^{(0)}$, we can consider the set $\alpha_0(i)$ as the set of possible values of $\mathbf{a}^{(0)}$ with the restrictions imposed by the ith step. In the case that β is a permutation,

and hence all steps are also permutations, for each element in $\alpha_{i-1}(i)$ there is one element in $\alpha_0(i)$. Both have the same relative size: $2^{-\mathrm{w_r}(i)}$.

Now consider a two-round differential trail. The first step imposes that $\mathbf{a}^{(0)} \in \alpha_0(1)$ and the second step that $\mathbf{a}^{(1)} \in \alpha_1(2)$. We can reduce this second restriction to $\mathbf{a}^{(0)} \in \alpha_0(2)$. The joint restriction imposed by both steps now becomes: $\mathbf{a}^{(0)} \in \alpha_0(1,2)$ where $\alpha_0(1,2) = \alpha_0(1) \cap \alpha_0(2)$. If

$$\mathrm{Prob}(\mathbf{x} \in \alpha_0(1) | \mathbf{x} \in \alpha_0(2)) = \mathrm{Prob}(\mathbf{x} \in \alpha_0(1)), \tag{8.20}$$

the restrictions imposed by the first and the second step are independent. In that case, the relative size of $\alpha_0(1,2)$ is equal to $2^{-(\mathrm{w_r}(1)+\mathrm{w_r}(2))}$. It turns out that for round transformations with a high diffusion and some non-linearity, restrictions due to the differential steps of a linear trails can be considered independent if the weight of the trail is below $n-1$. The relative size of the set of values $\mathbf{a}^{(0)}$ that satisfy the restrictions imposed by all the differential steps of a differential trail Q is by definition the probability of Q.

While it is easy to compute the weight of a differential trail, computing its probability is in general difficult. If we neglect the correlations between the restrictions of the different steps, the probability of the differential trail is approximated by

$$\mathrm{Prob}(\mathsf{Q}) \approx 2^{-\mathrm{w_r}(\mathsf{Q})}. \tag{8.21}$$

For actual ciphers the approximation in (8.21) is generally very good if the weight of the trail is significantly below $n-1$. If $\mathrm{w_r}(\mathsf{Q})$ is of the order $n-1$ or larger, (8.21) can no longer be a valid approximation. In this situation, the inevitable (albeit small) correlations between the restrictions come into play. The probability multiplied by 2^n is the absolute number of inputs $\mathbf{a}^{(0)}$ for which the initial difference pattern propagates along the specified differential trail. For this reason, it must therefore be an (even) integer. Of the differential trails Q with a weight $\mathrm{w_r}(\mathsf{Q})$ above $n-1$, only a fraction $2^{n-1-\mathrm{w_r}(\mathsf{Q})}$ can be expected to actually occur for some $\mathbf{a}^{(0)}$.

Differential cryptanalysis exploits difference propagations $\left(\mathbf{q}^{(0)}, \mathbf{q}^{(r)}\right)$ with large probabilities. Since, for a given input value $\mathbf{a}^{(0)}$, exactly one differential trail is followed, the probability of difference propagation $(\mathbf{a}', \mathbf{b}')$ is the sum of the probabilities of all r-round differential trails with initial difference $\mathbf{a}'$ and terminal difference $\mathbf{b}'$. We have

$$\mathrm{Prob}(\mathbf{a}', \mathbf{b}') = \sum_{\mathbf{q}^{(0)}=\mathbf{a}', \mathbf{q}^{(r)}=\mathbf{b}'} \mathrm{Prob}(\mathsf{Q}). \tag{8.22}$$

8.5 Key-Alternating Cipher

As the round transformation is independent of the key, so is the weight of a differential step over a round. A key addition step has no impact on the

difference propagation pattern or the weight. Since the weight of a differential trail is the total of the weight of its differential steps, it is independent of the round keys and hence of the cipher key.

The reduction of the restrictions imposed upon $\mathbf{a}^{(i-1)}$ by $(\mathbf{q}^{(i-1)}, \mathbf{q}^{(i)})$, to restrictions on $\mathbf{a}^{(0)}$, involves the round keys. As the signs of the correlations between the different restrictions do depend on the round keys, the probability of a differential trail is in general not independent of the cipher key.

For a key-alternating cipher, the approximation given by (8.21) is key independent. Therefore, in key-alternating ciphers with a high-diffusion round transformation, differential trails with weights significantly below n have a probability that is practically independent of the round keys.

For differential trails Q with a weight $\mathrm{w_r}(\mathsf{Q})$ above $n-1$, only for an expected portion $2^{n-1-\mathrm{w_r}(\mathsf{Q})}$ of the cipher keys, there will exist a right pair.

8.6 The Effect of the Key Schedule

If we use the total weight over all differential steps to predict the difference propagation probability, we make the assumption that the restrictions due to the steps are independent. If we make an assumption for the value of the round keys, we can reduce the restrictions of all differential steps to restrictions on $\mathbf{a}^{(0)}$. It may turn out that the different restrictions are not independent. The reduction of the restrictions from all differential steps to $\mathbf{a}^{(0)}$ involves the round keys, that are in turn a function of the cipher key by the key schedule. Hence, the key schedule influences the reduction of restrictions of all differential steps to $\mathbf{a}^{(0)}$.

8.7 Differential Trails and Differential Cryptanalysis Literature

In this section we match our formalism with the terminology of the original description of differential cryptanalysis and with the widely accepted concept of Markov ciphers.

8.7.1 Differential Cryptanalysis of the DES Revisited

In this section we match the elements of differential cryptanalysis as described in Sect. 6.2 with those of our framework.

The characteristics with their characteristic probability described in [9] correspond to what we call differential trails and the approximation of its

probability based on its weight. In the cryptanalysis of the DES, the difference propagation probability from the initial difference pattern to the final difference pattern is approximated by the probability of the differential trail. This is a valid approximation because of the low relative weight of the differential trail:

1. The odd-round differential steps have a weight equal to 0 and do not impose any restrictions.
2. The even-round differential steps only impose restrictions on few state bits.
3. The state bits after round $i+2$ depend on many state bits after round i. In other words, the correlation between the different restrictions is very weak, if there is any.

For the DES round transformation the distribution of the differential steps and their weight are not independent of the round keys. This dependence was already recognized in [9] where in the analysis the weight of the differential steps are approximated by an average value. The two-round iterative differential trail with approximate probability 1/234 has in fact a probability that is either 1/146 or 1/585, depending on the value of a linear combination of round key bits.

8.7.2 Markov Ciphers

In Sect. 8.4.2 we discussed the determination of the probability of a multiple-round differential trail. This problem has been studied before. A commonly used approach was proposed by X. Lai, J. Massey and S. Murphy in [56]. We briefly introduce their approach here, and explain why we prefer our own formalism.

The most valuable contribution of [56] is that it is the first paper to make the difference between *differentials* and *characteristics*. A differential is a difference propagation from an input difference pattern to an output difference pattern. A characteristic is a differential trail along a number of rounds of a block cipher. In [56] it is shown that the probability of a differential over a sequence of rounds of a block cipher is equal to the sum of the probabilities of all characteristics (differential trails) over those rounds.

However, we do not fully agree with the general approach taken in [56]. It is based on the following three concepts:

1. **Markov cipher.** A Markov cipher is an iterative cipher whose round transformation satisfies the condition that the differential probability is independent of the choice of one of the component plaintexts under an appropriate definition of difference.

2. **Hypothesis of stochastic equivalence.** This hypothesis states that, for virtually all values of the cipher key, the probability of a differential trail can be approximated by the expected value of the probability of the differential trail, averaged over all possible values of the cipher key.
3. **Hypothesis of independent round keys.** The round keys of a cipher are derived from the cipher key by means of the key schedule. One can also study a cipher where all the round keys are specified independently. The hypothesis states that the expected probability of a differential trail, averaged over all possible values of the cipher key, can be approximated by the expected probability of the differential trail, averaged over all independently specified round key values.

Under the assumption of independent round keys, for a Markov cipher the sequence of round differences forms a Markov chain. From this it follows that the study of expected probabilities of differential trails in an r-round Markov cipher is reduced to the study of the transition probabilities created by its round transformation (assuming that it is an iterated cipher). Under the hypothesis of stochastic equivalence, conclusions can be drawn about the probability of differential trails under a fixed key. We have two major objections against Markov cipher theory.

As a first objection, we point out that the given conditions are *sufficient*, but not necessary. Similar to the theory of linear hulls (see Sect. 7.10.2), the condition of independent round keys for the Markov cipher theory is often misunderstood by cipher designers as an incentive to design heavy key schedules, in order to make the relations between round keys very complicated, or 'very random'. We have shown that for iterative Boolean transformations, independence of the differential steps can be defined even when there are no keys. In our discussion, we have shown that this independence does in no way require round keys that are supposed to be *independent* or, even worse, *random*. It is even quite meaningful to speak about the probability of differential trails in an iterative transformation with round keys that are fixed and known.

For our second objection, let us consider the validity of the hypothesis of stochastic equivalence. For certain popular Markov ciphers, this hypothesis does not seem to hold. The first example is the block cipher IDEA, that has been shown to have 2^{32} weak keys. For these weak keys, differential trails over the complete cipher exist that have a probability equal to 1 [19]. A second example is the AES candidate DFC [35]. The round transformation of DFC is based on a key-dependent linear transformation. For each given key, there are a few trails with probability 1, and many trails with probability 0. Averaged over all keys, all trails have a low, non-zero probability. However, this property does not give the cipher the predicted resistance against differential attacks [54]. It seems that the hypothesis of stochastic equivalence seems to hold

best in the case of key-alternating ciphers as for these ciphers the weight of differential trails is independent of the values of the round keys.

8.8 Conclusions

We have described the propagation of differences in Boolean functions, in iterated Boolean transformations and in block ciphers in general. We have introduced the *differential trail* as the basic building block of difference propagation in block ciphers.

9. The Wide Trail Strategy

In this chapter we explain the strategy that underlies many choices made in the design of Rijndael and its related ciphers.

We start with a detailed explanation of how linear correlations and difference propagation probabilities are built up in key-alternating block ciphers. This is followed by an explanation of the basic principles of the wide trail strategy. Then we introduce an important diffusion measure, the *branch number* and describe how it is relevant in providing bounds for the probability of differential trails and the correlation of linear trails over two rounds. This is followed by a key-alternating cipher structure that combines efficiency with high resistance against linear and differential cryptanalysis. We apply the same principles to the Rijndael cipher structure and prove a theorem that provides a lower bound on the number of active S-boxes in any four-round trail for these ciphers. Finally we provide some concrete constructions for the components used in the described cipher structures, using coding theory and geometrical representations.

9.1 Propagation in Key-alternating Block Ciphers

In this section we describe the anatomy of correlations and difference propagations in key-alternating block ciphers. This is used to determine the number of rounds required to provide resistance against linear and differential cryptanalysis. In this section we assume that the round transformations do not exhibit correlations with an amplitude of 1 or difference propagations with a probability of 1.

Limiting ourselves to the key-alternating structure allows us to reason more easily about linear and differential trails, since the effect of a key addition on the propagation is quite simple.

9.1.1 Linear Cryptanalysis

For a successful classical linear cryptanalysis attack, the cryptanalyst needs to know a correlation over all but a few rounds of the cipher with an amplitude

that is significantly larger than $2^{-n_b/2}$. To avoid this, we choose the number of rounds so that there are no such linear trails with a correlation contribution above $n_k^{-1}2^{-n_b/2}$.

This does not guarantee that there are no high correlations over r rounds. In Chap. 7 we have shown that each output parity of a Boolean function is correlated to a number of input parities. Parseval's theorem (7.15) states that the sum of the correlation potentials with all input parities is 1. In the assumption that the output parity is equally correlated to all 2^{n_b} possible input parities, the correlation to each of these input parities has amplitude $2^{-n_b/2}$. In practice it is very unlikely that such a uniform distribution will be attained, and so correlations will exist that are orders of magnitude higher than $2^{-n_b/2}$. This also applies to the Boolean permutation formed by a cipher for a given value of the cipher key. Hence, the presence of high correlations over (all but a few rounds of) the cipher is a mathematical fact rather than something that may be avoided by design.

However, when we impose an upper bound on the amplitude of the correlation contributions of linear trials, high correlations can only occur as the result of constructive interference of many linear trails that share the same initial and final selection patterns. If this upper bound is $n_k^{-1}2^{-n_b/2}$, any such correlation with an amplitude above $2^{-n_b/2}$ must be the result of at least n_k different linear trails. The condition that a linear trail in this set contributes constructively to the resulting correlation imposes a linear relation on the round key bits. From the point that more than n_k linear trails are combined, it is very unlikely that all such conditions can be satisfied by choosing the appropriate cipher key value.

The strong key-dependence of this interference makes it very unlikely that if a specific output parity has a high correlation with a specific input parity for a given key, this will also be the case for another value of the key. In other words, although it follows from Parseval's theorem that high correlations over the cipher will exist whatever the number of rounds, the strong round key dependence of interference makes locating the input and output selection patterns for which high correlations occur practically infeasible. This is true if the key is known, and even more so if it is unknown.

In the above discussion we have neglected possible linear trail clustering: the fact that sets of linear trails tend to propagate along common intermediate selection patterns. If linear trails tend to cluster, this must be taken into account in the upper bounds for the correlation contributions. Possible clustering of linear trails in Rijndael and its relatives is treated in Appendix B. As explained in Sect. 7.9.4, the key schedule has little relevance in this discussion. In our opinion, linear cryptanalysis does not give much guidance on how to design a key schedule.

9.1.2 Differential Cryptanalysis

For a successful classical differential cryptanalysis attack, the cryptanalyst needs to know an input difference pattern that propagates to an output difference pattern over all but a few (two or three) rounds of the cipher, with a probability that is significantly larger than 2^{1-n_b}. To avoid this, we choose the number of rounds so that there are no such differential trails with a weight below n_b.

This strategy does not guarantee that there are no such difference propagations with a high probability. For any Boolean function, a difference pattern at the input must propagate to some difference pattern at the output, and the sum of the difference propagation probabilities over all possible output differences is 1. Hence, there must be difference propagations with probabilities equal to or larger than 2^{1-n_b}. This also applies to the Boolean permutation formed by a cipher for a given value of the cipher key. Hence, similar to what we have for correlations, the presence of difference propagations with a high probability over (all but a few rounds of) the cipher is a mathematical fact that cannot be avoided by a careful design.

Let us analyse how, for a given key value, a difference pattern at the input propagates to a difference pattern at the output with some probability y. By definition, there are exactly $y2^{n_b-1}$ pairs with the given input difference pattern and the given output difference pattern. Each of these pairs follows a particular differential trail.

Assuming that the pairs are distributed over the trails according to a Poisson distribution, the expected number of pairs that, for a given key value, follow a differential trail with weight z is 2^{n_b-1-z}. Consider a differential trail with a weight z larger than $n_b - 1$ that is followed by at least one pair. The probability that this trail is followed by more than one pair is approximately 2^{n_b-1-z}. It follows that if there are no differential trails with a weight below $n_b - 1$, the $y2^{n_b-1}$ pairs that have the correct input difference pattern and output difference pattern follow almost $y2^{n_b-1}$ different differential trails.

Hence, if there are no differential trails with a low weight, difference propagations with a large probability are the result of multiple differential trails that happen to be followed by a pair in the given circumstances, i.e. for the given key value. For another key value, each of these individual differential trails may be followed by a pair or may not. This makes predicting the input difference patterns and output difference patterns that have large difference propagation probabilities practically infeasible. This is true if the key is known, and even more so if it is unknown.

In the above discussion we have neglected possible differential trail clustering: the fact that sets of differential trails tend to propagate along common intermediate difference patterns. If differential trails tend to cluster, this must be taken into account in the lower bounds for the weight of the differential

trails. Possible clustering of differential trails in Rijndael and its relatives is treated in Appendix B.

9.1.3 Differences between Linear Trails and Differential Trails

Linear and differential trails propagate in a very similar way. Still, when they are combined to form correlations and difference propagations, respectively, there are a number of very important differences.

The *impact* of a linear trail is its correlation contribution. The correlation contribution can easily be computed and its amplitude is independent of the value of the key. The problem with computing correlations over many rounds is that a correlation may be the result of many linear trails whose interference — constructive or destructive — is strongly key-dependent.

The *impact* of a differential trail is its probability, that is in general infeasible to compute precisely. However, it can be approximated using the *weight* of the differential trail. Unlike the probability, the weight of a differential trail is easy to compute. However, the approximation is only valid for differential trails in which the restrictions imposed by the differential steps are mutually independent and hence that have a weight below $n_b - 1$. If the probability of the individual differential trails would be known for a given key, difference propagation probabilities would be easy to compute. For differential trails, destructive interference does not exist.

9.2 The Wide Trail Strategy

The wide trail strategy is an approach used to design the round transformations of key-alternating block ciphers that combine efficiency and resistance against differential and linear cryptanalysis. In this book we describe the strategy for key-alternating block ciphers, but it can also be extended to more general block cipher structures.

We build the round transformations as a sequence of two invertible steps:

1. γ. A local non-linear transformation. By local, we mean that any output bit depends on only a limited number of input bits and that neighbouring output bits depend on neighbouring input bits.
2. λ. A linear mixing transformation providing high diffusion. What is meant by high diffusion will be explained in Sect. 9.2.3.

Hence we have a round transformation ρ:

$$\rho = \lambda \circ \gamma. \tag{9.1}$$

and refer to this as a $\gamma\lambda$ round transformation.

9.2.1 The $\gamma\lambda$ Round Structure in Block Ciphers

In block cipher design γ is a bricklayer permutation consisting of a number of S-boxes. The state bits of $\mathbf{a}$ are partitioned into n_t m-bit *bundles* $\mathbf{a}_i \in Z_2^m$ with $i \in \mathcal{I}$ according to the so-called *bundle partition*. $\mathcal{I}$ is called the *index space*. The block size of the cipher is given by $n_b = mn_t$.

Example 9.2.1. Let $\mathcal{X}_1$ be a cipher with a block length of 48 bits. Let the input be divided into six 8-bit bundles.

$$\mathbf{a} = \left[a_1 \; a_2 \; a_3 \; a_4 \; a_5 \; a_6 \right]$$

The index space is $\mathcal{I} = \{1, 2, 3, 4, 5, 6\}$.

Figure 9.1 illustrates the different steps of a round and a key addition for a simple example. The block cipher example has a block size of 27 bits. The non-linear S-boxes operate on $m = 3$ bits at a time. The linear transformation mixes the outputs of the $n_t = 9$ S-boxes. Figure 9.2 gives a more schematic representation, which we will use in the remainder of this chapter.

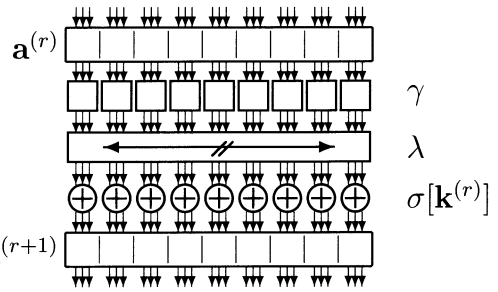

Fig. 9.1. Steps of an example block cipher.

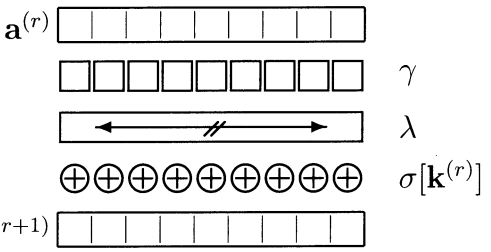

Fig. 9.2. Schematic representation of the different steps of a block cipher.

The step γ is a bricklayer permutation composed of S-boxes:

$$\gamma : \mathbf{b} = \gamma(\mathbf{a}) \Leftrightarrow \mathbf{b}_i = S_\gamma(\mathbf{a}_i), \tag{9.2}$$

where S_γ is an invertible non-linear m-bit substitution box. For the purpose of this analysis, S_γ does not need not to be specified. Clearly, the inverse of γ consists of applying the inverse substitution S_γ^{-1} to all bundles. The results of this chapter can easily be generalized to include non-linear permutations that use different S-boxes for different bundle positions. However, this does not result in a plausible improvement of the resistance against known attacks. The use of different S-boxes also increases the program size in software implementations and the required chip area in hardware implementations.

The step λ combines the bundles linearly: each bundle at its output is a linear function of bundles at its input. λ can be specified at the bit level by a simple $n_\mathrm{b} \times n_\mathrm{b}$ binary matrix M. We have

$$\lambda : \mathbf{b} = \lambda(\mathbf{a}) \Leftrightarrow \mathbf{b} = \mathsf{M}\mathbf{a}. \tag{9.3}$$

λ can also be specified at the bundle level. For this purpose the bundles are assumed to code elements in $\mathrm{GF}(2^m)$ with respect to some basis. In its most general form, we have:

$$\lambda : \mathbf{b} = \lambda(\mathbf{a}) \Leftrightarrow b_i = \bigoplus_j \bigoplus_{0 \le \ell < m} \mathsf{C}_{i,j,\ell} a_j^{2^\ell}. \tag{9.4}$$

In most instances a more simple linear function is chosen that is a special case of (9.4):

$$\lambda : \mathbf{b} = \lambda(\mathbf{a}) \Leftrightarrow b_i = \bigoplus_j \mathsf{C}_{i,j} a_j. \tag{9.5}$$

If we consider the state as an array of bundles, this can be expressed as a matrix multiplication:

$$\lambda : \mathbf{b} = \lambda(\mathbf{a}) \Leftrightarrow \mathbf{b} = \mathsf{C} \cdot \mathbf{a} \tag{9.6}$$

where C is an $n_\mathrm{t} \times n_\mathrm{t}$ matrix with elements in $\mathrm{GF}(2^m)$. The jth column of C is denoted by C_j. The inverse of λ is specified by the matrix C^{-1}.

Example 9.2.2. In $\mathcal{X}_1$, λ could be defined as

$$\lambda\left(\left[a_1\ a_2\ a_3\ a_4\ a_5\ a_6\right]\right) = \left[2 \cdot a_1\ a_1 \oplus a_2\ a_2 \oplus a_3 \oplus a_4 \oplus a_5\ a_4 \oplus a_5 \oplus a_6\ a_3 \oplus a_5 \oplus a_6\ a_2 \oplus a_3\right].$$

The C matrix is then given by

$$\mathsf{C} = \begin{bmatrix} 2\,0\,0\,0\,0\,0 \\ 1\,1\,0\,0\,0\,0 \\ 1\,0\,1\,1\,1\,0 \\ 0\,0\,0\,1\,1\,1 \\ 0\,0\,1\,0\,1\,1 \\ 0\,1\,1\,0\,0\,0 \end{bmatrix}.$$

9.2.2 Weight of a Trail

γ is a bricklayer permutation consisting of S-boxes. Hence, as explained in Sect. 7.4.3, the correlation over γ is the product of the correlations over the different S-box positions for the given input and output selection patterns. We define the *weight* of a correlation as the negative logarithm of its amplitude. The correlation weight for an input selection pattern and output selection pattern is the sum of the correlation weights of the different S-box positions. If the output selection pattern is non-zero for a particular S-box position or bundle, we call this S-box or bundle *active*.

Similarly, the weight of the difference propagation over γ is the sum of the weights of the difference propagations of the S-box positions for the given input difference pattern and output difference pattern. If the input difference pattern is non-zero for a particular S-box position or bundle, we call this S-box or bundle *active*.

We take S-boxes that have good non-linearity properties. For linear cryptanalysis, the relevant property is the maximum amplitude of correlations over the S-box. For differential cryptanalysis, the relevant property is the maximum difference propagation probability. Once a single S-box has been found with good properties, this can be used for all S-box positions in the non-linear permutation.

A linear trail is defined by a series of selection patterns. The weight of such a trail is the sum of the weights of the selection patterns of the trail. As the weight of the selection patterns is the sum of the weight of its active S-box positions, the weight of a linear trail is the sum of that of its active S-boxes. An upper bound to the correlation is a lower bound to the weight per S-box. Hence, the weight of a linear trail is equal to or larger than the number of active bundles in all its selection patterns times the minimum (correlation) weight per S-box. We call the number of active bundles in a pattern or a trail its *bundle weight*.

A differential trail is defined by a series of difference patterns. The weight of such a trail is the sum of the weights of the difference patterns of the trail. Completely analogous to linear trails, the weight of a differential trail is equal to or larger than the number of active S-boxes times the minimum (differential) weight per S-box.

This suggests two possible mechanisms of eliminating low-weight trails:

1. Choose S-boxes with high minimum differential and correlation weight.
2. Design the round transformation(s) in such a way that there are no relevant trails with a low bundle weight.

The maximum correlation amplitude of an m-bit invertible S-box is above $2^{-m/2}$, yielding an upper bound for the minimum (correlation) weight of $m/2$. The maximum difference propagation probability is at least 2^{2-m}, yielding an upper bound for the minimum (differential) weight of $m-2$. This seems to suggest that one should take large S-boxes.

Instead of spending most of its resources on large S-boxes, the wide trail strategy aims at designing the round transformation(s) such that there are no trails with a low bundle weight. In ciphers designed by the wide trail strategy, a relatively large amount of resources are spent in the linear step to provide high multiple-round diffusion.

9.2.3 Diffusion

Diffusion is the term used by C. Shannon to denote the quantitative spreading of information [86]. The exact meaning of the term diffusion depends strongly on the context in which it is used. In this section we will explain what we mean by diffusion in the context of the wide trail strategy.

Inevitably, the non-linear step γ provides some interaction between the different bits within the bundles that may be referred to as diffusion. However, it does not provide any inter-bundle interaction: difference propagation and correlation over γ stays confined within the bundles. In the context of the wide trail strategy, it is not this kind of diffusion we are interested in. We use the term diffusion to indicate properties of a Boolean function that increase the minimum bundle weight of linear and differential trails. In this sense, all diffusion is realized by λ; γ does not provide any diffusion at all.

Let us start by considering single-round trails. Obviously, the bundle weight of a single round trail — differential or linear — is equal to the number of active bundles at its input. It follows that the minimum bundle weight of a single-round trail is 1, independent of λ.

In two-round trails, the bundle weight is the sum of the number of active bundles in the (selection or difference) patterns in the state at the input of the first and at the second round. The state at the input of the second round is equal to the XOR of the output of the first and a round key. This key addition has no impact on the selection or difference pattern and hence does not impact on their bundle weight. In this context a relevant diffusion measure of ρ is the minimum number of active bundles at the input and output of ρ. We call this the (bundle) *branch number* of ρ. Basically, this

branch number provides a lower bound for the minimum bundle weight of any two-round trail. The bundle branch number ranges between two ('no diffusion at all') and the total number of bundles in the state n_t plus one.

In trails of more than two rounds, the desired diffusion properties of ρ are less trivial. It is clear that any $2n$-round trail is a sequence of n two-round trails and hence that its bundle weight is lower bounded by n times the branch number of ρ. One approach would be to design a round transformation with a maximum branch number. However, similar to large S-boxes, transformations that provide high branch numbers have a tendency to have a high implementation cost. More efficient designs can be achieved using a round structure with a limited branch number but with some other particular propagation properties.

For this purpose, λ can be built as a sequence of two steps:

1. θ. A step that provides high local diffusion.
2. π. A step that provides high *dispersion*.

In block cipher design, the mixing step θ is a linear bricklayer permutation. Its component each permutations operate on a limited number of bundles and have a branch number that is high with respect to their dimensions. The step π takes care of *dispersion*. By dispersion we mean the operation by which bits or bundles that are close to each other in the context of θ are moved to positions that are distant.

Jointly, θ and π have a spectacular effect on patterns with a low Hamming weight: through θ this propagates to a localized pattern with high Hamming weight that is dispersed all over the state by π. There are several approaches on how θ and π are selected. One of these approaches has lead to Rijndael and its relatives.

9.3 Branch Numbers and Two-Round Trails

In this section we formally define the *branch number* of a Boolean transformation with respect to a bundle partition.

The bundle weight of a state is equal to the number of non-zero bundles. This is denoted by $w_b(\mathbf{a})$. If this is applied to a difference pattern $\mathbf{a}'$, $w_b(\mathbf{a}')$ is the number of active bundles in $\mathbf{a}'$. Applied to a selection pattern $\mathbf{v}$, $w_b(\mathbf{v})$ is the number of active bundles in $\mathbf{v}$. We make a distinction between the differential and the linear branch number of a transformation.

Definition 9.3.1. *The* differential branch number *of a transformation ϕ is given by*

$$\mathcal{B}_d(\phi) = \min_{\mathbf{a},\mathbf{b}\neq\mathbf{a}} \{w_b(\mathbf{a} \oplus \mathbf{b}) + w_b(\phi(\mathbf{a}) \oplus \phi(\mathbf{b}))\}. \tag{9.7}$$

For a linear transformation $\lambda(\mathbf{a}) \oplus \lambda(\mathbf{b}) = \lambda(\mathbf{a} \oplus \mathbf{b})$, and (9.7) reduces to:

$$\mathcal{B}_{\mathrm{d}}(\lambda) = \min_{\mathbf{a}' \neq 0} \{\mathrm{w_b}(\mathbf{a}') + \mathrm{w_b}(\lambda(\mathbf{a}'))\}. \tag{9.8}$$

Analogous to the differential branch number, we can define the linear branch number.

Definition 9.3.2. *The* linear branch number *of a transformation ϕ is given by*

$$\mathcal{B}_{\mathrm{l}}(\phi, \alpha) = \min_{\alpha, \beta, \mathrm{C}(\alpha^{\mathrm{T}}\mathbf{x}, \beta^{\mathrm{T}}\phi(\mathbf{x})) \neq 0} \{\mathrm{w_b}(\alpha) + \mathrm{w_b}(\beta)\}. \tag{9.9}$$

If ϕ is a linear transformation denoted by λ, here exists a matrix M such that $\lambda(\mathbf{x}) = \mathsf{M} \cdot \mathbf{x}$. Equation (9.9) can then be simplified to (see Sect. 7.4).

$$\mathcal{B}_{\mathrm{l}}(\lambda) = \min_{\alpha \neq 0} \{\mathrm{w_b}(\alpha) + \mathrm{w_b}(\mathsf{M}^{\mathrm{T}}\alpha)\}. \tag{9.10}$$

It follows that the linear branch number of the linear transformation specified by the matrix M is equal to the differential branch number of the linear transformation specified by the matrix M^{T}. Many of the following discussions are valid both for differential and linear branch numbers, and both $\mathcal{B}_{\mathrm{d}}$ and $\mathcal{B}_{\mathrm{l}}$ are denoted simply by $\mathcal{B}$.

An upper bound for the (differential or linear) branch number of a Boolean transformation ϕ is given by the total number of bundles in the state, denoted by n_α. For the output difference or selection pattern corresponding to an input difference or selection pattern with a single non-zero bundle can have a maximum weight of n_α. Hence, the branch number of ϕ is upper bounded by

$$\mathcal{B}(\phi) \leq n_\alpha + 1. \tag{9.11}$$

In general, the linear and differential branch number of a transformation with respect to a partition are not equal. This is illustrated in Example 9.3.1. However, if the step λ satisfies certain conditions it can be shown that the differential and linear branch numbers are equal. An obvious sufficient condition is the requirement that M be symmetric. Also, if a Boolean transformation has the maximal possible differential or linear branch number, then both branch numbers are equal. This is proven for the case of linear transformations in Sect. 9.6 and for the general case in Appendix B.

Example 9.3.1. Consider the transformation $\lambda : \mathbf{x} \mapsto \mathsf{A} \cdot \mathbf{x}$ over GF(4), where

$$\mathsf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}. \tag{9.12}$$

Since $\mathsf{A} \cdot (1,0,0,0)^{\mathrm{T}} = (1,0,0,0)^{\mathrm{T}}$, it follows that $\mathcal{B}_{\mathrm{d}}(\theta) \leq 2$. However, simple enumeration shows that there is no α for which $\mathrm{w_b}(\alpha) + \mathrm{w_b}(\mathsf{A}^{\mathrm{T}}\alpha) \leq 2$. Therefore, $\mathcal{B}_{\mathrm{l}}(\theta) \geq 3$.

9.3.1 Derived Properties

From the symmetry of Definitions 9.3.1 and 9.3.2 it follows that the branch number of a transformation and that of its inverse are the same. Moreover, we have the following properties:

1. a (differential or selection) pattern $\mathbf{a}$ is not affected by a key addition and hence its bundle weight $\mathrm{w_b}(\mathbf{a})$ is not affected.
2. a bricklayer permutation operating on individual bundles cannot turn an active bundle into a non-active bundle or vice versa. Hence, it does not affect the bundle weight $\mathrm{w_b}$.

Assume that we have a transformation ϕ that is a sequence of a transformation ϕ_1 and a bricklayer transformation ϕ_2 operating on bundles, i.e. $\phi = \phi_2 \circ \phi_1$. As ϕ_2 does not affect the number of active bundles in a propagation pattern, the branch number of ϕ and ϕ_1 are the same. More generally, if propagation of patterns is analysed at the level of bundles (columns), bricklayer transformations operating on individual bundles (or columns) may be ignored as they leave the difference patterns and selection patterns unchanged.

If we apply this to the bundle weight of a $\gamma\lambda$ round transformation ρ, it follows immediately that the (linear or differential) bundle branch number of ρ is that of its linear part λ.

9.3.2 A Two-Round Propagation Theorem

The following theorem relates the value of $\mathcal{B}(\lambda)$ to a bound on the number of active bundles in a trail. The proof is valid both for linear and differential trails: in the case of linear trails $\mathcal{B}$ stands for $\mathcal{B}_{\mathrm{l}}$ and in the case of differential trails $\mathcal{B}$ stands for $\mathcal{B}_{\mathrm{d}}$.

Theorem 9.3.1 (Two-Round Propagation Theorem).
For a key-alternating block cipher with a $\gamma\lambda$ round structure, the number of active bundles of any two-round trail is lower bounded by the (bundle) branch number of λ.

Proof. Figure 9.3 depicts two rounds. Since the steps γ and $\sigma[\mathbf{k}]$ operate on each bundle individually, they do not affect the propagation of patterns. Hence it follows that $\mathrm{w_b}(\mathbf{a}^1) + \mathrm{w_b}(\mathbf{a}^2)$ is only bounded by the properties of the linear step λ of the first round. Definitions 9.3.1 and 9.3.2 imply that

the sum of the active bundles before and after λ of the first round is lower bounded by $\mathcal{B}(\lambda)$. □

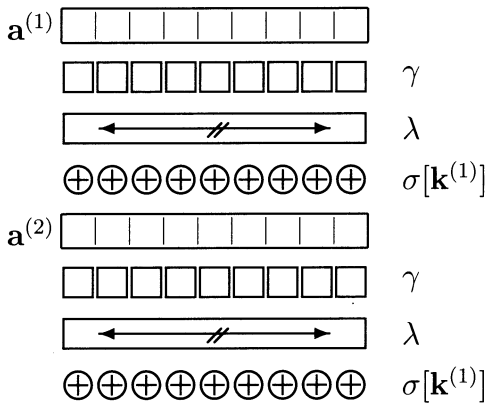

Fig. 9.3. Steps relevant in the proof of Theorem 9.3.1.

9.4 An Efficient Key-Alternating Structure

Theorem 9.3.1 seems to suggest that to obtain high lower bounds on the bundle weight of multiple round trails, a transformation λ must be used with a high branch number. However, realizing a high branch number has its computational cost. In this section we elaborate on a cipher structure that is more efficient in providing lower bounds.

We build a key-alternating block cipher that consists of an alternation of two different round transformations defined by

$$\rho^a = \theta \circ \gamma \text{ and} \tag{9.13}$$

$$\rho^b = \Theta \circ \gamma. \tag{9.14}$$

The step γ is defined as before and operates on n_t m-bit bundles.

9.4.1 The Diffusion Step θ

With respect to θ, the bundles of the state are grouped into a number of *columns* by a partition Ξ of the index space $\mathcal{I}$. We denote a column by ξ and the number of columns by n_Ξ. The column containing an index i is denoted by $\xi(i)$, and the number of indices in a column ξ by n_ξ. The size of the columns relates to the block length by

$$m \sum_{\xi \in \Xi} n_\xi = m n_\mathrm{t}.$$

θ is a bricklayer permutation with component permutations that each operate on a column. Within each column, bundles are linearly combined. We have:

$$\theta : \mathbf{b} = \theta(\mathbf{a}) \Leftrightarrow b_i = \bigoplus_{j \in \xi(i)} C_{i,j} a_j. \tag{9.15}$$

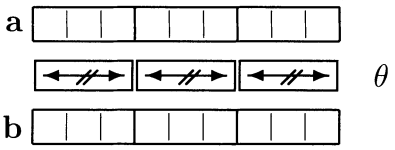

Fig. 9.4. The diffusion step θ.

If the array of bundles with indices in ξ is denoted by $\mathbf{a}_\xi$, we have

$$\theta : \mathbf{b} = \theta(\mathbf{a}) \Leftrightarrow \mathbf{b}_\xi = \mathsf{C}_\xi \mathbf{a}_\xi \tag{9.16}$$

where C_ξ is an $n_\xi \times n_\xi$ matrix. The jth column of C_ξ is denoted by $\mathsf{C}_{\xi|j}$. The inverse of θ is specified by the partition Ξ and the matrices C_ξ^{-1}. The bricklayer transformation θ only needs to realize diffusion within the columns and hence has an implementation cost that is much lower.

Similar to active bundles, we can speak of active columns. The number of active columns of a propagation pattern $\mathbf{a}$ is denoted by $\mathrm{w_s}(\mathbf{a})$.

The round transformation $\rho^{(a)} = \theta \circ \gamma$ is a bricklayer transformation operating independently on a number of columns. Taking this bricklayer structure into account, we can extend the result of Sect. 9.3 slightly. The branch number of θ is given by the minimum branch number of its component transformations. Applying (9.11) to the component permutations defined by the matrices C_ξ results in the following upper bound:

$$\mathcal{B}(\theta) \leq \min_\xi n_\xi + 1. \tag{9.17}$$

Hence, the smallest column imposes the upper limit for the branch number.

The two-round propagation theorem (Theorem 9.3.1) implies the following lemma.

Lemma 9.4.1. *The bundle weight of any two-round trail in which the first round has a $\gamma\theta$ round transformation is lower bounded by $N\mathcal{B}(\theta)$, where N is the number of active columns at the input of the second round.*

Proof. Theorem 9.3.1 can be applied separately to each of the component transformations of the bricklayer transformation $\rho^{(a)}$. For each active column there are at least $\mathcal{B}(\theta)$ active bundles in the two-round trail. If the number of active columns is denoted by N, we obtain the proof. □

Example 9.4.1. In $\mathcal{X}_2$, the partition Ξ has two elements. θ can be defined as

$$\theta\left(\begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \\ & a_5 \\ & a_6 \end{bmatrix}\right) = \begin{bmatrix} 2a_1 \oplus a_2 & a_3 \oplus a_4 \oplus a_5 \\ a_1 \oplus a_2 & a_4 \oplus a_5 \oplus a_6 \\ & a_3 \oplus a_5 \oplus a_6 \\ & a_3 \oplus a_4 \oplus a_6 \end{bmatrix}.$$

In this case there are two matrices C_ξ:

$$\mathsf{C}_{\xi(0)} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}, \qquad \text{and} \qquad \mathsf{C}_{\xi(2)} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}.$$

9.4.2 The Linear Step Θ

Θ mixes bundles across columns:

$$\Theta : \mathbf{b} = \Theta(\mathbf{a}) \Leftrightarrow b_i = \bigoplus_j C_{i,j} a_j \tag{9.18}$$

The goal of Θ is to provide inter-column diffusion. Its design criterion is to have a high branch number with respect to the column partition. This is denoted by $\mathcal{B}^c(\Theta)$ and called its *column branch number.*

9.4.3 A Lower Bound on the Bundle Weight of Four-Round Trails

The combination of the bundle branch number of θ and the column branch number of Θ allows us to prove a lower bound on the bundle weight of any trail over four rounds starting with $\rho^{(a)}$.

Theorem 9.4.1 (Four-Round Propagation Theorem for $\theta\Theta$ Construction). *For a key-alternating block cipher with round transformations as defined in (9.13) and (9.14), the bundle weight of any trail over $\rho^{(b)} \circ \rho^{(a)} \circ \rho^{(b)} \circ \rho^{(a)}$ is lower bounded by $\mathcal{B}(\theta) \times \mathcal{B}^c(\Theta)$.*

Proof. Figure 9.5 depicts four rounds with the key addition steps and the nonlinear steps removed, since these play no role in the trail propagation. It is easy to see that the linear step of the fourth round plays no role. The sum of the number of active columns in $\mathbf{a}^{(2)}$ and $\mathbf{a}^{(3)}$ is lower bounded by $\mathcal{B}^c(\Theta)$. According to Lemma 9.4.1, for each active column in $\mathbf{a}^{(2)}$ there are at least $\mathcal{B}(\theta)$ active bundles in the corresponding columns of $\mathbf{a}^{(1)}$ and $\mathbf{a}^{(2)}$. Similarly, for each active column in $\mathbf{a}^{(3)}$ there are at least $\mathcal{B}(\theta)$ active bundles in the corresponding columns of $\mathbf{a}^{(3)}$ and $\mathbf{a}^{(4)}$. Hence the total number of active bundles is lower bounded by $\mathcal{B}(\theta) \times \mathcal{B}^c(\Theta)$. □

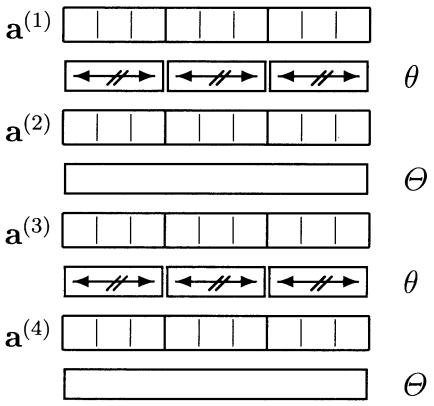

Fig. 9.5. Steps relevant in the proof of Theorem 9.4.1.

9.4.4 An Efficient Construction for Θ

As opposed to θ, Θ does not operate on different columns independently and hence may have a much higher implementation cost. In this section we present a construction of Θ in terms of θ and bundle transpositions denoted by π. We have

$$\Theta = \pi \circ \theta \circ \pi. \tag{9.19}$$

In the following we will define π, and prove that if π is well chosen the column branch number of Θ can be made equal to the bundle branch number of θ.

The bundle transposition π. The bundle transposition π is defined as

$$\pi : \mathbf{b} = \pi(\mathbf{a}) \Leftrightarrow b_i = a_{p(i)}, \tag{9.20}$$

where $p(i)$ is a permutation of the index space $\mathcal{I}$. The inverse of π is defined by $p^{-1}(i)$.

Example 9.4.2. In the cipher $\mathcal{X}_2$, we define π as the transformation that leaves the first row unchanged and shifts the second row one place to the right:

$$\pi\left(\begin{bmatrix} a_1 & a_3 & a_5 \\ a_2 & a_4 & a_6 \end{bmatrix}\right) = \begin{bmatrix} a_1 & a_3 & a_5 \\ a_6 & a_2 & a_4 \end{bmatrix}.$$

Observe that a bundle transposition π does not affect the bundle weight of a propagation pattern and hence that the branch number of a transformation is not affected if it is composed with π.

As opposed to θ, π provides inter-column *diffusion*. Intuitively, good diffusion for π would mean that it distributes the different bundles of a column to as many different columns as possible.

We say π is *diffusion optimal* if the different bundles in each column are distributed over all different columns. More formally, we have:

Definition 9.4.1. *π is* diffusion optimal *iff*

$$\forall i, j \in \mathcal{I}, i \neq j : (\xi(i) = \xi(j)) \Rightarrow (\xi(p(i)) \neq \xi(p(j))). \tag{9.21}$$

It is easy to see that this implies the same condition for π^{-1}. A diffusion optimal bundle transposition π implies $\mathrm{w_s}(\pi(\mathbf{a})) \geq \max_\xi(\mathrm{w_b}(a_\xi))$. Therefore a diffusion optimal transformation can only exist if $n_\Xi \geq \max_i(n_{\xi_i})$. In words, π can only be diffusion optimal if there are at least as many columns as there are bundles in the largest column.

If π is diffusion optimal, we can prove that the column branch number of the transformation Θ is equal to the branch number of θ.

Lemma 9.4.2. *If π is a diffusion optimal transposition of bundles, the column branch number of $\pi \circ \phi \circ \pi$ is equal to the bundle branch number of ϕ*

Proof. We refer to Fig. 9.6 for the notation used in this proof. Firstly, we demonstrate that

$$\mathrm{w_s}(\mathbf{a}) + \mathrm{w_s}(\mathbf{d}) \geq \mathcal{B}(\phi). \tag{9.22}$$

For any active column in **b**, the number of active bundles in that column and the corresponding column of **c** is at least $\mathcal{B}(\phi)$. π moves all active bundles in an active column of **c** to different columns in **d**, and π^{-1} moves all active bundles in an active column of **b** to different columns in **a**. It follows that the sum of the number of active columns in **a** and in **d** is lower bounded by the bundle branch number of ϕ.
Now we only have to prove that the sum of the number of active columns in **a** and in **d** is upper bounded by the bundle branch number of ϕ. Assume that **b**, and equivalently **c**, only have one active column and that ϕ restricted to this column has branch number $\mathcal{B}(\phi)$. In that case, there exists a configuration in which the sum of the number of active bundles in **b** and **c** is equal to $\mathcal{B}(\phi)$. π moves the active bundles in the active column of **c** to different columns in **d**, and π^{-1} moves the active bundles in the active column of **b** to different columns in **a**, and hence the total number of active columns in **a** and **d** is equal to $\mathcal{B}(\phi)$. □

9.5 The Round Structure of Rijndael

9.5.1 A Key-Iterated Structure

The efficient structure described in Sect. 9.4 uses two different round transformations. It is possible to define a block cipher structure with only one round

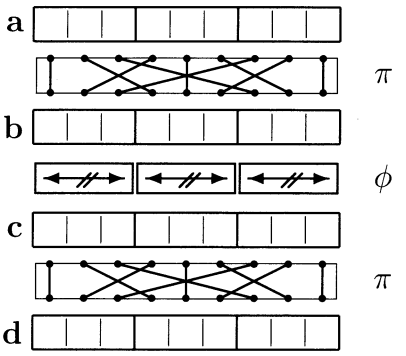

Fig. 9.6. Steps relevant in the proof of Lemma 9.4.2.

transformation that achieves the same bound. This is the round structure used in Rijndael and most of the related ciphers. The advantage of having a single round transformation is a reduction in program size in software implementations and chip area in dedicated hardware implementations. For this purpose, λ can be built as the sequence of two steps:

1. θ. The linear bricklayer transformation that provides high local diffusion, as defined in Sect. 9.4.1.
2. π. the bundle transposition that provides high *dispersion*, as defined in Sect. 9.4.4.

Hence we have the following for the round transformation:

$$\rho^c = \theta \circ \pi \circ \gamma \tag{9.23}$$

Figure 9.7 gives a schematic representation of the different steps of a round. The steps of the round transformation are defined in such a way that they

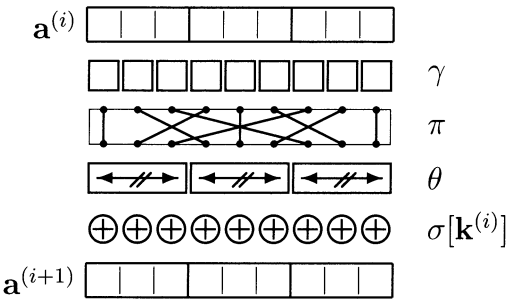

Fig. 9.7. Sequence of steps of a $\gamma\pi\theta$ round transformation, followed by a key addition.

impose strict lower bounds on the number of active S-boxes in four-round trails.

For two-round trails the number of active bundles is lower bounded by $\mathcal{B}(\rho) = \mathcal{B}(\lambda) = \mathcal{B}(\theta)$. For four rounds, we can prove the following important theorem:

Theorem 9.5.1 (Four-Round Propagation Theorem).
For a key-iterated block cipher with a $\gamma\pi\theta$ round transformation and diffusion optimal π, the number of active S-boxes in a four-round trail is lower bounded by $(\mathcal{B}(\theta))^2$.

Proof. Firstly, we show that the transformation consisting of four rounds ρ^c as defined in (9.23) is equivalent to four rounds of the construction with ρ^a and ρ^b as defined in (9.13) and (9.14). For simplicity, we leave out the key addition steps, but the proof works in the same way if the key addition steps are present. Let $\mathcal{A}$ be defined as

$$\begin{aligned}\mathcal{A} &= \rho^c \circ \rho^c \circ \rho^c \circ \rho^c \\ &= (\theta \circ \pi \circ \gamma) \circ (\theta \circ \pi \circ \gamma) \circ (\theta \circ \pi \circ \gamma) \circ (\theta \circ \pi \circ \gamma).\end{aligned}$$

γ is a bricklayer permutation, operating on every bundle separately and operating independently of the bundle's position. Therefore γ commutes with π, which only moves the bundles to different positions. We get

$$\begin{aligned}\mathcal{A} &= (\theta \circ \gamma) \circ (\pi \circ \theta \circ \pi \circ \gamma) \circ (\theta \circ \gamma) \circ (\pi \circ \theta \circ \pi \circ \gamma) \\ &= \rho^a \circ \rho^b \circ \rho^a \circ \rho^b,\end{aligned}$$

where Θ of ρ^b is defined exactly as in (9.19). Now we can apply Lemma 9.4.2 and Theorem 9.4.1 to finish the proof. □

The following is an alternative proof of Theorem 9.5.1. It does not use the results of the previous sections. In order to clarify the discussion, the steps γ and $\sigma[\mathbf{k}]$ have been left out of the picture.

Proof. Figure 9.8 depicts four rounds. It is easy to see that the linear steps of the fourth round play no role. By applying Lemma 9.4.2 on $\mathbf{a}^{(2)}$ and $\mathbf{b}^{(3)}$, it follows that the sum of the number of active columns in $\mathbf{a}^{(2)}$ and $\mathbf{b}^{(3)}$ is lower bounded by $\mathcal{B}(\theta)$. As the number of active columns in $\mathbf{b}^{(3)}$ and the number of active columns in $\mathbf{a}^4$ are equal, we have

$$\mathrm{w_s}\left(\mathbf{a}^{(2)}\right) + \mathrm{w_s}\left(\mathbf{a}^{(4)}\right) \geq \mathcal{B}(\theta)$$

By applying Lemma 9.4.1 to $\mathbf{b}^{(1)}$ and $\mathbf{a}^{(2)}$ we obtain

$$\mathrm{w_b}\left(\mathbf{b}^{(1)}\right) + \mathrm{w_b}\left(\mathbf{a}^{(2)}\right) \geq \mathrm{w_s}\left(\mathbf{a}^{(2)}\right) \mathcal{B}(\theta)$$

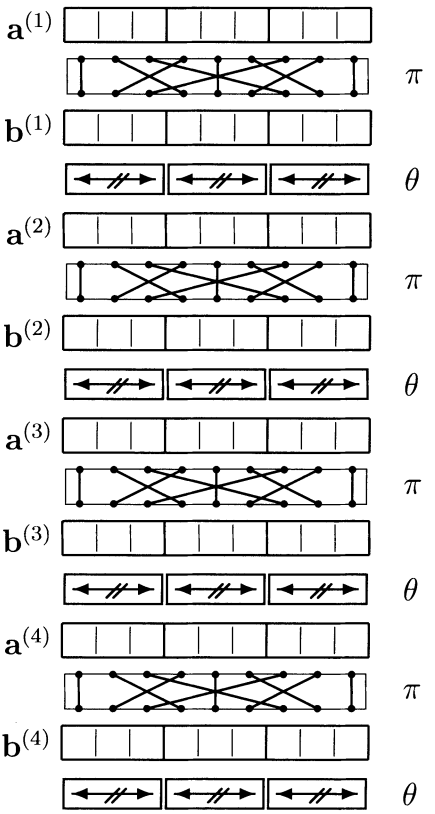

Fig. 9.8. Steps relevant in the proof of Theorem 9.5.1.

and applying Lemma 9.4.1 to $\mathbf{b}^{(3)}$ and $\mathbf{a}^{(4)}$ we obtain

$$\mathrm{w_b}\left(\mathbf{b}^{(3)}\right) + \mathrm{w_b}\left(\mathbf{a}^{(4)}\right) \geq \mathrm{w_s}\left(\mathbf{a}^{(4)}\right) \mathcal{B}(\theta)$$

Combining the three equations yields

$$\begin{aligned} \mathrm{w_b}\left(\mathbf{b}^{(1)}\right) + \mathrm{w_b}\left(\mathbf{a}^{(2)}\right) + \mathrm{w_b}\left(\mathbf{b}^{(3)}\right) + \mathrm{w_b}\left(\mathbf{a}^{(4)}\right) \\ \geq \left(\mathrm{w_s}\left(\mathbf{a}^{(2)}\right) + \mathrm{w_s}\left(\mathbf{a}^{(4)}\right)\right) \mathcal{B}(\theta) \\ \geq \left(\mathcal{B}(\theta)\right)^2 \end{aligned} \tag{9.24}$$

As a bundle transposition does not affect the bundle weight, we have $\mathrm{w_b}\left(\mathbf{b}^{(1)}\right) = \mathrm{w_b}\left(\mathbf{a}^{(1)}\right)$ and $\mathrm{w_b}\left(\mathbf{b}^{(3)}\right) = \mathrm{w_b}\left(\mathbf{a}^{(3)}\right)$. Substitution into (9.24) yields

$$\mathrm{w_b}\left(\mathbf{a}^{(1)}\right) + \mathrm{w_b}\left(\mathbf{a}^{(2)}\right) + \mathrm{w_b}\left(\mathbf{a}^{(3)}\right) + \mathrm{w_b}\left(\mathbf{a}^{(4)}\right) \geq \left(\mathcal{B}(\theta)\right)^2,$$

proving the theorem. □

In a four-round trail there can be only $4n_\mathrm{t}$ active bundles. One may wonder how the lower bound of Theorem 9.5.1 relates to this upper bound. From (9.11) we have that $\mathcal{B}^2 \leq \min(n_\xi + 1)^2 = \min n_\xi^2 + 2\min n_\xi + 1$. Diffusion-optimality implies that $\min(n_\xi + 1)^2 \leq \min n_\xi n_\Xi + 2\min n_\xi + 1 \leq n_\mathrm{t} + 2n_\mathrm{t} + n_\mathrm{t} = 4n_\mathrm{t}$. Hence, the lower bound of Theorem 9.5.1 is always below the upper bound of $4n_\mathrm{t}$.

9.5.2 Applying the Wide Trail Strategy to Rijndael

To provide resistance against differential and linear cryptanalysis, Rijndael has been designed according to the wide trail strategy: the four-round propagation theorem is applicable to Rijndael. It exhibits the key-iterated round structure described above:

1. `SubBytes`: the non-linear step γ, operating on the state bytes in parallel.
2. `ShiftRows`: the transposition step π.
3. `MixColumns`: the mixing step θ, operating on columns of four bytes each.

The coefficients of `MixColumns` have been selected in such a way that both the differential branch number and the linear branch number (see Definitions 9.3.1 and 9.3.2) of `MixColumns` are equal to 5. Since `ShiftRows` moves the bytes of each column to four different columns, it is diffusion optimal (see Definition 9.4.1). Hence, the four-round propagation theorem (Theorem 9.5.1) proves that the number of active S-boxes in a four-round differential trail or linear trail is lower bounded by 25.

S_{RD} has been selected in such a way that the maximum correlation over it is at most 2^{-3}, and that the difference propagation probability is at most 2^{-6}, in other words, that the weight of any difference propagation is at least 6.

This gives a minimum weight of 150 for any four-round differential trail and a maximum of 2^{-75} for the correlation contribution for any four-round linear trail. These results hold for all block lengths of Rijndael and are independent of the value of the round keys. Hence there are no eight-round trails with a weight below 300 or a correlation contribution above 2^{-150}. We consider this sufficient to resist differential and linear attacks.

9.6 Constructions for θ

For memory-efficient implementations, all columns preferably have the same size. The fact that the branch number is upper bounded by the smallest column (see Eq. (9.11)) points in the same direction. Hence we will consider in the following only the case where all columns have the same size.

Additionally we can reduce program and chip size by imposing the requirement that θ acts in the same way on each column. In this case the same matrix C_ξ is used for all columns.

Imposing additional symmetry conditions on the matrix C_ξ can lead to even more compact implementations. For instance, C_ξ can be defined as a circulant matrix.

Definition 9.6.1. *An $n \times n$ matrix* A *is* circulant *if there exist n constants $a_1, \ldots, a_n$ and a 'step' $c \neq 0$ such that for all i, j $(0 \leq i, j < n)$*

$$a_{i,j} = a_{i+cj \bmod n} \,. \tag{9.25}$$

If $\gcd(c, n) = 1$ it can be proven that $\mathcal{B}_l(\lambda) = \mathcal{B}_d(\lambda)$.

We can construct matrices C_Ξ giving rise to a maximum branch number from an MDS code.

The branch numbers of linear functions can be studied using the framework of linear codes over GF(2^p). Codes can be associated with Boolean transformations in the following way.

Definition 9.6.2. *Let θ be a transformation from $GF(2^p)^n$ to $GF(2^p)^n$. The* associated code *of θ, $\mathcal{C}_\theta$, is the code that has codewords given by the vectors $(\mathbf{x}, \theta(\mathbf{x}))^{\mathrm{T}}$. The code $\mathcal{C}_\theta$ has 2^n codewords and has length $2n$.*

If θ is defined as $\theta(\mathbf{x}) = \mathsf{A} \cdot \mathbf{x}$, then $\mathcal{C}_\theta$ is a linear $[2n, n, d]$ code. Code $\mathcal{C}_\theta$ consists of the vectors $(\mathbf{x}, \mathsf{A} \cdot \mathbf{x})^{\mathrm{T}}$, where $\mathbf{x}$ takes all possible input values.

Equivalently, the generator matrix G_θ of $\mathcal{C}_\theta$ is given by

$$\mathsf{G}_\theta = [\mathsf{I}\ \mathsf{A}]\ , \tag{9.26}$$

and the parity-check matrix H_θ is given by

$$\mathsf{H}_\theta = \left[-\mathsf{A}^t\ \mathsf{I}\right] = \left[\mathsf{A}^t\ \mathsf{I}\right]\ . \tag{9.27}$$

It follows from Definition 9.3.1 that the differential branch number of a transformation θ equals the minimal distance between two different codewords of its associated code $\mathcal{C}_\theta$. The theory of linear codes addresses the problems of determining the distance of a linear code and the construction of linear codes with a given distance. The relations between linear transformations and linear codes allow us to construct efficiently transformations with high branch numbers. As a first example, the upper bound on the differential branch number given as in (9.11) corresponds to the Singleton bound for codes (Theorem 2.2.1). Theorem 2.2.2 states that a linear code has distance d if and only if every $d-1$ columns of the parity-check matrix H are linearly independent and there exists some set of d columns that are linearly dependent. Reconsidering the matrix A of Example 9.3.1, all sets of two columns in $\mathsf{H} = [-\mathsf{A}^t\ \mathsf{I}]$ are independent, but no set of three independent columns exists. Therefore the differential branch number equals two. Theorem 2.2.3 states that a linear code with maximal distance requires that every square submatrix of A is nonsingular. An immediate consequence is that transformations can have maximal branch numbers only if they are invertible. Furthermore, a transformation with maximal linear branch number has also maximal differential branch number, and vice versa. Indeed, if all submatrices of A are non-singular, then this holds also for A^{T}.

The following theorem relates the linear branch number of a linear transformation to the dual of the associated code:

Theorem 9.6.1. *If $\mathcal{C}_\theta$ is the associated code of the linear transformation θ, then the linear branch number of θ is equal to the distance of the dual code of $\mathcal{C}_\theta$.*

Proof. We give the proof for binary codes only. If θ is specified by the matrix A, then $[\mathsf{I}\ \mathsf{A}]$ is a generator matrix for $\mathcal{C}_\theta$, and $\left[\mathsf{A}^{\mathrm{T}}\ \mathsf{I}\right]$ is a generator matrix for the dual of $\mathcal{C}_\theta$. It follows from (9.10) that the minimal distance of the code generated by $\left[\mathsf{A}^{\mathrm{T}}\ \mathsf{I}\right]$ equals the linear branch number of θ. □

It follows that transformations that have an associated code that is MDS, have equal differential and linear branch numbers.

9.7 Choices for the Structure of $\mathcal{I}$ and π

In this section we present several concrete constructions for π and the implications with respect to trails.

We present two general structures for $\mathcal{I}$ and π. In the first structure the different bundles of a state are arranged in a multidimensional regular array or hypercube of dimension d and side n_ξ. Ciphers constructed in this way have a block size of mn_ξ^d. In the second structure the bundles of a state are arranged in a rectangle with one side equal to n_ξ. This gives more freedom for the choice of the block size of the cipher.

9.7.1 The Hypercube Structure

In this construction the columns ξ are arranged in a hypercube. The step π corresponds to a rotation of the hypercube around a diagonal axis (called the p-axis).

The indices $i \in \mathcal{I}$ are represented by a vector of length d and elements i_j between 0 and $n_\xi - 1$. We have

$$\mathbf{i} = (i_1, i_2, \ldots, i_d). \tag{9.28}$$

The columns ξ are given by

$$\mathbf{j} \in \xi(\mathbf{i}) \text{ if } j_1 = i_1, j_2 = i_2, \ldots \text{ and } j_{d-1} = i_{d-1}. \tag{9.29}$$

$p(i)$, defining π, is given by

$$p : \mathbf{j} = p(\mathbf{i}) \Leftrightarrow (j_1, j_2, \ldots, j_{d-1}, j_d) = (i_2, i_3, \ldots, i_d, i_1). \tag{9.30}$$

Clearly, π is diffusion optimal (if $d > 1$). We will briefly illustrate this for d equal to 1, 2 and 3.

Dimension 1.. Dimension 1 is a degenerate case because the partition counts only one column, and π cannot be diffusion optimal. SHARK [81] is an example where $n_t = n_\xi = 8$ and $m = 8$, resulting in a block size of 64 bits.

Dimension 2.. Figure 9.9 shows the two-dimensional array, the transposition π, and the partition Ξ.

The two-dimensional structure is adopted in Square[21], with $m = 8$ and $n_\xi = 4$, resulting in a block cipher with a block size of 128 bits in which every four-round trail has at least $\mathcal{B}^2 = 25$ active S-boxes.

Crypton [59] (see Sect. 11.5.1) has the same structure and transposition π as SQUARE, but it uses a different step θ. Since for Crypton $\mathcal{B}(\theta) = 4$, there are at least 16 active S-boxes in every four-round trail.

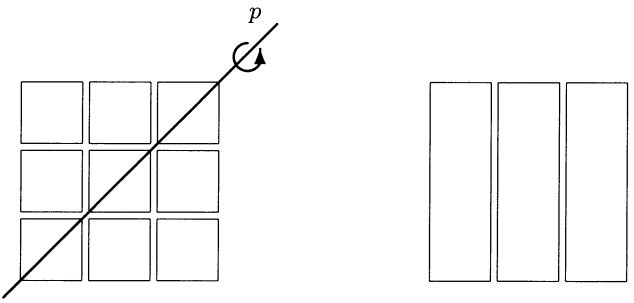

Fig. 9.9. Example of the hypercube structure with $d = 2$ and $n_\xi = 3$. The p-axis is indicated on the left.

Dimension 3.. For dimension three, with $n_\xi = 2$ and $m = 8$, we get a 64-bit cipher that has some similarity to the block cipher SAFER designed by J. Massey [64], however the round transformation of SAFER actually looks more like a triple application of $\theta \circ \pi$ for every application of γ. Therefore SAFER also can (almost) be seen as an example of a cipher with a diffusion layer of dimension 1.

Theorem 9.5.1 guarantees for our constructions a lower bound on the number of active S-boxes per four rounds of 9. For trails of more than four rounds, the minimum number of active S-boxes per round rises significantly: after six rounds for instance there are already minimum 18 active S-boxes. Figure 9.10 shows an example for the arrangement of the bundles and the columns.

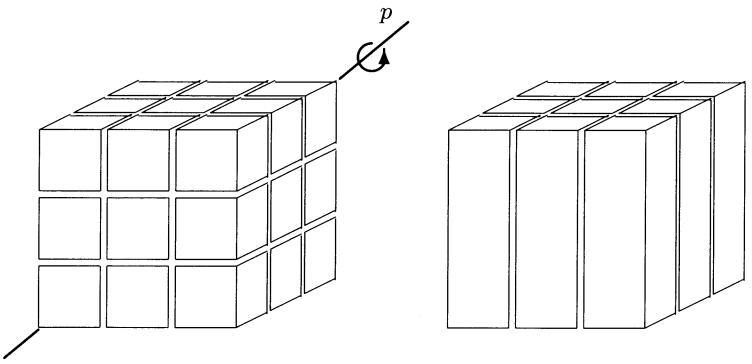

Fig. 9.10. Example for the hypercube structure with $d = 3$ and $n_\xi = 3$. The bundles are shown on the left and the colunms are shown on the right.

9.7.2 The Rectangular Structure

In this construction the columns ξ are arranged in a rectangle. The other dimensions of the array are determined from the required block size of the cipher. Figure 9.11 shows the arrangement of the bundles and the columns for an example where $n_\xi = 3$ and $n_\Xi = 5$. The step π leaves the first row invariant, shifts the second row by one position, and the third row by two positions.

Generally, if the step π shifts every row by a different number of bundles, the diffusion of π is optimal. (Note that this is only possible if $n_\Xi \geq n_\xi$, i.e. if the number of rows is at most the number of columns.)

If a bundle has $m = 8$ bits and every column contains $n_\xi = 4$ bundles, then setting the number of columns n_Ξ to 4, 5, 6, 7 or 8 gives a block size of 128, 160, 192, 224 or 256 bits, respectively. This is exactly the structure adopted in Rijndael [26]. BKSQ [25] is a cipher tailored for smart cards. Therefore its dimensions are kept small: $m = 8, n_\xi = 3$ and $n_\Xi = 4$ to give a block length of 96 bits.

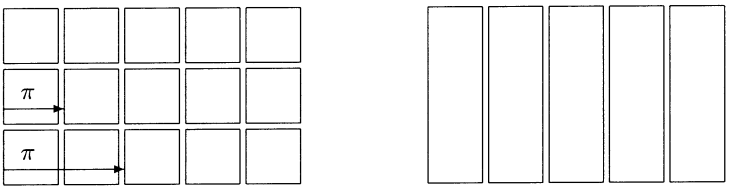

Fig. 9.11. Example of the rectangular structure. The bundles are shown on the left and the columns are on the right.

9.8 Conclusions

In this chapter we have given the design strategy that is the fundament of the Rijndael structure. The proposed cipher structure allows us to give provable bounds on the correlation of linear trails and the weight of differential trails, while at the same time allowing efficient implementations.

Finally, we show that Rijndael and its related ciphers are instances of a cipher family that allows a large flexibility in block length without losing the properties of efficiency and high resistance against cryptanalysis.

10. Cryptanalysis

The resistance of Rijndael against linear and differential cryptanalysis has been treated extensively in Chaps. 7 to 9 . In this chapter we discuss the resistance of Rijndael against various other cryptanalytic attacks. None of these attacks poses a threat to Rijndael, not in an academic, theoretical sense, and certainly not in a practical sense. We also touch briefly on the topic of implementation attacks.

10.1 Truncated Differentials

The concept of truncated differentials was described by L. Knudsen in [53]. The corresponding class of attacks exploit the fact that in some ciphers, differential trails (see Chap. 8) tend to cluster. We refer to Appendix B for a treatment in depth. In short, clustering takes place if for certain sets of input difference patterns and output difference patterns, the number of differential trails is exceedingly large. The expected probability that a differential trail stays within the boundaries of the cluster can be computed independently of the probabilities of the individual differential trails. Ciphers in which all steps operate on the state in bundles are prone to be susceptible to this type of attack. Since this is the case for Rijndael, with all steps operating on bytes rather than individual bits, we investigated its resistance against truncated differentials. For six rounds or more, no attacks faster than exhaustive key search have been found.

10.2 Saturation Attacks

In the paper presenting the block cipher Square [21], a dedicated attack by L. Knudsen on reduced versions of Square is described. The attack is often referred to as the 'Square' attack. The attack exploits the byte-oriented structure of Square, and is also applicable to reduced versions of Rijndael. N. Ferguson et al. [31] proposed some optimizations that reduce the work factor of the attack. In [61], S. Lucks proposes the name *'saturation attack'* for

this type of attack. More recently, these attacks have been called 'Structural attacks' by A. Biryukov and A. Shamir [11].

The saturation attack is a chosen-plaintext attack on ciphers with the Rijndael round structure. It can be mounted independently of the choice of the S-box in the non-linear step and the key schedule. The version we describe here is for the case that the columns of mixing step `MixColumns` have a maximum branch number and that the byte transposition `MixColumns` is diffusion optimal. If one of these two conditions is not fulfilled, the attack is slightly different but has comparable complexity. In this section we describe the attack on a cipher in which the bundles are bytes. Generalizing the attack to other bundle sizes is trivial.

Applied to Rijndael, the saturation attack is faster than an exhaustive key search for reduced-round versions of up to six rounds. After describing the basic attack on four rounds, we will show how it can be extended to five and six rounds.

10.2.1 Preliminaries

Let a Λ-set be a set of 256 states that are all different in some of the state bytes (the *active bytes*) and all equal in the other state bytes (the *passive bytes*) We have

$$\forall \mathbf{x}, \mathbf{y} \in \Lambda : \begin{cases} x_{i,j} \neq y_{i,j} \text{ if } (i,j) \text{ active} \\ x_{i,j} = y_{i,j} \text{ otherwise} \end{cases} . \tag{10.1}$$

Since the bytes of a Λ-set are either constant or range over all possible values, we have

$$\bigoplus_{\mathbf{x} \in \Lambda} x_{i,j} = 0, \quad \forall i, j. \tag{10.2}$$

Application of the steps `SubBytes` or `AddRoundKey` on the states of a Λ-set results in a different Λ-set with the positions of the active bytes unchanged. Application of the step `ShiftRows` results in a Λ-set in which the active bytes are transposed by `ShiftRows`. Application of the step `MixColumns` to a Λ-set does not necessarily result in a Λ-set. However, since every output byte of `MixColumns` is a linear combination with invertible coefficients of the four input bytes in the same column, an input column with a single active byte gives rise to an output column with all four bytes active. Hence, the output of `MixColumns` is a Λ-set if the Λ-set at its input has a maximum of one active byte per column.

10.2.2 The Basic Attack

Consider a Λ-set in which only one byte is active. We will now trace the evolution of the positions of the active bytes through three rounds. In the

first round, `MixColumns` converts the active byte to a complete column of active bytes. In the second round, the four active bytes of this column are spread over four distinct columns by `ShiftRows`. Subsequently, `MixColumns` of the second round converts this to 4 columns of only active bytes. The set stays a Λ-set until the `MixColumns` in the third round. Let the inputs of `MixColumns` in the third round be denoted by $\mathbf{a}^l$, and the outputs by $\mathbf{b}^l$. Then we have for all values of i, j that:

$$\begin{aligned}\bigoplus_l b_{i,j}^l &= \bigoplus_l \texttt{MixColumns}(a_{i,j}^l) \\ &= \bigoplus_l \left(02 \cdot a_{i,j}^l \oplus 03 \cdot a_{i+1,j}^l \oplus a_{i+2,j}^l \oplus a_{i+3,j}^l\right) \\ &= 02 \cdot \bigoplus_l a_{i,j}^l \oplus 03 \cdot \bigoplus_l a_{i+1,j}^l \oplus \bigoplus_l a_{i+2,j}^l \oplus \bigoplus_l a_{i+3,j}^l \\ &= 0 \oplus 0 \oplus 0 \oplus 0 = 0.\end{aligned}$$

Hence, all bytes at the input of the fourth round sum to zero. This property is in general destroyed by the subsequent application of `SubBytes`.

We assume that the fourth round is a `FinalRound`, i.e. it does not include a `MixColumns` operation. Every output byte of the fourth round depends on only one input byte of the fourth round. Let the input of the fourth round be denoted by $\mathbf{c}$, the output by $\mathbf{d}$ and the round key of the fourth round by $\mathbf{k}$. We have:

$$\mathbf{d} = \texttt{AddRoundKey}\left(\texttt{ShiftRows}\left(\texttt{SubBytes}(\mathbf{c})\right), \mathbf{k}\right) \tag{10.3}$$

$$d_{i,j} = \mathrm{S_{RD}}[c_{i,j+C_i}] \oplus k_{i,j}, \quad \forall\, i, j \tag{10.4}$$

$$c_{i,j} = \mathrm{S_{RD}}^{-1}[d_{i,j-C_i} \oplus k_{i,j-C_i}], \quad \forall\, i, j, \tag{10.5}$$

where the operations on the column index are, as always, performed in modulo $\mathrm{N_b}$. Using (10.5), the value of $c_{i,j}$ can be calculated from the ciphertexts for all elements of the Λ-set by assuming a value for $k_{i,j-C_i}$. If the assumed value for $k_{i,j-C_i}$ is equal to the correct round key byte, the following equations must hold:

$$\bigoplus_l c_{i,j}^l = 0, \quad \forall\, i, j. \tag{10.6}$$

If (10.6) does not hold, the assumed value for the key byte must be wrong. This is expected to eliminate all but approximately 1 key value. This can be repeated for the other bytes of $\mathbf{k}$. Since checking (10.6) for a single Λ-set leaves only $1/256$ of the wrong key assumptions as possible candidates, the cipher key can be found with overwhelming probability with only two Λ-sets. The work factor of the attack is determined by the processing of the first set of 2^8 plaintexts. For all possible values of one round key byte, (10.6) has to be evaluated. This means 2^{16} XOR operations and S-box look-ups. This

corresponds to roughly 2^{10} executions of the four-round cipher. A negligible amount of possible values for the round key byte has to be checked against the second set of plaintexts. In order to recover a full round key, the attack needs to be repeated 16 times. This results in a total complexity of 2^{14} cipher executions.

10.2.3 Influence of the Final Round

At first sight, it seems that the removal of the operation `MixColumns` in the final round of Rijndael makes the cipher weaker against the saturation attack. We will now show that adding a `MixColumns` operation in the last round would not increase the resistance. Let the input of the fourth round still be denoted by $\mathbf{c}$, and the output of a 'full' fourth round (including `MixColumns`) by $\mathbf{e}$. We have:

$$\mathbf{e} = \texttt{AddRoundKey}\left(\texttt{MixColumns}\left(\texttt{ShiftRows}\left(\texttt{SubBytes}(\mathbf{c})\right)\right), \mathbf{k}\right) \tag{10.7}$$

$$\begin{aligned} e_{i,j} = {} & \texttt{02} \cdot \mathrm{S_{RD}}[c_{i,j+C_i}] \oplus \texttt{03} \cdot \mathrm{S_{RD}}[c_{i+1,j+C_{i+1}}] \\ & \oplus \mathrm{S_{RD}}[c_{i+2,j+C_{i+2}}] \oplus \mathrm{S_{RD}}[c_{i+3,j+C_{i+3}}] \oplus k_{i,j}, \quad \forall i,j. \end{aligned} \tag{10.8}$$

There are $4\mathrm{N_b}$ equations (10.8): one for each value of i, j. The equations can be solved for the bytes of $\mathbf{c}$, e.g., for $c_{0,0}$:

$$\begin{aligned} c_{0,0} = {} & \mathrm{S_{RD}}^{-1}[\texttt{0E} \cdot (e_{0,0} \oplus k_{0,0}) \oplus \texttt{0B} \cdot (e_{1,-C_1} \oplus k_{1,-C_1}) \\ & \oplus \texttt{0D} \cdot (e_{2,-C_2} \oplus k_{2,-C_2}) \oplus \texttt{09} \cdot (e_{3,-C_3} \oplus k_{3,-C_3})] \end{aligned} \tag{10.9}$$

$$\begin{aligned} = {} & \mathrm{S_{RD}}^{-1}[\texttt{0E} \cdot e_{0,0} \oplus \texttt{0B} \cdot e_{1,-C_1} \oplus \texttt{0D} \cdot e_{2,-C_2} \oplus \texttt{09} \cdot e_{3,-C_3} \\ & \oplus k'_{0,0}], \end{aligned} \tag{10.10}$$

where the equivalent key $\mathbf{k}'$ is defined as

$$\mathbf{k}' = \texttt{InvMixColumns}(\texttt{InvShiftRows}(\mathbf{k})). \tag{10.11}$$

Similar equations hold for the other bytes of $\mathbf{c}$. The value of $c_{0,0}$ in all elements of the Λ-set can be calculated from the value of the ciphertext by assuming a value for one byte of the equivalent key $\mathbf{k}'$, and the same attack as before can be applied in order to recover the bytes of the equivalent key $\mathbf{k}'$. When all bytes of $\mathbf{k}'$ have been determined, (10.11) can be used to determine $\mathbf{k}$.

We conclude that the removal of the `MixColumns` step in the final round does *not* weaken Rijndael with respect to the four-round saturation attack. This conclusion agrees with the results of Sect. 3.7.2. Since the order of the steps `MixColumns` and `AddRoundKey` in the final round can be inverted, `MixColumns` can be moved after the last key addition and thus a cryptanalyst can easily factor it out, even if he does not know the round key.

10.2.4 Extension at the End

If a round is added, we have to calculate the value of $c_{i,j+C_i}$ from (10.5) using the output of the fifth round instead of the fourth round. This can be done by additionally assuming a value for a set of 4 bytes of the fifth round key. As in the case of the four-round attack, wrong key assumptions are eliminated by verifying (10.6).

In this five-round attack, 2^{40} key values must be checked, and this must be repeated four times. Since checking (10.6) for a single Λ-set leaves only $1/256$ of the wrong key assumptions as possible candidates, the cipher key can be found with overwhelming probability with only five Λ-sets. The complexity of the attack can be estimated at four runs $\times$ 2^{40} possible values for five key bytes $\times$ 2^8 ciphertexts in the set $\times$ five S-box look-ups per test, or 2^{46} five-round cipher executions.

10.2.5 Extension at the Beginning

The basic idea of this extension is to work with sets of plaintexts that result in a Λ-set with a single active byte at the output of the first round.

We consider a set of 2^{32} plaintexts, such that one column of bytes at the input of `MixColumns` in the first round ranges over all possible values and all other bytes are constant. Since `MixColumns` and `AddRoundKey` are invertible and work independently on the four columns, this property will be conserved: at the output of the first round the states will have constant values for 3 columns, and the value of the fourth column will range over all 2^{32} possibilities. This set of 2^{32} plaintexts can be considered as a union of 2^{24} Λ-sets, where each Λ-set has one active byte at the output of the first round. It is not possible to separate the plaintexts of the different Λ-sets, but evidently, since (10.6) must hold for every individual Λ-set, it must also hold when the sum goes over all 2^{32} values. Therefore, the round key of the final round can be recovered byte by byte, in the same way as for the 4-round attack. This five-round attack requires two structures of 2^{32} chosen plaintexts. The work factor of this attack can be estimated at 16 runs $\times$ 2^{32} ciphertexts in the set $\times 2^8$ possible values for the key byte $\times$ one S-box look-up per test, or 2^{38} five-round cipher executions.

10.2.6 Attacks on Six Rounds

Combining both extensions results in a 6-round attack. The work factor can be estimated at four runs $\times$ 2^{32} ciphertexts in the set $\times$ 2^{40} possible values for 5 key bytes $\times$ 5 S-box look-ups per test, or 2^{70} six-round cipher executions. N. Ferguson et al. explain in [31] a way to do the required calculations more efficiently. In this way, the work factor of the six-round attack can be further

reduced to 2^{46} six-round cipher executions. The work factor and memory requirements are summarized in Table 10.1. S. Lucks observes that for Rijndael with key lengths of 192 or 256, the six-round attack can be extended with one more round by guessing an additional round key [62]. The work factor of the attack increases accordingly.

Table 10.1. Complexity of saturation attacks applied to Rijndael.

Attack	No. of plaintexts	No. of cipher executions	Memory
Basic (four rounds)	2^9	2^{14}	small
Extension at end	2^{11}	2^{46}	small
Extension at beginning	2^{33}	2^{38}	2^{32}
Both extensions	2^{35}	2^{46}	2^{32}

10.2.7 The Herds Attack

N. Ferguson et al. describe in [31] a further extension of the saturation attack, known as the herds attack. Because of the attack's complexity, we did not verify its correctness. We simply copy here the attack's requirements. The authors describe a seven-round attack that requires $2^{128} - 2^{119}$ chosen plaintexts and 2^{64} bits of memory. The attack has a workload comparable to 2^{120} encryptions.

The attack can be extended into an eight-round attack with the same plaintext requirements and using 2^{104} bits of memory. The workload is too large to be applicable to the case of 128-bit keys. For 192-bit keys, the workload is comparable to 2^{188} encryptions. For 256-bit keys, this becomes 2^{204} encryptions.

10.3 Gilbert–Minier Attack

The saturation attack on Rijndael reduced to six rounds is based on the fact that three rounds of Rijndael can be distinguished from a random permutation. H. Gilbert and M. Minier developed a four-round distinguisher that allows an attack on Rijndael that is reduced to seven rounds [36]. Due to the increased work factor of the attack, it is more efficient than exhaustive key search for only some of the key lengths.

10.3.1 The Four-Round Distinguisher

Let the input of the first round be denoted by **a**, the input of the second round by **b**, the input of the third round by **c**, the input of the fourth round by **d** and

the output of the fourth round by $\mathbf{e}$. Let $R4_k$ denote the action of Rijndael, reduced to four rounds, under the unknown key $\mathbf{k}$. We will investigate the behaviour of a family of functions $f_{uvw,\mathbf{k}}$, that is parameterised by three bytes u, v and w and a key $\mathbf{k}$. The functions take a byte value x as input and have a byte value y as output. The functions $f_{uvw,\mathbf{k}}$ are defined as follows:

$$f_{uvw,\mathbf{k}}(\mathbf{x}) = \mathbf{y} \Leftrightarrow \begin{cases} a_{0,0} = x, a_{1,0} = u, a_{2,0} = v, a_{3,0} = w, \\ \text{the other } a_{i,j} \text{ are unknown, but constant,} \\ R4_k(a) = e, \\ y = \texttt{0E}e_{0,0} \oplus \texttt{0B}e_{1,0} \oplus \texttt{0D}e_{2,0} \oplus \texttt{09}e_{3,0} \end{cases} . \quad (10.12)$$

It can be shown that the inherent structure in the transformation `Round` imposes restrictions on the family of functions $f_{uvw,\mathbf{k}}$: if 2^{16} different values for the parameters u, v and w are selected, with large probability at least two sets of parameters will result in the same function f. This property holds for all values of the key $\mathbf{k}$ and can be used to distinguish $R4_k$ from a random permutation. Note that the distinguisher does not work with probability 1. More information on the construction of this four-round distinguisher can be found in [36].

10.3.2 The Attack on Seven Rounds

In the same way as the six-round saturation attack, the seven-round attack is mounted by adding one round before the distinguisher and two rounds after it.

By assuming a value for 4 key bytes of the first round key, it is possible to determine a set of plaintexts such that the inputs of the second round are constant in three columns. This set is divided into subsets with constant values for the 'parameters' u, v and w at the input of the second round. There should be 2^{16} subsets, with 16 plaintexts in each subset. 16 values for x suffices to determine whether two sets of parameters result in identical functions, with negligible false-alarm probability. It can be shown that the required plaintexts for all 2^{32} values of the 4 bytes of the first round key can be drawn from a set of 2^{32} plaintexts.

Each of the bytes $e_{i,j}$ can be expressed as a function of 4 ciphertext bytes and 5 key bytes. Hence the y values depend on 20 key bytes, that have to be guessed in order to perform the attack. The work complexity of the attack can be estimated at about 2^{192} executions of the round transformation, which is below the complexity of an exhaustive search for a 256-bit key, and approximately equal to the complexity of an exhaustive search for a 192-bit key.

A variant of this attack works only for 128-bit keys, and is claimed to be marginally faster than an exhaustive search for a 128-bit key.

10.4 Interpolation Attacks

In [46] T. Jakobsen and L. Knudsen introduced a new attack on block ciphers. In this attack, the attacker constructs polynomials using cipher input/output pairs. This attack is feasible if the components in the cipher have a compact algebraic expression and can be combined to give expressions with manageable complexity. The basis of the attack is that if the constructed polynomials have a small degree, only a few cipher input/output pairs are necessary to solve for the (key-dependent) coefficients of the polynomial.

S_{RD} takes bytes as input and produces bytes as output. Like any other transformation with this input size and output size, it can be expressed as a polynomial over GF(2^8). The polynomial expression of S_{RD} can, for example, be found by means of the Lagrange interpolation technique. The polynomial expression for S_{RD} is given by

$$\begin{aligned} S_{RD}[x] = {} & \mathtt{63} \oplus \mathtt{8F}x^{127} \oplus \mathtt{B5}x^{191} \oplus \mathtt{01}x^{223} \oplus \mathtt{F4}x^{239} \\ & \oplus \mathtt{25}x^{247} \oplus \mathtt{F9}x^{251} \oplus \mathtt{09}x^{253} \oplus \mathtt{05}x^{254}. \end{aligned} \tag{10.13}$$

This complicated expression of S_{RD} in GF(2^8), in combination with the effect of the mixing and transposition steps, prohibits interpolation attacks on more than a few rounds of Rijndael.

The techniques in [46] can be extended to use rational expressions or in fact any other type of expression. We found no simple rational expression for S_{RD} but it seems impossible to prove that no usable expression can be found. A second possible extension of this attack is the use of approximate expressions, as proposed by T. Jakobsen in [45]. It remains an open problem whether any useful expression can be derived in this way.

N. Ferguson et al. describe how one can derive an algebraic expression for ten-round Rijndael [32]. The expression would count 2^{50} terms. Although this is certainly an interesting result, the authors are not aware of ways to use this expression in an actual attack. Another interesting and as yet unanswered question is how this compares with other block ciphers.

10.5 Symmetry Properties and Weak Keys as in the DES

Despite the large amount of symmetry, care has been taken to eliminate symmetry in the behaviour of the cipher. This is obtained by the round constants that are different for each round. The fact that the cipher and its inverse use different transformations practically eliminates the possibility for weak and semi-weak keys, as described by D. Davies for the DES [28]. The non-linearity of the key expansion practically eliminates the possibility of equivalent keys.

10.6 Weak keys as in IDEA

The weak keys discussed in this subsection are keys that result in a permutation with detectable weaknesses. The best-known case of this type of weak keys are those of IDEA [19]. Typically, this weakness occurs when a cipher depends heavily on the key application for its non-linearity. Rijndael is a key-iterated cipher with all non-linearity provided by a key-independent S-box, and hence does not exhibit this type of weak keys (see Sect. 5.7).

10.7 Related-Key Attacks

In [6], E. Biham introduced a related-key attack. Later it was demonstrated by J. Kelsey et al. that several ciphers have related-key weaknesses [49]. In related-key attacks, the cryptanalyst is assumed to have access to the ciphertexts that result from encryptions using different (unknown or partly unknown) keys with a chosen relation. The key schedule of Rijndael, with its high diffusion and non-linearity, makes it difficult to mount attacks of this type. N. Ferguson et al. describe a related-key attack on Rijndael reduced to nine rounds [31]. The attack works for a Rijndael version with 128-bit blocks and a 256-bit key. It requires 2^{72} chosen plaintexts and has a work factor of 2^{227} encryptions, which indeed is faster than an exhaustive search for the key.

10.8 Implementation Attacks

Implementation attacks are based not only on mathematical properties of the cipher, but also on physical characteristics of the implementation. Typical examples are timing attacks [50], introduced by P. Kocher, and power analysis [51], introduced by P. Kocher et al. In timing attacks, measuring the total execution time of the encryption algorithm is used to derive key information. In power analysis attacks, measurements of the power consumption of the device executing the encryption algorithm is used to derive key information. Power analysis attacks can be generalized to other measurable quantities such as radiation or heat dissipation emanating from the device.

10.8.1 Timing Attacks

A timing attack can be mounted if the execution time of the encryption algorithm depends on the value of the key. Let us illustrate this by an example. Assume that we have a cipher implementation in which an instruction is executed on the condition that a certain key-dependent intermediate result b

takes a specific value. If no special precautions are taken, the total execution time of the cipher will vary depending on whether or not the conditional instruction is executed. Hence, it is possible to deduce the value of b from carefully measuring the execution time. It suffices to compare the encryption time for different values of b, while taking care that all other parameters influencing the encryption time are kept constant or averaged out.

An implementation can be protected against timing attacks by ensuring that the encryption time is independent of the value of the key. For conditional instructions, this can be done by inserting dummy instructions in the shortest path until all paths take the same time. However, this solution might leave the cipher unprotected against power analysis attacks.

In Rijndael, the only possible weakness with respect to timing attacks is the implementation of the finite field multiplications in `MixColumns`, namely the subroutine `xtime`. All other operations in Rijndael are implemented naturally by instructions that take a constant time. The weakness in `xtime` can easily be eliminated by defining a 256-byte table and using a look-up table to implement `xtime` (see Sect. 4.1.1).

10.8.2 Power Analysis

Simple power analysis (SPA) is an attack where the attacker obtains measurements of the power consumption of the device during the execution of one encryption. Typically, this type of attack is applicable to devices that depend on external power supplies, e.g. smart cards. If the power consumption pattern of the hardware depends on the *instruction* being executed, the attacker can deduce the sequence of instructions. If the sequence or the type of instructions depends on the value of the key, then the power consumption pattern leaks information about the key. Rijndael can easily be implemented with a fixed sequence of instructions, which prevents this type of attack.

In most processors, the power consumption pattern of an instruction depends on the value of the *operands*. For example, setting a bit in a register might consume more power than clearing it. Usually, the variation in the power consumption due to the difference in operand value is so small that it is buried in noise and is not revealed in power consumption measurements. However, by combining measurements of many encryptions, the attacker can average out the noise and obtain information about the value of the operand. This class of attack are called *differential power analysis* (DPA). Protecting implementations against these sophisticated attacks is much harder than for timing attacks and SPA, especially if the signal-to-noise ratio is high. Proposed countermeasures can be divided into three classes.

Protection of individual instructions. It is possible to reduce the vulnerability of each individual instruction against power analysis. A first example is *load balancing* which was proposed in [27]. Load balancing can be

achieved by a redesign of the hardware to minimize or eliminate completely the dependency of the power consumption on the value of the operands. This redesign can also be simulated by changing the software in such a way that all data words contain at all times the complement of each of the data bits as well as the data bits themselves. In this way, the correlation between power consumption and input values can be diminished. It seems unlikely that the dependency can be eliminated completely since there will always be small physical variations in the devices.

Masking of operands is another technique. In this approach, instructions on the operand x are replaced by instructions on operands x' and x'', where x' and x'' are unpredictable for the attacker. Only the joint knowledge of x' and x'' reveals information on the value of x. Several approaches have been proposed [2, 16, 23, 39, 69].

Protection of individual instructions has the disadvantage that it has to be repeated for each instruction that the algorithm uses. The fact that Rijndael can be implemented using only the XOR instruction and look-up tables is definitely an advantage here. This is illustrated by T. Messerges in [69], where the performance decrease for protected implementations of the five AES finalists is compared. It is shown that the performance penalty for Rijndael is very modest. M.-L. Akkar and C. Giraud present in [2] another masking technique, which uses the mathematical structure of Rijndael to generate masks.

Desynchronization. Instead of focussing on the protection of individual instructions separately, one can also try to limit the impact of the weaknesses of the individual instructions. A first approach is *desynchronization:* by changing the instruction sequence for every encryption, or part of the encryption, it becomes more difficult for the attacker to obtain meaningful statistics. The parallelism in the round transformation of Rijndael allows for some variation in the instruction sequence. However, the number of sequences is limited.

Key schedule complexity. In [15] S. Chari et al. argue that a complex key expansion scheme helps to increase the resistance against power attacks. If the knowledge of a round key does not allow reconstruction the cipher key, an attacker will have to recover more — or all — round keys in order to be able to forge or read new messages. It is argued that the simple key expansion of Rijndael is a disadvantage in this respect. However, it should be clear to the reader that if one round key can be recovered, the other round keys can be recovered with a similar effort. Furthermore, it is extremely likely that the extra effort to recover more round keys will only be computational effort — the number of required power measurements, which is the limiting factor in this class of attacks, will not increase significantly. Moreover, the computations of the key schedule by themselves are a target of power analysis attacks. In this respect, it is a disadvantage to have a complex key schedule.

10.9 Conclusion

Resistance against linear and differential attacks was a design criterion of Rijndael. From the number of publications alone, we can conclude that during the AES selection process, Rijndael attracted a significant amount of attention from the cryptographic community. SQUARE, the direct predecessor of Rijndael, has also been scrutinized vigorously for weaknesses. The complexity of the published attacks on reduced versions of Rijndael indicate that with the current state-of-the-art cryptographic techniques, no attacks can be mounted on a full version of Rijndael.

In order to resist implementation attacks, care has to be taken when implementing the algorithm. Because of its simplicity, Rijndael has a number of advantages when it comes to protecting its implementation against this kind of attack.

11. Related Block Ciphers

We did not design Rijndael from scratch. In fact, prior to the design of Rijndael, we had already published three block ciphers that are similar to Rijndael. Each of these ciphers inherits properties from its predecessor and enriches them with new ideas. Moreover, since the publication of Rijndael and its predecessors, a substantial number of cryptographers have based block cipher designs on ideas that were introduced in the Rijndael family. Hence, Rijndael can be seen as a step in an evolution, with predecessors and successors.

In this chapter, we discuss the similarities and differences between Rijndael and its different predecessors, and discuss briefly some of the recent block cipher designs that are based on Rijndael or use some elements of its round transformation.

11.1 Overview

The design of Rijndael is only one step in a long process of our research on the design of secure and efficient block ciphers using the wide trail design strategy. In this section, we present the different ciphers that we designed along the way. We also discuss common elements of the round transformation structure, and the differences in the first or the last round.

11.1.1 Evolution

SHARK. The first cipher in the series was SHARK, which was published in [81]. In this cipher, we first used MDS codes to build a mixing step. The mixing step of SHARK has the one-dimensional structure described in Sect. 9.7.1. The round transformation of SHARK is modular and in principle easily extendible to any block length that is a multiple of 8. However, for a block length of $8n$ bits, an efficient implementation of the round transformation uses tables that require $n^2 \times 256$ bytes of memory. For block lengths of 128 bits, this becomes inefficient on most common processors.

Square. The cipher Square was published in [21]. It has a block length of 128 bits, yet requires only sixteen 8-bit to 32-bit table look-ups per round, whereas an extension of SHARK to this block length would require sixteen 8-bit to 128-bit table look-ups per round. The increased efficiency is achieved by using a two-dimensional structure, as discussed in Sect. 9.7.1, and the introduction of a transposition step. The round transformation of Square uses tables that require $n \times 256$ bytes in total, for a block length of $8n$. Note that n has to be a square number. For Square, n has been fixed to 16.

Another improvement in Square concerns the implementations on processors with limited RAM. These processors typically have no space for the extended tables. By restricting the coefficients in the mixing step to small values, the performance on these limited processors becomes acceptable for practical applications.

A fourth improvement in Square is the introduction of an efficient and elegant key schedule.

BKSQ. The cipher BKSQ was published in [25]. In this cipher, the round transformation structure of Square is further generalized. The state is no longer 'square', but can become 'rectangular'. This allows defining ciphers with block lengths of $8n_1n_2$ bits.

A second modification with respect to Square is the introduction of non-linearity in the key schedule.

11.1.2 The Round Transformation

SHARK, Square, BKSQ and Rijndael are key-iterated block ciphers: they consist of the alternation of a key-independent round transformation ρ with a key addition, here denoted by $\sigma[\mathbf{k}]$. The round transformation is the sequence of a non-linear bricklayer permutation, here denoted by γ, and a linear step, here denoted by λ. The three operations $\sigma[\mathbf{k}]$, γ and λ can be ordered in six different ways in the round transformation. However, we will show that with respect to security, all the orderings are equivalent.

Equivalence of orderings. Firstly, we recall from Sect. 3.7.2 that

$$\sigma[\lambda(\mathbf{k})] \circ \lambda \equiv \lambda \circ \sigma[\mathbf{k}]. \tag{11.1}$$

Both orderings can be chosen in the definition of the cipher's round transformation, without making a difference in the security analysis or performance of the cipher.

Secondly, consider the following key-dependent round transformations that are rotated versions of one another:

$$\rho_1 = \sigma[\mathbf{k}] \circ \lambda \circ \gamma \tag{11.2}$$

$$\rho_2 = \lambda \circ \gamma \circ \sigma[\mathbf{k}]. \tag{11.3}$$

A cipher defined as the iteration of R ρ_1 round transformations can also be described as an iteration of ρ_2 round transformations, with a special definition for the first round and the last round.

We conclude that the same ordering of operations in the cipher can follow from different definitions of the round transformation. In fact, from the previous arguments it follows that all six orderings of the operations in the round transformation result in equivalent ciphers, except for the definition of the key schedule and for the definition of the first and the last round.

Boundary effects. The first and/or the last round of the ciphers can differ from the other rounds in several ways. Firstly, operations performed before the first key application or after the last key application can usually be factored out by the cryptanalyst and hence do not contribute to the security of the cipher. The only exception to this rule are the modes of operation where only a part of the state may be output, e.g. in the CFB mode. Therefore, if in the definition of the cipher, the round transformation does not start (end) with a round key application, an extra round key application has to be added to the beginning (end) of the cipher.

Secondly, because of (11.1) it is usually possible to leave out one application of λ in the first or the last round, since it does not improve the security of the cipher. Removing one application of λ usually helps to give the inverse of the cipher the same structure as the cipher.

11.2 SHARK

Both the block length and the key length of SHARK can easily be varied. In [81] it is proposed to use a block length of 8 bytes, or 64 bits. Let the number of bytes in the input be denoted by n. For a block length of 64 bits, $n = 8$.

The structure. The round transformation of SHARK has the simple $\gamma\lambda$ structure, as defined in Sect. 9.2.1. The elements of a state $\mathbf{a}$ are denoted by a_i, $0 \leq i < n$. The cipher consists of eight rounds.

The linear transformation. The mixing step of SHARK is derived from a linear code over GF(2^8) with length $2n$, dimension n and minimal distance $n+1$. This construction corresponds to the one-dimensional structure discussed in Sect. 9.7.1. The transformation is denoted by λ. For $n = 8$, we have

$$\lambda(\mathbf{a}) = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \end{bmatrix} \times \begin{bmatrix} \texttt{CE} & \texttt{95} & \texttt{57} & \texttt{82} & \texttt{8A} & \texttt{19} & \texttt{B0} & \texttt{01} \\ \texttt{E7} & \texttt{FE} & \texttt{05} & \texttt{D2} & \texttt{52} & \texttt{C1} & \texttt{88} & \texttt{F1} \\ \texttt{B9} & \texttt{DA} & \texttt{4D} & \texttt{D1} & \texttt{9E} & \texttt{17} & \texttt{83} & \texttt{86} \\ \texttt{D0} & \texttt{9D} & \texttt{26} & \texttt{2C} & \texttt{5D} & \texttt{9F} & \texttt{6D} & \texttt{75} \\ \texttt{52} & \texttt{A9} & \texttt{07} & \texttt{6C} & \texttt{B9} & \texttt{8F} & \texttt{70} & \texttt{17} \\ \texttt{87} & \texttt{28} & \texttt{3A} & \texttt{5A} & \texttt{F4} & \texttt{33} & \texttt{0B} & \texttt{6C} \\ \texttt{74} & \texttt{51} & \texttt{15} & \texttt{CF} & \texttt{09} & \texttt{A4} & \texttt{62} & \texttt{09} \\ \texttt{0B} & \texttt{31} & \texttt{7F} & \texttt{86} & \texttt{BE} & \texttt{05} & \texttt{83} & \texttt{34} \end{bmatrix} \tag{11.4}$$

The branch number of λ is 9 $(= n + 1)$.

As explained in Chap. 4, λ can be implemented efficiently by extending the tables that specify the substitution boxes. In SHARK, there are n tables, requiring $n \times 256$ bytes of memory each. When $n = 8$, this gives a total of 16 kB.

The non-linear transformation. The non-linear transformation is a bricklayer permutation of S-boxes operating on bytes, denoted by γ. The same S-box is used for all byte positions. We have

$$\gamma : \mathbf{b} = \gamma(\mathbf{a}) \Leftrightarrow b_i = \mathrm{S}_\gamma(a_i), \tag{11.5}$$

where S_γ is an invertible 8-bit substitution table or S-box.

As in Rijndael, the S-box of SHARK is based on the function $F(x) = x^{-1}$ over GF(2^8), as proposed by K. Nyberg in [74]. An affine transformation is added in order to make the description of the S-boxes less simple. This transformation is not equivalent to the transformation that is applied in the S-boxes of Rijndael.

The round key application. In [81], two alternative ways to introduce the round key in the round transformation are proposed. The first is a key addition in the form of a bitwise XOR of the state with a round key, the second version uses a key-controlled affine transform.

XOR. In the first alternative, the 64 state bits are modified by means of an XOR with a 64-bit round key. This operation is denoted $\sigma_\oplus[\mathbf{k}^{(r)}]$. The resulting cipher is a key-iterated cipher with all its advantages, see Chap. 9. A limitation of the simple scheme is that the entropy of the round key is 'only' 64 bits.

Affine transformation. Let $\kappa^{(t)}$ be a key dependent invertible 8×8 matrix over GF(2^8). The second alternative for the key application is then denoted by $\sigma_{\mathrm{AT}}[\kappa^{(t)}, \mathbf{k}^{(t)}]$ and defined as

$$\sigma_{\mathrm{AT}}[\kappa^{(t)}, \mathbf{k}^{(t)}] : \mathbf{b} = \sigma_{\mathrm{AT}}[\kappa^{(t)}, \mathbf{k}^{(t)}](\mathbf{a}) \Leftrightarrow \mathbf{b} = \kappa^{(t)} \times \mathbf{a} \oplus \mathbf{k}^{(t)}. \tag{11.6}$$

The resulting operation on the state is linear. Since the operation has to be invertible, it must be ensured that all $\kappa^{(t)}$ are invertible matrices. Each round now introduces more key material, increasing the amount of round key bits introduced in the key application to 9×64 bits. The computational overhead of this operation is very large. We can restrict the $\kappa^{(t)}$ to a certain subspace, for instance by letting the $\kappa^{(t)}$ be diagonal matrices. The amount of round key bits introduced in the key application then becomes close to 2×64 bits.

The cipher. The round transformation, denoted by ρ, consists of a sequence of two steps:

$$\rho = \lambda \circ \gamma. \tag{11.7}$$

SHARK is defined with seven rounds, followed by a final round where the mixing step is absent. The applications of the round transformation are interleaved with nine round key applications.

The key schedule. The key schedule expands the key $\mathbf{K}$ to the round keys $\mathbf{K}^{(t)}$. The key schedule of SHARK operates in the following way. The cipher key is concatenated with itself until it has a length of 9×64 bits, or 9×128 bits for the extended version. This string is encrypted with SHARK in CFB mode, using a fixed key. The first 448 bits of the output form the round keys $\mathbf{k}^{(t)}$. For the extended version, the next 448 bits are used to form the diagonal elements of the matrices $\kappa^{(t)}$. If one of these elements is zero, then it is discarded and all the following values are shifted down one place. An extra encryption of the all-zero string is added at the end to provide the extra diagonal elements. The fixed key used during the key schedule is formed in the following way. The matrices $\kappa^{(t)}$ are equal to the identity matrix. The vectors $\mathbf{k}^{(t)}$ are taken from an expanded substitution table, that is used in the combined implementation of the non-linear step and the mixing step.

While this mechanism for round key generation in principle makes it possible to use a key of $64 \times 9 (\times 2)$ bits, it is suggested that the key length should not exceed 128 bits.

11.3 Square

Square can be considered as an extension of the simple SHARK variant where the mixing step is changed, a byte transposition step has been introduced, and an efficient and elegant key schedule has been introduced. Square has a block length of 128 bits and a key length of 128 bits.

The structure. The round transformation of Square is almost identical to the round transformation of Rijndael when the block length equals 128 bits. The round transformation consists of a sequence of three distinct steps that operate on the *state*: a 4×4 array of bytes. The element of a state $\mathbf{a}$ in row i and column j is specified as $a_{i,j}$. Both indexes start from 0. The steps are illustrated in Fig. 11.1.

The mixing step. The mixing step θ is similar to `MixColumns` in Rijndael, except that it operates on the rows of the state instead of the columns. We have

$$\theta(a) = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \times \begin{bmatrix} 02 & 01 & 01 & 03 \\ 03 & 02 & 01 & 01 \\ 01 & 03 & 02 & 01 \\ 01 & 01 & 03 & 02 \end{bmatrix}, \tag{11.8}$$

where the multiplication is in $\mathrm{GF}(2^8)$. The coefficients have been chosen to maximise the branch number of θ, and to facilitate the implementation on 8-bit processors.

The byte transposition. The byte transposition π interchanges rows and columns of a state. If the state is considered as a matrix, it corresponds to the matrix transposition operation. We have

$$\pi : \mathbf{b} = \pi(\mathbf{a}) \Leftrightarrow b_{i,j} = a_{j,i}. \tag{11.9}$$

π is an involution, hence $\pi^{-1} = \pi$.

The non-linear step. The non-linear step γ is a bricklayer permutation operating on bytes. We have

$$\gamma : \mathbf{b} = \gamma(\mathbf{a}) \Leftrightarrow b_{i,j} = \mathrm{S}_\gamma(a_{i,j}), \tag{11.10}$$

where S_γ is an invertible 8-bit substitution table or S-box. The S-box of Square is exactly the same as the S-box of SHARK.

The key addition. The key addition with round key $\mathbf{k}^{(t)}$ is denoted by $\sigma[\mathbf{k}^{(t)}]$. It is identical to the key addition in Rijndael, and the simple key application of SHARK.

The cipher. The round transformation ρ is a sequence of three steps:

$$\rho = \pi \circ \gamma \circ \theta. \tag{11.11}$$

Square is defined as eight rounds interleaved with nine key addition steps. These transformations are preceded by an initial application of θ^{-1}. Note that the θ^{-1} before $\sigma[\mathbf{k}^{(0)}]$ can be incorporated in the first round. The initial θ^{-1} can be discarded by omitting θ in the first round and applying $\theta(\mathbf{k}^{(0)})$ instead of $\mathbf{k}^{(0)}$. The same simplification can be applied to the algorithm for decryption.

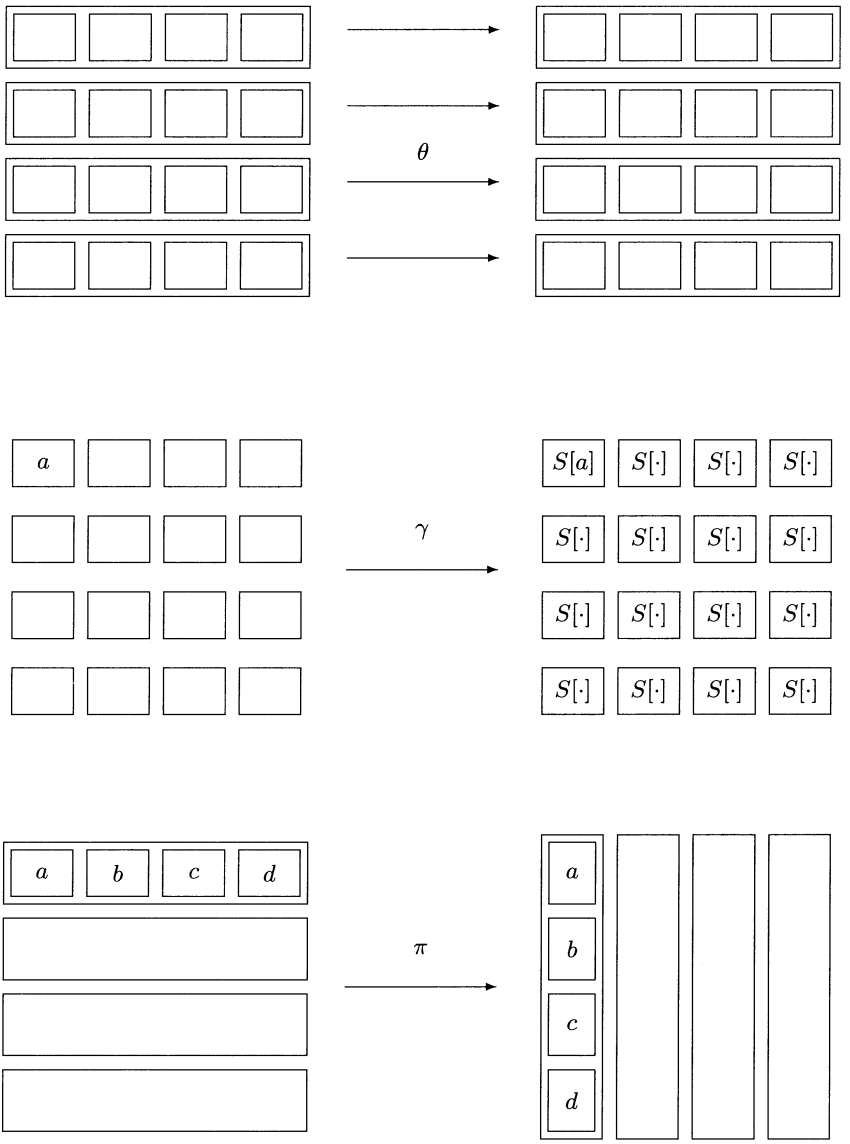

Fig. 11.1. The basic operations of Square. θ is a mixing step with 4 parallel linear transformations. γ consists of 16 separate substitutions. π is a transposition.

The key schedule. The key schedule is linear. It is defined in terms of the rows of the key. We can define a left byte-rotation operation $\mathsf{rotl}(a_i)$ on a row as

$$\mathsf{rotl}[a_{i,0}a_{i,1}a_{i,2}a_{i,3}] = [a_{i,1}a_{i,2}a_{i,3}a_{i,0}] \tag{11.12}$$

and a right byte-rotation $\mathsf{rotr}(a_i)$ as its inverse.

The round keys $\mathbf{k}^{(t)}$ are derived from the cipher key $\mathbf{K}$ in the following way. $\mathbf{k}^{(0)}$ equals the cipher key $\mathbf{K}$. The other round keys are derived iteratively by means of an invertible affine transformation, called 'the round key evolution' and denoted by ψ.

$$\psi : \mathbf{k}^{(t)} = \psi(\mathbf{k}^{(t-1)}) \tag{11.13}$$

The round key evolution ψ and is defined by

$$\begin{aligned}
\mathbf{k}_0^{(t+1)} &= \mathbf{k}_0^{(t)} \oplus \mathsf{rotl}(\mathbf{k}_3^{(t)}) \oplus C^{(t)} \\
\mathbf{k}_1^{(t+1)} &= \mathbf{k}_1^{(t)} \oplus \mathbf{k}_0^{(t+1)} \\
\mathbf{k}_2^{(t+1)} &= \mathbf{k}_2^{(t)} \oplus \mathbf{k}_1^{(t+1)} \\
\mathbf{k}_3^{(t+1)} &= \mathbf{k}_3^{(t)} \oplus \mathbf{k}_2^{(t+1)}
\end{aligned} \tag{11.14}$$

The round constants $C^{(t)}$ are also defined iteratively. We have $C^{(0)} = 1$ and $C^{(t)} = 2 \cdot C^{(t-1)}$.

11.4 BKSQ

BKSQ is an iterated block cipher with a block length of 96 bits and a key length of 96, 144 or 192 bits. It is especially suited to be implemented on a smart card. Its block length of 96 bits allows it to be used as a (second) pre-image resistant one-way function. Most available block ciphers have block lengths of 64 or 128 bits. A block length of 64 bits is currently perceived as being too small for a secure one-way function. Using a block cipher with a block length of 128 bits often leads to one-way functions that are significantly slower. BKSQ is tailored towards these applications. Still, it can also be used for efficient MACing and encryption on a smart card.

The structure. Let the input of the cipher be denoted by a string of 12 bytes: $p_0p_1 \ldots p_{11}$. These bytes can be rearranged into a 3×4 array, or state $\mathbf{a}$.

$$\mathbf{a} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix} = \begin{bmatrix} p_0 & p_3 & p_6 & p_9 \\ p_1 & p_4 & p_7 & p_{10} \\ p_2 & p_5 & p_8 & p_{11} \end{bmatrix} \tag{11.15}$$

The basic building blocks of the cipher operate on this array. Figure 11.2 gives a graphical illustration of the building blocks.

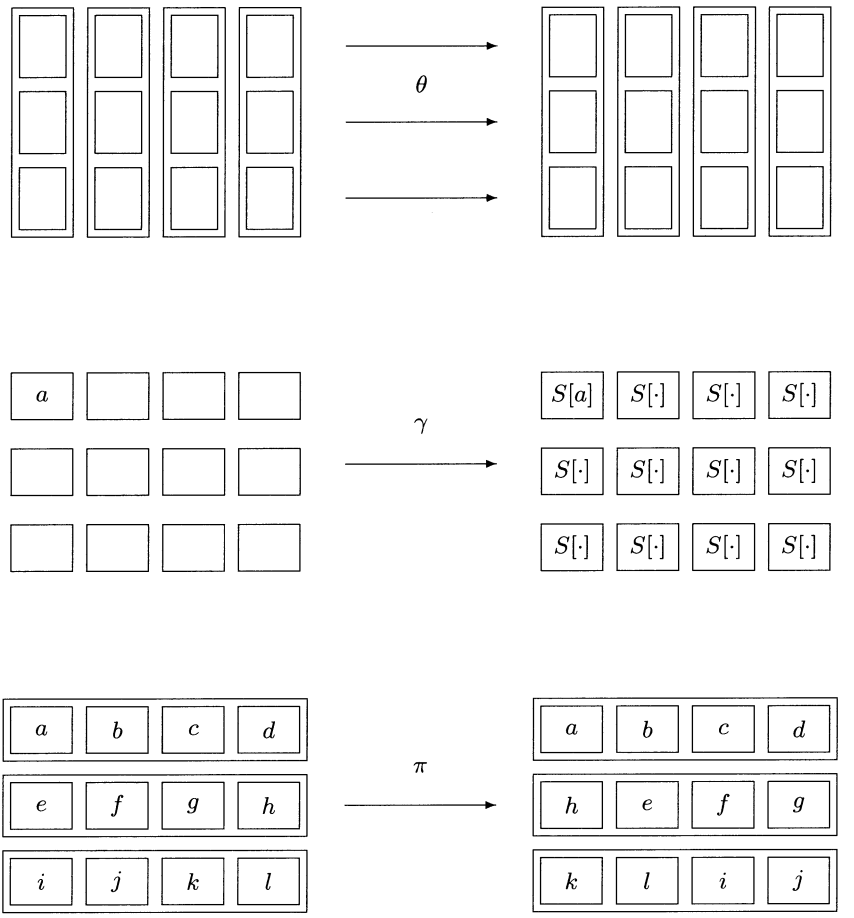

Fig. 11.2. The basic operations of BKSQ. θ is a mixing step with 4 parallel linear transformations. γ consists of 12 separate substitutions. π is a shift of the rows.

The linear transformations. BKSQ uses two linear transformations. The first transformation is similar to `MixColumns` in Rijndael, except that it operates on columns of length 3 instead of length 4. This transformation is denoted by θ. We have

$$\theta(\mathbf{a}) = \begin{bmatrix} 03\ 02\ 02 \\ 02\ 03\ 02 \\ 02\ 02\ 03 \end{bmatrix} \times \begin{bmatrix} a_{0,0}\ a_{0,1}\ a_{0,2}\ a_{0,3} \\ a_{1,0}\ a_{1,1}\ a_{1,2}\ a_{1,3} \\ a_{2,0}\ a_{2,1}\ a_{2,2}\ a_{2,3} \end{bmatrix}. \tag{11.16}$$

This choice for the coefficients makes it possible to implement θ very efficiently on an 8-bit processor with limited working memory.

The second linear transformation is a byte permutation, denoted by π. The effect of π is a shift of the rows of a state. Every row is shifted a different amount. We have

$$\pi : \mathbf{b} = \pi(\mathbf{a}) \Leftrightarrow b_{i,j} = a_{i,j-i}. \tag{11.17}$$

The effect of π is that for every column of $\mathbf{a}$, the three elements are moved to three different columns in $\pi(\mathbf{a})$.

The non-linear transformation. The non-linear transformation is a bricklayer permutation operating on bytes, denoted by γ. It operates on all bytes in the same way. We have

$$\gamma : \mathbf{b} = \gamma(\mathbf{a}) \Leftrightarrow b_{i,j} = \mathrm{S}_\gamma(a_{i,j}), \tag{11.18}$$

where S_γ is an invertible 8-bit substitution table or S-box. The inverse of γ consists of the application of the inverse substitution S_γ^{-1} to all bytes of a state. The S-box of BKSQ is exactly the same as the S-box of Rijndael.

The key addition. The key addition with key $\mathbf{k}^{(t)}$ is denoted by $\sigma[\mathbf{k}^{(t)}]$. It is defined analogously to the key addition of Square and Rijndael.

The cipher. The round transformation denoted by ρ is a sequence of three steps:

$$\rho = \pi \circ \gamma \circ \theta. \tag{11.19}$$

BKSQ is defined as R times the round operation, interleaved with $R + 1$ applications of the key addition and preceded by θ^{-1}:

$$\mathrm{BKSQ}[\mathbf{k}] = \sigma[\mathbf{k}^{(R)}] \circ \rho \circ \sigma[\mathbf{k}^{(R-1)}] \circ \rho \circ \ldots \circ \rho \circ \sigma[\mathbf{k}^{(0)}] \circ \theta^{-1} \tag{11.20}$$

The number of rounds R depends on the key length that is used. For 96-bit keys, there are 10 rounds; for 144-bit keys, there are 14 rounds; and for 192-bit keys, the number of rounds is 18.

The key schedule. The derivation of the round keys $\mathbf{k}^{(t)}$ from the cipher key K is very similar to the key schedule of Rijndael. The round keys $\mathbf{k}^{(t)}$ are extracted from an expanded key array, denoted by W:

$$\mathbf{k}^{(t)} = \mathsf{W}[\cdot][4t] \parallel \mathsf{W}[\cdot][4t+1] \parallel \mathsf{W}[\cdot][4t+2] \parallel \mathsf{W}[\cdot][4t+3]. \tag{11.21}$$

As in Rijndael, the expansion of the cipher key $\mathbf{K}$ into the expanded key array W depends on the length of the cipher key. Let L denote the key length divided by 24. The array is constructed by repeated application of an invertible non-linear transformation ψ: the first L columns are the columns of $\mathbf{K}$, the next L are given by $\psi(\mathbf{K})$, the following columns are given by $\psi(\psi(\mathbf{K}))$, etc. The transformation ψ operates on blocks of L columns and is defined in terms of the XOR operation, a byte-rotation rot that rotates the bytes of a column, and a non-linear substitution γ' that operates in exactly the same way as γ, but takes as argument column vectors instead of arrays. For a detailed description of the key schedule, we refer to [25].

11.5 Children of Rijndael

The design principles of Square and, more recently, Rijndael have been incorporated in other block cipher designs. We list here some recently designed block ciphers that are based on Square and Rijndael, or that have inherited some features from them.

11.5.1 Crypton

Crypton was designed by C. Lim [59, 60]. It is one of the 15 block ciphers that was accepted as an AES candidate. The round transformation of Crypton resembles to a large extent the round transformation of Square. Different versions of Crypton have been published. We discuss here the version that was submitted to the AES process. The differences between Crypton and Square are the following:

1. **non-linear step.** Crypton uses an S-box that is constructed from a three-round Feistel structure with three different 4×4 S-boxes. Both in encryption and decryption operation, eight state bytes are transformed with the S-box, and eight with the inverse S-box.
2. **transposition step.** Crypton keeps the transposition from Square (denoted by π in this book),

3. **mixing step.** Crypton replaces θ by two different linear transformations: one for use in even-numbered rounds and one for use in odd-numbered rounds. These mixing steps have branch number 4, where Square and Rijndael use a mixing step with branch number 5. As opposed to Rijndael and its predecessors, the mixing steps of Crypton are specified by means of mask and XOR operations, and cannot be described by a simple matrix multiplication over GF(2^8).
4. **number of rounds.** Crypton uses 12 rounds, whereas Square uses eight.
5. **key schedule.** The key schedule of Crypton is more complex than that of Square. A disadvantage is the existence of 2^{32} weak keys, as described by J. Borst in [13].

11.5.2 Twofish

Twofish was designed by N. Ferguson et al. [83]. It is one of the five finalists in the AES evaluation process. Its round transformation is based on the Feistel structure. The designers of Twofish used elements and ideas from several other block ciphers. From Square, they used the idea to mix the outputs of 4 non-linear S-boxes by means of a linear transformation that is based on an MDS code.

11.5.3 Anubis

Anubis [4] is one of the block ciphers that has been submitted to the NESSIE process [73]. Its structure is very similar to that of Rijndael, but different choices have been made for various modules. The most important differences are the following:

1. **The involutional structure.** All steps of the round transformation of Anubis are involutions. This should in principle reduce the program size in software or chip area in hardware applications that implement both encryption and decryption.
2. **The different S-box.** The S-box of Anubis is constructed by connecting five 4-bit S-boxes. This choice makes it easier to design efficient implementations in hardware. Furthermore, the polynomial expression for the S-box becomes more complex. An expected disadvantage is the suboptimal behaviour with respect to differential and linear cryptanalysis.
3. **A more complex key schedule.** The expected advantage is the improved resistance against key-based attacks, in particular the shortcuts for long keys. The disadvantage is the higher cost: slower execution, a reduced key agility, a longer program or a higher gate count.

11.5.4 Grand Cru

Grand Cru was designed by J. Borst [14]. It is one of the block ciphers that have been submitted to the NESSIE process [73]. Its structure is very similar to that of Rijndael:

1. The transformations `SubBytes`, `AddRoundKey` and `MixColumns` are copied unchanged.
2. The transformation `ShiftRows` is replaced by a transformation that shifts the rows and the columns over amounts that depend on the value of the round key.
3. A keyed byte-wise rotation is added, rotating each byte of the state by an amount that depends on the value of the round key.
4. Extra initial and final key addition are added, using bytewise modular addition. An extra non-linear transformation is added at the beginning of the cipher, and its inverse is added at the end.

11.5.5 Hierocrypt

Hierocrypt-3 and Hierocrypt-L1 are block ciphers that have been submitted to the NESSIE process [88, 89]. The round transformation of the ciphers incorporates two different linear transformations to mix the outputs of the non-linear S-boxes. One of the linear transformations is based on an MDS code over $\mathrm{GF}(2^8)$, and the other one is based on an MDS code over $\mathrm{GF}(2^{32})$.

11.6 Conclusion

Rijndael is the result of a long design process with continuous improvements along the way. The earliest related design, SHARK, dates back to 1995. Most of the predecessors of Rijndael have been scrutinized intensively by cryptanalysts looking for security flaws, and by programmers interested in efficient implementations. The result of all this work has been taken into account in the design of Rijndael.

The design approach we used for Square and Rijndael has been adopted enthusiastically by a number of cipher designers all over the world. This demonstrates a worldwide belief that the strategy used is sound.

A. Propagation Analysis in Galois Fields

In the specification of Rijndael, we have extensively used operations in a finite field, where the bytes of the state and key represent elements of $\mathrm{GF}(2^8)$. Still, as for most block ciphers, Rijndael operates on plaintext blocks, ciphertext blocks and keys that are strings of bits. Apart from some exceptions such as interpolation attacks, cryptanalysis of ciphers is also generally conducted at the bit level. For example, linear cryptanalysis (see Chap. 7) exploits high correlations between linear combinations of bits of the state in different stages of the encryption process. Differential cryptanalysis (see Chap. 8) exploits high propagation probabilities between bitwise differences in the state in different stages of the encryption.

Later in this appendix we demonstrate how Rijndael can be specified completely with operations in $\mathrm{GF}(2^8)$. How the elements of $\mathrm{GF}(2^8)$ are represented in bytes can be seen as a detail of the specification. Addressing this representation issue in the specifications is important for different implementations of Rijndael to be interoperable, but not more so than for instance the ordering of the bits within the bytes, or the way the bytes of the plaintext and ciphertext blocks are mapped onto the state bytes. It may well be possible that taking a different finite field representation may lead to more efficient implementations.

We can make abstraction from the representation of the elements of $\mathrm{GF}(2^8)$ and consider a block cipher that operates on strings of elements of $\mathrm{GF}(2^8)$. We call this generalisation RIJNDAEL-GF. Rijndael can be seen as an instance of RIJNDAEL-GF, where the representation of the elements has been specified. In principle, this can be applied to most block ciphers. Each block cipher for which the block length and the key length are a multiple of n can in principle be generalized to operate on strings of elements of $\mathrm{GF}(2^n)$. However, unlike for Rijndael, the specification of these generalized ciphers may become quite complicated.

Intuitively, it seems obvious that if Rijndael has a cryptographic weakness, this is inherited by RIJNDAEL-GF and any instance of it, whatever the representation of the elements of $\mathrm{GF}(2^8)$. Still, in the propagation analysis as described in Chap. 7 and 8, we work at the bit level and must assume a specific representation to study the propagation properties. In this appendix we demonstrate how to conduct differential and linear propagation analysis at the level of elements of $\mathrm{GF}(2^n)$, without having to deal with representation issues.

This appendix is mainly devoted to functions over fields with a characteristic of two. However, some of the properties and theorems are valid for any finite field. In those cases, we have treated the more general case; the fields with characteristic two are just a particular case. We start by describing difference propagation and correlation properties of functions over $\mathrm{GF}(2^n)$, with the focus on linear functions. This is further generalized to functions over $\mathrm{GF}(2^n)^\ell$. We then discuss representations and bases in $\mathrm{GF}(p)^n$. Subsequently we show how propagation in functions over $\mathrm{GF}(2^n)$ maps to propagation in Boolean functions by the choice of a basis. Subsequently, we prove two theorems that relate representations of linear functions in $\mathrm{GF}(p)^n$ and functions in $\mathrm{GF}(p^n)$ that are linear over $\mathrm{GF}(p)$. We illustrate this with an example of a function over $\mathrm{GF}(2^3)$. We conclude by specifying RIJNDAEL-GF.

For readability, the notation we use in this appendix differs slightly from the notation in the rest of the book. The addition in a finite field is denoted by the symbol $+$ and the corresponding summation by $\sum$.

A.1 Functions over $\mathrm{GF}(2^n)$

In this section we study the differential propagation and correlation properties of the functions over $\mathrm{GF}(2^n)$:

$$f : \mathrm{GF}(2^n) \to \mathrm{GF}(2^n) : a \mapsto b = f(a). \tag{A.1}$$

All functions over $\mathrm{GF}(2^n)$ can be expressed as a polynomial over $\mathrm{GF}(2^n)$ of degree at most $2^n - 1$:

$$f(a) = \sum_{i=0}^{2^n-1} c_i a^i. \tag{A.2}$$

Given a table specification where the output value $f(a)$ is given for the 2^n different input values a, the 2^n coefficients of this polynomial can be found by applying Lagrange interpolation [58, p. 28]. On the other hand, given a polynomial expression, the table specification can be found by evaluating the polynomial for all values of a.

A.1.1 Difference Propagation

As for Boolean functions (see Sect. 8.1), we have difference propagation probabilities over f:

$$\text{Prob}^f(a', b') = 2^{-n} \sum_a \delta(b' + f(a \oplus a') + f(a)), \tag{A.3}$$

where $+$ denotes the addition in GF(2^n). A difference a' propagates to a difference b' through f iff:

$$f(a) + f(a + a') = b'. \tag{A.4}$$

The difference propagation probability $\text{Prob}^f(a', b')$ is equal to the total number of values a that satisfy this equation, divided by 2^n. By using the polynomial expression for f, Equation (A.4) becomes:

$$\sum_i c_i a^i + \sum c_i (a + a')^i + b' = 0. \tag{A.5}$$

This is a polynomial equation in a. For certain special cases, the number of solutions of these polynomials can be analytically determined providing provable bounds for difference propagation probability. Examples can be found in the paper of K. Nyberg on S-boxes [74].

A.1.2 Correlation

In Boolean functions, correlation is defined between parities (see Chap. 7). Parities are linear combinations of bits, as determined by a selection pattern. For a function over GF(2^n), individual bits cannot be distinguished without adopting a representation, and hence speaking about parities does not make sense. A parity is a function that maps GF$(2)^n$ to GF(2), which is linear over GF(2). In GF(2^n), we can find functions with the same properties. For that purpose, we first must introduce the *trace* function in a finite field.

Definition A.1.1. *Let x be an element of* GF(p^n). *The* trace *of x over* GF(p) *is defined by*

$$\text{Tr}(x) = x + x^p + x^{p^2} + x^{p^3} + \cdots + x^{p^{n-1}}. \tag{A.6}$$

The trace is linear over GF(p) (see Sect. 2.1.2):

$$\forall\, x, y \in \text{GF}(p^n) : \ \text{Tr}(x + y) = \text{Tr}(x) + \text{Tr}(y) \tag{A.7}$$

$$\forall\, a \in \text{GF}(p), \forall\, x \in \text{GF}(p^n) : \ \text{Tr}(ax) = a\text{Tr}(x). \tag{A.8}$$

From (A.6) we can derive

$$(\mathrm{Tr}(x))^p = \mathrm{Tr}(x^p) = \mathrm{Tr}(x) \tag{A.9}$$

and

$$(\mathrm{Tr}(x))^{p^t} = \mathrm{Tr}(x^{p^t}) = \mathrm{Tr}(x), \quad t = 2, \ldots, n. \tag{A.10}$$

It follows that the trace of x has an order that divides p and hence is an element of GF(p). In the field GF(2^n), the trace of an element is in GF(2). As the trace map is linear over GF(2), it follows that all functions of the form

$$f(a) = \mathrm{Tr}(wa) \tag{A.11}$$

are two-valued functions of GF(2^n), which are linear over GF(2). There are exactly 2^n such functions, one for each value of w. We call the function $\mathrm{Tr}(wa)$ a *trace parity*, and the corresponding value w a *trace pattern*. In the analysis of correlation properties of functions over GF(2^n), trace parities play the role that is played by the parities in the correlation analysis of Boolean functions (see Chap. 7). When a representation is chosen, these functions can be mapped one-to-one to parities (see Sect. A.4.2).

By working with trace patterns, it is possible to study correlation properties in functions over GF(2^n) without having to specify a basis. Hence, the obtained results are valid for all choices of basis. Once a basis is chosen, trace patterns can be converted to selection patterns (see Theorem A.4.1).

For a function f over GF(2^n), we denote the correlation between an input trace parity $\mathrm{Tr}(wa)$ and an output trace parity $\mathrm{Tr}(uf(a))$ by $C^f_{u,w}$. We have

$$C^f_{u,w} = 2^{-n} \sum_a (-1)^{\mathrm{Tr}(wa)} (-1)^{\mathrm{Tr}(uf(a))} \tag{A.12}$$

$$= 2^{-n} \sum_a (-1)^{\mathrm{Tr}(wa)+\mathrm{Tr}(uf(a))} \tag{A.13}$$

$$= 2^{-n} \sum_a (-1)^{\mathrm{Tr}(wa+uf(a))}. \tag{A.14}$$

The value of this correlation is determined by the number of values a that satisfy

$$\mathrm{Tr}(wa + uf(a)) = 0. \tag{A.15}$$

If this equation is satisfied by r values a, the correlation $C^f_{u,w}$ is equal to $2^{1-n}r$. If it has no solutions, the correlation is -1; if it is satisfied by all values a, the correlation is 1; and if it is satisfied by exactly half of the possible values a, the correlation is 0. By using the polynomial expression for f, (A.15) becomes a polynomial equation in a:

$$\mathrm{Tr}(wa + u \sum_i c_i a^i) = 0. \tag{A.16}$$

As in the case for differential propagation, for some cases the number of solutions of these polynomials can be analytically determined providing provable bounds for correlation properties [74].

Example A.1.1. Let us consider the following operation:

$$b = f(a) = a + \mathrm{c}, \tag{A.17}$$

where c is a constant. A difference in a fully determines the difference in b:

$$b' = f(a) + f(a + a') = (a + \mathrm{c}) + (a + a' + \mathrm{c}) = a'. \tag{A.18}$$

Hence the addition of a constant has no effect on the difference pattern. For the correlation we can find the number of solutions of

$$\mathrm{Tr}(wa + uf(a)) = 0, \tag{A.19}$$

which is equivalent to

$$\mathrm{Tr}(wa + ua + u\mathrm{c}) = \mathrm{Tr}((w + u)a + u\mathrm{c}) = 0. \tag{A.20}$$

If $w + u$ is different from 0, the trace is zero for exactly half of the values of a, and the correlation is 0. If $w = u$ this becomes

$$\mathrm{Tr}(u\mathrm{c}) = 0. \tag{A.21}$$

This equation is true for all values of a if $\mathrm{Tr}(u\mathrm{c}) = 0$, and has no solutions if $\mathrm{Tr}(u\mathrm{c}) = 1$. It follows that the addition of a constant has no effect on the trace pattern and that the sign of the correlation is equal to $(-1)^{\mathrm{Tr}(u\mathrm{c})}$.

A.1.3 Functions that are Linear over GF(2^n)

The functions of GF(2^n) that are linear over GF(2^n) are of the form

$$f(a) = l^{(0)}a, \tag{A.22}$$

where $l^{(0)}$ is an element of GF(2^n). Hence, there are exactly 2^n functions over GF(2^n) that are linear over GF(2^n). A difference in a fully determines the difference in b:

$$b' = l^{(0)}a'. \tag{A.23}$$

For the correlation we can find the number of solutions of

$$\mathrm{Tr}(wa + uf(a)) = 0, \tag{A.24}$$

which is equivalent to

$$\mathrm{Tr}(wa + ul^{(0)}a) = \mathrm{Tr}((w + ul^{(0)})a) = 0. \tag{A.25}$$

If the factor of a is different from 0, the correlation is 0. The correlation between $\mathrm{Tr}(wa)$ and $\mathrm{Tr}(ub)$ is equal to 1 if

$$w = l^{(0)}u. \tag{A.26}$$

A.1.4 Functions that are Linear over GF(2)

A function over $\mathrm{GF}(2^n)$ is linear over $\mathrm{GF}(2)$ if it satisfies the following

$$\forall\, x, y \in \mathrm{GF}(2^n) : \; f(x + y) = f(x) + f(y) \tag{A.27}$$

$$\forall\, a \in \mathrm{GF}(2), \forall\, x \in \mathrm{GF}(2^n) : \; \mathrm{Tr}(ax) = a\mathrm{Tr}(x). \tag{A.28}$$

Observe that a function that is linear over $\mathrm{GF}(2^n)$ is also linear over $\mathrm{GF}(2)$, but a function that is linear over $\mathrm{GF}(2)$ is not necessarily linear over $\mathrm{GF}(2^n)$. Moreover, a function that satisfies (A.27) automatically satisfies (A.28). For example, the function $f(x) = x^2$ is linear over $\mathrm{GF}(2)$, but not over $\mathrm{GF}(2^n)$:

$$\begin{aligned} f(x + y) &= (x + y)^2 = x^2 + xy + yx + y^2 = x^2 + y^2 \\ &= f(x) + f(y) \\ f(ax) &= a^2 f(x) \neq af(x) \text{ if } a \notin \mathrm{GF}(2). \end{aligned}$$

This can be extended to all functions of the form $f(x) = x^{2^t}$. Moreover, any linear combination of these functions is linear over $\mathrm{GF}(2)$. It follows that all functions of the form

$$f(a) = \sum_{t=0}^{n-1} l^{(t)} a^{2^t} \;, \text{ with } l^{(t)} \in \mathrm{GF}(2^n), \tag{A.29}$$

are linear over $\mathrm{GF}(2)$. Moreover, all functions of $\mathrm{GF}(2^n)$ that are linear over $\mathrm{GF}(2)$ can be represented in this way.

From (A.27) it follows that a difference in a fully determines the difference in b by:

$$b' = \sum_{t=0}^{n-1} l^{(t)} a'^{2^t}. \tag{A.30}$$

The relation between the trace pattern at the input and the trace pattern at the output is less trivial.

Theorem A.1.1. *For a function* $b = \sum_{t=0}^{n-1} l^{(t)} a^{2^t}$ *an output trace parity* $\mathrm{Tr}(ub)$ *is correlated to input trace parity* $\mathrm{Tr}(wa)$ *with a correlation of 1 iff*

$$w = \sum_{t=0}^{n-1} (l^{(n-t \bmod n)} u)^{2^t}. \tag{A.31}$$

Proof. We will prove that $\mathrm{Tr}(wa) = \mathrm{Tr}(ub)$ and hence that $\mathrm{Tr}(wa + ub) = 0$ for all values of a if w is given by (A.31). All computations with variables t, s and r are performed in modulo n, and all summations are from 0 to $n-1$.

$$\begin{aligned}
\mathrm{Tr}(wa) &= \mathrm{Tr}(ub) \\
\mathrm{Tr}\left(\sum_t (l^{(n-t)} u)^{2^t} a\right) &= \mathrm{Tr}\left(u \sum_t l^{(t)} a^{2^t}\right) \\
\sum_s \left(\sum_t l^{(n-t)^{2^t}} u^{2^t} a\right)^{2^s} &= \sum_s \left(\sum_t l^{(t)} u a^{2^t}\right)^{2^s} \\
\sum_s \sum_t l^{(n-t)^{2^{s+t}}} u^{2^{s+t}} a^{2^s} &= \sum_s \sum_t l^{(t)^{2^s}} u^{2^s} a^{2^{s+t}} \\
\sum_s \sum_t l^{(n-t)^{2^{s+t}}} u^{2^{s+t}} a^{2^s} &= \sum_{r=s+t} \sum_t l^{(t)^{2^{r-t}}} u^{2^{r-t}} a^{2^r} \\
\sum_s \sum_{r=n-t} l^{(r)^{2^{s-r}}} u^{2^{s-r}} a^{2^s} &= \sum_s \sum_t l^{(t)^{2^{s-t}}} u^{2^{s-t}} a^{2^s} \\
\sum_s \sum_t l^{(t)^{2^{s-t}}} u^{2^{s-t}} a^{2^s} &= \sum_s \sum_t l^{(t)^{2^{s-t}}} u^{2^{s-t}} a^{2^s}.
\end{aligned}$$

□

A.2 Functions over $(\mathrm{GF}(2^n))^\ell$

In this section we treat the difference propagation and correlation properties of functions that operate on arrays of ℓ elements of $\mathrm{GF}(2^n)$. We denote the arrays by

$$\mathbf{A} = [a_1 \; a_2 \; a_3 \; \ldots \; a_\ell \;]^{\mathrm{T}}. \tag{A.32}$$

where the elements $a_i \in \mathrm{GF}(2^n)$. We have

$$F : \mathrm{GF}(2^n)^\ell \to \mathrm{GF}(2^n)^\ell : \mathbf{A} \mapsto \mathbf{B} = F(\mathbf{A}). \tag{A.33}$$

A.2.1 Difference Propagation

A difference $\mathbf{A}'$ propagates to a difference $\mathbf{B}'$ through F iff

$$F(\mathbf{A}) + F(\mathbf{A} + \mathbf{A}') = \mathbf{B}'. \tag{A.34}$$

The difference propagation probability $\text{Prob}^F(\mathbf{A}', \mathbf{B}')$ is equal to the total number of values $\mathbf{A}$ that satisfy this equation, divided by $2^{n\ell}$.

A.2.2 Correlation

The trace parities can be extended to vectors. We can define a trace pattern vector as

$$\mathbf{W} = [w_1 \; w_2 \; w_3 \; \ldots \; w_\ell]^{\mathrm{T}}. \tag{A.35}$$

where the elements $w_i \in \text{GF}(2^n)$. The trace parities for a vector are of the form

$$\sum \text{Tr}(w_i a_i) = \text{Tr}\left(\sum_i w_i a_i\right) = \text{Tr}(\mathbf{W}^{\mathrm{T}}\mathbf{A}). \tag{A.36}$$

We can define a correlation between an input trace parity $\text{Tr}(\mathbf{W}^{\mathrm{T}}\mathbf{A})$ and an output trace parity $\text{Tr}(\mathbf{U}^{\mathrm{T}}\mathbf{A})$:

$$C^F_{\mathbf{U},\mathbf{W}} = 2^{-n\ell} \sum_{\mathbf{A}} (-1)^{\text{Tr}(\mathbf{W}^{\mathrm{T}}\mathbf{A})} (-1)^{\text{Tr}(\mathbf{W}^{\mathrm{T}} F(\mathbf{A}))} \tag{A.37}$$

$$= 2^{-n\ell} \sum_{\mathbf{A}} (-1)^{\text{Tr}(\mathbf{W}^{\mathrm{T}}\mathbf{A}) + \text{Tr}(\mathbf{W}^{\mathrm{T}} F(\mathbf{A}))} \tag{A.38}$$

$$= 2^{-n\ell} \sum_{\mathbf{A}} (-1)^{\text{Tr}(\mathbf{W}^{\mathrm{T}}\mathbf{A} + \mathbf{W}^{\mathrm{T}} F(\mathbf{A}))}. \tag{A.39}$$

A.2.3 Functions that are Linear over GF(2^n)

If F is linear over $\text{GF}(2^n)$, it can be denoted by a matrix multiplication. We have

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_\ell \end{bmatrix} = \begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} & \cdots & l_{1,\ell} \\ l_{2,1} & l_{2,2} & l_{2,3} & \cdots & l_{2,\ell} \\ l_{3,1} & l_{3,2} & l_{3,3} & \cdots & l_{3,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{\ell,1} & l_{\ell,2} & l_{\ell,3} & \cdots & l_{\ell,\ell} \end{bmatrix} \times \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_\ell \end{bmatrix}. \tag{A.40}$$

The elements of the matrix are elements of $\text{GF}(2^n)$.

Or

$$\mathbf{B} = \mathsf{L}\mathbf{A} \tag{A.41}$$

for short. A difference in $\mathbf{A}$ fully determines the difference in $\mathbf{B}$:

$$\mathbf{B}' = \mathsf{L}\mathbf{A}' \tag{A.42}$$

For the correlation, we have:

$$\begin{aligned} \mathrm{Tr}(\mathbf{W}^{\mathrm{T}}\mathbf{A} + \mathbf{U}^{\mathrm{T}}\mathsf{L}\mathbf{A}) &= \mathrm{Tr}(\mathbf{W}^{\mathrm{T}}\mathbf{A} + (\mathsf{L}^{\mathrm{T}}\mathbf{U})^{\mathrm{T}}\mathbf{A}) && \text{(A.43)} \\ &= \mathrm{Tr}((\mathbf{W} + \mathsf{L}^{\mathrm{T}}\mathbf{U})^{\mathrm{T}}\mathbf{A}). && \text{(A.44)} \end{aligned}$$

Hence, the correlation between $\mathrm{Tr}(\mathbf{W}^{\mathrm{T}}\mathbf{A})$ and $\mathrm{Tr}(\mathbf{U}^{\mathrm{T}}\mathbf{B})$ is equal to 1 if

$$\mathbf{W} = \mathsf{L}^{\mathrm{T}}\mathbf{U}. \tag{A.45}$$

A.2.4 Functions that are Linear over GF(2)

Generalizing equation (A.29) to vectors of $\mathrm{GF}(2^n)$ yields

$$b_i = \sum_j \sum_t l_{i,j}^{(t)} a_j^{2^t} \quad 0 \le i < n. \tag{A.46}$$

If we introduce the following notation:

$$\mathbf{A}^{2^t} = \left[a_1^{2^t} \; a_2^{2^t} \; a_3^{2^t} \; \ldots \; a_\ell^{2^t} \right], \tag{A.47}$$

this can be written as

$$\mathbf{B} = \sum_t \mathsf{L}^{(t)} \mathbf{A}^{2^t}. \tag{A.48}$$

An example of such a linear function is the mixing transformation in the AES candidate Crypton [59].

From (A.27) it follows that a difference in A fully determines the difference in B by

$$\mathbf{B}' = \sum_t \mathsf{L}^{(t)} \mathbf{A}'^{2^t}. \tag{A.49}$$

For the relation between the input trace pattern and the output trace pattern, it can be proven that

$$\mathbf{W} = \sum_t (\mathsf{L}^{(n-t \bmod n)^{\mathrm{T}}} \mathbf{U})^{2^t}. \tag{A.50}$$

A.3 Representations of GF(p^n)

In this section we treat different representations of GF(p^n). We first describe the cyclic representation that simplifies the operation of multiplication. This is followed by a treatment of vector representations of GF(p^n) and dual bases. These play an essential role in the mapping of propagation properties from functions over GF(2^n) to those of Boolean functions.

A.3.1 Cyclic Representation of GF(p^n)

It can be proven that the multiplicative group of GF(p^n) is cyclic. The elements of this group (different from $\mathbf{0}$) can be represented as the $p^n - 1$ powers of a generator $\alpha \in \mathrm{GF}(p^n)$:

$$\forall x \in \mathrm{GF}(p^n) \backslash \{0\}, \exists\, a_x \in \mathbb{Z}_{p^n-1} : x = \alpha^{a_x}. \tag{A.51}$$

In this representation, multiplication of two non-zero elements corresponds to addition of their powers modulo $p^n - 1$:

$$x \cdot y = \alpha^{a_x} \cdot \alpha^{a_y} = \alpha^{a_x + a_y \bmod p^n - 1}. \tag{A.52}$$

Operations such as taking the multiplicative inverse and exponentiation are trivial in this representation. For addition, however, the vector representation (see Sect. A.3.2) is more appropriate. In computations involving both addition and multiplication, one may switch between the two different representations by means of conversion tables. The table used for conversion from the vector representation to the cyclic representation is called a *log table*, and the table for the inverse conversion is called an *alog table*. We have used this principle in our reference implementation (see Appendix E).

A.3.2 Vector Space Representation of GF(p^n)

The elements of a finite field GF(p^n) can be represented as vectors that are elements of an n-dimensional vector space over GF(p), commonly denoted by GF$(p)^n$. The addition of vectors in this vector space corresponds to the addition in GF(p^n). We can choose a basis consisting of n elements $e^{(i)} \in \mathrm{GF}(p^n)$. We depict a basis e by a column vector that has as elements the elements of the basis:

$$\mathsf{e} = \left[e^{(1)}\ e^{(2)}\ \cdots\ e^{(n)} \right]^{\mathrm{T}} \tag{A.53}$$

The elements of GF(p^n)can be represented by their coordinates with respect to this basis. We have

$$a = \sum_i a_i e^{(i)} = \boldsymbol{a}^{\mathrm{T}} \mathsf{e}. \tag{A.54}$$

where a_i is the coordinates of a with respect to the basis e and where $\boldsymbol{a}$ is the column vector consisting of coordinates a_i. The coordinates are elements of GF(p). Given a basis, there is a one-to-one correspondence between field elements in GF(p^n) and their coordinates. The zero element of the field GF(p^n), denoted by $\mathbf{0}$, has coordinates all equal to the zero element of the field GF(p):

$$\mathbf{0} = [0\ 0\ \cdots\ 0]. \tag{A.55}$$

The coordinates of the sum of two vectors are given by the vector sum of the coordinates of the two vectors:

$$\mathbf{d} = \mathbf{b} + \mathbf{c} \Leftrightarrow d_i = b_i + c_i,\ \ 0 < i \leq n. \tag{A.56}$$

Here, the summing of the coefficients occurs in the field GF(p). It follows that the coordinates of the inverse element of $\mathbf{b}$ can be calculated by replacing every coordinate by its inverse in GF(p). In a finite field with a characteristic of 2, each coordinate is its own inverse element with respect to addition. Hence, this holds for each element of the field as well. The polynomial representation is a special case of the vector representation. The basis is of the form $1, x, x^2, x^3, \ldots, x^{n-1}$.

A.3.3 Dual Bases

Coordinates of a field element with respect to a basis can be expressed in terms of the *dual basis* and the *trace* map.

Definition A.3.1. *Two bases* e *and* d *are called* dual bases *if for all* i *and* j *with* $1 \leq i$ *and* $j \leq n$, *it holds that*

$$\mathrm{Tr}(d^{(i)}e^{(j)}) = \delta(i \oplus j), \tag{A.57}$$

Every base has exactly one dual base. Let e and d be dual bases. Then we have

$$\mathrm{Tr}(d^{(j)}a) = \mathrm{Tr}\left(d^{(j)}\sum_{i=1}^{n} a_i e^{(i)}\right) = \sum_{i=1}^{n} a_i \mathrm{Tr}(d^{(j)}e^{(i)}) = a_j. \tag{A.58}$$

Hence the coordinates with respect to basis e can be expressed in an elegant way by means of the trace map and the dual basis d:

$$\boldsymbol{a} = \left[\mathrm{Tr}(d^{(1)}a)\ \mathrm{Tr}(d^{(2)}a)\ \ldots\ \mathrm{Tr}(d^{(n)}a)\right]. \tag{A.59}$$

Applying (A.54) gives:

$$a = \sum_{i=1}^{n} \mathrm{Tr}(d^{(i)}a)e^{(i)} = \sum_{i=1}^{n} \mathrm{Tr}(e^{(i)}a)d^{(i)}. \tag{A.60}$$

A.4 Boolean Functions and Functions in GF(2^n)

Functions of GF(2^n) can be mapped to functions of GF(2)^n by choosing a basis e in GF(2^n). Given

$$f : \mathrm{GF}(2^n) \to \mathrm{GF}(2^n) : a \mapsto b = f(a), \tag{A.61}$$

we can define a Boolean function ϕ:

$$\phi : \mathrm{GF}(2)^n \to \mathrm{GF}(2)^n : \boldsymbol{a} \mapsto \boldsymbol{b} = \phi(\boldsymbol{a}) \tag{A.62}$$

where

$$\begin{aligned} \boldsymbol{a} &= [a_1\ a_2\ \ldots\ a_n] \\ \boldsymbol{b} &= [b_1\ b_2\ \ldots\ b_n], \end{aligned}$$

and

$$\begin{aligned} a_i &= \mathrm{Tr}(ad^{(i)}) \\ b_i &= \mathrm{Tr}(bd^{(i)}). \end{aligned}$$

On the other hand, given a Boolean function ϕ, a function over GF(2^n) can be defined as

$$a = \boldsymbol{a}^{\mathrm{T}}\mathsf{e} \tag{A.63}$$

$$b = \boldsymbol{b}^{\mathrm{T}}\mathsf{e}. \tag{A.64}$$

This can be extended to functions operating on vectors of elements of GF(2^n).

A.4.1 Differences in GF(2)^n and GF(2^n)

Now we can consider how a difference pattern in GF(2)^n maps to a difference pattern in GF(2^n). Thanks to the linearity of the trace map, the mapping between a difference pattern a' in GF(2^n) and a difference pattern $\boldsymbol{a}'$ in GF(2)^n is given by

$$\boldsymbol{a}' = [a'_1\ a'_2\ \ldots\ a'_n], \tag{A.65}$$

where

$$a'_i = \mathrm{Tr}(a'd^{(i)}) \tag{A.66}$$

and

$$a' = \boldsymbol{a}'^{\mathrm{T}}\mathsf{e}. \tag{A.67}$$

A.4.2 Relationship Between Trace Patterns and Selection Patterns

In the following theorem we prove that for every selection pattern a corresponding trace pattern exists. Hence, when studying the propagation of correlations, we can use trace patterns. In this way we avoid the specification of a basis, which is necessary when using selection patterns.

Let the coordinates of a with respect to e be denoted by $\boldsymbol{a}$, and the coordinates of w with respect to d be denoted by $\boldsymbol{w}_\mathrm{d}$, where d and e are dual bases.

Theorem A.4.1. *The relationship between a trace pattern and the corresponding selection pattern is given by*

$$\mathrm{Tr}(wa) = \boldsymbol{w}_\mathrm{d}{}^\mathrm{T}\boldsymbol{a}. \tag{A.68}$$

Proof. Applying (A.60) to w and a, we get

$$\mathrm{Tr}(wa) = \mathrm{Tr}\left(\left(\sum_i \mathrm{Tr}(e^{(i)}w)d^{(i)}\right)\left(\sum_j \mathrm{Tr}(d^{(j)}a)e^{(j)}\right)\right).$$

Since the output of the trace map lies in GF(2), and since the trace map is linear over GF(2), we can convert this to:

$$\begin{aligned}\mathrm{Tr}(wa) &= \sum_i \mathrm{Tr}(e^{(i)}w)\sum_j \mathrm{Tr}(d^{(j)}a)\mathrm{Tr}(d^{(i)}e^{(j)})\\ &= \sum_i \mathrm{Tr}(e^{(i)}w)\sum_j \mathrm{Tr}(d^{(j)}a)\delta(i\oplus j)\\ &= \sum_i \mathrm{Tr}(e^{(i)}w)\mathrm{Tr}(d^{(i)}a).\end{aligned}$$

Applying (A.59) twice completes the proof. □

A.4.3 Relationship Between Linear Functions in GF$(p)^n$ and GF(p^n)

A linear function of GF$(p)^n$ is completely specified by an $n \times n$ matrix M:

$$\boldsymbol{b} = \mathsf{M}\boldsymbol{a}. \tag{A.69}$$

A linear function of GF(p^n) is specified by the n coefficients $l^{(t)} \in \mathrm{GF}(p^n)$ in

$$b = \sum_{t=0}^{n-1} l^{(t)}a^{p^t}. \tag{A.70}$$

After choosing a basis e over GF(p^n), these two representations can be converted to one another.

Theorem A.4.2. *Given the coefficients* $l^{(t)}$ *and a basis* e*, the elements of the matrix* M *are given by*

$$M_{ij} = \sum_{t=0}^{n-1} \mathrm{Tr}\left(l^{(t)} d^{(i)} e^{(j)^{p^t}}\right). \tag{A.71}$$

Proof. We will derive an expression of b_i as a linear combination of a_j in terms of the factors $l^{(t)}$. For a component b_i we have

$$\begin{aligned} b_i &= \mathrm{Tr}(b d^{(i)}) \\ &= \mathrm{Tr}\left(\sum_t l^{(t)} a^{p^t} d^{(i)}\right) \\ &= \sum_t \mathrm{Tr}(l^{(t)} a^{p^t} d^{(i)}). \end{aligned} \tag{A.72}$$

The powers of a can be expressed in terms of the components a_j:

$$\begin{aligned} a^{p^t} &= \left(\sum_j a_j e^{(j)}\right)^{p^t} \\ &= \sum_j a_j e^{(j)^{p^t}}, \end{aligned} \tag{A.73}$$

where we use the fact that exponentiation by p^t is linear over $\mathrm{GF}(p)$ to obtain (A.73). Substituting (A.73) in (A.72) yields

$$\begin{aligned} b_i &= \sum_t \mathrm{Tr}\left(l^{(t)} \sum_j a_j e^{(j)^{p^t}} d^{(i)}\right) \\ &= \sum_t \sum_j \mathrm{Tr}\left(l^{(t)} e^{(j)^{p^t}} d^{(i)} a_j\right) \\ &= \sum_j \left(\sum_t \mathrm{Tr}(l^{(t)} e^{(j)^{p^t}} d^{(i)})\right) a_j. \end{aligned} \tag{A.74}$$

It follows that

$$M_{ij} = \sum_t \mathrm{Tr}\left(l^{(t)} e^{(j)^{p^t}} d^{(i)}\right),$$

proving the theorem. □

Theorem A.4.3. *Given matrix* M *and a basis* e*, the elements* $l^{(t)}$ *are given by*

$$l^{(t)} = \sum_{i=1}^{n} \sum_{j=1}^{n} M_{ij} d^{(j)^{p^t}} e^{(i)}. \tag{A.75}$$

Proof. We will express b as a function of powers of a in terms of the elements of the matrix M. We have

$$b = \sum_i b_i e^{(i)}, \tag{A.76}$$

and

$$\begin{aligned} b_i &= \sum_j M_{ij} a_j \\ &= \sum_j M_{ij} \mathrm{Tr}(a d^{(j)}) \\ &= \sum_j M_{ij} \sum_t a^{p^t} d^{(j)^{p^t}}. \end{aligned} \tag{A.77}$$

Substituting (A.77) into (A.76) yields

$$\begin{aligned} b &= \sum_i \sum_j M_{ij} \sum_t a^{p^t} d^{(j)^{p^t}} e^{(i)} \\ &= \sum_t \left(\sum_i \sum_j M_{ij} d^{(j)^{p^t}} e^{(i)} \right) a^{p^t}. \end{aligned} \tag{A.78}$$

It follows that

$$l^{(t)} = \sum_i \sum_j M_{ij} d^{(j)^{p^t}} e^{(i)},$$

proving the theorem. □

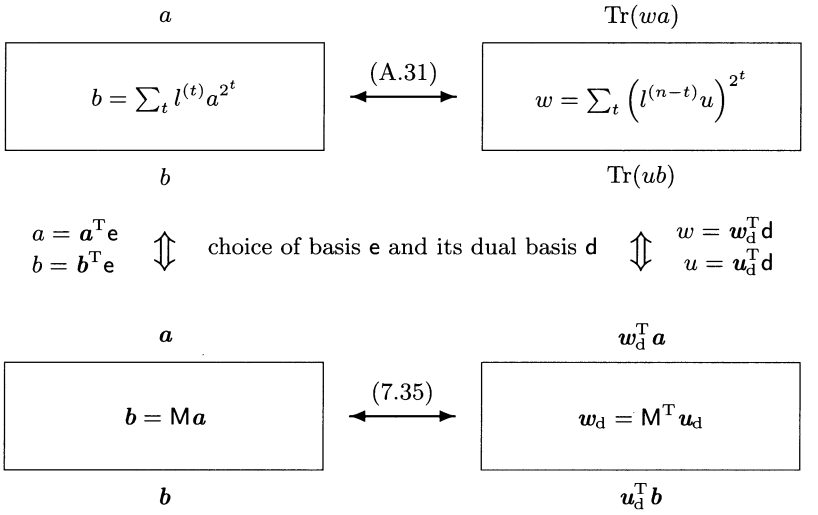

Fig. A.1. The linear trail propagation through a linear function.

A.4.4 Illustration

Figure A.1 and Example A.4.1 illustrate the results for the propagation of linear trails through linear functions of GF(p^n). Remember that we always express the *input* pattern w as a function of the *output* pattern u.

Example A.4.1. We consider the field GF(2^3). Let α be a root of $x^3+x+1 = 0$. Then the elements of GF(2^3) can be denoted by $0, 1, \alpha, \alpha+1, \alpha^2, \alpha^2+1, \alpha^2+\alpha$ and $\alpha^2+\alpha+1$. We consider two transformations f and g, defined by

$$f(a) = \alpha a \tag{A.79}$$

$$g(a) = a^4 + (\alpha^2 + \alpha + 1)a^2. \tag{A.80}$$

For both functions, we want to derive a general expression that for any output trace pattern u gives the input trace pattern w it correlates with. We will denote these expressions by f_{d} and g_{d}, respectively, where the superscript 'd' stands for 'dual'. We can derive f_{d} and g_{d} without having to consider representation issues by applying Theorem A.1.1. For $f(a)$ we have

$$l^{(0)} = \alpha,\ l^{(1)} = l^{(2)} = 0, \tag{A.81}$$

and hence

$$w = f_{\mathrm{d}}(u) = \alpha u. \tag{A.82}$$

For $g(a)$ we have

$$l^{(0)} = 0,\ l^{(1)} = \alpha^2 + \alpha + 1,\ l^{(2)} = 1, \tag{A.83}$$

and hence

$$w = g_{\mathrm{d}}(u) = u^2 + ((\alpha^2 + \alpha + 1)u)^4 = u^2 + (\alpha^2 + 1)u^4. \tag{A.84}$$

Alternatively, we can apply the formulas derived in Chap. 7 by choosing a basis and going via the vector space representation. We start by choosing the basis:

$$\mathsf{e} = [\alpha^2 + \alpha + 1, \alpha + 1, 1]^{\mathrm{T}}. \tag{A.85}$$

Table A.1 shows the coordinates of the elements of GF(2^3), as well as the coordinates of the images of f and g with respect to this basis.

Table A.1. Coordinates of the field elements, and the images of f and g with respect to the basis e.

x	$\boldsymbol{x}$	$\boldsymbol{y} = f(x)$	$\boldsymbol{y} = g(x)$
0	000	000	000
1	001	011	101
$\alpha + 1$	010	101	001
α	011	110	100
$\alpha^2 + \alpha + 1$	100	111	100
$\alpha^2 + \alpha$	101	100	001
α^2	110	010	101
$\alpha^2 + 1$	111	001	000

Once the coordinates of the inputs and outputs of f and g have been determined, we can derive the matrices M and N that describe the functions f and g in the vector space:

$$\mathsf{M} = \begin{bmatrix} 1\,1\,0 \\ 1\,0\,1 \\ 1\,1\,1 \end{bmatrix},\quad \mathsf{N} = \begin{bmatrix} 1\,0\,1 \\ 0\,0\,0 \\ 0\,1\,1 \end{bmatrix}. \tag{A.86}$$

Using (7.36), the transformations to derive input selection patterns from output selection patterns are determined by M^{T} and N^{T}:

$$f_{\mathrm{d}}(\boldsymbol{u}_{\mathrm{d}}) = \mathsf{M}^{\mathrm{T}}\boldsymbol{u}_{\mathrm{d}} \tag{A.87}$$

$$g_{\mathrm{d}}(\boldsymbol{u}_{\mathrm{d}}) = \mathsf{N}^{\mathrm{T}}\boldsymbol{u}_{\mathrm{d}}. \tag{A.88}$$

Table A.2 shows the coordinates of the elements of GF(2^3), as well as the coordinates of the images of f_{d} and g_{d} calculated according to (A.87) and (A.88).

Table A.2. The functions f_{d} and g_{d}.

u_{d}	$w_{\mathrm{d}} = f_{\mathrm{d}}(u_{\mathrm{d}})$	$w_{\mathrm{d}} = g_{\mathrm{d}}(u_{\mathrm{d}})$
000	000	000
001	111	011
010	101	000
011	010	011
100	110	101
101	001	110
110	011	101
111	100	110

The dual basis of e can be determined by solving (A.57). It is given by

$$\mathsf{d} = [\alpha, \alpha^2 + \alpha, \alpha^2 + 1]^{\mathrm{T}}. \tag{A.89}$$

It can now be verified that the coordinates in Table A.2 correspond to the definitions of f_{d} and g_{d} of (A.82)–(A.84), provided that the coordinates of the variables u and w are determined with respect to basis d.

A.5 Rijndael-GF

The Rijndael round transformation operates on a state in $\mathrm{GF}(2)^{8n_{\mathrm{t}}}$ where $n_{\mathrm{t}} \in \{16, 20, 24, 28, 32\}$. We will now describe how each of its steps can be generalized to operations in $\mathrm{GF}(2^8)^{n_{\mathrm{t}}}$. The non-linear step `SubBytes` operates on the individual elements of the state. It is composed of two steps. The first step is taking the multiplicative inverse in $\mathrm{GF}(2^n)$:

$$g(x) = x^{-1}, \tag{A.90}$$

where the bytes represent polynomials. K. Nyberg has studied the non-linearity properties of this and similar functions over $\mathrm{GF}(2^n)$ in [74].

The second step is an affine function defined by a matrix multiplication and the subsequent addition of a constant. If we apply Theorem A.4.3, we obtain the following expression for the affine function:

$$\begin{aligned} f(x) = {} & \mathtt{05} \cdot x + \mathtt{09} \cdot x^2 + \mathtt{F9} \cdot x^{2^2} + \\ & \mathtt{25} \cdot x^{2^3} + \mathtt{F4} \cdot x^{2^4} + \mathtt{01} \cdot x^{2^5} + \\ & \mathtt{B5} \cdot x^{2^6} + \mathtt{8F} \cdot x^{2^7} + \mathtt{63}. \end{aligned} \tag{A.91}$$

Composing (A.90) and (A.91) yields the expression of the RIJNDAEL-GF S-box:

$$\begin{aligned} f_{\mathrm{RD}}(x) = \mathtt{05} \cdot x^{254} &+ \mathtt{09} \cdot x^{253} + \mathtt{F9} \cdot x^{251} + \\ \mathtt{25} \cdot x^{247} &+ \mathtt{F4} \cdot x^{239} + \mathtt{01} \cdot x^{223} + \\ \mathtt{B5} \cdot x^{191} &+ \mathtt{8F} \cdot x^{127} + \mathtt{63}. \end{aligned} \tag{A.92}$$

In this expression the coefficients are elements of $\mathrm{GF}(2^8)$. We will denote this polynomial by $f_{\mathrm{RD}}(x)$. For the generalisation of `SubBytes` we then have that it replaces each element of the state a_i by $f_{\mathrm{RD}}(a_i)$.

The step `ShiftRows` is a byte transposition that does not modify the values of the bytes but merely changes their order. Hence, also its generalization merely changes the order of the elements a_i of the state without changing their order. The mixing step `MixColumns` operates independently on 4-byte columns and mixes them linearly. It has been defined as a matrix multiplication operating on vectors of 4 elements of $\mathrm{GF}(2^8)$ which fully defines the generalization. The addition of a round key `AddRoundKey` consists of a simple bitwise XOR, which corresponds to the addition in all vector representations of $\mathrm{GF}(2^8)$.

The key expansion only makes use of bitwise XOR, byte transpositions, the Rijndael S-box and the bitwise XOR with round constants. For the first three operations we have explained how they can be generalized. The round constants have in turn been defined in terms of elements in $\mathrm{GF}(2^8)$.

RIJNDAEL-GF, together with the choice of a representation of the elements of $\mathrm{GF}(2^8)$ as bytes constitutes a block cipher. For example, Rijndael is the representation of RIJNDAEL-GF where the elements of $\mathrm{GF}(2^8)$ are coded as bytes denoting binary polynomials of degree less than eight where the field multiplication is defined modulo the following irreducible polynomial:

$$m(x) = x^8 + x^4 + x^3 + x + 1. \tag{A.93}$$

B. Trail Clustering

In Chaps. 7 and 8 we explain how the correlation and the difference propagation over a number of rounds of an iterative block cipher is composed of a number of linear trails or differential trails respectively. We show that in key-alternating ciphers the correlation contribution of linear trails and the weight of a differential trail are both independent of the value of the key. Section 9.1 explains how to choose the number of rounds of a key-alternating cipher to offer resistance against linear and differential cryptanalysis. Although the existence of high correlations and difference propagation probabilities cannot be avoided, taking a number of rounds so that the contributions of the individual trails are below some limit, makes the values of the patterns that exhibit large difference propagation probabilities or correlations very key-dependent. We count on this key-dependence to make the exploitation of these high correlations and difference propagation probabilities in cryptanalysis infeasible.

In our analysis in Sect. 9.1, we have neglected possible trail clustering: the fact that sets of trails tend to propagate along common intermediate patterns. If clustering of trails occurs, the small contributions of the individual trails may be compensated by the fact that there are so many trails between an input pattern and an output pattern. The structure of Rijndael, and any cipher that operates on bundles rather than bits, can be suspected of trail clustering.

In this appendix, we prove some properties of Boolean transformations with a maximum branch number. Subsequently, we give provable upper bounds for the expected difference propagation probability and correlation potential for two rounds and four rounds of ciphers with the Rijndael structure. Finally, we study a particular case of differential and linear propagation over two rounds of Rijndael, and illustrate these with some experimental results.

B.1 Transformations with Maximum Branch Number

Consider a Boolean transformation ϕ operating on vectors of n_t bundles, and let the number of bits per bundle be denoted by v. We have

$$\left[b_{(1)}\ b_{(2)}\ b_{(3)} \dots\ b_{(n_t)}\right]^{\mathrm{T}} = \phi(a_{(1)}, a_{(2)}, a_{(3)}, \dots, a_{(n_t)}) \tag{B.1}$$

Figure B.1 illustrates this with an example.

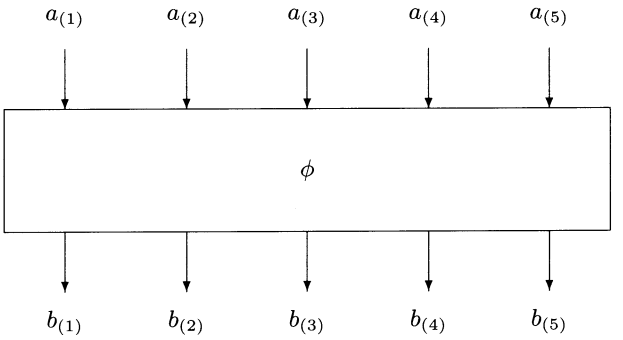

Fig. B.1. Boolean transformation ϕ operating on 5-bundle vectors.

Consider the following equation:

$$\phi(x_{(1)}, x_{(2)}, x_{(3)}, \dots, x_{(n_t)}) = \left[x_{(n_t+1)}\ x_{(n_t+2)}\ x_{(n_t+3)} \dots\ x_{(2n_t)}\right]^{\mathrm{T}} \tag{B.2}$$

Clearly, (B.2) has exactly 2^{vn_t} solutions, one for each choice of the vector $[x_{(1)}\ x_{(2)}\ x_{(3)} \dots\ x_{(n_t)}]$.

We consider a partition Ξ of the set $\{1, 2, 3, \dots, 2n_t\}$ that divides the set of indices in two equally sized subsets ξ and $\bar{\xi}$. We denote the vector with components x_i with $i \in \xi$ by x_ξ. Given such a partition and a value for $\mathbf{x}_\xi$, we define the following set of equations:

$$\begin{cases} \phi(y_{(1)}, y_{(2)}, \dots, y_{(n_t)}) = \left[y_{(n_t+1)}\ y_{(n_t+2)}\ \dots\ y_{(2n_t)}\right]^{\mathrm{T}} \\ \mathbf{y}_\xi = \mathbf{x}_\xi \end{cases} . \tag{B.3}$$

Theorem B.1.1. *A Boolean transformation ϕ has a maximum differential branch number, i.e. $\mathcal{B}(\phi) = n_t + 1$, iff any set of equations of the form (B.3) has exactly one solution, whatever the choice of ξ (with $\#\xi = n_t$) and $\mathbf{x}_\xi$.*

Proof.

$\Rightarrow$ Assume that $\mathcal{B}(\phi) = n_t + 1$, and that there is a choice of ξ and a value of $\mathbf{x}_\xi$ for which (B.3) has more than one solution. The solutions can only differ in at most n_t bundles, since the n_t components of $\mathbf{y}_\xi$ are fixed by $\mathbf{y}_\xi = \mathbf{x}_\xi$. However, if ϕ has a differential branch number equal to $n_t + 1$, (B.2) cannot have two solutions that differ in less than $n_t + 1$ bundles. Hence, (B.3) has at most one solution.

Now consider the $2^{n_t v}$ solutions of (B.2). For some given choice of ξ, each of these solutions $\mathbf{a}$ is also a solution of exactly one set of equations of type (B.3), i.e. the one with $\mathbf{x}_\xi = \mathbf{a}_\xi$. As each set of equations (as in Eq. B.3) has at most one solution and as the total number of sets of equations of type (B.3) for a given ξ is $2^{n_t v}$, each of these sets has exactly one solution.

$\Leftarrow$ Assume the Boolean transformation ϕ has a differential branch number that is smaller than $n_t + 1$. This implies that there must exist at least two solutions of (B.2) that differ in at most n_t bundles. We can now construct a set of equations (as in Eq. B.3) that has two solutions as follows. We choose ξ to contain only bundle positions in which the two solutions are the same, and $\mathbf{x}_\xi$ the vector containing the value of those bundles for the two solutions. This contradicts the premise and hence our initial hypothesis is proven to be false. □

Corollary B.1.1. *For a Boolean transformation operating on n_t-bundle vectors and with a maximum branch number, any set of n_t input and/or output bundles determines the remaining n_t output and/or input bundles completely.*

Hence, if we have a Boolean transformation ϕ with a maximum branch number, any partition Ξ that divides the input and output bundles into two sets with an equal number of elements ξ and $\bar{\xi}$ also defines a Boolean transformation. We call this function ϕ_ξ. This is illustrated with an example in Fig. B.2. As for any value of ξ both ϕ_ξ and $\phi_{\bar{\xi}}$ are Boolean tranformations, it follows that all ϕ_ξ are Boolean permutations. Note that with this convention, the permutation ϕ corresponds to ϕ_ξ with $\xi = \{1, 2, 3, \ldots, n_t\}$, and its inverse ϕ^{-1} with ϕ_ξ with $\xi = \{n_t + 1, n_t + 2, n_t + 3, \ldots, 2n_t\}$.

In [90], S. Vaudenay defines the similar concept of *multipermutations*. An (r, n)-*multipermutation* is a function that maps a vector of r bundles to n bundles with a differential branch number that is larger than r. A Boolean transformation ϕ with maximum differential branch number is hence a (n_t, n_t)-multipermutation. The name multipermutation is very appropriate

for such a transformation, since it defines a permutation from any set of n_t input and/or output bundles to the complementary set.

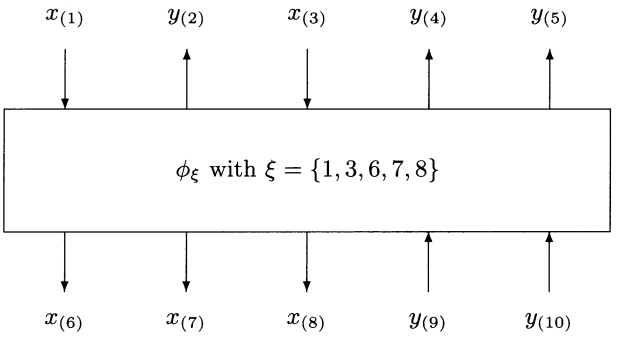

Fig. B.2. Boolean function ϕ

Theorem B.1.2. *A Boolean transformation ϕ has a maximum differential branch number iff it has a maximum linear branch number.*

Proof.

$\Rightarrow$ Assume that ϕ has a maximum differential branch number and not a maximum linear branch number. Consider now (B.2). If ϕ does not have a maximum linear branch number, there is a selection pattern w with a bundle weight of less than $n_t + 1$ such that the parity $\mathbf{w}^T\mathbf{x}$ is correlated to 0. If we now consider ϕ_ξ with only positions in ξ for which $w_i = 0$, this implies that a parity of output bits of ϕ_ξ is correlated to 0, or in other words, is unbalanced. As ϕ has a maximum differential branch number, any ϕ_ξ must be a permutation and hence according to Theorem 7.5.1 all its output parities must be balanced. It follows that ϕ cannot have a maximum differential branch number and a non-maximum linear branch number.

$\Leftarrow$ Assume that ϕ has a maximum linear branch number and not a maximum differential branch number. If ϕ does not have a maximum differential branch number, (B.2) has at least two solutions that differ in at most n_t bundles. If we choose ξ such that the bundle positions in which these two solutions differ are all in ξ, this means that the function ϕ_ξ has two inputs with the same output and hence is no permutation. According to Theorem 7.5.1, this ϕ_ξ must have output parities ${w_{\bar{\xi}}}^T x_{\bar{\xi}}$ that are not balanced. Hence, ϕ must have a linear branch number that is maximally n_t. It follows that ϕ cannot have a maximum linear branch number and a non-maximum differential branch number. □

B.2 Bounds for Two Rounds

For a cipher with a $\gamma\lambda$-round structure we can prove upper bounds for the expected difference propagation probability and the expected correlation potential (see Sect. 7.9.3) over two rounds.

In Fig. B.3 we have depicted the sequence of steps in two rounds of a cipher with the $\gamma\lambda$-structure. We study the probability of propagation of a difference in $\mathbf{a}^{(1)}$ to a difference in $\mathbf{a}^{(3)}$. The difference pattern in $\mathbf{a}^{(3)}$ completely determines the difference pattern in $\mathbf{b}^{(2)}$. Hence for this study we only have to consider the first round and the non-linear step of the second round. For the correlation potentials, we study the correlation between parities of $\mathbf{a}^{(3)}$ and parities of $\mathbf{a}^{(1)}$. A parity of $\mathbf{a}^{(3)}$ is correlated to exactly one parity of $\mathbf{b}^{(2)}$ with a correlation of 1 or -1 depending on the value of a parity of round key $\mathbf{k}^{(2)}$. As we are not interested in the sign, again we can limit ourselves to studying the first round and the non-linear step of the second round.

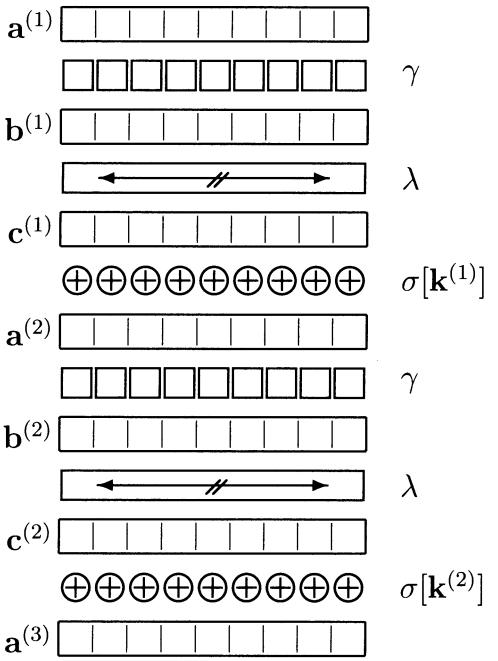

Fig. B.3. Two rounds of a cipher with the $\gamma\lambda$-structure.

Fig. B.4 depicts the sequence of steps relevant in our analysis of difference and linear propagation over two rounds. In the remainder of this section we denote the difference pattern and selection patterns in the state at the different intermediate stages by $\mathbf{a}, \mathbf{b}, \mathbf{c}$ and $\mathbf{d}$.

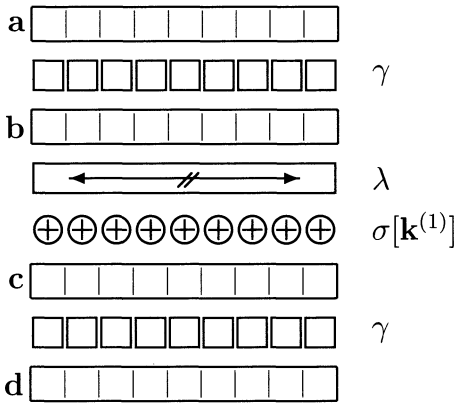

Fig. B.4. Steps relevant in the study of two-round difference propagation.

B.2.1 Difference Propagation

In this section we will denote the difference propagation probabilities of the γ S-box by $\mathrm{p_s}(x, y)$. Moreover, we will assume that these difference propagation probabilities are below an upper bound, denoted $\mathrm{p_m}$.

Consider now a differential trail from a difference pattern $\mathbf{a}$ to a difference pattern $\mathbf{d}$. We denote the set of positions with active bundles in $\mathbf{a}$ by α, and the set of positions with active bundles in $\mathbf{d}$ by δ. The number of active bundles in $\mathbf{a}$ is denoted by $\#\alpha$, and the number of active bundles in $\mathbf{d}$ is denoted by $\#\delta$. The active bundles of $\mathbf{a}$ propagate to active bundles in $\mathbf{b}$ through the γ S-boxes. The difference pattern $\mathbf{b}$ fully determines $\mathbf{c}$ by $\mathbf{c} = \lambda(\mathbf{b})$. The difference pattern $\mathbf{c}$ propagates to $\mathbf{d}$ through the γ S-boxes. The difference patterns $\mathbf{c}$ and $\mathbf{d}$ have active bundles in the same positions. As $\mathbf{b}$ completely determines the trail, together with $\mathbf{a}$ and $\mathbf{d}$, we denote this trail by $(\mathbf{a}, \mathbf{b}, \mathbf{d})$. The weight $\mathrm{w_r}(\mathbf{a}, \mathbf{b}, \mathbf{c})$ of this differential trail is the sum of the weights of the difference propagation over active S-boxes corresponding to the active bundles in $\mathbf{a}$ (or equivalently $\mathbf{b}$) and $\mathbf{c}$ (or equivalently $\mathbf{d}$). The sum of the number of active bundles in $\mathbf{b}$ and $\mathbf{c}$ is lower bounded by $\mathcal{B}(\lambda)$. Since $\mathbf{a}$ and $\mathbf{d}$ have active bundles at the same positions as $\mathbf{b}$ and $\mathbf{c}$, respectively, it follows that $\#\alpha + \#\delta$ is lower bounded by $\mathcal{B}(\lambda)$. An approximation for the probability of a differential trail is $2^{-\mathrm{w_r}(\mathbf{a},\mathbf{b},\mathbf{d})}$. We will denote this approximation by $\mathrm{Prob_e}(\mathbf{a}, \mathbf{b}, \mathbf{d})$. These approximations should be interpreted with care, as they are made under the assumption that the restrictions are independent (see Sect. 8.4.2). We have:

$$\mathrm{Prob_e}(\mathbf{a}, \mathbf{b}, \mathbf{d}) = \prod_{i \in \alpha} \mathrm{p_s}(a_i, b_i) \prod_{j \in \delta} \mathrm{p_s}(c_j, d_j). \tag{B.4}$$

The expected probability that $\mathbf{a}$ propagates to $\mathbf{d}$ is denoted by $\mathrm{Prob_e}(\mathbf{a}, \mathbf{d})$ and it is given by the sum of the expected probabilities of all differential trails between $\mathbf{a}$ and $\mathbf{d}$:

$$\mathrm{Prob}_{rme}(\mathbf{a}, \mathbf{d}) = \sum_{b} \mathrm{Prob}_{rme}(\mathbf{a}, \mathbf{b}, \mathbf{d}). \tag{B.5}$$

Example B.2.1. Consider the propagation from a difference pattern $\mathbf{a}$ with a single active bundle, in position 1. Equation (B.4) simplifies to

$$\mathrm{Prob_e}(\mathbf{a}, \mathbf{b}, \mathbf{d}) = \mathrm{p_s}(a_1, b_1) \prod_{j \in \delta} \mathrm{p_s}(c_i, d_i). \tag{B.6}$$

By using the upper bound for the difference propagation probability for the S-box, $\mathrm{p_s}(x, y) \leq \mathrm{p_m}$, this can be reduced to

$$\mathrm{Prob_e}(\mathbf{a}, \mathbf{b}, \mathbf{d}) \leq \mathrm{p_s}(a_1, b_1) {\mathrm{p_m}}^{\#\delta}. \tag{B.7}$$

Substitution into (B.5) yields

$$\mathrm{Prob_e}(\mathbf{a}, \mathbf{d}) \leq \sum_{b_1} \mathrm{p_s}(a_1, b_1) {\mathrm{p_m}}^{\#\delta}. \tag{B.8}$$

By using the fact that $\sum_y \mathrm{p_s}(x, y) = 1$, we obtain

$$\mathrm{Prob_e}(\mathbf{a}, \mathbf{d}) \leq {\mathrm{p_m}}^{\#\delta}. \tag{B.9}$$

Since $\#\delta + 1$ is the sum of the number of active S-boxes of $\mathbf{b}$ and $\mathbf{c}$, it is lower bounded by $\mathcal{B}(\lambda)$. It follows that

$$\mathrm{Prob_e}(\mathbf{a}, \mathbf{d}) \leq p^{\mathcal{B}-1}. \tag{B.10}$$

We can now prove the following theorem:

Theorem B.2.1. *If λ has a maximum branch number, the expected maximum difference propagation probability over two rounds is upper bounded by* ${\mathrm{p_s}}^{n_t}$.

Proof. Clearly $\#\alpha + \#\delta$ is lower bounded by $n_t + 1$. Let us now partition the bundle positions of $\mathbf{b}$ and $\mathbf{c}$ in two equally sized sets ξ and $\bar{\xi}$, such that $\bar{\xi}$ has only active bundles. This is always possible as there must at least be $n_t + 1$ active bundles in $\mathbf{b}$ and $\mathbf{c}$ together. We have $\mathbf{c} = \lambda(\mathbf{b})$, and that λ has a maximum differential branch number. Hence according to Theorem B.1.1, the values of the bundles of $\mathbf{b}$ and $\mathbf{c}$ that are in positions in ξ completely determine the values of the bundles with positions in $\bar{\xi}$.

We can convert (B.4) to

$$\mathrm{Prob_e}(\mathbf{a},\mathbf{b},\mathbf{d}) \approx \prod_{i\in\alpha\cap\xi} \mathrm{p_s}(a_i,b_i) \prod_{j\in\delta\cap\xi} \mathrm{p_s}(c_j,d_j) \times \tag{B.11}$$

$$\prod_{i\in\alpha\cap\bar{\xi}} \mathrm{p_s}(a_i,b_i) \prod_{j\in\delta\cap\bar{\xi}} \mathrm{p_s}(c_j,d_j). \tag{B.12}$$

Since all bundles in positions in $\bar{\xi}$ are active, the factor (B.12) is upper bounded by $\mathrm{p_m}^n_{\mathrm{t}}$. We obtain

$$\mathrm{Prob_e}(\mathbf{a},\mathbf{b},\mathbf{d}) \leq {\mathrm{p_m}}^{n_{\mathrm{t}}} \prod_{i\in\alpha\cap\xi} \mathrm{p_s}(a_i,b_i) \prod_{j\in\delta\cap\xi} \mathrm{p_s}(c_j,d_j). \tag{B.13}$$

The expected difference propagation probability from $\mathbf{a}$ to $\mathbf{d}$ can be found by summing over all possible trails. In this case, this implies summing over all possible values of the active bundles in $\mathbf{b}$ and $\mathbf{c}$ that have positions in ξ. We have

$$\mathrm{Prob_e}(\mathbf{a},\mathbf{d}) \leq {\mathrm{p_m}}^{n_{\mathrm{t}}} \prod_{i\in\alpha\cap\xi} \sum_{b_i} \mathrm{p_s}(a_i,b_i) \prod_{j\in\delta\cap\xi} \sum_{c_j} \mathrm{p_s}(c_j,d_j). \tag{B.14}$$

We can apply $\sum_y \mathrm{p_s}(x,y) = 1$ to the factors $\sum_{b_i} \mathrm{p_s}(a_i,b_i)$. Moreover, as the S-box is invertible, we also have $\sum_x \mathrm{p_s}(x,y) = 1$. This can be applied to the factors $\sum_{c_j} \mathrm{p_s}(c_j,d_j)$. We obtain

$$\mathrm{Prob_e}(\mathbf{a},\mathbf{d}) \leq {\mathrm{p_m}}^{n_{\mathrm{t}}},$$

proving the theorem. □

B.2.2 Correlation

In this section we will denote the input-output correlation potentials of the γ S-box by $\mathrm{c_s}(x,y)$. We will assume that these correlation potentials are below an upper bound, denoted $\mathrm{c_m}$. For an introduction to correlation potentials we refer to Chap. 7.

Consider now a linear trail from a selection pattern $\mathbf{a}$ to a selection pattern $\mathbf{d}$. We denote the set of positions with active bundles in $\mathbf{a}$ by α, and the set of positions with active bundles in $\mathbf{d}$ by δ. The active bundles of $\mathbf{a}$ propagate to active bundles in $\mathbf{b}$ through the γ S-boxes. Since λ is linear, the selection pattern $\mathbf{b}$ fully determines $\mathbf{c}$. The selection pattern $\mathbf{c}$ propagates to $\mathbf{d}$ through n_{t} S-boxes. The selection patterns $\mathbf{c}$ and $\mathbf{d}$ have active bundles in the same positions. The correlation potential of this linear trail is the product of the correlation potentials of the active S-boxes corresponding to the active bundles in $\mathbf{a}$ (or equivalently $\mathbf{b}$) and $\mathbf{c}$ (or equivalently $\mathbf{d}$). The sum of the number of active bundles in $\mathbf{b}$ and $\mathbf{c}$ is lower bounded by $\mathcal{B}(\lambda)$. Since $\mathbf{a}$ and $\mathbf{d}$

have active bundles at the same positions as $\mathbf{b}$ and $\mathbf{c}$, respectively, it follows that $\#\alpha + \#\delta$ is lower bounded by $\mathcal{B}(\lambda)$. We have

$$\mathrm{c}(\mathbf{a}, \mathbf{b}, \mathbf{d}) = \prod_{i \in \alpha} \mathrm{c_s}(a_i, b_i) \prod_{j \in \delta} \mathrm{c_s}(c_j, d_j). \tag{B.15}$$

The expected potential of the correlation between a selection pattern $\mathbf{a}$ and a selection pattern $\mathbf{d}$ is denoted by $\mathrm{c_e}(\mathbf{a}, \mathbf{d})$ and it is given by the sum of the correlation potentials of all linear trails between $\mathbf{a}$ and $\mathbf{d}$:

$$\mathrm{c_e}(\mathbf{a}, \mathbf{d}) = \sum_{b} \mathrm{c}(\mathbf{a}, \mathbf{b}, \mathbf{d}). \tag{B.16}$$

We can now prove the following theorem:

Theorem B.2.2. *If λ has a maximum branch number, the expected correlation potential over two rounds is upper bounded by $\mathrm{c_s}^{n_\mathrm{t}}$.*

Proof. Clearly $\#\alpha + \#\delta$ is lower bounded by $n_\mathrm{t} + 1$. Let us now partition the bundle positions of $\mathbf{b}$ and $\mathbf{c}$ into two equally sized sets ξ and $\bar{\xi}$ such that $\bar{\xi}$ has only active bundles. This is always possible as there must at least be $n_\mathrm{t} + 1$ active bundles in $\mathbf{b}$ and $\mathbf{c}$ together. As λ is a linear transformation with a maximum branch number, according to Theorem B.1.1 the values of the bundles of $\mathbf{b}$ and $\mathbf{c}$ that are in positions in ξ completely determine the values of the bundles with positions in $\bar{\xi}$. We can convert (B.15) to

$$\mathrm{c}(\mathbf{a}, \mathbf{b}, \mathbf{d}) \approx \prod_{i \in \alpha \cap \xi} \mathrm{c_s}(a_i, b_i) \prod_{j \in \delta \cap \xi} \mathrm{c_s}(c_j, d_j) \times \tag{B.17}$$

$$\prod_{i \in \alpha \cap \bar{\xi}} \mathrm{c_s}(a_i, b_i) \prod_{j \in \delta \cap \bar{\xi}} \mathrm{c_s}(c_j, d_j). \tag{B.18}$$

As all bundles in positions in $\bar{\xi}$ are active, the factor (B.18) is upper bounded by $\mathrm{c_m}^n_\mathrm{t}$. We obtain

$$\mathrm{c}(\mathbf{a}, \mathbf{b}, \mathbf{d}) \leq \mathrm{c_m}^{n_\mathrm{t}} \prod_{i \in \alpha \cap \xi} \mathrm{c_s}(a_i, b_i) \prod_{j \in \delta \cap \xi} \mathrm{c_s}(c_j, d_j). \tag{B.19}$$

The expected potential of the correlation between $\mathbf{a}$ to $\mathbf{d}$ can be found by summing over all possible trails. In this case, this implies summing over all possible values of the active bundles in $\mathbf{b}$ and $\mathbf{c}$ that have positions in ξ. We have

$$\mathrm{c_e}(\mathbf{a}, \mathbf{d}) \leq \mathrm{c_m}^{n_\mathrm{t}} \prod_{i \in \alpha \cap \xi} \sum_{b_i} \mathrm{c_s}(a_i, b_i) \prod_{j \in \delta \cap \xi} \sum_{c_j} \mathrm{c_s}(c_j, d_j). \tag{B.20}$$

Applying Parseval's theorem to the γ S-box yields $\sum_y \mathrm{c_s}(x, y) = 1$ and applying it to the inverse of the γ S-box yields $\sum_x \mathrm{c_s}(x, y) = 1$. Using this, we obtain

$$c_e(\mathbf{a}, \mathbf{d}) \leq {c_m}^{n_t},$$

proving the theorem. □

B.3 Bounds for Four Rounds

For key-iterated ciphers with a $\gamma\pi\theta$-structure, we can prove similar bounds for four rounds. In Theorem 9.5.1 we have shown that the analysis of such a cipher can be reduced to the analysis of a key-alternating cipher with two round transformations. In this section, we will study this key-alternating cipher structure.

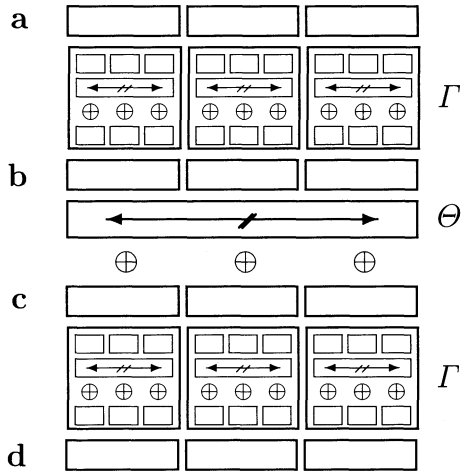

Fig. B.5. Steps relevant in the study of upper bounds for four rounds.

As illustrated in Fig. B.5, the relevant steps of four rounds can be grouped into a number of supersteps. The first step and last step consists of an application of γ, θ, key addition and again γ. This step operates independently on the columns of the state and can be considered as a γ step with big S-boxes. If θ has a maximum branch number at the level of the columns, the theorems of Sect. B.2 provide upper bounds for the big S-box. The expected difference propagation probability is upper bounded by ${p_m}^{\mathcal{B}-1}$ and the expected correlation potential is upper bounded by ${c_m}^{\mathcal{B}-1}$.

In the four-round structure, the two steps in-between are a linear mixing step and a key addition. If the mixing step has a maximum branch number at the level of the columns, the theorems of Sect. B.2 are also applicable at this level, giving upper bounds for four rounds. The expected difference propagation probability and correlation potential is upper bounded by ${p_m}'^{\mathcal{B}'-1}$ and

the expected correlation potential is upper bounded by ${c_m}'^{\mathcal{B}'-1}$. In these expressions $\mathcal{B}'$ is the branch number of Θ, and ${p_m}'$ and ${c_m}'$ refer to the S-boxes of Γ. By substituting the values for the Γ S-boxes, we obtain upper limits of ${p_m}^{(\mathcal{B}'-1)(\mathcal{B}-1)}$ and ${c_m}^{(\mathcal{B}'-1)(\mathcal{B}-1)}$. In the case where the branch number of Θ and θ are the same, this is reduced to ${p_m}^{(\mathcal{B}-1)^2}$ and ${c_m}^{(\mathcal{B}-1)^2}$. This applies to Rijndael with a block length of 128 bits.

B.4 Two Case Studies

If we apply Theorems B.2.1 and B.2.2 to Rijndael, we find that the expected difference propagation probability and correlation potential over two rounds is upper bounded by 2^{-24}, and over four rounds is upper bounded by 2^{-96}. These upper bounds are, however, very seldomly attained. In this section we will illustrate this with a quantitative description of difference and linear propagation over two rounds for a configuration in which there are only 5 active S-boxes.

B.4.1 Differential Trails

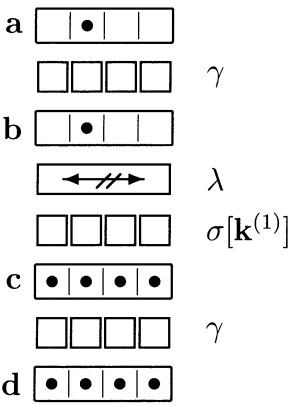

Fig. B.6. Difference propagation over two rounds of Rijndael. Active bytes are indicated with bullets.

In S_{RD}, each non-zero input difference can propagate to exactly 127 output differences. For the propagation to 126 of these output differences the probability is 2^{-7}, and for one it is 2^{-6}. In the former case, there is exactly one pair of input values for which the outputs have the given difference. In the latter case, there are two pairs of input values.

Let us now consider a difference propagation over two rounds of Rijndael, where the difference pattern **a** at the input of the first round has a single active byte a_2. We restrict ourselves to a single column, as illustrated in Fig. B.6. The difference pattern **a** propagates to a difference pattern **b** with a single active byte b_2. The number of possible values of b_2 is 127. As the branch number of `MixColumns` is 5, **b** propagates to a difference pattern **c** with 4 active bytes. These values of these bytes are completely determined by the value of b_2. Each of these bytes of c_i can propagate to 127 possible values d_i. In total, there are 127^5 possible trails starting from the given **a**. There are only 2^{31} possible input pairs, hence at most 2^{31} of these 127^5 trails will be followed for a given key. The fact that a certain trail will be followed depends on the value of the key. the probability that a certain trail will be followed depends on the weight of the trail.

The weight of a differential trail is the sum of the weights of its active S-boxes. In the S-box a given input propagates to one output difference pattern with a weight of 6 and to 126 other output differences with a weight of 7. Hence, the weight of the differential trails described above ranges between 30 and 35. This depends on the number of S-boxes ℓ in which there is a difference propagation with probability 2^{-7}. The distribution of the number of trails as described above as a function of ℓ is given in Table B.1.

Let us now consider the set of trails that may be followed for some given value of the key. For a trail with a weight of 30, we expect it to be followed for exactly two pairs of inputs. However, in general the inputs to the S-boxes in the two subsequent rounds are not independent: we expect to have no trails for some key values, while there will be some more trails for other key values. The trails with a weight of 31 are all expected to be followed once. We expect that about half of the trails of weight 32, one-fourth of the trails with weight 33, 1/8 of the trails with weight 34 and 1/16 of the trails with weight 35 to be followed. Except for the single trail a with of weight 30, all of the trails only occur for a single pair of inputs. for the trails with higher weight, the part of the trails that are followed depends on the value of the round key.

Table B.1. Differential trail statistics

ℓ	w_r $30+\ell$	No. existing $\binom{5}{\ell}126^\ell$	Prob. of being followed: $2^{1-\ell}$	No. followed $2\binom{5}{\ell}63^\ell$
0	30	1	2	2
1	31	630	1	630
2	32	2.4×10^5	2^{-1}	1.2×10^5
3	33	3×10^8	2^{-2}	7.5×10^7
4	34	1.2×10^9	2^{-3}	1.5×10^8
5	35	3×10^{10}	2^{-4}	2×10^9
All		127^5		2^{31}

B.4.2 Linear Trails

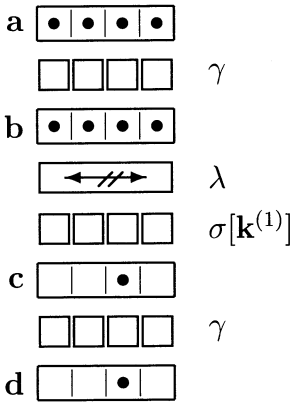

Fig. B.7. Correlation over two rounds of Rijndael. Active bytes are indicated with bullets.

In S_{RD}, each non-zero output selection pattern is correlated to exactly 239 input selection patterns. The amplitude of these correlations ranges from 2^{-6} to $8 \times 2^{-6} = 2^{-3}$.

Let us now consider a selection pattern **d** with a single active byte d_2 and see to which selection patterns in **c**, **b** and **a** there is a non-zero correlation. Again, we restrict ourselves to a single column, as illustrated in Fig. B.7. The parity defined by d_2 is correlated to 239 different parities defined by a selection pattern **c** with a single active byte c_2. As the branch number of `MixColumns` is 5, the parity defined by c_2 is correlated with a parity defined by a selection pattern **b** with 4 active bytes. The values of these bytes are completely determined by the value of c_2. The parities corresponding to the selection patterns b_i are each correlated with the parities corresponding with 239 different values for d_i. In total, there are 239^5 linear trails arriving in **d**. These linear trails all start from a selection pattern in **a** with four active bytes. As there are 255^4 such selection patterns, given a particular selection pattern **a** we expect there to be $239^5/255^4 \approx 184$ linear trails between **a** and **d**. The correlation contribution of such a linear trail is the product of the correlations for its active S-boxes. This ranges between $(2^{-3})^5 = 2^{-15}$ and $(2^{-6})^5 = 2^{-30}$. The sign of these linear trails depends on the value of $\mathbf{k}_2^{(1)}$, and hence the fact that the trails interfere constructively or destructively depends on the value of this key byte.

We have conducted a series of experiments in which we measured correlation values for linear trails of the type described above. We have summarized the results of these experiments in a Tables B.2 to B.4.

Table B.2 lists the distribution of the number of trails between given pairs of input/output selection patterns of the type described above. Observe that the distribution is centered around 184 trails per pair, as predicted by the computations above. The maximum number of linear trails $(\mathbf{a}, \mathbf{b}, \mathbf{d})$ for a single pair of input/output selection patterns $\mathbf{a}$ and $\mathbf{d}$ we have observed is 210.

Table B.3 lists the distribution of the expected correlation potentials. For a given input selection pattern $\mathbf{a}$ and output selection pattern $\mathbf{d}$ we have computed the expected correlation potential as the sum of the correlation potentials of all trails between $\mathbf{a}$ and $\mathbf{d}$. The average value for this expected correlation potential is 2^{-32}, the value one would expect for a random mapping. The maximum value that we observed was 2^{-29}. This is a factor of 2^5 smaller than the upper bound given by Theorem B.2.2.

Table B.4 lists the distribution of the amplitude of measured correlations between $\mathbf{a}$ and $\mathbf{d}$ for given values of the key. These correlations were computed by adding the correlation contributions of the linear trails between $\mathbf{a}$ and $\mathbf{d}$. More than 64% of the correlations have a value below 2^{-16}. The maximum correlation we observed has value $2^{-13.5}$.

Table B.2. Distribution of the number of trails between **a** and **d**.

Value	Proportion	Value	Proportion	Value	Proportion
175	0.0014%	187	8.69%	199	0.039%
176	0.02%	188	5.49%	200	0.022%
177	0.12%	189	3.07%	201	0.01%
178	0.50%	190	1.53%	202	0.0044%
179	1.54%	191	0.69%	203	0.0016%
180	3.63%	192	0.29%	204	0.0006%
181	6.88%	193	0.12%	205	0.0002%
182	10.7%	194	0.07%	206	0.00006%
183	14.0%	195	0.07%	207	0.00002%
184	15.5%	196	0.08%	208	0.000004%
185	14.7%	197	0.07%	209	0.000002%
186	12.1%	198	0.06%	210	0.0000005%

Table B.3. Distribution of expected correlation potentials.

Value	Proportion	Value	Proportion	Value	Proportion
$2^{-29.0}$	0.0000043%	$2^{-31.0}$	2.0%	$2^{-33.0}$	1.0%
$2^{-29.2}$	0.000031%	$2^{-31.2}$	4.4%	$2^{-33.2}$	0.26%
$2^{-29.4}$	0.00018%	$2^{-31.4}$	8.0%	$2^{-33.4}$	0.051%
$2^{-29.6}$	0.00084%	$2^{-31.6}$	12.4%	$2^{-33.6}$	0.007%
$2^{-29.8}$	0.00333%	$2^{-31.8}$	16.3%	$2^{-33.8}$	0.0007%
$2^{-30.0}$	0.012%	$2^{-32.0}$	17.7%	$2^{-34.0}$	0.00005%
$2^{-30.2}$	0.038%	$2^{-32.2}$	15.7%	$2^{-34.2}$	0.000003%
$2^{-30.4}$	0.11%	$2^{-32.4}$	11.3%	$2^{-34.4}$	0.0000002%
$2^{-30.6}$	0.32%	$2^{-32.6}$	6.5%		
$2^{-30.8}$	0.86%	$2^{-32.8}$	2.9%		

Table B.4. Distribution of measured correlation amplitudes.

Value	Proportion	Value	Proportion	Value	Proportion
$2^{-13.5}$	0.0001%	$2^{-14.4}$	0.24%	$2^{-15.3}$	2.52%
$2^{-13.6}$	0.0007%	$2^{-14.5}$	0.40%	$2^{-15.4}$	2.79%
$2^{-13.7}$	0.0005%	$2^{-14.6}$	0.59%	$2^{-15.5}$	2.95%
$2^{-13.8}$	0.0036%	$2^{-14.7}$	0.81%	$2^{-15.6}$	3.13%
$2^{-13.9}$	0.0078%	$2^{-14.8}$	1.06%	$2^{-15.7}$	3.21%
$2^{-14.0}$	0.018%	$2^{-14.9}$	1.38%	$2^{-15.8}$	3.35%
$2^{-14.1}$	0.038%	$2^{-15.0}$	1.67%	$2^{-15.9}$	3.33%
$2^{-14.2}$	0.079%	$2^{-15.1}$	1.96%	$2^{-16.0}$	3.39%
$2^{-14.3}$	0.15%	$2^{-15.2}$	2.26%	$< 2^{-16}$	64.7%

C. Substitution Tables

In this appendix, we list some tables that represent various mappings used in Rijndael.

C.1 S_{RD}

This section includes several representations of S_{RD} and related mappings. More explanation about the alternative representations for the mappings used in the definition of S_{RD} can be found in Sect. 3.4.1. Tabular representations of S_{RD} and ${S_{RD}}^{-1}$ are given in Tables C.1 and C.2.

Table C.1. Tabular representation of $S_{RD}(xy)$.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

For hardware implementations, it might be useful to use the following decomposition of S_{RD}:

$$S_{RD}[a] = f(g(a)), \tag{C.1}$$

where $g(a)$ is the mapping

$$a \to a^{-1} \text{ in } \mathrm{GF}(2^8), \tag{C.2}$$

Table C.2. Tabular representation of $S_{RD}{}^{-1}(xy)$.

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	A	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	B	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	C	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	D	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	E	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	F	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

and $f(a)$ is an affine mapping. Since $g(a)$ is self-inverse, we have

$$S_{RD}{}^{-1}[a] = g^{-1}(f^{-1}(a)) = g(f^{-1}(a)). \tag{C.3}$$

The tabular representations of f, f^{-1} and g are given in Tables C.3–C.5.

Algebraic representations of S_{RD} have also received a lot of attention in the literature, specially in cryptanalytic literature. Mappings over a finite domain can *always* be represented by polynomials functions with a finite number of terms. As a consequence, mappings from GF(2^8) to GF(2^8) can always be represented by a polynomial function over GF(2^8). A general way to derive this polynomial representation is given by the *Lagrange interpolation formula*. Applying Lagrange interpolation to S_{RD} gives the following result:

$$\begin{aligned} S_{RD}[x] = {} & 05 \cdot x^{254} + 09 \cdot x^{253} + F9 \cdot x^{251} + 25 \cdot x^{247} \\ & + F4 \cdot x^{239} + 01 \cdot x^{223} + B5 \cdot x^{191} + 8F \cdot x^{127} + 63. \end{aligned} \tag{C.4}$$

The coefficients are elements of GF(2^8).

C.2 Other Tables

C.2.1 xtime

More explanation about the mapping `xtime` can be found in Sect. 4.1.1. The tabular representation is given in Table C.6.

C.2.2 Round Constants

The key expansion routine uses round constants. Further explanation can be found in Sect. 3.6. Table C.7 lists the first 30 round constants. Note that

Table C.3. Tabular representation of $f(xy)$.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7C	5D	42	1F	00	21	3E	9B	84	A5	BA	E7	F8	D9	C6
	1	92	8D	AC	B3	EE	F1	D0	CF	6A	75	54	4B	16	09	28	37
	2	80	9F	BE	A1	FC	E3	C2	DD	78	67	46	59	04	1B	3A	25
	3	71	6E	4F	50	0D	12	33	2C	89	96	B7	A8	F5	EA	CB	D4
	4	A4	BB	9A	85	D8	C7	E6	F9	5C	43	62	7D	20	3F	1E	01
	5	55	4A	6B	74	29	36	17	08	AD	B2	93	8C	D1	CE	EF	F0
	6	47	58	79	66	3B	24	05	1A	BF	A0	81	9E	C3	DC	FD	E2
	7	B6	A9	88	97	CA	D5	F4	EB	4E	51	70	6F	32	2D	0C	13
	8	EC	F3	D2	CD	90	8F	AE	B1	14	0B	2A	35	68	77	56	49
	9	1D	02	23	3C	61	7E	5F	40	E5	FA	DB	C4	99	86	A7	B8
	A	0F	10	31	2E	73	6C	4D	52	F7	E8	C9	D6	8B	94	B5	AA
	B	FE	E1	C0	DF	82	9D	BC	A3	06	19	38	27	7A	65	44	5B
	C	2B	34	15	0A	57	48	69	76	D3	CC	ED	F2	AF	B0	91	8E
	D	DA	C5	E4	FB	A6	B9	98	87	22	3D	1C	03	5E	41	60	7F
	E	C8	D7	F6	E9	B4	AB	8A	95	30	2F	0E	11	4C	53	72	6D
	F	39	26	07	18	45	5A	7B	64	C1	DE	FF	E0	BD	A2	83	9C

Table C.4. Tabular representation of $f^{-1}(xy)$.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	05	4F	91	DB	2C	66	B8	F2	57	1D	C3	89	7E	34	EA	A0
	1	A1	EB	35	7F	88	C2	1C	56	F3	B9	67	2D	DA	90	4E	04
	2	4C	06	D8	92	65	2F	F1	BB	1E	54	8A	C0	37	7D	A3	E9
	3	E8	A2	7C	36	C1	8B	55	1F	BA	F0	2E	64	93	D9	07	4D
	4	97	DD	03	49	BE	F4	2A	60	C5	8F	51	1B	EC	A6	78	32
	5	33	79	A7	ED	1A	50	8E	C4	61	2B	F5	BF	48	02	DC	96
	6	DE	94	4A	00	F7	BD	63	29	8C	C6	18	52	A5	EF	31	7B
	7	7A	30	EE	A4	53	19	C7	8D	28	62	BC	F6	01	4B	95	DF
	8	20	6A	B4	FE	09	43	9D	D7	72	38	E6	AC	5B	11	CF	85
	9	84	CE	10	5A	AD	E7	39	73	D6	9C	42	08	FF	B5	6B	21
	A	69	23	FD	B7	40	0A	D4	9E	3B	71	AF	E5	12	58	86	CC
	B	CD	87	59	13	E4	AE	70	3A	9F	D5	0B	41	B6	FC	22	68
	C	B2	F8	26	6C	9B	D1	0F	45	E0	AA	74	3E	C9	83	5D	17
	D	16	5C	82	C8	3F	75	AB	E1	44	0E	D0	9A	6D	27	F9	B3
	E	FB	B1	6F	25	D2	98	46	0C	A9	E3	3D	77	80	CA	14	5E
	F	5F	15	CB	81	76	3C	E2	A8	0D	47	99	D3	24	6E	B0	FA

Table C.5. Tabular representation of $g(xy)$.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	00	01	8D	F6	CB	52	7B	D1	E8	4F	29	C0	B0	E1	E5	C7
	1	74	B4	AA	4B	99	2B	60	5F	58	3F	FD	CC	FF	40	EE	B2
	2	3A	6E	5A	F1	55	4D	A8	C9	C1	0A	98	15	30	44	A2	C2
	3	2C	45	92	6C	F3	39	66	42	F2	35	20	6F	77	BB	59	19
	4	1D	FE	37	67	2D	31	F5	69	A7	64	AB	13	54	25	E9	09
	5	ED	5C	05	CA	4C	24	87	BF	18	3E	22	F0	51	EC	61	17
	6	16	5E	AF	D3	49	A6	36	43	F4	47	91	DF	33	93	21	3B
	7	79	B7	97	85	10	B5	BA	3C	B6	70	D0	06	A1	FA	81	82
	8	83	7E	7F	80	96	73	BE	56	9B	9E	95	D9	F7	02	B9	A4
	9	DE	6A	32	6D	D8	8A	84	72	2A	14	9F	88	F9	DC	89	9A
	A	FB	7C	2E	C3	8F	B8	65	48	26	C8	12	4A	CE	E7	D2	62
	B	0C	E0	1F	EF	11	75	78	71	A5	8E	76	3D	BD	BC	86	57
	C	0B	28	2F	A3	DA	D4	E4	0F	A9	27	53	04	1B	FC	AC	E6
	D	7A	07	AE	63	C5	DB	E2	EA	94	8B	C4	D5	9D	F8	90	6B
	E	B1	0D	D6	EB	C6	0E	CF	AD	08	4E	D7	E3	5D	50	1E	B3
	F	5B	23	38	34	68	46	03	8C	DD	9C	7D	A0	CD	1A	41	1C

Table C.6. Tabular representation of `xtime`(xy).

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	00	02	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
	1	20	22	24	26	28	2A	2C	2E	30	32	34	36	38	3A	3C	3E
	2	40	42	44	46	48	4A	4C	4E	50	52	54	56	58	5A	5C	5E
	3	60	62	64	66	68	6A	6C	6E	70	72	74	76	78	7A	7C	7E
	4	80	82	84	86	88	8A	8C	8E	90	92	94	96	98	9A	9C	9E
	5	A0	A2	A4	A6	A8	AA	AC	AE	B0	B2	B4	B6	B8	BA	BC	BE
	6	C0	C2	C4	C6	C8	CA	CC	CE	D0	D2	D4	D6	D8	DA	DC	DE
	7	E0	E2	E4	E6	E8	EA	EC	EE	F0	F2	F4	F6	F8	FA	FC	FE
	8	1B	19	1F	1D	13	11	17	15	0B	09	0F	0D	03	01	07	05
	9	3B	39	3F	3D	33	31	37	35	2B	29	2F	2D	23	21	27	25
	A	5B	59	5F	5D	53	51	57	55	4B	49	4F	4D	43	41	47	45
	B	7B	79	7F	7D	73	71	77	75	6B	69	6F	6D	63	61	67	65
	C	9B	99	9F	9D	93	91	97	95	8B	89	8F	8D	83	81	87	85
	D	BB	B9	BF	BD	B3	B1	B7	B5	AB	A9	AF	AD	A3	A1	A7	A5
	E	DB	D9	DF	DD	D3	D1	D7	D5	CB	C9	CF	CD	C3	C1	C7	C5
	F	FB	F9	FF	FD	F3	F1	F7	F5	EB	E9	EF	ED	E3	E1	E7	E5

RC[0] is never used. In the unlikely case that more values are required, they should be generated according to (3.19).

Table C.7. Round constants for the key generation.

i	0	1	2	3	4	5	6	7
RC[i]	00	01	02	04	08	10	20	40
i	8	9	10	11	12	13	14	15
RC[i]	80	1B	36	6C	D8	AB	4D	9A
i	16	17	18	19	20	21	22	23
RC[i]	2F	5E	BC	63	C6	97	35	6A
i	24	25	26	27	28	29	30	31
$RC[i]$	D4	B3	7D	FA	EF	C5	91	39

D. Test Vectors

D.1 KeyExpansion

In this section we give test vectors for the key expansion in the case where both block length and key length are equal to 128. The all-zero key is expanded into the following:

```
 0 00000000000000000000000000000000
 1 62636363626363636263636362636363
 2 9B9898C9F9FBFBAA9B9898C9F9FBFBAA
 3 90973450696CCFFAF2F457330B0FAC99
 4 EE06DA7B876A1581759E42B27E91EE2B
 5 7F2E2B88F8443E098DDA7CBBF34B9290
 6 EC614B851425758C99FF09376AB49BA7
 7 217517873550620BACAF6B3CC61BF09B
 8 0EF903333BA9613897060A04511DFA9F
 9 B1D4D8E28A7DB9DA1D7BB3DE4C664941
10 B4EF5BCB3E92E21123E951CF6F8F188E
```

D.2 Rijndael(128,128)

In this section we gives test vectors for all intermediate steps of one encryption. A 128-bit plaintext is encrypted under a 128-bit key. These test vectors are a subset of the extensive set of test vectors generated by Brian Gladman.

```
LEGEND - round r = 0 to 10
input:   cipher input
start:   state at start of round[r]
s_box:   state after s_box substitution
s_row:   state after shift row transformation
m_col:   state after mix column transformation
k_sch:   key schedule value for round[r]
output:  cipher output

PLAINTEXT:     3243f6a8885a308d313198a2e0370734
KEY:           2b7e151628aed2a6abf7158809cf4f3c
```

```
ENCRYPT        16 byte block, 16 byte key
R[00].input    3243f6a8885a308d313198a2e0370734
R[00].k_sch    2b7e151628aed2a6abf7158809cf4f3c
R[01].start    193de3bea0f4e22b9ac68d2ae9f84808
R[01].s_box    d42711aee0bf98f1b8b45de51e415230
R[01].s_row    d4bf5d30e0b452aeb84111f11e2798e5
R[01].m_col    046681e5e0cb199a48f8d37a2806264c
R[01].k_sch    a0fafe1788542cb123a339392a6c7605
R[02].start    a49c7ff2689f352b6b5bea43026a5049
R[02].s_box    49ded28945db96f17f39871a7702533b
R[02].s_row    49db873b453953897f02d2f177de961a
R[02].m_col    584dcaf11b4b5aacdbe7caa81b6bb0e5
R[02].k_sch    f2c295f27a96b9435935807a7359f67f
R[03].start    aa8f5f0361dde3ef82d24ad26832469a
R[03].s_box    ac73cf7befc111df13b5d6b545235ab8
R[03].s_row    acc1d6b8efb55a7b1323cfdf457311b5
R[03].m_col    75ec0993200b633353c0cf7cbb25d0dc
R[03].k_sch    3d80477d4716fe3e1e237e446d7a883b
R[04].start    486c4eee671d9d0d4de3b138d65f58e7
R[04].s_box    52502f2885a45ed7e311c807f6cf6a94
R[04].s_row    52a4c89485116a28e3cf2fd7f6505e07
R[04].m_col    0fd6daa9603138bf6fc0106b5eb31301
R[04].k_sch    ef44a541a8525b7fb671253bdb0bad00
R[05].start    e0927fe8c86363c0d9b1355085b8be01
R[05].s_box    e14fd29be8fbfbba35c89653976cae7c
R[05].s_row    e1fb967ce8c8ae9b356cd2ba974ffb53
R[05].m_col    25d1a9adbd11d168b63a338e4c4cc0b0
R[05].k_sch    d4d1c6f87c839d87caf2b8bc11f915bc
R[06].start    f1006f55c1924cef7cc88b325db5d50c
R[06].s_box    a163a8fc784f29df10e83d234cd503fe
R[06].s_row    a14f3dfe78e803fc10d5a8df4c632923
R[06].m_col    4b868d6d2c4a8980339df4e837d218d8
R[06].k_sch    6d88a37a110b3efddbf98641ca0093fd
R[07].start    260e2e173d41b77de86472a9fdd28b25
R[07].s_box    f7ab31f02783a9ff9b4340d354b53d3f
R[07].s_row    f783403f27433df09bb531ff54aba9d3
R[07].m_col    1415b5bf461615ec274656d7342ad843
R[07].k_sch    4e54f70e5f5fc9f384a64fb24ea6dc4f
R[08].start    5a4142b11949dc1fa3e019657a8c040c
R[08].s_box    be832cc8d43b86c00ae1d44dda64f2fe
R[08].s_row    be3bd4fed4e1f2c80a642cc0da83864d
R[08].m_col    00512fd1b1c889ff54766dcdfa1b99ea
R[08].k_sch    ead27321b58dbad2312bf5607f8d292f
R[09].start    ea835cf00445332d655d98ad8596b0c5
R[09].s_box    87ec4a8cf26ec3d84d4c46959790e7a6
R[09].s_row    876e46a6f24ce78c4d904ad897ecc395
R[09].m_col    473794ed40d4e4a5a3703aa64c9f42bc
R[09].k_sch    ac7766f319fadc2128d12941575c006e
R[10].start    eb40f21e592e38848ba113e71bc342d2
R[10].s_box    e9098972cb31075f3d327d94af2e2cb5
R[10].s_row    e9317db5cb322c723d2e895faf090794
R[10].k_sch    d014f9a8c9ee2589e13f0cc8b6630ca6
R[10].output   3925841d02dc09fbdc118597196a0b32
```

D.3 Other Block Lengths and Key Lengths

The values in this section correspond to the ciphertexts obtained by encrypting the all-zero string with the all-zero key (values on the first lines), and by encrypting the result again with the all-zero key (values on the second lines). The values are given for the five different block lengths and the five different key lengths. The values were generated with the program listed in Appendix E.

```
block length 128  key length 128
66E94BD4EF8A2C3B884CFA59CA342B2E
F795BD4A52E29ED713D313FA20E98DBC

block length 160  key length 128
9E38B8EB1D2025A1665AD4B1F5438BB5CAE1AC3F
939C167E7F916D45670EE21BFC939E1055054A96

block length 192  key length 128
A92732EB488D8BB98ECD8D95DC9C02E052F250AD369B3849
106F34179C3982DDC6750AA01936B7A180E6B0B9D8D690EC

block length 224  key length 128
0623522D88F7B9C63437537157F625DD5697AB628A3B9BE2549895C8
93F93CBDABE23415620E6990B0443D621F6AFBD6EDEFD6990A1965A8

block length 256  key length 128
A693B288DF7DAE5B1757640276439230DB77C4CD7A871E24D6162E54AF434891
5F05857C80B68EA42CCBC759D42C28D5CD490F1D180C7A9397EE585BEA770391

block length 128  key length 160
94B434F8F57B9780F0EFF1A9EC4C112C
35A00EC955DF43417CEAC2AB2B3F3E76

block length 160  key length 160
33B12AB81DB7972E8FDC529DDA46FCB529B31826
97F03EB018C0BB9195BF37C6A0AECE8E4CB8DE5F

block length 192  key length 160
528E2FFF6005427B67BB1ED31ECC09A69EF41531DF5BA5B2
71C7687A4C93EBC35601E3662256E10115BEED56A410D7AC

block length 224  key length 160
58A0C53F3822A32464704D409C2FD0521F3A93E1F6FCFD4C87F1C551
D8E93EF2EB49857049D6F6E0F40B67516D2696F94013C065283F7F01

block length 256  key length 160
938D36E0CB6B7937841DAB7F1668E47B485D3ACD6B3F6D598B0A9F923823331D
7B44491D1B24A93B904D171F074AD69669C2B70B134A4D2D773250A4414D78BE

block length 128  key length 192
AAE06992ACBF52A3E8F4A96EC9300BD7
52F674B7B9030FDAB13D18DC214EB331
```

```
block length 160  key length 192
33060F9D4705DDD2C7675F0099140E5A98729257
012CAB64982156A5710E790F85EC442CE13C520F

block length 192  key length 192
C6348BE20007BAC4A8BD62890C8147A2432E760E9A9F9AB8
EB9DEF13C253F81C1FC2829426ED166A65A105C6A04CA33D

block length 224  key length 192
3856B17BEA77C4611E3397066828AADDA004706A2C8009DF40A811FE
160AD76A97AE2C1E05942FDE3DA2962684A92CCC74B8DC23BDE4F469

block length 256  key length 192
F927363EF5B3B4984A9EB9109844152EC167F08102644E3F9028070433DF9F2A
4E03389C68B2E3F623AD8F7F6BFC88613B86F334F4148029AE25F50DB144B80C

block length 128  key length 224
73F8DFF62A36F3EBF31D6F73A56FF279
3A72F21E10B6473EA9FF14A232E675B4

block length 160  key length 224
E9F5EA0FA39BB6AD7339F28E58E2E7535F261827
06EF9BC82905306D45810E12D0807796A3D338F9

block length 192  key length 224
ECBE9942CD6703E16D358A829D542456D71BD3408EB23C56
FD10458ED034368A34047905165B78A6F0591FFEEBF47CC7

block length 224  key length 224
FE1CF0C8DDAD24E3D751933100E8E89B61CD5D31C96ABFF7209C495C
515D8E2F2B9C5708F112C6DE31CACA47AFB86838B716975A24A09CD4

block length 256  key length 224
BC18BF6D369C955BBB271CBCDD66C368356DBA5B33C0005550D2320B1C617E21
60ABA1D2BE45D8ABFDCF97BCB39F6C17DF29985CF321BAB75E26A26100AC00AF

block length 128  key length 256
DC95C078A2408989AD48A21492842087
08C374848C228233C2B34F332BD2E9D3

block length 160  key length 256
30991844F72973B3B2161F1F11E7F8D9863C5118
EEF8B7CC9DBE0F03A1FE9D82E9A759FD281C67E0

block length 192  key length 256
17004E806FAEF168FC9CD56F98F070982075C70C8132B945
BED33B0AF364DBF15F9C2F3FB24FBDF1D36129C586EEA6B7

block length 224  key length 256
9BF26FAD5680D56B572067EC2FE162F449404C86303F8BE38FAB6E02
658F144A34AF44AAE66CFDDAB955C483DFBCB4EE9A19A6701F158A66
```

```
block length 256  key length 256
C6227E7740B7E53B5CB77865278EAB0726F62366D9AABAD908936123A1FC8AF3
9843E807319C32AD1EA3935EF56A2BA96E4BF19C30E47D88A2B97CBBF2E159E7
```

E. Reference Code

```
/* Rijndael code   August '01
 *
 * author: Vincent Rijmen,

 * This code is based on the official reference code
 * by Paulo Barreto and Vincent Rijmen
 *
 * This code is placed in the public domain.
 * Without any warranty of fitness for any purpose.
 */

#include <stdio.h>

typedef unsigned char word8;
typedef unsigned int word32;

  /* The tables Logtable and Alogtable are used to perform
   * multiplications in GF(256)
   */
word8 Logtable[256] = {
  0,  0, 25,  1, 50,  2, 26,198, 75,199, 27,104, 51,238,223,  3,
100,  4,224, 14, 52,141,129,239, 76,113,  8,200,248,105, 28,193,
125,194, 29,181,249,185, 39,106, 77,228,166,114,154,201,  9,120,
101, 47,138,  5, 33, 15,225, 36, 18,240,130, 69, 53,147,218,142,
150,143,219,189, 54,208,206,148, 19, 92,210,241, 64, 70,131, 56,
102,221,253, 48,191,  6,139, 98,179, 37,226,152, 34,136,145, 16,
126,110, 72,195,163,182, 30, 66, 58,107, 40, 84,250,133, 61,186,
 43,121, 10, 21,155,159, 94,202, 78,212,172,229,243,115,167, 87,
175, 88,168, 80,244,234,214,116, 79,174,233,213,231,230,173,232,
 44,215,117,122,235, 22, 11,245, 89,203, 95,176,156,169, 81,160,
127, 12,246,111, 23,196, 73,236,216, 67, 31, 45,164,118,123,183,
204,187, 62, 90,251, 96,177,134, 59, 82,161,108,170, 85, 41,157,
151,178,135,144, 97,190,220,252,188,149,207,205, 55, 63, 91,209,
 83, 57,132, 60, 65,162,109, 71, 20, 42,158, 93, 86,242,211,171,
 68, 17,146,217, 35, 32, 46,137,180,124,184, 38,119,153,227,165,
103, 74,237,222,197, 49,254, 24, 13, 99,140,128,192,247,112,  7};
```

```
word8 Alogtable[256] = {
  1,  3,  5, 15, 17, 51, 85,255, 26, 46,114,150,161,248, 19, 53,
 95,225, 56, 72,216,115,149,164,247,  2,  6, 10, 30, 34,102,170,
229, 52, 92,228, 55, 89,235, 38,106,190,217,112,144,171,230, 49,
 83,245,  4, 12, 20, 60, 68,204, 79,209,104,184,211,110,178,205,
 76,212,103,169,224, 59, 77,215, 98,166,241,  8, 24, 40,120,136,
131,158,185,208,107,189,220,127,129,152,179,206, 73,219,118,154,
181,196, 87,249, 16, 48, 80,240, 11, 29, 39,105,187,214, 97,163,
254, 25, 43,125,135,146,173,236, 47,113,147,174,233, 32, 96,160,
251, 22, 58, 78,210,109,183,194, 93,231, 50, 86,250, 21, 63, 65,
195, 94,226, 61, 71,201, 64,192, 91,237, 44,116,156,191,218,117,
159,186,213,100,172,239, 42,126,130,157,188,223,122,142,137,128,
155,182,193, 88,232, 35,101,175,234, 37,111,177,200, 67,197, 84,
252, 31, 33, 99,165,244,  7,  9, 27, 45,119,153,176,203, 70,202,
 69,207, 74,222,121,139,134,145,168,227, 62, 66,198, 81,243, 14,
 18, 54, 90,238, 41,123,141,140,143,138,133,148,167,242, 13, 23,
 57, 75,221,124,132,151,162,253, 28, 36,108,180,199, 82,246,  1};

word8 S[256] = {
 99,124,119,123,242,107,111,197, 48,  1,103, 43,254,215,171,118,
202,130,201,125,250, 89, 71,240,173,212,162,175,156,164,114,192,
183,253,147, 38, 54, 63,247,204, 52,165,229,241,113,216, 49, 21,
  4,199, 35,195, 24,150,  5,154,  7, 18,128,226,235, 39,178,117,
  9,131, 44, 26, 27,110, 90,160, 82, 59,214,179, 41,227, 47,132,
 83,209,  0,237, 32,252,177, 91,106,203,190, 57, 74, 76, 88,207,
208,239,170,251, 67, 77, 51,133, 69,249,  2,127, 80, 60,159,168,
 81,163, 64,143,146,157, 56,245,188,182,218, 33, 16,255,243,210,
205, 12, 19,236, 95,151, 68, 23,196,167,126, 61,100, 93, 25,115,
 96,129, 79,220, 34, 42,144,136, 70,238,184, 20,222, 94, 11,219,
224, 50, 58, 10, 73,  6, 36, 92,194,211,172, 98,145,149,228,121,
231,200, 55,109,141,213, 78,169,108, 86,244,234,101,122,174,  8,
186,120, 37, 46, 28,166,180,198,232,221,116, 31, 75,189,139,138,
112, 62,181,102, 72,  3,246, 14, 97, 53, 87,185,134,193, 29,158,
225,248,152, 17,105,217,142,148,155, 30,135,233,206, 85, 40,223,
140,161,137, 13,191,230, 66,104, 65,153, 45, 15,176, 84,187, 22};

word8 Si[256] = {
 82,  9,106,213, 48, 54,165, 56,191, 64,163,158,129,243,215,251,
124,227, 57,130,155, 47,255,135, 52,142, 67, 68,196,222,233,203,
 84,123,148, 50,166,194, 35, 61,238, 76,149, 11, 66,250,195, 78,
  8, 46,161,102, 40,217, 36,178,118, 91,162, 73,109,139,209, 37,
114,248,246,100,134,104,152, 22,212,164, 92,204, 93,101,182,146,
108,112, 72, 80,253,237,185,218, 94, 21, 70, 87,167,141,157,132,
144,216,171,  0,140,188,211, 10,247,228, 88,  5,184,179, 69,  6,
208, 44, 30,143,202, 63, 15,  2,193,175,189,  3,  1, 19,138,107,
 58,145, 17, 65, 79,103,220,234,151,242,207,206,240,180,230,115,
150,172,116, 34,231,173, 53,133,226,249, 55,232, 28,117,223,110,
 71,241, 26,113, 29, 41,197,137,111,183, 98, 14,170, 24,190, 27,
252, 86, 62, 75,198,210,121, 32,154,219,192,254,120,205, 90,244,
 31,221,168, 51,136,  7,199, 49,177, 18, 16, 89, 39,128,236, 95,
 96, 81,127,169, 25,181, 74, 13, 45,229,122,159,147,201,156,239,
160,224, 59, 77,174, 42,245,176,200,235,187, 60,131, 83,153, 97,
 23, 43,  4,126,186,119,214, 38,225,105, 20, 99, 85, 33, 12,125};
```

```
word32 RC[30] = {
0x00,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,
0x1B,0x36,0x6C,0xD8,0xAB,0x4D,0x9A,0x2F,0x5E,
0xBC,0x63,0xC6,0x97,0x35,0x6A,0xD4,0xB3,0x7D,
0xFA,0xEF,0xC5};

#define MAXBC               8
#define MAXKC               8
#define MAXROUNDS           14

static word8 shifts[5][4] = {
   0, 1, 2, 3,
   0, 1, 2, 3,
   0, 1, 2, 3,
   0, 1, 2, 4,
   0, 1, 3, 4};

static int numrounds[5][5] = {
    10, 11, 12, 13, 14,
    11, 11, 12, 13, 14,
    12, 12, 12, 13, 14,
    13, 13, 13, 13, 14,
    14, 14, 14, 14, 14};

int BC, KC, ROUNDS;

word8 mul(word8 a, word8 b) {
   /* multiply two elements of GF(256)
    * required for MixColumns and InvMixColumns
    */
    if (a && b) return Alogtable[(Logtable[a] + Logtable[b])%255];
    else return 0;
}

void AddRoundKey(word8 a[4][MAXBC], word8 rk[4][MAXBC]) {
    /* XOR corresponding text input and round key input bytes
     */
    int i, j;

    for(i = 0; i < 4; i++)
        for(j = 0; j < BC; j++) a[i][j] ^= rk[i][j];
}

void SubBytes(word8 a[4][MAXBC], word8 box[256]) {
    /* Replace every byte of the input by the byte at that place
     * in the non-linear S-box
     */
    int i, j;

    for(i = 0; i < 4; i++)
        for(j = 0; j < BC; j++) a[i][j] = box[a[i][j]] ;
}
```

```
void ShiftRows(word8 a[4][MAXBC], word8 d) {
    /* Row 0 remains unchanged
     * The other three rows are shifted a variable amount
     */
    word8 tmp[MAXBC];
    int i, j;

    if (d == 0) {
        for(i = 1; i < 4; i++) {
            for(j = 0; j < BC; j++)
               tmp[j] = a[i][(j + shifts[BC-4][i]) % BC];
            for(j = 0; j < BC; j++) a[i][j] = tmp[j];
        }
    }
    else {
        for(i = 1; i < 4; i++) {
            for(j = 0; j < BC; j++)
               tmp[j] = a[i][(BC + j - shifts[BC-4][i]) % BC];
            for(j = 0; j < BC; j++) a[i][j] = tmp[j];
        }
    }
}

void MixColumns(word8 a[4][MAXBC]) {
    /* Mix the four bytes of every column in a linear way
     */
    word8 b[4][MAXBC];
    int i, j;

    for(j = 0; j < BC; j++)
        for(i = 0; i < 4; i++)
            b[i][j] = mul(2,a[i][j])
                ^ mul(3,a[(i + 1) % 4][j])
                ^ a[(i + 2) % 4][j]
                ^ a[(i + 3) % 4][j];
    for(i = 0; i < 4; i++)
        for(j = 0; j < BC; j++) a[i][j] = b[i][j];
}

void InvMixColumns(word8 a[4][MAXBC]) {
    /* Mix the four bytes of every column in a linear way
     * This is the opposite operation of Mixcolumns
     */
    word8 b[4][MAXBC];
    int i, j;

    for(j = 0; j < BC; j++)
    for(i = 0; i < 4; i++)
        b[i][j] = mul(0xe,a[i][j])
            ^ mul(0xb,a[(i + 1) % 4][j])
            ^ mul(0xd,a[(i + 2) % 4][j])
            ^ mul(0x9,a[(i + 3) % 4][j]);
```

```
    for(i = 0; i < 4; i++)
        for(j = 0; j < BC; j++) a[i][j] = b[i][j];
}

int KeyExpansion (word8 k[4][MAXKC],
         word8 W[MAXROUNDS+1][4][MAXBC]) {
    /* Calculate the required round keys
     */
    int i, j, t, RCpointer = 1;
    word8 tk[4][MAXKC];


    for(j = 0; j < KC; j++)
        for(i = 0; i < 4; i++)
            tk[i][j] = k[i][j];
    t = 0;
    /* copy values into round key array */
    for(j = 0; (j < KC) && (t < (ROUNDS+1)*BC); j++, t++)
        for(i = 0; i < 4; i++) W[t / BC][i][t % BC] = tk[i][j];

    while (t < (ROUNDS+1)*BC) {
        /* while not enough round key material calculated,
         * calculate new values
         */
        for(i = 0; i < 4; i++)
            tk[i][0] ^= S[tk[(i+1)%4][KC-1]];
        tk[0][0] ^= RC[RCpointer++];

        if (KC <= 6)
            for(j = 1; j < KC; j++)
                for(i = 0; i < 4; i++) tk[i][j] ^= tk[i][j-1];
        else {
            for(j = 1; j < 4; j++)
                for(i = 0; i < 4; i++) tk[i][j] ^= tk[i][j-1];
            for(i = 0; i < 4; i++) tk[i][4] ^= S[tk[i][3]];
            for(j = 5; j < KC; j++)
                for(i = 0; i < 4; i++) tk[i][j] ^= tk[i][j-1];
    }
    /* copy values into round key array */
    for(j = 0; (j < KC) && (t < (ROUNDS+1)*BC); j++, t++)
        for(i = 0; i < 4; i++) W[t / BC][i][t % BC] = tk[i][j];
    }

    return 0;
}

int Encrypt (word8 a[4][MAXBC], word8 rk[MAXROUNDS+1][4][MAXBC])
{
    /* Encryption of one block.
     */
    int r;

    /* begin with a key addition
```

```
     */
    AddRoundKey(a,rk[0]);

        /* ROUNDS-1 ordinary rounds
     */
    for(r = 1; r < ROUNDS; r++) {
        SubBytes(a,S);
        ShiftRows(a,0);
        MixColumns(a);
        AddRoundKey(a,rk[r]);
    }

    /* Last round is special: there is no MixColumns
     */
    SubBytes(a,S);
    ShiftRows(a,0);
    AddRoundKey(a,rk[ROUNDS]);

    return 0;
}


int Decrypt (word8 a[4][MAXBC], word8 rk[MAXROUNDS+1][4][MAXBC])
{
    int r;

    /* To decrypt:
     *   apply the inverse operations of the encrypt routine,
     *   in opposite order
     *
     * - AddRoundKey is equal to its inverse)
     * - the inverse of SubBytes with table S is
     *             SubBytes with the inverse table of S)
     * - the inverse of Shiftrows is Shiftrows over
     *       a suitable distance)
     */

    /* First the special round:
     *   without InvMixColumns
     *   with extra AddRoundKey
     */
    AddRoundKey(a,rk[ROUNDS]);
    SubBytes(a,Si);
    ShiftRows(a,1);

    /* ROUNDS-1 ordinary rounds
     */
    for(r = ROUNDS-1; r > 0; r--) {
        AddRoundKey(a,rk[r]);
        InvMixColumns(a);
        SubBytes(a,Si);
        ShiftRows(a,1);
    }
```

```
    /* End with the extra key addition
     */
    AddRoundKey(a,rk[0]);

    return 0;
}


int main() {

    int i, j;
    word8 a[4][MAXBC], rk[MAXROUNDS+1][4][MAXBC], sk[4][MAXKC];

    for(KC = 4; KC <= 8; KC++)
        for(BC = 4; BC <= 8; BC++) {
            ROUNDS = numrounds[KC-4][BC-4];
            for(j = 0; j < BC; j++)
                for(i = 0; i < 4; i++) a[i][j] = 0;
            for(j = 0; j < KC; j++)
                for(i = 0; i < 4; i++) sk[i][j] = 0;
            KeyExpansion(sk,rk);
            Encrypt(a,rk);
            printf("block length %d  key length %d\n",32*BC,32*KC);
            for(j = 0; j < BC; j++)
                for(i = 0; i < 4; i++) printf("%02X",a[i][j]);
            printf("\n");

            Encrypt(a,rk);
            for(j = 0; j < 4; j++)
                for(i = 0; i < 4; i++) printf("%02X",a[i][j]);
            printf("\n");
            printf("\n");
        }

    return 0;
}
```

Bibliography

1. C. Adams, S. Tavares, "The Structured Design of Cryptographically Good S-Boxes," *Journal of Cryptology*, Vol. 3, No. 1, 1990, pp. 27–42.
2. M.-L. Akkar, C. Giraud, "An implementation of DES and AES, secure against some attacks," *Cryptographic Hardware and Embedded Systems CHES 2001, LNCS 2162*, Ç.K. Koç, D. Naccache, C. Paar, Eds., Springer-Verlag, 2001, pp. 315–324.
3. R.A. Anderson, E. Biham, L.R. Knudsen, "Serpent", *Proc. of the 1st AES candidate conference,* CD-1: Documentation, August 20–22, 1998, Ventura.
4. P.S.L.M. Barreto, V. Rijmen, "The Anubis block cipher," available from the NESSIE homepage, URL: `http://cryptonessie.org`.
5. O. Baudron, H. Gilbert, L. Granboulan, H. Handschuh, A. Joux, P. Nguyen, F. Noilhan, D. Pointcheval, T. Pornin, G. Poupard, J. Stern, S. Vaudenay, "Report on the AES candidates," *Proc. of the 2nd AES candidate conference,* March 22–23, 1999, Rome, pp. 53–67.
6. E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys," *Advances in Cryptology, Proc. Eurocrypt'93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 398–409.
7. E. Biham, "A note on comparing the AES candidates," *Proc. of the 2nd AES candidate conference,* March 22–23, 1999, Rome, pp. 85–92.
8. E. Biham, O. Dunkelman, N. Keller, "Linear Cryptanalysis of Reduced Round Serpent," *Fast Software Encryption 2001, LNCS,* M. Matsui, Ed., Springer-Verlag, to appear.
9. E. Biham, A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Journal of Cryptology*, Vol. 4, No. 1, 1991, pp. 3–72.
10. E. Biham, A. Shamir, "Power Analysis of the Key Scheduling of the AES Candidates," *Proc. of the 2nd AES candidate conference,* March 22–23, 1999, Rome, pp. 115–121.
11. A. Biryukov, A. Shamir, "Structural cryptanalysis of SASAS," *Advances in Cryptology, Proc. Eurocrypt'01, LNCS 2045*, B. Pfitzmann, Ed., Springer-Verlag, 2001, pp. 394–405.
12. A. Biryukov, D. Wagner, "Slide attacks," *Fast Software Encryption '99, LNCS 1636*, L. Knudsen, Ed., Springer-Verlag, 1999, pp. 245–259.
13. J. Borst, "Weak keys of Crypton," available from NIST's AES homepage, URL: `http://www.nist.gov/aes`.

14. J. Borst, "The block cipher: Grand Cru," available from the NESSIE homepage, URL: `http://cryptonessie.org`.

15. S. Chari, C. Jutla, J.R. Rao, P.J. Rohatgi, "A cautionary note regarding evaluation of AES candidates on smart cards," *Proc. of the 2nd AES candidate conference,* March 22–23, 1999, Rome, pp. 133–150.

16. S. Chari, C. Jutla, J.R. Rao, P.J. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," *Advances in Cryptology, Proc. Crypto'99, LNCS 1666*, M. Wiener, Ed., Springer-Verlag, 1999, pp. 398–412.

17. C.S.K. Clapp, "Instruction-level parallelism in AES candidates,", *Proc. of the 2nd AES candidate conference,* March 22–23, 1999, Rome, pp. 68–84.

18. J. Daemen, "Cipher and hash function design strategies based on linear and differential cryptanalysis," *Doctoral Dissertation*, March 1995, K.U.Leuven.

19. J. Daemen, R. Govaerts, J. Vandewalle, "Weak Keys of IDEA," *Advances in Cryptology, Proc. Crypto'93, LNCS 773*, D. Stinson, Ed., Springer-Verlag, 1994, pp. 224–231.

20. J. Daemen, R. Govaerts, J. Vandewalle, "A New Approach towards Block Cipher Design," *Fast Software Encryption '93, LNCS 809*, R. Anderson, Ed., Springer-Verlag, 1994, pp. 18–33.

21. J. Daemen, L.R. Knudsen, V. Rijmen, "The block cipher Square," *Fast Software Encryption '97, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 149–165.

22. J. Daemen, L.R. Knudsen, V. Rijmen, "Linear frameworks for block ciphers," *Designs, Codes and Cryptography*, Vol. 22, No. 1, January 2001, pp. 65–87.

23. J. Daemen, M. Peeters, G. Van Assche, "Bitslice ciphers and Implementation attacks," *Fast Software Encryption 2000, LNCS 1978,* B. Schneier, Ed., Springer-Verlag, 2001, pp. 134–149.

24. J. Daemen, M. Peeters, V. Rijmen, G. Van Assche, "Nessie Proposal: Noekeon," URL: `http://www.protonworld.com/research/`.

25. J. Daemen, V. Rijmen, "The Block Cipher BKSQ," *Smart Card Research and Applications, LNCS 1820,* J.-J. Quisquater and B. Schneier, Eds., Springer-Verlag, 2000, pp. 247–256.

26. J. Daemen, V. Rijmen, "The block cipher Rijndael," available from NIST's AES homepage, URL: `http://www.nist.gov/aes`.

27. J. Daemen, V. Rijmen, "Resistance against implementation attacks: a comparative study of the AES proposals," *Proc. of the 2nd AES candidate conference,* March 22–23, 1999, Rome, pp. 122–132.

28. D.W. Davies, Some Regular Properties of the DES, in *Advances in Cryptology, Proc. Crypto'82*, D. Chaum, R. Rivest and A. Sherman, Eds., Plenum Press, 1983, pp. 89–96.

29. A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar, "An FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalists," *Proc. of the 2nd AES candidate conference,* March 22–23, 1999, Rome, pp. 13–27.

30. H. Feistel, W.A. Notz, J.L. Smith, "Some cryptographic techniques for machine-to-machine data communications," *Proc. IEEE*, Vol. 63, No. 11, 1975, pp. 1545–1554.

31. N. Ferguson, J. Kelsey, B. Schneier, M. Stay, D. Wagner, D. Whiting, "Improved cryptanalysis of Rijndael," *Fast Software Encryption 2000, LNCS 1978,* B. Schneier, Ed., Springer-Verlag, 2001, pp. 213–231.

32. N. Ferguson, R. Schroeppel, D. Whiting, "A simple algebraic representation of Rijndael," *Selected Areas in Cryptography '01, LNCS*, Springer-Verlag, to appear.

33. *Data Encryption Standard*, Federal Information Processing Standard (FIPS), Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C. , January 1977.

34. K. Gaj, P. Chodowiec, "Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays," RSA conference 2001, April 2001.

35. H. Gilbert, M. Girault, P. Hoogvorst, F. Noilhan, T. Pornin, G. Poupard, J, Stern, S. Vaudenay, "Decorrelated fast cipher: an AES candidate," *Proc. of the 1st AES candidate conference,* CD-1: Documentation, August 20–22, 1998, Ventura.

36. H. Gilbert, M. Minier, "A collision attack on 7 rounds of Rijndael," *Proc. of the 3rd AES candidate conference,* April 13–14, 2000, New York, pp. 230–241.

37. B. Gladman, "Implementation experience with the AES candidate algorithms," *Proc. of the 2nd AES candidate conference,* March 22–23, 1999, Rome, pp. 7–14.

38. S.W. Golomb, *Shift Register Sequences*, Holden–Day Inc., San Francisco, 1967.

39. L. Goubin, J. Patarin, "DES and differential power analysis – the duplication method," *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems,* Springer-Verlag, 1999, pp. 158–172.

40. G. Hachez, F. Koeune, J.-J. Quisquater, "cAESar results: Implementation of Four AES Candidates on Two Smart Cards," *Proc. of the 2nd AES candidate conference,* March 22–23, 1999, Rome, pp. 95–108.

41. M. Hellman, R. Merkle, R. Schroeppel, L. Washington, W. Diffie, S. Pohlig, P. Schweitzer, Results of an initial attempt to cryptanalyze the NBS Data Encryption Standard, *Information Systems Lab., Dept. of Electrical Eng., Stanford Univ.*, 1976.

42. ISO/IEC 9797-1, *"Information technology – Security Techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher,"* International Organisation for Standardization, Geneva, Switzerland, 1999.

43. ISO/IEC 10116, *Information technology – Security techniques – Modes of operation of an n-bit block cipher algorithm*, International Organisation for Standardization, Geneva, Switzerland, 1997.

44. ISO/IEC 10118-2 *Information technology - Security techniques - Hash-functions, Part 2: Hash-functions using an n-bit block cipher algorithm*, International Organisation for Standardization, Geneva, Switzerland, 1994.

45. T. Jakobsen, "Cryptanalysis of block ciphers with probabilistic non-linear relations of low degree," *Advances in Cryptology, Proc. Crypto'98, LNCS 1462*, H. Krawczyk, Ed., Springer-Verlag, 1998, pp. 212–222.

46. T. Jakobsen, L.R. Knudsen, "The interpolation attack on block ciphers," *Fast Software Encryption '97, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 28–40.

47. J.B. Kam, G.I. Davida, Structured design of substitution-permutation encryption networks, *IEEE Trans. on Computers*, Vol. C–28, 1979, pp. 747–753.

48. G. Keating, "Performance analysis of AES candidates on the 6805 CPU core" URL: `http://www.ozemail.com.au/~geoffk/aes-6805/`.

49. J. Kelsey, B. Schneier, D. Wagner, "Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES," *Advances in Cryptology, Proc. Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 237–252.

50. P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," *Advances in Cryptology, Proc. Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 104–113.

51. P. Kocher, J. Jaffe, B. Jun, "Differential power analysis," *Advances in Cryptology, Proc. Crypto'99, LNCS 1666*, M. Wiener, Ed., Springer-Verlag, 1999, pp. 388–397.

52. L.R. Knudsen, "A key-schedule weakness in SAFER-K64," *Advances in Cryptology, Proc. Crypto'95, LNCS 963*, D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 275–286.

53. L.R. Knudsen, "Truncated and higher order differentials," *Fast Software Encryption '94, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 196–211.

54. L.R. Knudsen, V. Rijmen, "On the decorrelated fast cipher (DFC) and its theory," *Fast Software Encryption '99, LNCS 1636*, L. Knudsen, Ed., Springer-Verlag, 1999, pp. 81–94.

55. H. Kuo, I. Verbauwhede, "Architectural optimization for a 3Gbit/sec VLSI implementation of the AES Rijndael algorithm," preprint.

56. X. Lai, J.L. Massey, S. Murphy, "Markov Ciphers and Differential Cryptanalysis," *Advances in Cryptology, Proc. Eurocrypt'91, LNCS 547*, D.W. Davies, Ed., Springer-Verlag, 1991, pp. 17–38.

57. H. Lipmaa, "AES candidates: A survey of implementations," URL: `http://www.tcs.hut.fi/~helger/aes/`.

58. R. Lidl, H. Niederreiter, *Introduction to finite fields and their applications,* Cambridge University Press, 1986 (Reprinted 1988).

59. C.H. Lim, "Crypton: a new 128-bit block cipher," available from NIST's AES homepage, URL: `http://www.nist.gov/aes`.

60. C.H. Lim, "A revised version of Crypton: Crypton v1.0," *Fast Software Encryption '99, LNCS 1636*, L. Knudsen, Ed., Springer-Verlag, 1999, pp. 31–45.

61. S. Lucks, "The saturation attack – a bait for Twofish," *Fast Software Encryption 2001, LNCS,* M. Matsui, Ed., Springer-Verlag, to appear..

62. S. Lucks, "Attacking 7 rounds of Rijndael under 192-bit and 256-bit keys," *Proc. of the 3rd AES candidate conference,* April 13–14, 2000, New York, pp. 215–229.

63. F.J. MacWilliams, N.J.A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Publishing Company, 1978.

64. J. Massey, "SAFER K-64: a byte-oriented block-ciphering algorithm," *Fast Software Encryption '93, LNCS 809*, R. Anderson, Ed., Springer-Verlag, 1994, pp. 1–17.

65. M. Matsui, "Linear Cryptanalysis Method for DES Cipher," *Advances in Cryptology, Proc. Eurocrypt'93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 386–397.

66. M. Matsui, "The First Experimental Cryptanalysis of the Data Encryption Standard," *Advances in Cryptology, Proc. Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 1–11.

67. R.J. McEliece, *Finite Fields for Computer Scientists and Engineers*, Kluwer Academic Publishers, 1987.

68. A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, October 1996.

69. T.S. Messerges, "Securing the AES finalists against power analysis attacks," *Fast Software Encryption 2000, LNCS 1978,* B. Schneier, Ed., Springer-Verlag, 2001, pp. 150–164.

70. C.H. Meyer, S.M. Matyas, *Cryptography*, John Wiley & Sons, 1982.

71. J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, E. Roback, "Report on the Development of the Advanced Encryption Standard (AES)," available from NIST's AES homepage, URL: `http://www.nist.gov/aes`.

72. J. Nechvatal, E. Barker, D. Dodson, M. Dworkin, J. Foti, E. Roback, "Status report on the first round of the development of the Advanced Encryption Standard," available from NIST's AES homepage, URL: `http://www.nist.gov/aes`.

73. New European schemes for signatures, integrity and encryption (NESSIE), available from the NESSIE homepage, URL: `http://cryptonessie.org`.

74. K. Nyberg, "Differentially uniform mappings for cryptography," *Advances in Cryptology, Proc. Eurocrypt'93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 55–64.

75. K. Nyberg, "Linear Approximation of Block Ciphers," *Advances in Cryptology, Proc. Eurocrypt'94, LNCS 950*, A. De Santis, Ed., Springer-Verlag, 1995, pp. 439–444.

76. K. Nyberg, L.R. Knudsen, "Provable security against a differential attack," *Journal of Cryptology*, Vol. 8, No. 1, 1995, pp. 27–38.

77. L. O'Connor, "On the distribution of characteristics in bijective mappings," *Journal of Cryptology*, Vol. 8, No. 2, 1995, pp. 67–86..

78. C. Paar, M. Rosner, "Comparison of arithmetic architectures for Reed-Solomon decoders in reconfigurable hardware," *Fifth annual IEEE symposium on field-programmable custom computing machines (FCCM '97).*

79. B. Preneel, *Analysis and Design of Cryptographic Hash Functions*, Doct. Dissertation KULeuven, 1993.

80. V. Rijmen, "Cryptanalysis and design of iterated block ciphers," *Doctoral Dissertation*, October 1997, K.U.Leuven.

81. V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, E. De Win, "The cipher SHARK," *Fast Software Encryption '96, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 99–111.

82. R.L. Rivest, M.J.B. Robshaw, R. Sidney, Y.L. Yin, "The RC6 block cipher," *Proc. of the 1st AES candidate conference,* CD-1: Documentation, August 20–22, 1998, Ventura.

83. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, "Twofish, a 128-bit block cipher," available from NIST's AES homepage, URL: `http://www.nist.gov/aes`.

84. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, "Performance comparison of the AES submissions," *Proc. of the 2nd AES candidate conference,* March 22–23, 1999, Rome, pp. 15–34.

85. C.E. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. Journal*, Vol. 27, No. 3, 1948, pp. 379–423 and pp. 623–656.

86. C.E. Shannon, "Communication Theory of Secrecy Systems," *Bell Syst. Tech. Journal*, Vol. 28, 1949, pp. 656–715.

87. J. Swift, *Gulliver's travels.*

88. Toshiba corporation, "Specification of Hierocrypt-L1," available from the NESSIE homepage, URL: `http://cryptonessie.org`.

89. Toshiba corporation, "Specification of Hierocrypt-L3," available from the NESSIE homepage, URL: `http://cryptonessie.org`.

90. S. Vaudenay, "On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER," *Fast Software Encryption '94, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 286–297.

91. B. Weeks et al., "Hardware performance simulations of round 2 advanced encryption standard algorithms," National Security Agency white paper, May 15, 2000, available from NIST's AES homepage, URL: `http://www.nist.gov/aes`.

Index

Λ-set, 150
$\gamma\lambda$ structure, 127
S_{RD}, 35
3-Way, 67
32-bit platforms
– implementation, 56
8-bit platforms
– implementation, 53

Abelian group, 10
active S-box, 129
addition, 14, 185
`AddRoundKey`, 40
AES, 31
AES process, 1
affine function, 114
algebraic complexity, 35
Anubis, 172
associative, 10
attack
– differential, 83
– differential power analysis, 158
– Gilbert–Minier, 154
– herds, 154
– implementation, 157
– interpolation, 156
– linear, 85
– power analysis, 158
– related-key, 77, 157
– saturation, 149
– shortcut, 71
– slide, 66
– Square, 149
– timing, 157
– truncated differential, 149
– weak keys, 157
autocorrelation, 102

balanced, 99
BaseKing, 67
basis, 12, 91
big endian, 70
binary Boolean
– function, 20
bit-slice, 67
BKSQ, 162, 168
block cipher, VII, 23
Boolean
– function, 20
– functions, 19
– transformation, 20
– variable, 19
– vector, 20
branch number, 39, 130–132, 142, 144
bricklayer function, 22
bricklayer functions, 98, 114
bundle, 20
bundle transposition, 21
bundle weight, 129

candidates, 3
characteristic, 84, 120
cipher key, 24
circulant, 143
closed, 10
code
– linear, 17
– maximal distance separable (MDS), 19
coefficient, 13
column, 32, 38, 134
commutative, 10
complementation, 19
component, 20
conference, 4
coordinates, 12
correlation, 35, 36, 89
– contribution, 103
– matrices, 94
– matrix, 115
correlation potential, 92
cost, 4

cross-correlation, 101
cryptanalysis
– loyal, 6
Cryptix, 3
Crypton, 145, 171
cyclic representation, 184

D-box, 22
decryption, 45
– equivalent algorithm, 45, 48
– straightforward algorithm, 45
DES, 81
– differential cryptanalysis of, 83, 119
– linear cryptanalysis of, 85, 108
design criteria, 35
desynchronization, 159
deviation, 85
difference propagation, 113
– probability, 35, 36, 84, 113
– weight, 114
differential
– cryptanalysis, 117, 125
–– of the DES, 83
– power analysis, 158
– trail, 117, 125
–– weight of, 117
– truncated, 149
differentials, 120
diffusion, 39, 130, 131
diffusion optimal, 37, 138
disjunct, 94
dispersion, 131
distinguisher, 154
distributive, 11
distributivity, 11
DPA, 158
dual bases, 185
dual code, 18
dyadic, 93

echelon form, 18
effectiveness of a linear expression, 85
efficiency, 64
endian neutral, 70
entropy, 114
`EqFinalRound`, 49
`EqKeyExpansion`, 49
`EqRound`, 49
equivalent decryption algorithm, 45, 48
Euclidean algorithm, 15
evaluation criteria, 4
expanded key, 24
`ExpandedKey`, 40, 43
expansion, 82

F-function, 81
field, 11, *see* finite field
final round, 152
finalist, 6
finite field, 13, 53
– basis, 185
– characteristic, 13
– dual basis, 185
– inversion, 61
– order, 13
– representation, 184
– trace, 177, 185
FIPS, 1
fixed point, 36
flexibility, 69

generator, 184
generator matrix, 18
Gilbert–Minier attack, 154
GRAND CRU, 173
group, 10

Hamming
– distance, 18
– weight, 17
herds, 154
hermetic, 72
Hierocrypt, 173
hypothesis
– of independent round keys, 121
– of stochastic equivalence, 121

implementation
– 32-bit platforms, 56
– 8-bit platforms, 53
– attack, 157
– hardware, 59
indeterminate, 13
inner product, 90
intermediate state, 23
interpolation attack, 156
inverse element, 10
invertibility, 34
`InvKeyExpansion`, 56
`InvMixColumns`, 40
– implementation, 55
`InvShiftRows`, 38
`InvSubBytes`, 37
irreducible, 15
irreducible polynomial, 15
isomorphic, 13
iterated block cipher, 24

iterative
– block cipher, 24
– Boolean permutation, 23
– Boolean transformation, 22, 23
– characteristic, 84

K-secure, 71
key agility, 5, 64
key expansion, 43, 55, 77
– inverse implementation, 56
key schedule, 24, 43, 76, 106, 119
key selection, 77
key-alternating cipher, 25, 104, 118
key-iterated block ciphers, 25
key-iterated cipher, 25

linear
– code, 17, 18, 143
– combination, 12
– cryptanalysis, 89, 123
–– of the DES, 85
– expression, 85
–– effectiveness of, 85
– function, 12, 98, 114
– trail, 102, 123
little endian, 70
load balancing, 158
loyal cryptanalysis, 6

Markov, 120
– assumption, 84
– cipher, 120
masking, 159
maximal distance separable code, 19
MDS code, 19
MILENAGE, 2
`MixColumns`, 38
– implementation, 54
modes of operation, 27
multipermutation, 197
multiplication, 14, 15, 53

N_b, 32
neutral element, 10
NIST, 1
N_k, 32
Noekeon, 67
non-linearity, 35, 43, 129
norm, 90
N_r, 42

offsets, 37
opposite fixed point, 36

parallelism, 61, 68
parity, 89
parity-check matrix, 18
Parseval, 92
performance figures, 62
permutation, 82
piling-up lemma, 109
pipelining, 62
polynomial, 13
power analysis, 158
propagation, 113

related-key attack, 77, 157
representation
– cyclic, 184
– vector space, 184
restriction, 100, 117
right pairs, 83
ring, 10
round, 24, 33
– constant, 44, 67, 214
– final, 33
– key, 24, 40
– number of, 41
– transformation, 24, 33, 162
round transformation, 24

S-box, 22, 82, 211
S/N ratio, 84
saturation attack, 37, 149
scalar, 12
– multiplication, 12
scope, 1
security, 4
security margin, 41, 63
selection pattern, 85, 89, 102, 129, 133
Serpent, 67
SHARK, 145, 161, 163
`ShiftRows`, 37
shortcut attacks, 71
signal-to-noise ratio, 84
simplicity, 5, 34, 65
Singleton bound, 19
slide attack, 66
smart card, 53
SPA, 158
spectrum, 91
Square, 145, 162, 165
Square attack, 149
S_{RD}, 211
state, 20, 31
stochastic equivalence, 121
straightforward decryption algorithm, 45

structural attack, *see* saturation attack
`SubBytes`, 34
– implementation, 59
submission requirements, 2
suggested keys, 83
support space, 94
symmetry, 65, 156

T-table, 58
test vectors, 215
timing attack, 157
trace parity, 178
trace pattern, 178
trail, 119, 123, 125, 130, 143
– differential, 117
– linear, 102
transposition, 21
trivial
– characteristic, 84
– expression, 86
truncated differential, 149
truncated function, 100, 115
Twofish, 172

unit element, 11

vector, 12
– addition, 12
vector space, 11
– basis, 12
– coordinates, 12
vector space representation, 184
versatility, 4, 64

Walsh-Hadamard, 91
weak keys, 157
weight
– of a difference propagation, 114
– of a trail, 117, 129
wide trail strategy, 126
wrong pairs, 83

XOR, 19
`xtime`, 53, 158, 212