KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT DER TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK-ESAT
Kasteelpark Arenberg 10 – 3001 Leuven (Heverlee)

BLOCK CIPHERS: DESIGN, ANALYSIS AND SIDE-CHANNEL ANALYSIS

Promotoren :
Prof. Dr. ir. B. PRENEEL
Prof. Dr. ir. J. VANDEWALLE

Proefschrift voorgedragen tot het behalen van het doctoraat in de toegepaste wetenschappen
door
Johan Borst

September 2001

D/2001/7515/24

ISBN 90-5682-314-0

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT DER TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK-ESAT
Kasteelpark Arenberg 10 – 3001 Leuven (Heverlee)

BLOCK CIPHERS: DESIGN, ANALYSIS AND SIDE-CHANNEL ANALYSIS

Proefschrift voorgedragen tot het behalen van het doctoraat in de toegepaste wetenschappen
door
Johan BORST

U.D.C. 621.391.7

September 2001

Foreword

All that we see or seem is but
a dream within a dream

In 1996, Johan visited our research group COSIC of the Katholieke Universiteit Leuven for a research project on the cryptanalysis of IDEA. After obtaining his Master's degree at the Technische Universiteit Eindhoven, he returned to COSIC in 1997 as a PhD student. Between 1997 and 2000 he did research in the area of block ciphers, resulting in several publications. He collaborated with L. Knudsen and V. Rijmen.

On October 19, 2000, Johan died in Breda (The Netherlands). He left us a PhD thesis that was almost complete. In order to honor his memory, we have decided to perform the final editing of his thesis, and we have requested the Faculty of Applied Sciences to award posthumously the degree of Doctor in Applied Sciences.

We will remember Johan as a smart researcher and an excellent chess player. He was a man of few words, but those who got to know him valued his sense of humor and his willingness to help other people.

We would like to thank the members of the reading committee Prof. H. van Tilborg and Prof. M. Van Barel for their efforts in reviewing the manuscript. We are also grateful to Prof. Y.D. Willems, for chairing the jury, and to Prof. J.-J. Quisquater and Prof. L.R. Knudsen for serving as jury members.

BP, JV

Abstract

This thesis analyzes the security of cryptographic primitives, and more in particular of block ciphers. A new model is introduced that takes into account side-channel attacks; these attacks exploit the characteristics of the implementation of a block cipher. Our research has also contributed to the selection process of the Advanced Encryption Standard (AES).

A first part of the thesis concentrates on cryptanalysis. A detailed analysis is made of Hellman's time-memory tradeoff and its optimization using the technique of distinguished points. Next an improved linear cryptanalysis is presented of the block ciphers RC5 and the AES candidate RC6. The analysis takes into account the fact that standard assumptions in linear cryptanalysis do not apply. A new tool is introduced, the non-uniformity function. Then key schedule weaknesses are demonstrated in the AES candidate CRYPTON.

In a second part techniques are studied that can improve resistance against side-channel analysis such as timing analysis, simple power analysis, and differential power analysis. Concrete countermeasures against these new attacks are proposed and analyzed.

The third part introduces a new conservative design strategy for a block cipher. This strategy takes into account from the beginning the resistance to side-channel attacks. The methodology consists of the insertion of several ciphers into a single structure while keeping an acceptable performance. The block cipher GRAND CRU is proposed as an example of such a design.

Notation

List of Abbreviations

3GPP	: 3rd Generation Partnership Project
ANSI	: American National Standards Institute
ASCII	: American Standard Code for Information Interchange
CBC	: Cipher Block Chaining
CFB	: Cipher FeedBack
CPU	: Central Processing Unit
DC	: Differential Cryptanalysis
DP	: Distinguished Point
DPA	: Differential Power Analysis
ECB	: Electronic Code Book
EEPROM	: Electrically Erasable Programmable ROM
EFF	: Electronic Frontier Foundation
EMV	: Eurocard Mastercard Visa
EP	: End Point
GSM	: Global System for Mobile Communications
IEC	: International Electrotechnical Commission
IETF	: Internet Engineering Task Force
IK	: Intermediate Key
ISO	: International Organisation for Standardisation
IV	: Initializing Value
LAN	: Local Area Network
LC	: Linear Cryptanalysis
LFSR	: Linear Feedback Shift Register
LSB	: Least Significant Bit
MAC	: Message Authentication Code
MK	: Master Key
MSB	: Most Significant Bit
NESSIE	: New European Schemes for Signature, Integrity and Encryption
NIST	: National Institute for Standards and Technology
NSA	: National Security Agency
OFB	: Output FeedBack

PGP	: Pretty Good Privacy
RAM	: Random Access Memory
RFC	: Request For Comments
ROM	: Read Only Memory
SIM	: Subscriber Identity Module
SK	: Session Key
S/MIME	: Secure MIME
SP	: Start Point
SPA	: Simple Power Analysis

List of Cryptographic Algorithms

3-DES	: triple-DES
3-Way	: block cipher [30]
A5/1	: GSM encryption algorithm [57]
AES	: Advanced Encryption Standard [2, 56]
BaseKing	: block cipher [30]
BKSQ	: block cipher [33]
Blowfish	: block cipher [123]
CAST-256	: AES candidate
COMP128	: GSM authentication algorithm (example) [57]
CRYPTON	: AES candidate
Davies-Meyer	: hash function based on block cipher
DEAL	: AES candidate
DES	: Data Encryption Standard [52]
DESX	: DES variant by Rivest [74]
DFC	: Decorrelated Fast Cipher, AES candidate
E2	: AES candidate
FEAL	: Fast Encryption Algorithm
GRAND CRU	: block cipher (Chapter 7)
IDEA	: Improved Data Encryption Algorithm [90]
KASUMI	: 3GPP encryption algorithm [46]
Mars	: AES finalist
Matyas-Meyer-Oseas	: hash function based on block cipher
MD4	: Message Digest 4 [113]
MD5	: Message Digest 5 [115]
MDC-2	: Manipulation Detection Code-2
MDC-4	: Manipulation Detection Code-4
Misty1, Misty2	: block ciphers [98]
PES	: Proposed Encryption Standard [89]
Rabin-Williams	: public-key encryption algorithm
RC2	: block cipher [83]
RC5	: block cipher [116, 117]
RC6	: AES finalist [120]

RIPEMD-160	: hash function [43]
Rijndael	: AES [34, 56]
RSA	: Rivest-Shamir-Adleman
SAFER	: Secure And Fast Encryption Routine [94, 95]
Serpent	: AES finalist
SHA	: Secure Hash Algorithm
SHA-1	: Secure Hash Algorithm-1 [54]
SHA-256	: Secure Hash Algorithm-256 [55]
SHA-384	: Secure Hash Algorithm-384 [55]
SHA-512	: Secure Hash Algorithm-512 [55]
Shark	: block cipher [111, 112]
Skipjack	: block cipher [8, 84]
SQUARE	: block cipher [31]
Twofish	: AES finalist

List of Mathematical Symbols

$D_K(C)$	: the decryption of the ciphertext C under the key K
$E_K(P)$	: the encryption of the plaintext P under the key K
$\mathrm{GF}(2^n)$	: Galois field with 2^n elements
K	: key
$\mathcal{K}$	: key space
k	: key length in bits
LSB_s	: s least significant bits
lg	: binary logarithm
MSB_s	: s most significant bits
trunc_s	: s leftmost bits
$\oplus$	: addition modulo two, exclusive or, xor
$\lceil x \rceil$	: smallest integer larger than or equal to x
$\lfloor x \rfloor$	: largest integer smaller than or equal to x
`0x0f`	: hexadecimal representation (e.g., the byte 00001111)
$\|\|$	: the concatenation of two bit strings (vectors)
$\#V$	: the cardinality of the set V

Contents

Chapter 1

Introduction

1.1 The Role of Security Analysis

In our current society an increasing amount of information is stored, transmitted or processed in a digital form. Amongst others this is caused by the increasing use of computers and local area networks, the growth of the Internet, the increasing popularity of mobile communications and the growing use of smart cards. However, digital information has different properties than its physical equivalents, which implies that to specify its role and functions in current society, several questions should be addressed and specific problems should be solved.

If not secured, digital information can be easily accessed by unauthorized persons, copied, changed, erased, etc. For example, messages sent over the Internet can be eavesdropped, files on a PC or laptop can usually be read by anyone who gains access to the machine, memory on smart cards has to be secured otherwise confidential and valuable information could be compromised.

Furthermore, digital information is not easily combined to ownership and it is not straightforward to make hard commitments based on transactions of digital information. For example, how do you know the name above an email belongs to the person who sent it to you purely based on the content of the email? How can one prevent that someone denies previous commitments if these were registered in a digital form?

The possibilities and success of ecommerce, the electronic 'variant' of commerce is partly based on answers to the above questions. The field of information security studies technical aspects of the above questions and provides solutions. Such a solution addresses specified security objectives and has a certain level of security. Many of these solutions make use of techniques that were traditionally used for encryption, i.e., keeping information secret for unauthorized persons by using a different representation (sometimes referred to as secret code).

There are many solutions that address or claim to address security objectives and many more are being developed. This situation is caused by several factors.

Firstly, for few solutions a mathematical or logical proof exists that the security objective is addressed, hence they are not ‘provable secure’. Besides that, most provable secure solutions are not practically usable due to implementation restrictions such as high memory requirements or low performance. Furthermore, solutions can be patented and hence (too) costly to use. Finally, the suitability of a solution partly depends on the specific application and platform.

Hence, we have a situation where many solutions are developed, but for which security claims are made on an *ad hoc* basis. It is important that the security level of a solution gets well analyzed, preferably in an early stadium of the design. This reduces the number of design flaws and all costly consequences of such as flaws and it increases the trust in a solution.

1.2 This Thesis

In this thesis we analyze building blocks or cryptographic primitives for the above mentioned solutions. In particular we perform security analysis on cryptographic algorithms including aspects concerning their implementation and function in an architecture. Part of the analysis is determining the level of security and pointing out design criteria that can be used to improve the level of security, either in concrete proposals or for generic designs.

The kind of primitive that will get most attention is the block cipher, but part of the results are extendable to other cryptographic primitives. Block ciphers can be used in several modes for encryption, but also as building block for other cryptographic primitives to provide different security services.

The aim of this analysis is to suggest improvements for or to build up trust in cryptographic primitives that are used in practice or that are proposed to be used in practice. The specific primitives and designs that are analyzed in this thesis all have in common that they are widely used in commercial products or standards, or that they are candidates for standardization.

Due to the rapidly evolving fields of information security and cryptography and the fact that this thesis was meant to deal with current practical issues, a more specified ultimate goal of the research has not been set. Instead two actual processes have influenced the direction of this thesis.

The first one is the AES standardization process to determine the new block cipher standard for use by the U.S. federal government. It started in 1997 and in 2000 the algorithm that was selected to be standardized, Rijndael, was announced. DES, the predecessor of the AES as American federal standard, was incorporated in several banking standards and has been used worldwide for almost all financial transactions and other applications for over 20 years.

A second development was the introduction of techniques that exploit characteristics of the implementation of an algorithm to break it. These are so-called side-channel attacks. Especially power attacks, which exploit differences in the power consumption of a cryptographic implementation, are particularly applicable to smart cards. This has led to a boost of research and development of countermeasures.

This thesis contains security analysis results concerning both processes, to which we have contributed. From these results as well as for other developments in these processes, design criteria and specific designs have been derived.

1.3 Outline and Main Contributions

The remainder of this thesis is organized as follows.

In Chapter 2 we introduce the field of information security. We determine its objectives, i.e., a formal explanation on security requirements, and classify methods to achieve these objectives. We also specify the role of cryptology in achieving security objectives and give an overview of the primitives relevant to the work in our thesis. Furthermore, we give a classification of evaluation criteria, security evaluation methods and introduce concepts to determine the level of security. A contribution of this chapter is that it integrates side channel analysis in the classical framework. We also provide an overview of current standardization processes and practical applications emphasizing on the primitives: block ciphers, one-way functions and hash functions.

In Chapter 3 we describe a generic or brute-force attack, which is applicable to block ciphers, one-way functions and stream ciphers. The method is based on a previous method introduced by Hellman. Building on an idea of Rivest we show that the method presented has significant practical advantages compared to Hellman's method. We present an analysis of the complexity and success probability of the algorithm, which depends on generic parameters of the underlying primitive. These results make it practically possible to choose the optimal parameters to mount an attack, based on available resources and specific requirements. This is the main contribution of this chapter. This chapter has evolved from joint work with Preneel and Vandewalle. Parts of it have been published in [22].

Chapter 4 presents results of a security analysis of the block cipher RC5 and the consequences for the former AES candidate RC6. We present an evaluation of the resistance of RC5 against linear cryptanalysis, taking into account the restrictions of standard methods caused by the fact that standard assumptions do not hold for the specific cipher. The main contribution of this chapter is the description of an attack that breaks reduced round versions of RC5. Additional contributions are a mathematical analysis of the complexity and success probability of the attack and the introduction of a new tool that is used in the attack algorithm. We also discuss the consequences for RC6, which has a better resistance against this attack. Part of this chapter has been published as a joint work with Preneel and Vandewalle in [23].

Chapter 5 presents weaknesses in the block cipher CRYPTON, that was an AES candidate. These weaknesses are caused by design flaws in the key schedule in combination with the structure of the algorithm. We also present how to exploit these weaknesses, when CRYPTON is used in a popular hash function construction. This is the main contribution of this chapter. Since CRYPTON was designed with similar security design criteria as the for AES selected algorithm Rijndael, this indicates that a good design strategy does not offer a guarantee for a secure design. Part of the results of this chapter have been sent as a comment to NIST [18] as input for the AES selection process. CRYPTON did not get selected for the second evaluation round of AES.

In Chapter 6 power attacks are studied. These attacks exploit differences in the power consumption of the implementation of a cryptographic algorithm. We give an extensive overview of the different kinds of power attacks and make a classification of possible countermeasures, which is a first contribution of this chapter. The remainder of the chapter studies the applicability of increasing resistance against power analysis by taking countermeasures at the protocol level. This is applied to a typical smart card application that uses triple-DES. These results are a second contribution of this chapter and have been developed in co-operation with Rijmen and Preneel. Parts of this chapter have been published in [21, 24].

Chapter 7 presents a new design strategy for block ciphers based on the new concept of multiple layered security. It also presents a concrete block cipher proposal based on this strategy and on the security and elegance of Rijndael. These are the two main contributions of this chapter. The proposed design strategy is conservative, in the sense that its aim is to base its security on several different and complementary security motivations. This is achieved by inserting several ciphers in a structure while keeping an acceptable performance. This design approach can be used to increase security against classical as well as against side-channel attacks. The block cipher proposal is submitted to the NESSIE project [104] under the name GRAND CRU.

We conclude in Chapter 8.

Chapter 2

Information Security and Cryptology

2.1 Introduction

Achieving information security for a system involves shifting the trust from untrusted components to trustworthy components. This can efficiently be achieved with cryptographic techniques. The design and evaluation of cryptographic techniques form an involved and interactive process.

In this chapter we introduce the field of cryptology, state its connection with information security and give all relevant definitions and applications, considering theory and practice. Furthermore we describe different areas and techniques in cryptology to situate our research area: security analysis and design of cryptosystems based on block ciphers. We show its relevance and point out the areas in which this thesis contributes.

For a more elaborate overview of cryptology we refer to [101, 131].

The remainder of this chapter is organized as follows. In Sect. 2.2 we describe the objectives of information security and the function of cryptology in achieving them. The tools of cryptology, primitives and evaluation criteria, are described in Sect. 2.3. In Sect. 2.4 we expand on the most important evaluation criterion: security. We give an overview, description, evaluation criteria and proposals of the classes of primitives, block ciphers, one-way functions and hash functions in Sect. 2.5 and 2.6. We conclude and summarize the contributions of this chapter in Sect. 2.7.

2.2 Objectives

Information was traditionally stored and transmitted using physical media, such as paper. In the past few decades this has drastically been changed due to the introduction and development of electronic storage media and (wireless) communication networks such as the Internet. Amongst others this has led to the development of mobile communications (GSM, UMTS) and electronic commerce. Compared to paper based information this makes it an easy target for copying, concealment, loss, fraud and other (possibly unintentional) abuse.

Information security[1] has many possible objectives. The most well-known objective is *confidentiality*, but besides that there are many other objectives, such as signatures, authorization, validation, ownership, voting, anonymity, witnessing, non-repudiation, time of occurrence, registration, (non)approval, etc.

Cryptology. Many security objectives can be achieved by technical means, possibly in combination with other means, such as legal, computer and network architectural means. The technical methods to secure information in electronic form are comprised in cryptology, which is defined as follows.

Definition 2.1 Cryptology *is the study of mathematical and logical aspects of (advanced) techniques that do not use physical characteristics of the information carrier or channel to achieve objectives of information security.*

Definition 2.2 *The set of mathematical and logical aspects, that do not use physical characteristics of the information carrier or channel, of a solution to achieve a security objective is called a* cryptosystem.

Cryptology is subdivided into two areas: *cryptography* and *cryptanalysis.* They are defined as follows.

Definition 2.3 Cryptography *is the science of designing cryptosystems.*

Definition 2.4 Cryptanalysis *is the science of the security analysis of cryptosystems.*

There is a strong interaction between the two areas: during the design of a system all relevant security analyses should be taken into account, designs are the subject of cryptanalysis, and cryptanalysis of a design can lead to new design guidelines. Often the term cryptography is used in the meaning of Definition 2.1.

According to [101] all objectives that can be met by cryptology can be derived from a framework based on the following four concepts:

- **Confidentiality**. This is a service to keep information secret from all but those who are authorized to see it.
- **Data integrity** is a service that addresses the unauthorized alteration of information. It provides mechanisms to detect unauthorized alteration, such as deletion, insertion and substitution.
- **Authentication**, that can be subdivided into
  - **Entity authentication** provides methods by which communicating parties can identify each other.

[1]Simmons uses the term *information integrity* in [131].

  - **Data origin authentication** provides methods by which data can be linked to the origin of the data.

- **Non-repudiation** is a service that prevents an entity from denying previous commitments or actions.

This classification is not strict: the objective of data integrity is implicitly met if data origin authentication is met.

The parties that are involved in achieving a security objective are also called *entities.*

2.3 Primitives

In order to achieve the objectives in the previous section one can distinguish between a number of *cryptographic tools* or *security primitives.* These primitives can be used as building blocks for other primitives, (cryptographic) protocols or cryptosystems. Cryptographic primitives and cryptosystems are also often referred to as cryptographic algorithms or designs. A *cryptographic protocol* is a distributed algorithm that specifies a sequence of actions required of two or more entities to achieve a security objective [101].

2.3.1 Encryption

An example of a cryptographic primitive is a *cipher.*

Definition 2.5 *Let $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{K}$ denote sets that we call plaintext space, ciphertext space and key space, resp. A* cipher *(or* encryption system*) is defined by*

- *a set $\{E_K : \mathcal{P} \to \mathcal{C} | K \in \mathcal{K}\}$ of* encryption *functions and*
- *a set $\{D_K : \mathcal{C} \to \mathcal{P} | K \in \mathcal{K}\}$ of* decryption *functions*

with the property that for each $K_1 \in \mathcal{K}$ there is a key $K_2 \in \mathcal{K}$ such that

$$D_{K_2}(E_{K_1}(P)) = P \text{ for all } P \in \mathcal{P}. \tag{2.1}$$

Such a corresponding pair (K_1, K_2) is called a key pair.

A cipher can be used to achieve confidentiality as follows. Suppose a sender A wants to send a message P to receiver B over an *insecure channel,* i.e. information sent over an insecure channel can be read by unauthorized parties. Assume they have a cipher according to Definition 2.5. Suppose A and B share the key pair (K_1, K_2); for example they have agreed on it via a secure channel. Then A can compute $C = E_{K_1}(P)$ (this computation is called *encryption*) and send it to B. Upon receipt of C, B can compute $P = D_{K_2}(C)$ (which is referred to as *decryption*). P is referred to as plaintext and C as ciphertext. This is visualized in Figure 2.1. Note that C can be read by unauthorized parties.

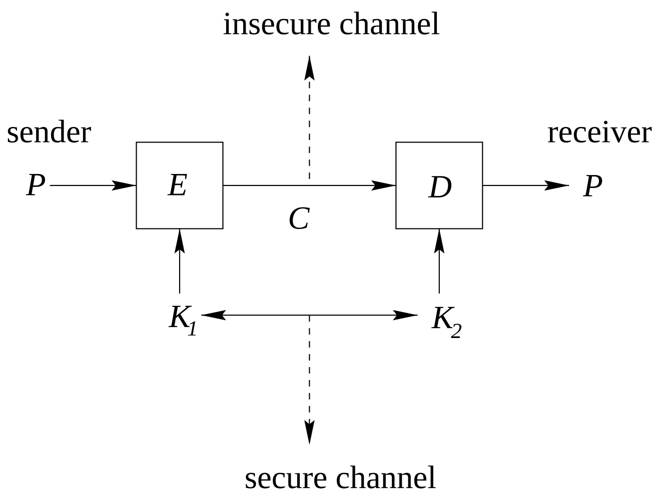

Figure 2.1: Using a cipher to provide confidentiality.

Confidentiality is achieved if no information about K_2 is known to any unauthorized entity and it is infeasible to obtain any previously unknown information about P from C without knowledge of K_2, including the information about K_2 that can directly be derived from K_1. Often the total of encryption, sending ciphertexts and decryption is referred to as the *encryption process.*

The technique of encryption can be seen as a basis for solving many other security objectives. The field of cryptology often is referred to as encryption technology.

2.3.2 Secret-key and Public-key Cryptography

Ciphers for which it is feasible to deduce K_2 from K_1 are called *symmetric* or *secret-key* ciphers. If it is infeasible they are called *asymmetric* or *public-key* ciphers and K_1 is called a public key and K_2 a private key. In that case there is no need for Alice and Bob to share the key pair (K_1, K_2): it is sufficient that Bob makes K_1 public and keeps the corresponding key K_2 private. Public-key cryptography was introduced by Diffie and Hellman in [38]. It makes the following possible. Many entities can encrypt messages that can only be decrypted by one entity and one entity can encrypt messages that many people can decrypt. The first capacity has its merits for key management. If A wants to communicate to B using a public key cipher, it is sufficient that A knows the public key of B. The system is secure as long as the private keys remain secret. Hence, contrary to secret-key ciphers the keys do not need to be established via a secure channel. Furthermore, not every pair of entities who want to communicate need to share a (secret-)key pair: one public-key pair for each entity is sufficient. The second capacity can be used to produce digital signatures in an efficient way. With public-key techniques it is possible to create digital signatures that are verifiable if one knows the public key of the signer. However, to make a public signature knowledge of the private key is required. Public-key cryptography has a major disadvantage: it is considerably slower than symmetric-key cryptography. For this reason for encryption of large amounts of data where efficiency is an important factor (*bulk encryption*) symmetric-key ciphers will be preferred. Many systems make use of both techniques: they use public-key cryptography for key establishment of the secret symmetric keys and secret-key cryptography for secure and efficient communication.

Cryptographic primitives can be classified as follows.

- **Symmetric-key (or secret-key) primitives**: For these primitives the security relies on the secrecy of all key material to unauthorized parties. Examples are: block ciphers, stream ciphers, MACs, pseudo-random string generators, etc.
- **Asymmetric-key (or public-key) primitives**: These primitives use a key pair of a private and a public key. The security relies on the secrecy of the private key to unauthorized parties. Examples are: public-key ciphers, digital signature schemes, key exchange protocols, etc. The best known asymmetric cryptographic primitive is RSA, named after its inventors Rivest, Shamir and Adleman. Currently, RSA Security Inc. is also the name of a world leading security company.
- **Unkeyed primitives**: These primitives do not use keys. Examples are: cryptographic hash-functions, one-way functions, secret sharing schemes, etc.

In this thesis we shall study symmetric key primitives, in particular block ciphers, and their applications.

2.3.3 Evaluation Criteria

Al primitives that are applied in a cryptosystem need to conform to certain criteria; in [101, p. 5–6] the following criteria are listed.

1. **Level of security**: It depends on the objective how well this can be quantified. Methods to express it are often given by computation power needed to defeat the objective, statements that defeating the objective is at least as difficult as a difficult problem or defeating the objective of another primitive, and the amount and quality of independent security evaluations.
2. **Functionality**: Some primitives meet several security objectives to a certain extent and to a certain level of security. Which primitive can be chosen to meet a certain security objective in combination with other primitives should be evaluated.
3. **Methods of operation**: Mostly a primitive has a basic mode, but primitives can often be used in different *modes of operation*, i.e. restricting

the output, combining input and output, etc. Different modes can even be used to obtain different objectives. It should be evaluated if a primitive can be used in a certain mode and how well it meets the intended (security) objectives.

4. **Performance**: This is the speed (an implementation of) a primitive can achieve, for example expressed in output bits per second or CPU clock cycles per operation. The performance is platform dependent. It is possible to optimize an implementation for performance on specific platforms.

5. **Ease of implementation**: Here one evaluates how well a primitive can be implemented given certain resources. One can for example have a situation where the code should fit in restricted memory or a hardware environment with a restricted number of gates. Often such restrictions also limit the performance of a primitive, since the optimal implementation would exceed the available resources.

Of course evaluation criteria should form the spine of a design process.

The level of security seems to be the most difficult to evaluate [101]. The design of a cryptosystem involves a combined evaluation on all criteria, but since the main goal is to meet a security objective, the design process is guided by the security evaluation. In this thesis we will meet aspects of all criteria, with an emphasis on security evaluation.

2.3.4 Standards

For many designs of cryptosystems it is necessary or preferable to develop new primitives and/or modes of use. Other cryptosystems benefit from known primitives that are well evaluated, known to be cryptographically sound and trusted. Besides that it is sound practice to (pro-actively) design a cryptosystem such that it is efficiently interoperable with other cryptosystems. These two requirements are supported by standards.

International and national organizations that have standardized cryptographic and information security techniques that are globally applied are:

- International Organisation for Standardisation (ISO).
- International Electrotechnical Commission (IEC).
- American National Standards Institute (ANSI).
- National Institute of Standards and Technology (NIST).
- Internet Engineering Task Force (IETF).

Many standards have been developed jointly by ISO and IEC. Banking security standards can be divided into *wholesale banking*, involving transactions between financial institutions and *retail banking* between institutions and private individuals. Such standards are developed by ISO or by its national members such as ANSI. NIST specifies Federal Information Processing Standards (FIPS), that provide guidelines for the U.S. federal government departments. The IETF makes recommendations for standards for use by the Internet community. Their official working notes are called Requests for Comments (RFCs). The standardization processes contain overlapping objectives and hence influence each other in a technical sense. For an overview and details of standards see [101].

Besides dedicated standardization organizations, there are also partnerships or projects founded by several organizations or companies in order to develop standards. Such standardization processes that include cryptographic and information security techniques are:

- Europay-MasterCard-Visa (EMV) [45]: A joint industry working group (formed by Europay, MasterCard and Visa) created to facilitate the introduction of smart card technology into the international payment systems environment by developing joint specifications for Integrated Circuit Cards (ICC) and terminals for payment systems. The EMV specifications serve as the global framework for chip card and terminal manufacturers worldwide.

- Third Generation Partnership Project (3GPP) [1]: A partnership of worldwide leading telecommunications companies to develop technical specifications for the third generation of mobile communication systems (the successors of GSM).

- New European Schemes for Signature, Integrity, and Encryption (NESSIE) [104]: A project within the Information Societies Technology (IST) programme of the European Commission. Participants are several European universities, in cooperation with an industry board. Goal of the project is to put forward a portfolio of strong cryptographic primitives for a number of different platforms.

Besides standards of cryptographic methods there also exist patents. On the one hand a patent (and corresponding evaluation) might induce trust in the soundness of a primitive or cryptosystem, but on the other hand it can be a reason not to choose a particular cryptographic solution to meet a security objective.

Hence, the existence of standards and patents can be seen as design or evaluation criteria.

In this thesis we shall present results that are of direct relevance to standardization processes of NIST, EMV and 3GPP. Furthermore we shall present a design that has been submitted to NESSIE.

2.4 Security Evaluation

As stated before the level of security is the most important evaluation criterion but also the one that is most difficult to quantify. The common model introduces an unauthorized entity, an *adversary* or *attacker*, and analyzes its possible actions, called *attacks*, and their consequences.

2.4.1 Approaches

There are different approaches to analyze or define security for primitives. We give the following categorization. All arguments also apply for the security of cryptosystems.

- **Information-theoretical approach**. Security in the information-theoretical sense is also called *unconditional security* and for encryption schemes *perfect secrecy*. Here an attacker is assumed to have unlimited computational resources. There are few cryptosystems resistant against an attack of such an opponent. The most well-known example is the *one-time pad*, proposed by Vernam in [137]. Encryption is performed by xoring an n-bit plaintext with an n-bit key. The best an attacker can do is to try each possible key to decrypt the ciphertext. As each key is assumed to be equally likely, no information can be gained about the plaintext. The disadvantage of this system is that for each message of n bits an n-bit key should be used. In general systems with unconditional security are not practically usable.

- **Complexity-theoretical approach**. Here an adversary is assumed to have polynomial limited computational power and memory resources, i.e. his resources are polynomial in the size of appropriate security parameters. Few practical methods have resulted from this approach.

- **Provable security**. In this approach one tries to link security to another difficult problem with statements as attacking a system is equivalent to or at least as difficult as solving that problem. We make a distinction between three approaches.

  - *Related to an attack scenario.* There exist different scenarios to attack a primitive or system. It is good practice to prove that an attack can not be successful in certain settings, especially if this attack was very successful for other primitives or systems of the same kind. Examples: Nyberg's and Knudsen's provable security against a differential attack [106], Vaudenay's decorrelation theory [136], the design strategy of Matsui for the block cipher Misty [98], the Wide Trail Strategy of Daemen as can be found in [30] and Chapter 5.
  - *Related to a difficult problem.* There exist mathematical problems that are not efficiently solvable. Examples are factoring the product of large primes and solving the discrete logarithm problem over certain large finite groups. For several cryptographic primitives it has been proven that breaking them would be equivalent to solving such a problem (for example, the Rabin-Williams public-key encryption algorithm [101]).
  - *Related to the security of another primitive.* This approach can often be used when constructing systems or primitives from other primitives. An example would be the design of triple-DES.

- **Computational and resource-based security** or **practical security**. Here an attacker is assumed to have limited computational power and limited resources. These assumptions are linked to practical considerations: which gain does an attacker have by breaking a system, how much can he afford to invest, does the duration of an attack need to be considered? The security analysis will then proceed according to the analysis of the best known and relevant attacks. Provable security can be seen as a special case of this. With few exceptions one can not get around this approach.

- **Ad hoc approaches** or **heuristic security**. This is the approach that uses any convincing arguments that do not fit in any of the above categories to make plausible that any attack will require more computational power or resources than available to an opponent. Although at first sight this might seem a suspicious approach, the trust in most systems and primitives that are currently used in practice is largely based on this concept, mostly in combination with security arguments of one of the above approaches. Shannon's concepts confusion and diffusion [127] can be seen as fitting in this approach. Another (non-mathematical) argument for security in this sense is linking the level of security (trust) of a primitive or system to the quality and quantity of independent security analysis it has received. Note that this assumes that the primitive has received attention from the cryptographic community and that it is suitable for analysis, i.e. has a clear structure. A drawback of this argument is that it will need some time before the primitive can be trusted to be secure.

- **Security through obscurity**. Nowadays, this is usually considered bad practice, with few exceptions. One can differentiate between two approaches.

  - *Keeping the design secret.* A motivation to do this would be that the secrecy would be an extra barrier for an attacker. However, there are two main reasons against this.

    Firstly, it is very difficult to keep a system secret, especially if it would be profitable to break it. Re-engineering and social engineering usually are sufficient. An example for this is the set of security algorithms used for the GSM, that were designed in secrecy but all specifications were reconstructed.

    Secondly, the important *ad hoc* security argument that trust in a cryptosystem's security depends on the independent analysis that it has received is difficult to support when the design is secret. An illustration of this is the GSM MAC-algorithm COMP128, that was easily broken after its specification was published [57]. Exception to this rule form organizations such as S.W.I.F.T. that can and do invest sufficiently in the security of their proprietary algorithms.

  - *Unmotivated complication of the design.* This has the following drawbacks. It makes a design more difficult to analyze and without analysis it is difficult to trust its security. Furthermore, complicated systems are often harder to implement efficiently. A secure design does not need any additional complication added. Besides that, complicated designs take more effort to implement and are in general less performant.

In practice security evaluation (and design) uses a combination of these different approaches. The best current practice seems to be: provable security arguments where and if possible, a relevant amount of computational security arguments and convincing heuristic arguments. The more weight is given to the heuristic arguments, the more time should be spent in independent analysis.

In practice a designer should make his design and corresponding claims concerning security (as well as other evaluation criteria, of course) available for analysis. If these claims are not refuted after considerable attention of independent experts (the cryptographic community), this will contribute to the trust in the validity of the claims.

In this thesis we shall present analysis results of provable, practical and *ad hoc* security of several primitives and systems. The analysis is based on mathematical and cryptographic techniques. The underlying assumptions shall be verified. Every analysis shall be compared with experimental results. We believe that this should be the approach for each security analysis.

Furthermore we shall present some designs of primitives and systems. The approach here is *conservative.* It shall include well studied and motivated arguments and principles from provable and practical security. Considering *ad hoc* security, only arguments will be used that are generally accepted and well known.

2.4.2 Security Analysis of Symmetric Ciphers

We give an overview of the possible settings and goals of an attack that have to be taken into considerations for the security analysis of symmetric ciphers.

Goals. We call the cipher under attack the *target cipher* and secret key that is used the *target key.* The purpose of an attack is to deduce information about plaintexts or target key that was not prior knowledge to the attacker. A first classification can be made, based on the achievement of an attack [76].

- **Key recovery:** The attack deduces the value of the key.
- **Global deduction:** Here the attacker derives an equivalent function of the encryption or decryption function, without deducing the key value.
- **Local deduction:** The attacker gains the ability to encrypt/decrypt part of the plaintext/ciphertext space.
- **Information deduction:** Here an attacker acquires some information about plaintext, ciphertext or key, that he did not know prior to the attack, not covered by the above possibilities.

Basic assumptions. Another classification can be made, based on the knowledge that an attacker can acquire about the encryption process. Since data is transmitted over an insecure channel, we assume the following:

Assumption 2.1 *An attacker has always knowledge of the ciphertexts.*

A classical assumption that is generally made is Kerckhoffs' assumption [101].

Assumption 2.2 [**Kerckhoffs**] *An attacker knows all details about the encryption (and decryption) function except for the value of the secret key.*

However, with side-channel analysis in mind we reformulate this assumption slightly to keep it more into accordance with current practice.

Assumption 2.3 [**Modified Kerckhoffs**] *An attacker knows all details about the mathematical method that is used to derive ciphertext from plaintext (and vice versa) except for the secret key.*

Note that Assumption 2.3 excludes an attacker to have knowledge about the implementation of the encryption function which was not anticipated in Assumption 2.2.

In the remainder of this thesis we make Assumptions 2.1 and 2.3. We further divide attacks into:

- **Classical attacks** are solely based on knowledge of plaintexts and ciphertexts (*information access*).
- **Side channel attacks** exploit characteristics of the implementation of an algorithm. These characteristics are called *side channels* and include power consumption and computation time.
- **Fault based attack** exploit (induced) faults in cryptographic computations.

Note that the last two items are in accordance with Definition 2.1: they exploit physical characteristics of the device that performs the cryptographic computations, they do not exploit physical characteristics of the information carrier or channel. In this thesis we shall present results for the first two items.

Information access. A classification of attacks based on the access and influence an attacker has to the input and output of the encryption process can be made as follows:

1. **Ciphertext only attack:** It is assumed that an attacker has only access to the ciphertexts and has some information about the plaintexts. This information consists for example of the knowledge that a sequence of plaintexts consists of an English or Dutch text, coded in ASCII, but not of specific or partly known values of plaintexts. Hence, the attacker has knowledge about the probability distribution of plaintext values.

2. **Known plaintext attack:** Here, the attacker has access to a number of plaintexts with corresponding ciphertexts.
3. **Chosen plaintext attack:** The attacker can choose a number of plaintexts with corresponding ciphertexts. A variation is an *adaptive chosen plaintext attack*, where the choice of plaintexts depends on previously obtained plaintext-ciphertext pairs.
4. **Chosen ciphertext attack:** The attacker has the ability to acquire corresponding plaintexts for ciphertexts of his choice. A variation is an *adaptive chosen ciphertext attack*.

The success of an attack typically depends on the number of texts that an attacker can access. Usually the probability to mount a successful attack will rise considering the scenario's from 1 to 4. For example, a chosen plaintext attack can be seen as a known plaintext attack with a lower success probability.

An alternative class of attacks are **related key attacks**. Here, an attacker has access to an encryption process performed with different keys, whose values are related. The significance of these kind of attacks depends on the application in which the cipher is used or on the key management.

Side channels. A side channel of a cipher is a characteristic of a cryptographic implementation that can be measured during a cryptographic computation by a certain device and that can be exploited. Side channels can typically only be measured in the proximity of the cryptographic device. Devices that are particularly suited for mounting side channel attacks are smart cards.

Security evaluation of a primitive or system against side channel attacks can not be done solely based on the mathematical specifications. Ideally an attack should be tested on an actual implementation. Note that to test an attack on an implementation it is not necessary to know how the target cryptographic primitive is implemented, i.e., Assumption 2.3 does not necessarily need to be extended for this attack model. Of course, detailed knowledge of the implementation would be more convenient to interpret the test results and to indicate changes to improve resistance of the implementation.

The most important side channel attacks are the following:

- **Timing attacks** were introduced in the open literature by Kocher in [85]. These attacks exploit the fact that the execution time of some implementations of cryptographic functions depends on the key material. Most of these weaknesses are caused by the fact that the implementation was optimized for speed.
- **Power attacks** are described by Kocher *et al.* in [86]; the idea is to extract the key of a cryptographic device by analyzing its power consumption during cryptographic computations.

In this thesis we will study power attacks and possible countermeasures; we will also present a cipher design strategy that results in designs with a high resistance against side channel attacks.

Complexity. To express the level of security one needs a means to measure it. This can be done by measuring the number of (chosen) ciphertexts and plaintexts, the computation time and the memory capacity that is required to achieve an attack goal with a certain probability. In particular we specify the following complexities.

- **Number of required texts** or **encryption/decryption samples**, if relevant divided into chosen/known ciphertexts/plaintexts. A suitable unit (bits, bytes, blocks) should be used. For side channel attacks this unit could be the amount of cryptographic computations for which a side channel measurement has been performed (samples).
- **Computation complexity**, subdivided into:
  - **Precomputation complexity**: the computation time of computations that can be performed before the acquired texts are processed.
  - **Processing complexity**: the computation time required to process the acquired texts.

  Typically this complexity is measured in the number of evaluations of the target cipher. This has as advantage that a complexity comparison to other attacks is easier; especially a comparison to brute force attacks since these are most naturally expressed in this way.
- **Memory complexity** or **storage complexity**: the amount of memory that is required for the attack. As for other complexities, a suitable unit could be specified, e.g., one block.
- **Complexity caused by the requirements of other resources**: an example is the amount of required memory accesses.
- **Success probability**: the probability that the goal of the attack is achieved.

It is also possible to include the specific goal into this list. We prefer not to do that since usually the achievement of any goal is considered a weakness of the primitive or cryptosystem.

It is not always necessary to take into account all items in the complexity analysis of an attack. Considering the practical application and the specific properties of the attack the relevant complexities have to be set. This deduction of relevant properties can even lead to considerable changes to improve an attack algorithm.

Methods. Contrary to the goals of an attack that can be clearly defined, it is impossible to present a complete and sufficiently specified overview of methods to achieve such a goal. Finding new methods to attack or new extensions of existing attacks is ongoing research. In this thesis we shall contribute to this area. Attacks on primitives can be divided into two categories:

- **Brute force attacks** or **generic attacks**. These are attacks that can be applied to any cipher and whose complexity and success probability only depend on the values of certain parameters of the cipher (such as key length, block size or internal memory).
- **Shortcut attacks**. These are attacks that exploit the internal structure of a specific cipher.

The results of brute force attacks indicate which values of parameters can be used in practice for which applications. The (lack of) results of shortcut attacks indicate the amount of trust a cipher deserves. In this thesis we shall present results in both categories.

A cipher is called *broken* if there exists an attack, such that the cipher does not meet the intended security objectives. We define the following gradations of how a cipher can be broken.

Definition 2.6 *A cipher is* theoretically broken *if there exists a shortcut attack that achieves better complexity than the best brute force attack.*

Attacks that break a cipher theoretically are also called *academic attacks*. If the complexity of the brute force attack is not reduced by a significant amount, then usually these kind of attacks do not have any consequences for the use of the cipher in practice. However, such results are valuable to get insight in the way the cipher provides security and hence to derive design and analysis criteria. Besides that they may lead to other attacks that decrease the complexities and eventually lead to a break in the sense of the following definitions. The process of making improvements to existing attacks on a cipher also may form a motivation for the amount of trust in a cipher and forms an illustration of the attention and of the amount of analysis that the cipher has received.

Definition 2.7 *A cipher is* broken concerning the claims of the designer *if there exists an attack with lower complexity than is claimed by the designer at the introduction of the cipher.*

This security criterion is amongst others used as a security evaluation criterion for NESSIE submissions. The NESSIE project has stated that a submission that gets broken in this sense will probably be disqualified.

Definition 2.8 *A cipher is* broken with respect to an application *if there exists an attack with lower complexity than the required minimum complexity for this application.*

Of course a cipher that gets broken in the sense of Definition 2.8 should not be used in the relevant application. It also seems sensible to prefer an alternative cipher for other applications.

Definition 2.9 *A cipher is* (practically) broken *if there exists an attack with lower complexity than the required minimum complexity for a typical application.*

In our opinion a cipher that is broken in the sense of Definition 2.9 should not be used in practice at all, that is, it should not even be used in an application for which it is not broken, provided that there is an alternative solution.

2.5 Block Ciphers

The class of symmetric ciphers can be divided into *block ciphers* and *stream ciphers.* Concerning encryption the distinction between the two is that stream ciphers encrypt *characters* with a transformation that changes in time and block ciphers encrypt *blocks* with a fixed transformation. A block can be seen as a sequence of characters; hence the set of possible blocks is considerably larger than the set of possible characters. Characters are typically bits or bytes, while a typical block typically consists of a sequence of 64 or 128 bits.

Encryption with a block cipher can be performed according to different modes. Furthermore, block ciphers can be used both in the construction of other symmetric-key primitives such as other block ciphers, stream ciphers, MACs, and unkeyed primitives such as hash functions and pseudo-random string generators.

2.5.1 Definitions and Properties

A block cipher is a symmetric cipher (according to Definition 2.5), where the plaintext $\mathcal{P}$ and ciphertext space $\mathcal{C}$ contain the same finite number of elements. Usually $\mathcal{P} = \mathcal{C}$. Typically the size of the set is a power of two. In that case, there exists an integer n such that any text can be uniquely expressed as an element of $\{0,1\}^n$. An n-bit block cipher is a symmetric cipher with $\mathcal{P} = \mathcal{C} = \{0,1\}^n$:

Definition 2.10 *Let $\mathcal{P} = \mathcal{C} = \{0,1\}^n$ be the plaintext and ciphertext space and $\mathcal{K} = \{0,1\}^k$ be the key space. Define a function $E : \mathcal{P} \times \mathcal{K} \to \mathcal{C}$. Then E is an n-bit block cipher if for each $K \in \mathcal{K}$, $E(.,K) = E_K(.)$ is invertible. The inverse of E is denoted with $D(.,K) = D_K(.)$. n is called the* block length *and k is called the* key length.

Usually the term block cipher is used for an n-bit block cipher. The elements of $\mathcal{P}$ and $\mathcal{C}$ are called *plaintext* and *ciphertext blocks*, resp., in the remainder also often referred to as plaintexts and ciphertexts, resp.

NIST has standardized four modes to use a block cipher for encryption in [53]. We assume that $P_1, P_2, \ldots$ is a stream of plaintext blocks of length m that are to be encrypted to a stream of ciphertext blocks $C_1, C_2, \ldots$.

- *Electronic Code Book (ECB):* each plaintext block of length $m = n$ is encrypted independently of the others.

$$\text{Encryption: } C_i = E_K(P_i)\,.$$

$$\text{Decryption: } P_i = D_K(C_i)\,.$$

- *Cipher Block Chaining (CBC):* each plaintext block of length $m = n$ is xored with the previous ciphertext block before encryption with the block cipher.

  $$\text{Encryption: } C_i = E_K(P_i \oplus C_{i-1})\,.$$

  $$\text{Decryption: } P_i = D_K(C_i) \oplus C_{i-1}\,.$$

  C_0 has a chosen initial value. This value also is referred to as initial vector or IV.

- *Cipher Feedback (CFB):* plaintext blocks of length $m \in \{1, \dots, n\}$ are encrypted as follows.

  $$\begin{aligned} \text{Encryption: } C_i &= P_i \oplus \text{MSB}_m(E_K(X_i)) \\ X_{i+1} &= \text{LSB}_{n-m}(X_i)||C_i\,. \end{aligned}$$

  $$\begin{aligned} \text{Decryption: } P_i &= C_i \oplus \text{MSB}_m(E_K(X_i)) \\ X_{i+1} &= \text{LSB}_{n-m}(X_i)||C_i\,. \end{aligned}$$

  Here X_1 has a chosen initial value, $||$ represents concatenation of strings and MSB_s and LSB_s refer to the s most significant or s least significant bits, respectively.

- *Output Feedback (OFB):* plaintext blocks of length n are encrypted as follows.

  $$\begin{aligned} \text{Encryption: } C_i &= P_i \oplus E_K(X_i) \\ X_{i+1} &= E_K(X_i)\,. \end{aligned}$$

  $$\begin{aligned} \text{Decryption: } P_i &= C_i \oplus E_K(X_i) \\ X_{i+1} &= E_K(X_i)\,. \end{aligned}$$

  X_1 has a chosen initial value. This mode is also defined for $m < n$.

A block cipher has one principal design criterion, concerning its security properties. For this we introduce the concept *random permutation models.*

Definition 2.11 *[101] Let $\mathcal{F}_n$ denote the set of all permutations from a finite domain of size n to a finite co-domain of size n. Models where random elements of $\mathcal{F}_n$ are considered are called* random permutation models.

Properties of random permutations are described in [51, 101]. The main design criterion of a block cipher is the following

Property 2.1 *A block cipher should be (practically) indistinguishable from a random permutation for every key.*

Hence, a block cipher can be modeled as a random permutation. Every successful attack on a block cipher is in fact a method to distinguish it from a random permutation. Note that not every method to distinguish a block cipher from a random permutation can be directly used to mount a key-recovery attack.

For many block ciphers the encryption algorithm E and the decryption algorithm D have a similar structure; this simplifies implementations. Furthermore, for efficient implementations most block ciphers have an iterated structure. A block cipher can transform a plaintext into a ciphertext by iterating a *round transformation* (or *round function*) r times. The round transformation is a (mostly insecure) block cipher, and hence takes a key as input in each iteration. These keys are called *round keys* and are derived from the original key (*user key*). Let E denote the block cipher, F the round function and $K_1, \ldots, K_r$ the round keys. Then a plaintext P is encrypted by

$$E_K(P) = F_{K_r}(F_{K_{r-1}}(\ldots(F_{K_1}(P)))) \,. \tag{2.2}$$

This is visualized in Figure 2.2. In some block ciphers the round function F can be different in different rounds. Furthermore many block ciphers have alternative initial or final transformations. All block ciphers that have such structures are called *iterated block ciphers.*

Usually the resistance of a block cipher against shortcut attacks increases if the number of rounds increases. On the other hand the performance of the cipher decreases with the addition of every round. An important part of the design for efficient iterated block ciphers is to set the appropriate number of rounds: too few is insecure, too many is inefficient. For this purpose reduced round versions are analyzed and tested for the randomness properties. The number of rounds is usually set to resist the best estimated attack with some extra rounds in order to provide some margin against new developments (or inaccurate analysis).

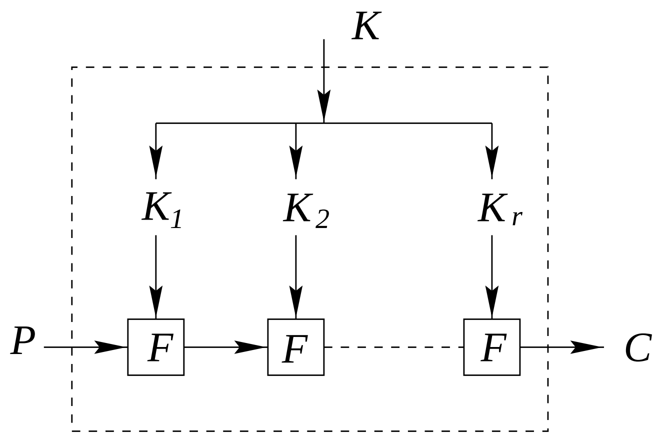

Figure 2.2: An iterated block cipher.

2.5.2 Brute Force Attacks

The generic parameters of a block cipher are its key length k and its block length n. The following attacks are always feasible for a block cipher.

- **Exhaustive key search:** This is a key recovery and known plaintext attack. An attacker acquires a plaintext/ciphertext pair and encrypts the plaintext with all possible keys until he finds the corresponding ciphertext. With high probability this key is the target key. The time needed for a successful key search depends on the key length and the available technology.
  - *Distributed software searches.* The use of general purpose machines for key search is particularly useful in distributed efforts. For comparable small key sizes a medium sized LAN is often sufficient. Another approach is to run key search clients that use idle CPU cycles on machines around the world that are connected via the internet.
  - *Dedicated hardware.* Several papers have been written studying the possibility to build a dedicated key search machine. The efficiency of the machine depends on the financial investment. A standard reference is the design of Wiener [139]. In 1998 the Electronic Frontier Foundation (EFF) built a machine to crack DES by exhaustive key search [44]. This machine was used to solve the RSA DES II Challenge Contest, a 56-bit key search, in less than 3 days. Total cost for design and equipment was less than $250,000.

  From 1997 RSA Laboratories has organized its secret-key challenge [122]. The challenge consists of several contests. Each contest provides a known plaintext, encrypted by a block cipher, and a ciphertext for which the plaintext is unknown. The goal is to find the unknown plaintext. In twelve of the contests the cipher is 12-round RC5, with a variable key length: 40, 48, 56, ... , 128 bits. Four of the contests used DES (56-bit key) for encryption. The contests solved in October 2000 are summarized in Table 2.1. It provides a quantification of the security offered by the key size of a symmetric cipher.

  A rough method to estimate the key length that is needed for sufficient security in the future is the cost variant of *Moore's law*, which states that the computing power for a given cost doubles every 18 months. An estimate of the security of symmetric key sizes depending on the resources of an attacker has been made in 1996 by Blaze *et al.* in [16]. Their conclusion was that for the next 20 years (starting from 1996) a key length of 90 bits would provide sufficient security against exhaustive key search attacks of any practical attacker.

  Based on these results we conclude that a key size of 128 bits provides sufficient security against exhaustive key search.

Table 2.1: Results of the RSA secret-key challenge (October 2000).

Contest	Time to solve	Percentage of key space	Method
RC5 40-bit	3.5 hrs		LAN
RC5 48-bit	313 hrs		Distributed over internet
RC5 56-bit	265 days		Distributed over internet
RC5 $\geq$ 64-bit	ongoing		
DES I	140 days	25%	Distributed over internet
DES II-1	40 days	87%	Distributed over internet
DES II-2	3 days		EFF DES cracker
DES III	22 hrs		EFF DES cracker & distributed over internet

- **Table precomputation:** This is a key recovery and chosen plaintext attack. An attacker takes a plaintext of his choice, encrypts it under all possible keys and stores the ciphertexts linked to the key values. This is the precomputation of the attack. Next the attacker acquires the encryption of his chosen plaintext with the target key. He then retrieves this ciphertext amongst his stored ciphertexts and hence retrieves the target key (or at least a few possibilities for the target key).

- **Hellman's time-memory tradeoff:** The aim of a time-memory tradeoff is to mount an attack which has a lower online processing complexity than exhaustive key search and a lower memory complexity than table precomputation, as indicated in the third column of Table 2.2. A first method was introduced by Hellman in [63]. Its complexity and success probability are summarized in the fourth column of Table 2.2. For the definition of the variables see Table 3.1. We shall elaborate on it in Chapter 3.

- **Distinguished points time-memory tradeoff:** This is a variation on the Hellman tradeoff, that was introduced by us in [22] and independently by Quisquater and Stern in [110]. It is based on an idea of Rivest [36, p.100]. The main advantage of this method over Hellman's is that it reduces the number of memory accesses significantly. We describe, discuss and analyze this method in Chapter 3. The results are summarized in the fifth column of Table 2.2; variables are defined in Table 3.2 on p. 47.

The complexities and success probabilities of these brute force attacks are summarized in Table 2.2. Note that the success probabilities for exhaustive key search and table precomputation are given as 1 with $\lceil \frac{k}{n} \rceil$. Here $\lceil x \rceil$ denotes the smallest integer larger than or equal to x. For key deduction the success probability is $1 - \frac{1}{e}$. With probability $\frac{1}{e}$ an attacker finds more than 1 possible key value. The target key can then be uniquely deduced by encrypting extra known plaintexts. In case of the time-memory trade-off methods, the error probability is a real error probability p. This means that, with probability p, a key cannot be identified based on the given memory content after the precomputation.

Table 2.2: Expected complexities of brute force attacks on block ciphers.

Attack	Key Search	Table	Tradeoff	Hellman	DP method
Known plaintexts	$\lceil\frac{k}{n}\rceil$	—	—	—	—
Chosen plaintexts	—	$\lceil\frac{k}{n}\rceil$	$\lceil\frac{k}{n}\rceil$	$\lceil\frac{k}{n}\rceil$	$\lceil\frac{k}{n}\rceil$
Precomp. complexity	—	2^k	?	$r_h m_h t_h$	$rm\gamma$
Memory complexity	—	2^k	$< 2^k$	$m_h r_h$	αr
Memory accesses	—	2^k	—	$m_h r_h$	r
Processing complexity	2^k	—	$< 2^k$	$r_h t_h$	$r\gamma$
Success probability	1	1	?	$1 - e^{-D(m_h,t_h)\frac{r_h}{2^k}}$	$1 - e^{-\frac{r\alpha\beta}{2^k}}$

2.5.3 Shortcut Attacks

Although shortcut attacks exploit the internal structure of a block cipher and hence need to be constructed specific for each cipher, there are several generic techniques for mounting an attack and for analyzing the resistance against it. Care has to be taken when trying to optimally apply these techniques, since each attack requires thorough analysis of all kinds of properties of the structure of an algorithm, a thourough understanding of the limitations of these techniques and last but not least a sufficient amount of intuition of the evaluator.

We summarize the techniques that have been proven to be successfully applicable to state-of-the-art designs of block ciphers.

- **Linear cryptanalysis (LC)** was introduced and developed by Matsui in [96, 97]. It is a known plaintext attack. Additional advanced techniques that are relevant can be found in [80, 71, 105, 135]. A basic linear attack makes use of a linear approximation between bits of the plaintext P, bits of the ciphertext C and bits of the expanded key K. Such a linear approximation is a probabilistic relation that can be denoted as

$$\alpha \cdot P \oplus \beta \cdot K = \gamma \cdot C\,, \tag{2.3}$$

where α, β and γ are binary vectors and $\chi \cdot X = \bigoplus_i \chi_i X_i$ for $\chi = (\chi_0, \chi_1, \ldots)$. Instead of P or C one can use intermediate values that can be computed from P or C by guessing (part of) a key value. If a linear approximation holds with probability $p = \frac{1}{2} + \delta$ with $\delta \neq 0$, a linear attack can be mounted which needs about $c|\delta|^{-2}$ plaintexts. The value of c depends on the attacking algorithm that is used. Here δ is called the deviation and $|\delta| = \epsilon$ is called the bias.[2] A predecessor of this technique is given by Matsui and Yamagishi in [99]. Here, all 'approximations' had bias $\frac{1}{2}$.

A cryptanalyst typically considers plaintexts and ciphertexts for a fixed but unknown key value K. For such a fixed key, $\beta \cdot K$ is a constant (either 0 or 1), hence (2.3) can be transformed into (2.4) without changing the bias:

$$\alpha \cdot P = \gamma \cdot C\,, \tag{2.4}$$

Note however that the probability of the approximation, that is, the sign of δ, will change for keys K with $\beta \cdot K = 1$. We say that *for certain values $\tilde{P}$ and $\tilde{C}$, (2.4) behaves in the deviation direction* if $\alpha \cdot \tilde{P} \oplus \gamma \cdot \tilde{C} = b$ and $\alpha \cdot P \oplus \gamma \cdot C = b$ has a positive deviation.

A linear approximation for the whole cipher can be derived by 'chaining' linear approximations between intermediate values. If the probabilities of these approximations are *independent*, the value of the deviation of the derived approximation can be computed with Matsui's Piling Up Lemma [96].

Theorem 2.1 [Matsui's Piling Up Lemma] *The deviation δ of n chained approximations with deviations δ_i is given by*

$$\delta = 2^{n-1} \prod_{i=1}^{n} \delta_i\,.$$

if all deviations δ_i are independent.

A proof can be found in [96].

- **Differential cryptanalysis (DC)** was introduced by Biham and Shamir in [9] and exploits the propagation of differences through a block cipher. To mount an attack one has to choose plaintext pairs with certain differences. Hence, differential cryptanalysis is viewed as a chosen plaintext attack. Mostly an attacker can choose plaintext values in an efficient way, such that from a set of chosen plaintexts many different pairs can be constructed with required differences. Such sets are called *structures*. The

[2] In the literature sometimes different names are used or the deviation is called bias and vice versa. Daemen introduced even different concepts such as correlation coefficients for linear cryptanalysis (for an overview see [30]) that are related to the concepts used throughout this work. Care has to be taken when comparing different sources.

chosen plaintext scenario can be changed into a known plaintext scenario that requires more plaintexts. This can be done as follows:

Theorem 2.2 *Let b denote the block length. Then a set of $\sqrt{2 \cdot N \cdot 2^b}$ texts is expected to contain about N pairs of each possible difference.*

A proof can be found in [9]. Theorem 2.2 implies the following.

Theorem 2.3 *Let b denote the block length. A differential attack that requires N pairs with a chosen plaintext difference can be performed as an attack that requires $\sqrt{2 \cdot N \cdot 2^b}$ known plaintexts.*

The technique of differential cryptanalysis has its merits in the analysis of several constructions, for example hash functions or one-way functions, and related key analysis of block ciphers.

- **LC and DC related attacks.** Some attacks have been described that generalize or combine LC or DC in some way. We mention the following, which have been successfully applied in practice.
  - *Truncated differentials* have been introduced by Knudsen in [77]. Like differential they consider difference propagation. Compared to differentials, only part of the difference propagation is taken into account, i.e., only part of the intermediate value is predicted. Truncated differentials seem to be most suited for attacking ciphers that mainly use operations on words of a small size.
  - *Linear-differential cryptanalysis* was introduced by Hellman and Langford in [64]. An attack is mounted by passing a first part of the cipher with a differential and the remaining part with a linear approximation.
  - *Impossible differentials* were introduced by Biham, Biryukov and Shamir in [8] and by Knudsen in the paper that analyzes the AES candidate DEAL. In these attacks the phenemenon of differences in intermediate values that never occur for a given input difference are exploited.
- χ^2 **cryptanalysis** was introduced by Vaudenay in [135]. Vaudenay's statistical model will be described later in this section. His χ^2 cryptanalysis is the most general instantiation that has resulted in practical attacks.
- **Others.**
  - *Meet-in-the-middle attack.* This method was originally introduced by Diffie and Hellman in [39] as an attack on multiple encryption schemes, i.e., the concatenation of several ciphers. Suppose E is a block cipher that can be written as

$$E_K = E^2_{K_2} \circ E^1_{K_1},$$

where the lengths of K_1 and K_2 are $k_1 < k$ and $k_2 < k$, with $k_1 \leq k_2$. Then a shortcut known plaintext attack can be mounted as follows. Suppose P, C is a corresponding plaintext/ciphertext pair.

```
1. For $K = 0$ to $2^{k_1}$
      a. Compute $M_K = E^1_K(P)$.
      b. Store $M_K, K$.
2. For $K = 0$ to $2^{k_2}$
      a. Compute $N_K = D^2_K(C)$.
      b. If $N_K$ is in the stored list,
         then check the suggested key with additional
              plaintext/ciphertext pairs.
```

This attack has a processing complexity of 2^{k_2} computations of E^2 and a memory complexity of 2^{k_1}. Time-memory tradeoffs are possible and the attack can be generalized to a concatenation of more than 2 ciphers, which is described in [101].

- *Interpolation attacks* were introduced by Jacobsen and Knudsen in [70] and further extended by Jakobsen in [69]. The basic idea is to find an expression for (part of) the target cipher as a polynomial with few coefficients that is a function of the input, output and key. If there is an expression that is suitable, interpolation techniques can be used to find an alternative representation of the cipher or for key recovery.
- *Slide attacks* were introduced by Biryukov and Wagner in [14]. These attacks exploit similarities in the round function and symmetries in the key schedule. An example of such an attack is given in Sect. 7.6.3.
- *Dedicated Square attack.* This attack was already described by the designers of Square at its introduction in [31]. All ciphers that are designed based on the same principles as Square are potentially vulnerable and hence need to be evaluated for resistance against this method. It exploits the slow diffusion and byte wise structure of a cipher.

Except for the last kind (others), all attacks exploit some property that propagates with a certain probability through the cipher, i.e., linear approximations, differentials, characteristics, truncated differentials, etc. Note, that finding such a property with sufficiently high probability implies that the target cipher can be distinguished from a random permutation. Typically a cryptanalyst that wants to mount an attack has two tasks.

- **Finding a 'good' propagation property.** Hence finding a linear approximation with high bias, characteristic with high probability, etc. The search has to be performed efficiently since trying all possibilities is computationally infeasible. The typical method is to analyze smaller components of the cipher for propagation properties and to combine this analysis to give an estimate for the propagation properties concerning the whole cipher. The iterated structure has resulted in strategies that base their approximation on the iteration of propagation properties for the individual rounds.

Care has to be taken when applying such an analysis. To facilitate the analysis mostly some assumptions are made. Three typical assumptions are:

Assumption 2.4 *In the propagation of a property through different components the individual probabilities are assumed to be independent.*

Assumption 2.5 [The hypothesis of stochastic equivalence [88]] *The propagation of the property through a key-dependent component is assumed to be stochastically independent of the specific key values. The computation of this property can then be done by averaging over all possible key values.*

Assumption 2.6 *The probability of propagation of a property through the different key-dependent components is assumed to be stochastically independent of the specific key values.*

Analysis where situations are pointed out where these assumptions do not hold and how they can be exploited can be found in [19, 30, 111]. An interesting overview concerning these assumptions and linear cryptanalysis can be found in [126]. In this thesis we shall show situations where these standard assumptions do not hold and how to perform a good analysis nevertheless.

- **Exploiting the property.** There are two approaches. If the property depends on subkey values, these can be deduced with statistical methods (assuming sufficient texts and a sufficiently high probability of the property). A method that can be more efficient to deduce key material is to make a statistical analysis by guessing key material in the outer rounds. This allows to 'peel off' one or more rounds. The assumption behind this method is that the statistical property will be more apparent for a correct guess than for a wrong guess. These techniques are called *outer round tricks* or *1r tricks.*

Hence, for research in the area of these attacks a distinction can be made between research in methods for the distinct tasks. In general one can say that the probability of a propagation property decreases if the number of components increases. This has led to a design process where the number of rounds is derived by taking the number of rounds that resists the best known (theoretically) exploitable property and adding several rounds to resist outer round tricks. Of course increasing the number of rounds implies a decrease in performance. Hence, much research in this area consists of cryptanalyzing (and breaking) iterated block ciphers with a reduced number of rounds.

1. Unless the claims of the designer are refuted, such attacks increase the amount of trust in the full round version of the cipher.

2. These attacks lead to design principles, since they show exploitable weaknesses in the components or their interaction.

Besides analyzing reduced round versions of an iterated cipher, results have also been published on reduced ciphers using a different approach: the cipher can also be reduced by omitting or changing components, such that the round structure of a cipher is changed.

Vaudenay's statistical cryptanalysis attack model. The attacks that use propagation properties all fall in the category of *statistical attacks* as defined by Vaudenay in [135]. Statistical cryptanalysis exploits statistically non-random properties that depend on key material. This non-random behavior can be measured from values that are computed using plaintext and ciphertext values and some key material that is known, i.e., in an attack this key material is guessed and for the right key guess non-random behavior is expected and for other key values random behavior. Furthermore, the attack is not only applicable to block ciphers, but also to stream ciphers. It is formalized in [135] as follows.

Let $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{K}$ denote the sets as in Definition 2.5 (for a block cipher as in Definition 2.10). Then in the statistical attack model three functions are defined:

- $h_1 : \mathcal{K} \to \mathcal{L}$, where $l = |\mathcal{L}| \leq |\mathcal{K}|$. This is typically a function that derives subkey material from the user key.

- $h_2 : \mathcal{P}^r \times \mathcal{C}^r \to \mathcal{S}$, where r is an integer (for the introduction of the model in [135] r was taken to be 1, but later implicitly generalized to any integer) and $|\mathcal{S}| \leq |\mathcal{P}|\cdot|\mathcal{C}|$. $\mathcal{S}$ is called the *sample space*; h_2 derives the 'exploitable' information from plaintext/ciphertext tuples.

- $h_3 : \mathcal{L} \times \mathcal{S} \to \mathcal{Q}$. Typically, the values of $\mathcal{Q}$ are intermediate values of the encryption.

The input of h_3 is a key material guess and a sample. Hence, for N samples $|\mathcal{L}|$ distributions on $\mathcal{Q}$ can be made. Typically $\mathcal{Q}$ contains all possibilities for an intermediate value. For the right key guess this intermediate value would be computed. In this thesis we will present a situation in which this is not the case. For a successful attack a statistical measurement function Σ is required that can distinguish the distribution of the target key value from that of the other distributions. As a generalization this function could rank all key guesses, according to the (computed) probability that they are the target key. We formalize this as

$$\Sigma : \mathcal{Q}^N \to [0,1], \text{ where } N \text{ is an integer and } \sum_{q \in \mathcal{Q}^N} \Sigma(q) = 1.$$

If Σ is chosen to be a χ^2 test, then analysis according to this model is called χ^2 *cryptanalysis*. It also is possible to describe the model where h_3 and Σ are integrated in one function. The general attack can be described as consisting of four phases:

1. **Counting phase.** N samples $S_i = h_2(P_i, C_i)$, $i = 1, \ldots, N$, are collected.

2. **Analysis phase.**
   For each candidate $\tilde{K}$ of the l candidates for the key material:
   1. For all i compute $X_i = h_3(\tilde{K}, S_i)$.
   2. Compute the 'mark' $M_k = \Sigma(X_1, \ldots, X_N)$.

3. **Sorting phase.** Sort the candidates according to their marks.

4. **Searching phase.** Exhaustively try all candidates according to their marks.

Note that this model does not allow adaptive chosen plaintext attacks. However, one could extend it by allowing alternate counting and analysis phases. Furthermore, it can be more efficient (for example for decreasing the required memory) in practical situations to compute X_i directly after obtaining (P_i, C_i), hence the counting and analysis phase can be performed in an interleaved way.

For more details on how to choose the characteristic and the statistical measurement function, and a more detailed general complexity analysis we refer to [135].

2.5.4 Standards and Proposals

Data Encryption Standard (DES) and Advanced Encryption Standard (AES). The best known and most widely used block cipher is DES. It has a 64-bit block size and a 56-bit key size. DES was designed in 1977 by IBM in collaboration with the National Security Agency (NSA), which designed the S-boxes. DES was standardized by NIST in [52]. This standard was followed by ANSI with [3].

The design criteria have always been kept secret, although some of them have more or less been revealed by Coppersmith (one of the designers) as a reaction on the discoveries of differential and linear cryptanalysis [29]. The first shortcut attack was a differential attack, published by Biham and Shamir in [9]. Currently the best shortcut attack is a known plaintext attack by Shimoyama and Kaneko [129], which is an improvement of the linear attack by Matsui [96, 97]. Knudsen and Mathiassen have developed a chosen plaintext variant [80] with lower complexity.

Nowadays the 56-bit key is considered too short for many applications and as a more secure alternative (two-key) triple-DES is used. This is a cipher for which encryption consists of a DES encryption, followed by a DES decryption, followed by a DES encryption. The DES encryptions are performed using the same key, hence the key length of triple-DES is 112 bits. This mode is standardized by

ANSI in [4] and by ISO in [66]. Another alternative, based on DES is DESX, that was introduced by Rivest and described in [74].

Because of the small key length of DES and the performance penalty induced by the use of triple-DES, in 1997 NIST has initiated a standardization process for a new block cipher standard to replace the DES: the Advanced Encryption Standard (AES) [2, 56]. The new standard should support key sizes of 128, 196 and 256 bits. An additional security related criterion is that the new standard should have a block size of 128 bits for resistance against dictionary like attacks. Furthermore it should be efficiently implementable on a variety of platforms.

For this purpose NIST has issued an open call for algorithms, i.e., the 'public' was asked to submit proposals for the new standard. In August 1998 fifteen algorithms were accepted that satisfied the submission criteria. In August 1999 after a first period of analysis, including two AES conferences, from the fifteen candidates five algorithms remained: MARS, RC6, Rijndael, Serpent, and Twofish. The choice for the final choice was made in October 2000. NIST selected the algorithm Rijndael, designed by Daemen and Rijmen for AES.

Intermediate era. Even before the initiative of NIST for a new block cipher standard, a number of other block ciphers have been proposed. We summarize the most prominent ones.

- *FEAL* is the name of a family of block ciphers that have played a critical role in the development of several cryptanalytic techniques, in particular linear and differential cryptanalysis. An overview of and references to design and analysis can be found in [101].

- *IDEA*. The original design criteria of this algorithm were introduced by Massey and Lai in [89], which lead to a design, called PES. In [90] design weaknesses were discovered, which lead to a design called IPES, that now is known as IDEA. It has a 64-bit block length and 128-bit key length. IDEA is used in many commercial products, amongst others in PGP. The best attacks are by the author, Knudsen and Rijmen in [19], and by Biham, Biryukov and Shamir in [7].

- *SAFER* is the name of several block ciphers, all (co)designed by Massey, originally introduced in [94, 95]. The best attacks on the first versions were published by Knudsen and Berson in [79]. Knudsen also gives an overview of analysis results and describes a new key schedule in [78]. All versions have 64-bit block size. These ciphers were the basis for SAFER+, one of the 15 original AES candidates, designed by Khachatrian, Kuregian and Massey.

- *Blowfish* was introduced by Schneier in [123]. It has gained much popularity, not in the last place because its designer is the author of [124]. Its security is based on large S-boxes, that are computed from the key using an involved computation. Although, this was initially an *ad hoc* argument, the fact that no impressive results are known makes it a trustworthy cipher. The best attacks are by Van Rompay and Verelst in [133], and Vaudenay in [134].

- *RC2* was designed by Rivest in the eighties and originally kept confidential. It was commercialized by RSA Inc. and used in many commercial products such as Netscape Communicator and Microsoft's Internet Explorer. A description and the best attack by Knudsen, Rijmen, Rivest and Robshaw can be found in [83].

- *RC5* was introduced by Rivest in [116, 117]. It has a simple structure and lends itself to analysis. Its security is based on the use of data dependent rotations; this method is even patented in [118, 119]. It has a variable block (64 or 128 bits) and key size. The best attacks are by Biryukov and Kushilevitz in [11], and the attack by the author, Preneel and Vandewalle, published in [23], that is described in Chapter 4. RC5 is commercialized by RSA Inc. From RC5 the former AES candidate RC6 was developed.

- *Square* was introduced by Daemen and Rijmen in [31]. It has a 128-bit block size and 128-bit key size. Its design and security are based on the *Wide Trail Strategy* as described by Daemen in [30] and Chapter 5. The best attack is given by its designers and Knudsen in [31]. Square was the basis for Rijndael, which was selected as AES.

- *Misty.* In [98] Matsui introduced two ciphers called Misty1 and Misty2. Both are constructions of ciphers that are provable secure against differential and linear cryptanalysis. This has also been the basis for the block cipher KASUMI [47], that is used as a primitive in the UMTS standard.

- *Skipjack.* Skipjack was originally a confidential design by the NSA to be embedded in tamper-resistant chips, that were commercially available on Fortezza PC cards. Recently it has been disclosed [8, 84]. It has a 64-bit block length and an 80-bit key size. The best attack is by Biham, Biryukov and Shamir [8].

2.6 One-Way Functions and Hash Functions

2.6.1 Definitions and Properties

One-way functions and cryptographic hash functions are unkeyed cryptographic primitives that share the following property.

Property 2.2 *A one-way function or cryptographic hash function f has the property* ease of computation*: for every input x (from the domain of f) $f(x)$ is 'easy' to compute.*

Furthermore a hash function has the extra property:

Property 2.3 *A hash function f maps an input x of arbitrary bit length to an output $h(x)$ of fixed bit length.*

Concerning their cryptographic properties; they (should) have at least one of the following three properties.

Definition 2.12 Preimage resistance *The function f is* preimage resistant *or* one-way *if the following holds. Given any image y, for which there exists an x with $f(x) = y$, it is computationally infeasible to compute any preimage x' with $f(x') = y$.*

Definition 2.13 Second-preimage resistance *or* **weak collision resistance**. *The function f is* second-preimage resistant *if the following holds. Given any preimage x it is computationally infeasible to find a 2nd-preimage $x' \neq x$ with $f(x) = f(x')$.*

Definition 2.14 Collision resistance *or* **strong collision resistance**. *The function f is* collision resistant *if the following holds. It is computationally infeasible to find any two distinct inputs x, x' such that $f(x) = f(x')$.*

Similar to the main design criterion for block ciphers, (well designed) one-way and hash functions have the following property.

Property 2.4 *A one-way function with or restricted to a finite domain or a hash-function restricted to a finite domain should be (practically) indistinguishable from a random mapping.*

Formalization of this property is a little bit more tricky: if a function is completely specified, it is easy to distinguish it from a random mapping; a way around this is to define a family of functions.

2.6.2 Brute Force Attacks

Considering the security evaluation of one-way and hash functions with respect to Definitions 2.12, 2.13 and 2.14, the following attacks are always feasible. Let f be such a function with a range of all n-bit values.

- **Preimage resistance:** To produce a preimage of y, an attacker computes the image of different input values, until he finds an x with $f(x) = y$. The processing complexity is approximately 2^n.
- **Second-preimage resistance:** Given an input x, an attacker computes the image of different input values (not equal to x), until he finds an $x' \neq x$ with $f(x') = f(x)$. The processing complexity is approximately 2^n.
- **Time-memory tradeoffs:** The online processing complexity (or success probability) for finding preimages and second preimages can be improved if an attacker invests in precomputations, i.e., he can precompute a number of images. This has important consequences if the target function has a restricted domain, for example only can take inputs of a given length. A description of such methods can be found in Chapter 3.

- **Collision resistance:** An attacker computes the images of different inputs until he has found two that collide. The processing complexity is approximately $2^{n/2}$, with a success probability of about 0.39, or $2^{(n/2)+1}$, with a success probability of about 0.63. Hence, a naive implementation of this attack would require a memory complexity of $2^{n/2}$. In [108, 109] Quisquater and Delescaille describe a method for which memory requirements are negligible.

The complexities are summarized in Table 2.3.

Table 2.3: Expected complexities of brute force attacks on one-way functions and hash functions.

Attack	Preimage	2nd Preimage	Tradeoff	Collision
Precomp. complexity			?	
Memory complexity			?	negligible
Memory accesses			?	negligible
Processing complexity	2^n	2^n	$< 2^n$	$2^{n/2+1}$
Success probability	1	1	?	0.63

2.6.3 Standards and Proposals

Most efficient and practically used one-way functions are hash functions with a restricted domain.

Almost all practically used hash functions are based on *iterated compression functions:*

Definition 2.15 *An* m, n-bit iterated compression function *is a function* $f : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^m$, *that is used to compress a string* $x_1, \ldots, x_t$ *of* n*-bit blocks to a* m*-bit value* H_t *that is iteratively computed by*

$$H_0 = IV \qquad (2.5)$$

$$H_i = f(H_{i-1}, x_i) \text{ for } i = 1, \ldots, t, \qquad (2.6)$$

where IV is called an initialization value.

The initialization value can be predefined or is to be chosen (randomly) for each hash computation, depending on the hash function specification. Concerning information access for a hash function attack one defines *free-start attacks* as attacks where an adversary can choose the initialization value.

The generic scheme of a hash function, that takes as input arbitrarily length input and gives as output an r-bit result, based on an iterated compression function is the following.

1. The input is padded to a string $x_1, \ldots, x_t$ of n-bit blocks.
2. The value H_t is computed using (2.6).
3. The hash value is given by the output $g(H_t)$, where $g : \{0,1\}^m \rightarrow \{0,1\}^r$ is called the *output function.* (Often $m = r$ and $g(H_t) = H_t$.)

A distinction can be made between two types of hash functions and one-way functions: dedicated constructions, and constructions using a block cipher as a building block. We will give a short overview of the most popular functions; all of them are based on an iterated compression function.

- *MD4, MD5.* MD4 was introduced by Rivest in [113]. It produces a 128-bit output. Due to analysis by Den Boer and Bosselaers in [15], Rivest proposed in [115] a strengthened version: MD5, which also produces a 128-bit output. Indeed, later in [40] Dobbertin presented an attack that found collisions for MD4. The analysis by Dobbertin in [42] also creates doubts about its preimage resistance; therefore MD4 has become obsolete. Dobbertin has also found collisions for the compression function of MD5 [41]; finding collisions for MD5 itself is now believed to be feasible.
- *SHA, SHA-1, SHA-256, SHA-384, SHA-512.* The NSA has designed the hash function SHA-1, which has a 160-bit output. It is standardized by NIST in [54]. Originally NSA had proposed the design SHA (sometimes referred to as SHA-0), but due to a weakness a slight modification has been introduced which resulted in SHA-1. This weakness as well as the other design motivations are confidential. Analysis that possibly motivates the change from SHA into SHA-1 can be found in [28]. Recently NIST has requested NSA to design extensions of SHA-1 with longer results: SHA-256, SHA-384 and SHA-512 [55].
- *RIPEMD-160.* RIPEMD-160 is a hash function with a 160-bit output, which was designed by Dobbertin, Bosselaers and Preneel. It is described in [43]. SHA-1 and RIPEMD-160 are also standardized by ISO/IEC in [68].
- *Block Cipher Constructions.* There are several (secure) constructions for hash functions based on block ciphers. A synthetic approach of generic schemes to construct a hash function from a block cipher where the output size is equal to the block size is given by Preneel in [107]. Three popular schemes that have good properties according to that analysis are the following. All schemes are defined by specifying the function f in equation (2.6). A characteristic of these schemes is their *rate:* this is the inverse of the number of block encryptions that have to be computed to process each message block.

Davies-Meyer:

$$H_i = H_{i-1} \oplus E_{x_i}(H_{i-1}) \,. \tag{2.7}$$

It has a rate of k/n, with k the key length (in bits) and n the block length (in bits). Hence, this mode seems particularly suitable when used with block ciphers that permit large key lengths.

Matyas-Meyer-Oseas:

$$H_i = x_i \oplus E_{h(H_{i-1})}(x_i) \,, \tag{2.8}$$

where h maps the chaining variable to a key of suitable length. It has rate 1.

Miyaguchi-Preneel:

$$H_i = H_{i-1} \oplus x_i \oplus E_{h(H_{i-1})}(x_i) \,. \tag{2.9}$$

This scheme has rate 1.

Popular constructions where the output size is twice the block length are MDC-2 and MDC-4 with rate $\frac{1}{2}$ and $\frac{1}{4}$, respectively. The Matyas-Meyer-Oseas and MDC-2 scheme are included in the ISO standard 10118-2 [67].

For an elaborate treatment of one-way functions and hash functions we refer to [107] by Preneel.

2.7 Conclusions

In this chapter we have introduced the field of information security, its applications and objectives. Cryptology offers important tools to achieve many objectives of information security with technical methods. We have given a framework from which the security objectives can be derived, that are achievable with cryptographic techniques. This framework consists of the concepts: confidentiality, data integrity, authentication and non-repudiation.

We have given a classification of the cryptographic tools or primitives that can be used to achieve these objectives and we have described their evaluation criteria. For reasons of interoperability and as a certification mark for the level of soundness and trust, primitives and their modes of use are described in standards, which are developed by several standardization bodies.

The design of cryptographic primitives or cryptosystems and their security evaluation are highly interactive. To design a cryptosystem a thorough knowledge of and experience with security evaluation techniques (generic as well as *ad hoc*) seems required. There are several approaches to evaluate and determine the level of security of a primitive or cryptosystem of which we have given an overview. At the moment provable security is out of reach. Hence, each design requires independent analysis. The most suitable design approach seems to be the conservative approach: use provable and practical security arguments if possible and use *ad hoc* arguments that are generally accepted.

We have given several classifications of security analysis and attacks on symmetric ciphers by determining the goals, assumptions, methods of information access, the use of classical and side channel analysis, the level of complexity and the exploited feature of the attack. This chapter contributes to the existing theory by fitting in side channel analysis in the classical framework. Another contribution is the determining of four definitions of breaking a cipher and their relevance. This gives an extra criterion to choose the most suitable cipher for an application.

An important class of cryptographic primitives is the class of block ciphers, since they can be used as building blocks for many other cryptographic primitives and cryptosystems. We have given a succinct overview of current state-of-the-art in design and analysis of block ciphers, one-way and hash functions, comprising of properties, brute force and shortcut attack methods as well as proposals and standards.

Chapter 3

A Time-Memory Tradeoff Attack

3.1 Introduction

This chapter studies a method to invert a one-way function (that may have a domain and range of different size), where precomputation of an amount of memory is used to reduce the time needed for inversion. This has as main application: improved brute force key search on symmetric encryption algorithms, which is mainly important for production cryptanalysis, where a large number of keys has to be recovered. We describe the method in the context of a chosen plaintext attack on a block cipher; the attack requires the encryption of a single fixed set of plaintexts, the value of which does not matter. We describe and analyze a variation that was introduced in [22] by the author, Preneel, and Vandewalle, and independently also by Quisquater and Stern in [110]. We also show its advantages in practical adaptation. Furthermore, we have implemented the variation and show experimental results that are in accordance with the analysis.

The remainder of this chapter is organized as follows. We summarize the previous work and our contribution in Sect. 3.2. In Sect. 3.3 we describe the application of the time-memory tradeoff. In Sect. 3.4 we show how to construct a one-way function from a block cipher, for which inverting it is equivalent to breaking the cipher. In Sect. 3.5 we describe Hellman's method to invert it and in Sect. 3.6 the method using distinguished points. The analysis of the distinguished points method is included in the remaining part: Sections 3.7 and 3.8 contain some preliminary work and in Sect. 3.9 the complexities and success probability of the method are derived. We compare the two methods in Sect. 3.11 and conclude in Sect. 3.12.

3.2 History

The tradeoff method was introduced and applied to DES in [63] by Hellman. In [50] Fiat and Naor propose a variant that is applicable to any cipher at the cost of extra workload and/or memory. Kusuda and Matsumoto generalize the Hellman method in [87]; they derive stricter bounds on the success probability and give relationships between the memory complexity, processing complexity, and success probability. This enables to optimize the tradeoff with respect to memory and processing cost and results in estimates for the cost and capacities of key search machines using dedicated hardware.

In all previous work the number of memory accesses was considered to be negligible. However, this is not always the case; one practical example would be when one uses Hellman's tradeoff method to perform a distributed key search over a LAN or over the Internet. A new tradeoff method was introduced independently by the author, Preneel, and Vandewalle in [22] and Quisquater and Stern in [110]. It is based on distinguished points and reduces the expected number of memory accesses with a large factor and thus overcomes these problems. It was based on an idea by Rivest [36, p.100].

The comparison of our method compared to Hellman's and the relationship with other brute force attacks is summarized in Table 2.2.

3.3 Applications

The aim of a time-memory tradeoff (for a block cipher attack) is to mount an attack which has a lower online processing complexity than exhaustive key search and a lower memory complexity than table precomputation, as indicated in the third column of Table 2.2.

Cryptosystems. The method as introduced by Hellman as well as our variation is actually an efficient method to compute the preimage of inputs of a function, if a certain amount of precomputation is invested. It can be applied to many cryptosystems:

- **One-way functions and hash functions:** The tradeoff method can be used to retrieve first and second preimages. The success probability and complexity depend on the input and output sizes of the function.

- **Asynchronous stream ciphers and key stream generators:** For a stream cipher a known plaintext attack can reduce part of the key stream. If sufficient key stream is known (part of) the internal state or seed can be deduced by exhaustive searching. For this the online complexity can be reduced using a time memory tradeoff. An interesting application of this kind of attack is given by Biryukov, Shamir and Wagner in [13] for the GSM encryption algorithm A5/1.

- **Block ciphers:** In the next section we will show a method to construct one-way functions from a block cipher for which holds that finding a certain preimage is (with high probability) equivalent for key recovery. It is a chosen plaintext attack.

Chosen plaintext attack on block ciphers. The attack as described in the next section is a chosen plaintext attack on a block cipher in ECB-mode and can be prevented by using CBC-mode with a random IV-vector. Still there are practical applications, which would be vulnerable to a chosen plaintext attack. We give the following examples.

- In the UNIX-password one-way function [101], a user password of 8 ASCII-characters is used to encrypt $0 \ldots 0$ with DES 25 times iterated. Although password salting gives 2^{12} variations of the block cipher, it is possible to make a database for part of the salt values or for a library of passwords of a given structure (for example passwords only consisting of a subset of all ASCII characters).
- S/MIME is an email security protocol that for example is used in Netscape Communicator and Microsoft's Internet Explorer. Besides DES it supports RC2 with a 40-bit key. The block cipher is used in CBC-mode, but the first encrypted block is always the ASCII-equivalent of 'Content-'. It is encrypted using a random IV-vector, which is then sent along with the message. If the random generator is predictable or can be influenced, then an attacker might profit by making tables for the corresponding IV-values.
- Many file formats begin with a standard header such as `%!PS-Adobe-3.0`, `\documentclass` and many Microsoft formats, although these are not in readable ASCII. This may always give rise to a chosen plaintext attack.

We conclude that time-memory tradeoff attacks need to be taken seriously into account in the design of secure systems.

3.4 A One-Way Function

In this section we show how to construct a one-way function f from a block cipher and a set of chosen plaintexts. It has the property that for ciphertexts corresponding to the chosen plaintexts, the target key can be found by inverting the function for a point that depends on the ciphertexts. In the next two sections we give two methods for inversion, namely Hellman's method [63] and the new method, denoted Distinguished Point (DP) method.

Let $E : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$ be a block cipher with block length n and key length k.

Define $q = \lceil \frac{k}{n} \rceil$, hence q is the required amount of known plaintexts or chosen plaintexts to determine the target key by exhaustive key search or table precomputation, for which the complexities are summarized in the first two columns of Table 2.2.

Define a function $g : \{0,1\}^{qn} \rightarrow \{0,1\}^k$ as

$$g(X) = g(X_1, \ldots, X_{qn}) = (X_{i_1}, \ldots, X_{i_k}) \text{ for all } X = (X_1, \ldots, X_{qn}) \in \{0,1\}^{qn},$$

where the i_j's are different elements of $\{1, \ldots, qn\}$. This function has the following two properties, which are important for our purpose. It is efficiently computable and there are many, namely $\frac{(qn)!}{(qn-k)!}$, different possibilities to define the function g.

Next, define on the key space a function $f : \{0,1\}^k \rightarrow \{0,1\}^k$ as

$$f(Y) = g(E_Y(P_1)| \ldots |E_Y(P_q)) \text{ for all } Y \in \{0,1\}^k, \tag{3.1}$$

where $P_1, \ldots, P_q$ are q different (chosen) plaintexts. This construction was introduced in [87] and is a generalization for the construction as given by Hellman in [63], which only holds for the case $k < n$. The function has the following property:

Property 3.1

$$\{C_i = E_K(P_i) \textit{ for } i = 1, \ldots, q\} \Rightarrow g(C_1| \ldots |C_q) = f(K)\,,$$

hence $g(C_1| \ldots |C_q)$ is the image of K under f, or in other words K is a predecessor of $g(C_1| \ldots |C_q)$. So a method to invert f is a method that can determine the secret key.

Property 3.2 *For a well designed block cipher the one-way function (3.1) can be modeled as a random mapping.*

This follows directly from Property 2.1.

Note that from $g(C_1| \ldots |C_q) = f(K)$ it does not necessarily follow that $C_i = E_K(P_i)$ for $i = 1, \ldots, q$, since $g(C)$ can have more than one predecessor, from which one is the target key. The situation where an inversion method finds another predecessor is called a *false alarm.* False alarms do make sense in the case of a block cipher inversion; they do not make sense if one just wants to find a preimage of a function (unless one wants to find a specific preimage).

3.5 Hellman's Tradeoff Method

For the sake of clarity the variables, definitions and functions for Hellman's method are summarized in Table 3.1.

Chains. For a random start point $SP \in \{0,1\}^k$, define a *chain* $K_0, K_1, \ldots, K_{\tilde{t}}$ of length $\tilde{t} + 1$ as

$$\begin{aligned} K_0 &= SP = f^0(K_0) \\ K_i &= f(K_{i-1}) = f^i(K_0) \text{ for } i = 1, \ldots, \tilde{t}. \end{aligned}$$

Table 3.1: Overview of variables, definitions and functions concerning Hellman's method.

Variable	Meaning	Pointer
t_h	Number of iterations that is done on a start point.	Alg. 1
m_h	Number of chains in a table.	Alg 1
r_h	Number of tables.	Alg. 1
$D(m_h, t_h)$	Number of different points in a table.	(3.2)
$E(F)$	Expected number of false alarms per table.	(3.5)

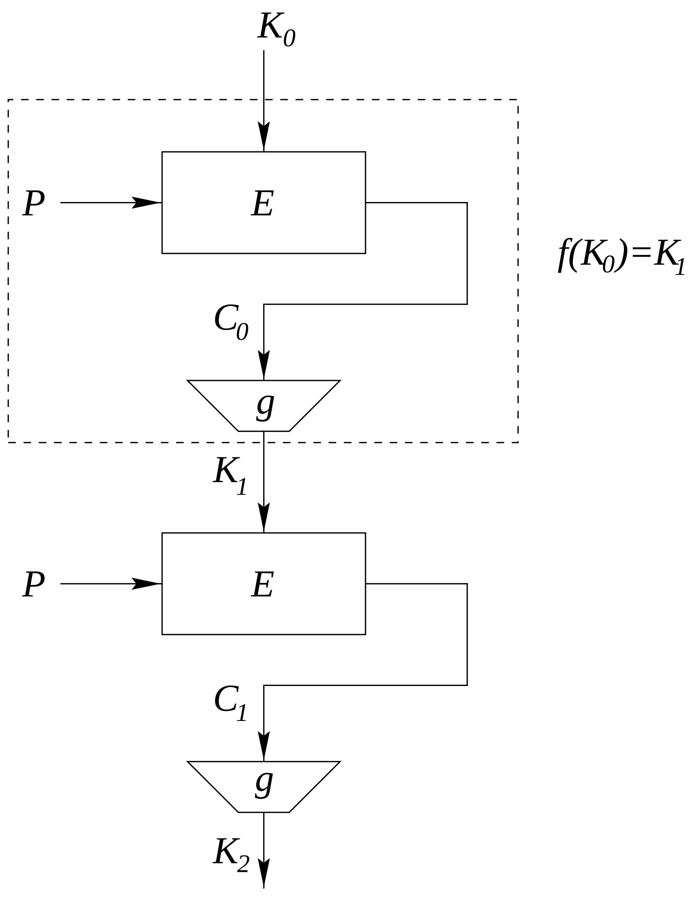

Figure 3.1: Iteration of the one-way function.

This iteration is visualized in Figure 3.1.

If only the start point and end point $EP = K_{\tilde{t}}$ are stored, then it is possible to determine whether a key K is equal to K_i for some $i \in \{1, \dots, \tilde{t}\}$ and, if so, to determine its predecessor in the chain, all with a complexity of $\tilde{t}$ operations. This can be achieved as follows.

Determining if a key is in a chain. Let $K \in \{0,1\}^k$.

1. For $l = 1$ to $\tilde{t}$ do
2. Check if $f^l(K) = EP$, if so then the predecessor of K is $f^{\tilde{t}-l}(SP)$.
3. Else, take next l.

In the tradeoff method one tries to store information about as many different predecessors as possible by taking $\tilde{m}$ different random start points and constructing thus $\tilde{m}$ chains of length $\tilde{t}+1$. A set of $\tilde{m}$ chains of length $\tilde{t}+1$ can contain information about at most $\tilde{m}\tilde{t}$ different predecessors, but can contain several predecessors that are equal, which we call *overlap*. Overlap is caused by one of the following two situations.

- A chain can *cycle*. This is the situation in which there are an i and j with $i \neq j$ and $K_{i+1} = K_j$. This is visualized in Figure 3.2.
- Two chains can *merge*. This is the situation in which two keys in different chains have the same image. Note that this means that from the moment of the merge till the end of at least one chain, both chains contain the same keys. This is visualized in Figure 3.3

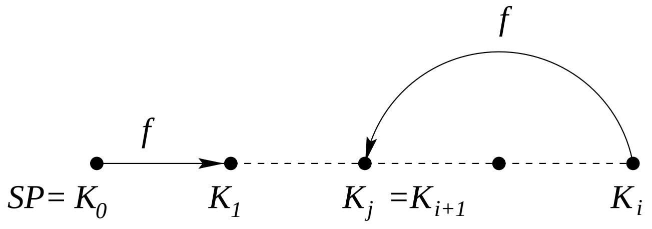

Figure 3.2: A cycling chain.

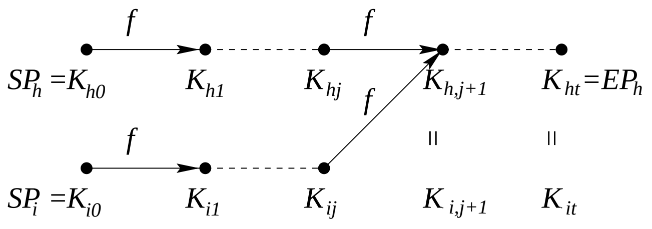

Figure 3.3: Two merging chains.

This implies that taking $\tilde{m}$ and $\tilde{t}$ large would mean that there would be a significant overlap. Hence, we take the values for $\tilde{m}$ and $\tilde{t}$ not too high. Furthermore, to reduce the overlap caused by the merging of two chains we take different functions f by choosing different functions g. We call a set of chains that are computed with the same function f a *table*.

Hellman's method. Hellman introduced the following precomputation algorithm and search algorithm, that use parameters m_h, t_h and r_h.

Precomputation algorithm 1
Make r_h tables with m_h (SP, EP)-pairs, sorted on EP.

```
1. Choose $r_h$ different random functions $g_i, i = 1, \dots, r_h$.
2. For $i = 1$ to $r_h$
        A. Choose $m_h$ different random starting points $SP_1^{(i)}, \dots, SP_{m_h}^{(i)}$.
        B. Compute $m_h$ end points $EP_j^{(i)} = f_i^{t_h}(SP_j^{(i)})$.
        C. Store in table $i$: $m_h$ $(SP, EP)$-pairs, sorted on end point.
```

Search algorithm 1
Given $C_i = E_K(P_i), i = 1, \dots, q$. Find K.

```
For $i = 1$ to $r_h$
    1. $Y = g_i(C_1|\dots|C_q)$.
    2. For $j = 0$ to $t_h - 1$
        A. If $Y = EP_l^{(i)}$ for some $l$ then
            a. Compute predecessor $\tilde{K} = f_i^{t_h-1-j}(SP_l^{(i)})$.
            b. If $C = E_{\tilde{K}}(P)$ then $K = \tilde{K}$: stop.
            c. If $C \neq E_{\tilde{K}}(P)$ then false alarm.
        B. Set $Y = f(Y)$.
```

Note 3.1 **Precomputation algorithm 1** *and* **Search algorithm 1** *are given in the typical setting of a (software) search on one machine; the tables are generated and searched sequentially. Of course this can be parallelized up to a parallel search in r_h tables.*

The attack has the following complexities. We shall analyze the complexities for the sequential case; adaptation to the parallel case is straightforward.

Memory complexity. Storage is needed for r_h tables. Each table contains m_h chains. These are in the form of pairs and other well-known tricks can be done to save on required space, such as choosing start points that need less storage. Hence, memory for $m_h r_h$ table entries is needed.

Success probability. In [87] the number of *different* points $D(m_h, t_h)$ covered by a Hellman table is estimated as

$$D(m_h, t_h) \approx \frac{2^k}{t_h} \int_0^{\frac{m_h t_h^2}{2^k}} \frac{1 - e^{-x}}{x} \, dx \,. \tag{3.2}$$

It holds under the restrictions that $m_h \gg 1$, $t_h \gg 1$, and $m_h t_h \ll 2^k$. Further, according to the experimental results in [87] the approximation gets worse if $\frac{m_h t_h^2}{2^k}$ gets larger than 1.

All points in the same table are different, but there can be overlap between different tables, i.e., a point can be covered by several tables. The success probability, the probability that some key K is at least in one of the tables, is given by

$$\begin{aligned} \text{Pr}_{\text{success}} &= 1 - \prod_{i=1}^{r_h} \text{Pr}(K \text{ not in table } i) \\ &\approx 1 - \prod_{i=1}^{r_h} (1 - \frac{D(m_h, t_h)}{2^k}) \\ &= 1 - (1 - \frac{D(m_h, t_h)}{2^k})^r \qquad (3.3) \\ &\approx 1 - e^{-\frac{rD(m_h, t_h)}{2^k}} \,. \qquad (3.4) \end{aligned}$$

Precomputation complexity. In the precomputation phase r_h tables are computed. For each table m_h chains of length t_h are computed. Hence, this gives a complexity of $r_h m_h t_h$ computations of the one-way function.

Processing complexity. The following analysis was given by Hellman in [63].

The expected number of false alarms per table tried is bounded by [63]:

$$E(F) \leq \frac{m_h t_h (t_h + 1)}{2^{k+1}} \,. \tag{3.5}$$

At most t_h operations are required to distinguish a false alarm from the recovery of the target key. If $m_h t_h^2 \approx 2^k$, then the expected processing complexity due to false alarms increases the expected processing complexity by at most 50 percent.

Hence, the processing complexity can be estimated to be in the order of $r_h t_h$.

The success probability and precomputation, memory and processing complexity are summarized in the fourth column of Table 2.2. Note that the processing complexity caused by false alarms can be neglected [63, 87]. When the attack is mounted with a precomputation complexity of 2^k and memory complexity equal to processing complexity, then $m = t = r = 2^{k/3}$ and the success probability is around 0.55 [63].

Table 3.2: Overview of variables, definitions and functions concerning the Distinguished Point method.

Variable	Meaning	Pointer
d	Order of the DP-property.	Alg. 2
t	Maximum number of iterations that is done on a start point.	Alg. 2
m	Initial number of start points to create a table.	Alg. 2
r	Number of tables.	Alg. 2
$l^i_{\max}$	Maximum length of a chain in table i.	Alg. 2
α	Expected (average) number of chains in a table.	(3.10), (3.12)
β	Expected (average) chain length.	(3.10), (3.12)
γ	Expected (average) amount of iterations made for a point during precomputation.	(3.15)
β_0	Expected (average) chain length before discarding chains with the same end point.	(3.8)
β_1	Expected (average) chain length of discarded chains.	(3.10), (3.12)
$p_1(s)$	Probability that a distinguished point is reached in $\leq s$ iterations.	(3.6), (3.7)
c	Expected number of tables to be checked in a search.	(3.17)
c_1	Expected fraction of tables that has to be checked before a success.	(2)

3.6 The Tradeoff Method Using Distinguished Points

For clarity we summarize the variables, definitions and functions that concern the distinguished points method and its complexity computations in Table 3.2.

For the distinguished points method, we need the following definition.

Definition 3.1 *Let $K \in \{0,1\}^k$ and $d \in \{1, \ldots, k-1\}$. Then K is a* distinguished point (DP) *of order d if there is an easily checked property which holds for K and which holds for 2^{k-d} different elements of $\{0,1\}^k$.*

Here 'easily checked' means a property that can be checked with a lower complexity than needed for a search in a table of about 2^d elements. An easily checked DP-property is for example having d bits with a fixed value on d specified places.

The new algorithm requires the choice of a DP-property of order d. Again r tables will be precomputed by choosing r different iteration functions f, but now each end point in each table will be a DP. For each iteration function m

different start points will be randomly chosen. For each start point a chain will be computed until a DP is encountered or until the chain has length $t+1$.

Only start points that iterate to a DP in less than t iterations will be stored (with the corresponding chain length), the others will be discarded. Hence, all chains that cycle are discarded. Moreover, if the same DP is an end point for different chains, then only the chain of maximal length will be stored. Hence, no two chains merge. Consequently there is no overlap in a single table. If the parameters d, t and m are chosen properly, the extra computation that has been done for discarded points is negligible.

Precomputation algorithm 2
Generate r tables with (SP, EP, l)-triples, sorted on end point.

```
1. Choose a DP-property of order $d$.
2. Choose $r$ different random functions $g_i$, $i = 1, \ldots, r$.
3. For $i = 1$ to $r$
        A. Choose $m$ random start points $SP_1^{(i)}, \ldots, SP_m^{(i)}$.
        B. For $j = 1$ to $m$, $l = 1$ to $t$
                a. Compute $f_i^l(SP_j^{(i)})$.
                b. If $f_i^l(SP_j^{(i)})$ is a DP, then
                        Store the triple $(SP_j^{(i)}, EP_j^{(i)} = f_i^l(SP_j^{(i)}), l)$ and take next $j$.
                c. If $l > t$ 'forget' $SP_j^{(i)}$ and take next $j$.
        C. Sort triples on end points.
           If several end points are the same, only store the triple
           with the largest $l$.
        D. Store maximum $l$ for each table: $l^i_{\max}$.
```

Tables constructed according to **Precomputation algorithm 2** have the following property.

Property 3.3 *For the search algorithm it holds that a table only has to be accessed when a DP is encountered during an iteration. Moreover, if the encountered DP is not in the table, then one will not find the target key by iterating further.*

Hence, then the current search can now skip the rest of this table.

Search algorithm 2
Given $C_i = E_K(P_i)$, $i = 1, \ldots, q$. Find K.

```
For $i = 1$ to $r$
    1. Look up $l^i_{\max}$.
    2. $Y = g_i(C_1, \ldots, C_q)$.
    3. For $j = 0$ to $l^i_{\max} - 1$
            A. If $Y$ is a DP then
                    a. If $Y$ in table $i$ and $j < l$, then
                            i. Compute predecessor $\tilde{K} = f_i^{l-1-j}(SP_l^{(j)})$.
            ii. If $C = E_{\tilde{K}}(P)$ then $K = \tilde{K}$: stop.
            iii. If $C \neq E_{\tilde{K}}(P)$, take next $i$.
        b. Else take next $i$.
    B. Set $Y = f(Y)$.
```

Note 3.2 **Precomputation algorithm 2** *and* **Search algorithm 2** *are given for the non-parallelized case. See also Note 3.1.*

Note 3.3 *It is also possible as suggested in [110] to discard chains that have a length that is lower than a certain lower bound. However, for practical parameters the consequences for the complexity and success probability are marginal.*

Experiments. The time-memory tradeoff using distinguished points has been implemented and tested on the block cipher RC2 for several key lengths and tradeoff parameters. RC2 has a block length of 64 bits. As key length a variable amount of bytes can be chosen. We have performed experiments for key lengths of 16, 24 and 32 bits. For the table generation, the choice of the g-functions and start points has been done pseudo-randomly, using a construction, based on the C `rand()` function.

We have chosen to do experiments with RC2 because it is for example used in the S/MIME-protocols of Netscape Communicator and Microsoft's Internet Explorer. In particular a 40-bit key length is supported. The block cipher is used in CBC-mode, but the first encrypted block is always the ASCII-equivalent of 'Content-'. It is encrypted using a random IV-vector, which is then sent along with the message. If the random generator is predictable or can be influenced, then an attacker might profit by making tables for the corresponding IV-values.

In the next section we shall start making an analysis of the tradeoff and compare our results to the experimental values.

3.7 Properties of the Iteration

Our analysis assumes Property 3.2.[1]

The precomputation creates chains with end points that are distinguished. In both the precomputation and processing algorithm we iterate from a (start) point until we reach a distinguished point or until we have reached a maximum number of iterations (either t in the precomputation or $l^i_{\max}$ in the search). Hence, to analyze the complexities of precomputation and search we study the probability that, when iterating from a random point, we reach a distinguished point in a certain number of steps. As a first step we will use these results to establish estimates for the expected chain length and the number of chains in a table, which we will do in the next section.

[1]In [13] Biryukov, Shamir and Wagner have found a one-way function that does not behave as a random mapping. They exploit this property to optimize their attack and pose a design criterion for similar constructions based on that.

The probability that we reach a distinguished point in $\leq s$ iterations is given by

$$\begin{aligned} p_1(s) &= \Pr(\rightarrow \text{DP in } \leq s \text{ iterations}) \\ &= 1 - \Pr(\not\rightarrow \text{DP in } \leq s \text{ iterations}) \\ &= 1 - (1 - 2^{-d})^s . \end{aligned} \tag{3.6}$$

According to [102, Prop.B2] the following holds

$$e^{-\frac{s}{2^d}} \cdot (1 - 2^{-2d} \cdot s) \leq (1 - 2^{-d})^s \leq e^{-\frac{s}{2^d}} ,$$

for $2^{-d} \leq 1$, $s \geq 1$ and $2^{-2d} \cdot s \ll 1$. Since in practical situation of the time-memory tradeoff this will always hold, we can make the following approximation, which we will need further on.

$$p_1(s) \approx 1 - e^{-\frac{s}{2^d}} . \tag{3.7}$$

Concerning the choice of the parameter t: we do not want to make any unnecessary computations in the precomputation. Hence t should be chosen such that the probability that a distinguished point is reached in less than t iterations is large. Hence $p_1(t)$ should be large. A reasonable choice is $t = 2^{d+3}$, since with the previous approximation

$$p_1(2^{d+3}) \approx 1 - e^{-2^3} \approx 1 .$$

On the other hand t should not be chosen too large since then there is a significant probability that we would continue to iterate when the chain is already cycling. But this is a less important restriction: we can easily upper bound it with $(1 - 2^{-d})^{t-1} \cdot \frac{t-1}{2^k} \approx e^{-8} \cdot \frac{t-1}{2^k} < \frac{t-1}{2^k}$. Hence, the expected number of starting points that will cycle is uppper bounded by $m \cdot \frac{t-1}{2^k}$. Practical values for m are much less than $\frac{2^k}{t-1}$. From this and our experiments it follows that $t = 2^{d+3}$ is a well-balanced value.

3.8 Expected Number of Chains and Chain Length

The questions we consider in his section are: consider one table which we make with variables t, m and d.

1. How many chains will be left in a table? We call this amount α.
2. What is the average length of a chain? We call this value β.

Note that it is easy to see that $\alpha \leq m$ and $\beta \leq t$. The parameters α and β play an important role in expressing the complexities and success probability of the tradeoff method as we will see in Sect. 3.9. In this section we will derive some relationships between α and β. Moreover, we will reduce the two questions to a single one: finding an expression for β_1 (defined in Sect. 3.8.1 below) or α or β

Table 3.3: Comparison of the experimental values obtained using the block cipher RC2 and the values computed with (3.12).

k	d	m	α	β	β_1	$\alpha \cdot \beta$	$D(\alpha, \beta_0)$ (3.12)
16	5	16	14	30	38	426	437
16	5	32	26	29	40	752	802
16	5	64	46	27	41	1250	1412
16	5	128	76	25	40	1921	2336
16	6	16	12	52	68	606	654
16	6	32	19	49	71	948	1097
16	6	64	31	44	69	1363	1725
16	6	128	48	39	68	1839	2616
20	6	32	30	62	88	1885	1833
20	6	64	57	61	90	3476	3457
20	6	128	106	58	89	6145	6090
20	6	256	187	55	87	10202	10018
20	7	32	27	115	162	3062	2820
20	7	64	47	108	165	5062	4922
20	7	128	79	99	164	7789	7538
20	7	256	127	88	160	11254	10672
24	7	128	121	124	183	14999	14962
24	7	256	230	121	182	27797	27684
24	7	512	423	116	180	49134	48790
24	7	1024	746	109	175	81368	80189
24	8	128	106	231	357	24516	24317
24	8	256	187	218	350	40656	40072
24	8	512	315	199	341	62732	61022
24	8	1024	507	178	328	90471	85712
24	9	128	79	396	664	31201	30273
24	9	256	127	353	643	44860	42679
24	9	512	198	307	618	61006	56167
24	9	1024	301	263	597	78883	69751

would result in expressions for α and β. We would like to propose this question as an interesting topic for further research. We give experimental values for α and β in the fourth and fifth column of Table 3.3.

In most practical cases one can efficiently and inexpensively experimentally compute α and β for given parameters k, d, m, and t. This will usually suffice to choose the optimal parameters based on the resources and goal of an attacker for a given application as we will motivate in Sect. 3.10.

3.8.1 First Relationship Between α and β

We consider the table of chains before sorting the triples (Step 3C. in **Precomputation algorithm 2**). It contains the chains that iterate to a distinguished point in $\leq t$ iterations. We argued in the previous section that this table contains about m chains, i.e., t is chosen such that this is expected to happen.

We call the average length of these chains β_0. Note that $\beta \leq \beta_0 \leq t$. The following holds for β_0:

$$\begin{aligned}
\beta_0 &= \sum_{l=1}^{t} l \cdot \Pr(\text{chain in table has length } l) \\
&= \sum_{l=1}^{t} l \cdot \{\prod_{i=1}^{l-1}(1-2^{-d}-\frac{i}{2^k}) - \prod_{i=1}^{l}(1-2^{-d}-\frac{i}{2^k})\} \\
&\approx \sum_{l=1}^{t} l \cdot \{(1-2^{-d}-\frac{l-1}{2^{k+1}})^{l-1} - (1-2^{-d}-\frac{l}{2^{k+1}})^{l}\} \\
&\approx \sum_{l=1}^{t} l \cdot \{(1-2^{-d}-\frac{t}{2^{k+2}})^{l-1} - (1-2^{-d}-\frac{t}{2^{k+2}})^{l}\} \\
&= f_1(2^{-d}+\frac{t}{2^{k+2}}, t)\,,
\end{aligned} \tag{3.8}$$

where

$$\begin{aligned}
f_1(x,y) &= \sum_{l=1}^{y} l \cdot ((1-x)^{l-1} - (1-x)^{l}) \\
&= 1 - y(1-x)^y + \sum_{l=1}^{y-1}(1-x)^l \\
&= 1 - y(1-x)^y + (1-x)\frac{1-(1-x)^{y-1}}{x} \\
&= 1 - y(1-x)^y + \frac{1}{x} - 1 - \frac{(1-x)^y}{x} \\
&= \frac{1}{x} - (\frac{1}{x}+y)\cdot(1-x)^y\,.
\end{aligned} \tag{3.9}$$

Table 3.4 compares the values for β_0, computed according to (3.8), with the values of the experiments we have done for several parameters. This shows that the approximations, made in the derivation were justified. Furthermore, the average length of a chain (that does not cycle) is expected to be 2^d. Since our algorithm stops after t iterations, β_0 is slightly smaller than 2^d: this also follows from (3.8).

Table 3.4: Expected chain length β_0 before throwing away chains.

k	d	β_0 experimental	β_0 according to (3.8)
16	5	31.1	31.0
16	6	57.2	56.8
20	6	63.6	63.3
20	7	123.7	123.8
24	7	127.3	127.4
24	8	253.6	253.3
24	9	498.4	495.3

The next step in constructing the (final) table is throwing away chains for which there is a longer chain in the table with the same end point. We call the average length of the thrown away chains β_1. In Table 3.3 the experimental values of β_1 are summarized. Note that (contrary to β_0) the value of β_1 also depends on m, not just on k and d. Now it is possible to express the average length of the chains in a table as

$$\beta = \frac{\beta_0 \cdot m - \beta_1 \cdot (m - \alpha)}{\alpha}. \tag{3.10}$$

This is the first relationship between α and β. In the next section we shall derive another one. Hence, with an expression for β_1 and these two relationships it is possible to derive values of α and β, given the parameters k, t and m of the tradeoff.

3.8.2 Second Relationship Between α and β

The second relationship between α and β makes use of (3.2). A table constructed according to our method covers $\alpha\beta_0$ points before sorting the triples. We estimate that in accordance with the Hellman method $D(\alpha, \beta_0)$ of these points are different. Furthermore we assume that most of the points covered by chains that are discarded because of same end points are still covered in the final table. (After all this was the purpose of throwing them away in the first place.) The construction implies that no point is covered twice in the final table.

The expected number of points in the final table can be estimated as approximately $\alpha\beta$ under the assumption that the distributions of values of α and β are independent. This independence increases if the values of m and k increase. Hence,

$$\text{E}(\#\text{ points covered by a table}) \approx \alpha\beta\,. \tag{3.11}$$

The final table covers approximately $\alpha\beta$ points. Using (3.2) this quantity can now be estimated as

$$\alpha\beta \approx D(\alpha, \beta_0)\,. \tag{3.12}$$

This is the second relationship between α and β. In Table 3.3 we provide experimental justification for (3.12). Note that when $\frac{\alpha\beta_0^2}{2^k}$ exceeds the value 1 the approximation is inaccurate. This is in accordance with the conditions of (3.2).

If an expression for β_1 is derived then with (3.10) and (3.12) it is possible to write down explicit expressions for α and β, depending on m and d (and k of course). They do not depend on t as long as t is chosen sufficiently large: $t \geq 2^{d+3}$.

3.9 Complexity and Success Probability

In this section we will derive estimates for the precomputation, memory and processing complexity and furthermore for the success probability. These are summarized in Table 2.2.

3.9.1 Memory Complexity

Storage is needed for r tables. Each table contains about α chains. These are in the form of triples or pairs and other well-known tricks can be done to save on needed space, such as choosing start points that need less storage. The storage for the maximal chain length per table $l^i_{\max}$ can be neglected. Hence, memory for αr table entries is needed.

3.9.2 Success Probability

We assume (3.11): a table is expected to contain $\alpha\beta$ points. All points in the same table are different, but there can be overlap between different tables, i.e., a point can be covered by several tables. The success probability, the probability that some key K is at least in one of the tables, is given by

$$\begin{aligned}
\Pr_{\text{success}} &= 1 - \prod_{i=1}^{r} \Pr(K \text{ not in table } i) \\
&\approx 1 - \prod_{i=1}^{r} (1 - \frac{\alpha\beta}{2^k}) \\
&= 1 - (1 - \frac{\alpha\beta}{2^k})^r \qquad (3.13) \\
&\approx 1 - e^{-\frac{r\alpha\beta}{2^k}}\,. \qquad (3.14)
\end{aligned}$$

This is in accordance with our experimental results.

3.9.3 Precomputation Complexity

In the precomputation for each point iterations are performed either until a distinguished point is reached or t iterations have been made. If a distinguished point is reached, then on the average β_0 iterations were computed. If a distinguished point is not reached then t iterations were computed. If we define the expected iterations for one point as γ then this can be estimated accordingly as

$$\gamma \approx t(1 - p_1(t)) + \beta_0 p_1(t)\,. \tag{3.15}$$

Note that $\gamma \approx \beta_0$ for $t \geq 2^{d+3}$. Iterations are made on rm start points, hence the precomputation complexity $\mathrm{C}_{\mathrm{prec}}$ can be estimated by

$$\mathrm{C}_{\mathrm{prec}} \approx rm\gamma\,. \tag{3.16}$$

3.9.4 Processing Complexity

We only consider the expected amount of iterations in the search, since the amount of table lookups is $\leq r$ and thus negligible. In the search we start with one point. For a table iterations are done on this point either until a distinguished point is reached or until l^i_{max} iterations have been performed. For a choice of $t \approx 2^{d+3}$ it will hold that $l^i_{\mathrm{max}} \approx t$, hence the expected amount of iterations per table is about γ. Not all r tables have to be checked: if the key is found in a table the algorithm will stop; an unsuccessful search will make iterations for all r tables. The expected fraction of tables that are checked is some variable c between 0 and 1, that depends on the success probability. It is given by

$$\begin{aligned} c &\approx c_1 \mathrm{Pr}_{\mathrm{success}} + (1 - \mathrm{Pr}_{\mathrm{success}}) \\ &= 1 - (1 - c_1)\mathrm{Pr}_{\mathrm{success}}\,, \end{aligned} \tag{3.17}$$

where c_1 is the expected fraction of tables that has to be checked before a success. We assume (3.11). Then the following expression can be derived:

$$\begin{aligned} c_1 &= \frac{1}{r}\sum_{l=1}^{r} l \cdot \mathrm{Pr}(\text{key in table } l \text{ and not in table } s < l) \\ &= \frac{1}{r}\sum_{l=1}^{r} l \cdot (1 - \frac{\alpha\beta}{2^k})^{l-1} \cdot \frac{\alpha\beta}{2^k} \\ &= \frac{\alpha\beta}{r \cdot 2^k} \cdot \sum_{l=1}^{r} l \cdot (1 - \frac{\alpha\beta}{2^k})^{l-1} \\ &= \frac{\alpha\beta}{r \cdot 2^k} \cdot f_2(\frac{\alpha\beta}{2^k}, r)\,, \end{aligned} \tag{3.18}$$

where

$$\begin{aligned} f_2(x,y) &= \sum_{l=1}^{y} l \cdot (1-x)^{l-1} \\ &= \frac{(1-x)^{y+1}(xy+1)-1+x}{(x-1)x^2} \,. \end{aligned} \tag{3.19}$$

Hence, the average processing complexity for a successful search would be

$$\begin{aligned} \mathrm{C_{proc}}(\text{successful search}) &= c_1 r\gamma \\ &\approx \frac{\alpha\beta}{2^k} \cdot f_2(\frac{\alpha\beta}{2^k}, r) \cdot \beta_0 \,, \end{aligned} \tag{3.20}$$

where we have used that $\gamma \approx \beta_0$ for $t \leq 2^{d+3}$, which was mentioned in the previous section and holds for the range of parameters used in the experiments. We verified (3.20) experimentally and summarize the results in Table 3.5. This leaves the following estimate for the processing complexity.

$$\mathrm{C_{proc}} \approx cr\gamma < r\gamma \,. \tag{3.21}$$

3.10 Choosing Optimal Tradeoff Parameters

Hellman concluded in [63] that an optimal choice of parameters for his method would be to take m_h, t_h and r_h all approximately equal to $2^{k/3}$. This was derived under the assumption of a precomputation complexity equal to exhaustive key search and the assumption that investment in memory and processing complexity should be about equal. Kusuda and Matsumoto showed in [87] that for the Hellman method the success probility can be optimized based on the cost of memory and processing complexity.

In order to implement the tradeoff, the following parameters have to be set: the order of the distinguished points d, the number of starting points in a table m, the number of tables r, and the maximal chain length t. In Sect. 3.7 we already motivated to choose $t = 2^{d+3}$.

The following items should be taken into account when choosing optimal parameters.

- The availability or investment in memory.
- The available computing power, for online processing as well as precomputation.
- The required or possible succes probability.

We have derived expressions for the memory, precomputation and processing complexity, as well as for the success probability. All these parameters depend on the values of α and β. In practice one can compute a few test tables for

Table 3.5: Processing complexity: on-line computation required to recover a key using the Distinguished Point method. Comparison of experimental values (obtained using RC2) and values computed with (3.20).

k	d	m	C_{proc}(successful search) experimental	C_{proc}(successful search) according to (3.20)
16	5	16	4218	4714
16	5	32	2285	2678
16	5	64	1732	1643
16	5	128	1079	1064
16	6	16	6126	6147
16	6	32	4194	3990
16	6	64	3584	2739
16	6	128	2783	2032
20	6	32	19845	19527
20	6	64	16460	16402
20	6	128	10188	10635
20	6	256	7810	6503
20	7	32	36360	33704
20	7	64	25935	24478
20	7	128	18830	16622
20	7	256	12379	11561
24	7	128	26869	33334
24	7	256	42079	38922
24	7	512	36736	34839
24	7	1024	27054	25123
24	8	128	81592	76396
24	8	256	67482	74306
24	8	512	56264	60842
24	8	1024	44520	45879
24	9	128	176463	152364
24	9	256	148447	141513
24	9	512	132195	121335
24	9	1024	105957	101121

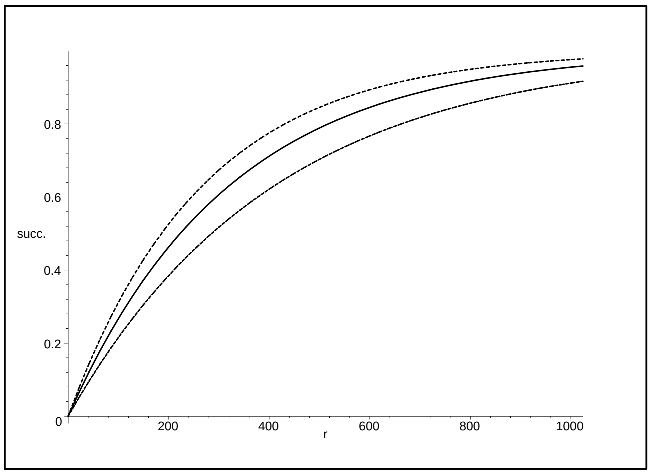

Figure 3.4: Success probability depending on the number of tables. Solid line: Hellman-method ($m_h = t_h = 2^8$). Other lines: Distinguished Point method (upper: $m = 2^9$, $d = 2^8$ and lower $m = d = 2^8$).

several choices of parameters and experimentally evaluate α and β. Taking into account which complexities should be optimized, from these values the optimal values for the parameters can be derived. The rough choices of parameters to make the test tables are mostly clear; for example optimization of the success probability under the assumption of a prexomputation equal to exhaustive key search and an equal investment in memory and online processing complexity would suggest values of $d \approx m \approx r \approx \frac{1}{3}k$ in accordance with Hellman's results.

3.11 Comparison with Hellman's Method

In Figure 3.4 we give the success probability, depending on the number of tables r for $k = 2^{24}$. The solid line is the theoretical result for Hellman's method with parameters $m_h = t_h = 2^8$ (the optimal settings [63]), the other two lines depict our algorithm with parameters $m = 2^9$ and $d = 2^8$ for the upper line and $m = d = 2^8$ for the lower one. The processing and memory complexity is the same in all three cases. But the precomputation complexity of the implementation with $m = 2^9$ would be two times larger than that of either one of the other two. We conclude that the optimal success probability of the Hellman method can be achieved for slightly more precomputation.

In the DP-method for each table the list of end points has to be checked for membership in the table at most once. Taking into consideration false alarms, in the Hellman method this number has an expected value of t_h. Less memory accesses are for example advantageous in a distributed key search over a LAN or over the Internet or for search machines using dedicated hardware.

Distributed key search. Assume that the precomputation is feasible, for example with a distributed key search over the Internet or a LAN or with the temporarily borrowed dedicated search machine of a friendly intelligence agency, and that the tables are stored somewhere safely. A possible way to speed up the actual attack is a distributed key search over the Internet or a LAN, where the tables are searched through in parallel. Hence, one central server distributes the tables over many clients. We examine two different methods for this.

In the first method the central server sends whole tables and corresponding g-functions (and of course the known ciphertext(s)). Each client performs one of the previously described search algorithms with his part of the tables.

This has the following disadvantages.

- In practice the central server has to distribute large amounts of data. This takes time.
- The tables that have to be sent contain sensitive information that should not fall in the wrong hands, and hence need to be sent encrypted,[2] especially when sent over the Internet. The needed encryption and decryption would again slow down the search considerably.

As alternative for this method there is a second one which only can be done with the DP-method and which does not have the disadvantages described above. In this method the central server distributes the tables by sending only the g-functions with corresponding $l_{\max}$. Now each client iterates f_i and there are the following possibilities.

- A DP is encountered after j iterations, then the client sends j, i and $f_i^j(K)$ to the central server, which checks whether the DP is an end point in the corresponding table and, if so, checks whether the target key is in the corresponding chain.
- If no DP is produced within $l^i_{\max}$ iterations the n the target key is not in table i: the table does not need to be searched.

The second method has as advantage that no table needs to be sent at all. If we assume that $2^d \approx m \approx 2^{k/3}$ the workload that the central server has to do, caused by checking the received DP's, can be loosely upper bounded by

$$\begin{aligned} r \cdot \Pr(\text{DP in table}) \cdot \mathrm{E}(\#\text{ iterations to find predecessor}) &< r \cdot \frac{\alpha}{2^{k-d}} \cdot \frac{1}{2}\beta \\ &< 2^d \cdot \frac{2^d \cdot m \cdot t}{2^{k+1}} \\ &\approx 2^d \cdot 4\,. \end{aligned}$$

Hence, in practical cases this workload can be neglected. Thus we have identified a situation in which the DP-method should be preferred over the Hellman method.

[2]In this case the encryption can be done provably secure. The security goal is to prevent an attacker to gain information on how to mount an attack on the target cipher. The solution to that is to encrypt the table that are to be sent with the target cipher.

Hardware. A similar situation exists for dedicated hardware. A key search machine that performs a time memory tradeoff search consists of three components:

- Memory for storing the tables. This is typically ROM.
- The encryption chips that perform the iterations and RAM memory.
- A controller that connects the two other units.

For a key search the controller loads the tables from ROM and stores them in the RAM. Each chip then proceeds by searching the tables in parallel. For the Hellman search the amount of memory accesses to the RAM would be about t_h for each table. In the case of the distinguished points method this would be decreased to at most one access per table.

3.12 Conclusions

Hellman's time-memory tradeoff is a brute force attack that requires lower processing complexity than exhaustive key search and has lower memory requirements than table precomputation. It can be used to mount attacks on one-way functions, cryptographic hash functions, stream ciphers and block ciphers.

The contribution of this chapter is the description of a new variation on Hellman's tradeoff. Its main advantage is a significant decrease of required memory accesses. This is advantageous in many situations: distributed key search as well as for the construction of key search machines using dedicated hardware.

We have analyzed this method and given expressions for its memory, precomputation and processing complexity and for its success probability, depending on the parameters of the target cipher and parameters of the search method, which is another contribution of this chapter. These results make it practically possible to choose the optimal parameters for a key search, based on available resources and specific requirements.

Of theoretical interest remains the open problem to deduce expressions for the parameters α and β. Our results have reduced the solution to this problem to finding only one parameter: either α, β or β_1. We would like to encourage further research on this topic.

Chapter 4

Cryptanalysis of reduced round versions of RC5 and RC6

4.1 Introduction

In this chapter we evaluate the resistance of the block cipher RC5 against linear cryptanalysis. Standard techniques from linear cryptanalysis to analyze this cipher are not applicable due to certain properties of the structure of the cipher. This is illustrated by the publication of a linear attack by Kaliski and Yin in [72]; after a few years Selçuk discovered that this attack did not work due to the special properties of the cipher [125]. The probabilities of linear approximations for the individual components of the cipher are highly dependent on each other and for the key dependent components they are highly key dependent as well. Hence, none of the Assumptions 2.4, 2.5 or 2.6 holds. Under these constraints we perform an analysis to find a good linear approximation. In order to do this we introduce the concept of a non-uniformity function and modify Vaudenay's statistical attack model slightly. Furthermore, to mount an attack we construct a method to perform a last round trick in order to exploit this approximation. This results in a known plaintext attack that can break RC5-32 (block size 64) with 10 rounds and RC5-64 (block size 128) with 15 rounds. These attacks break RC5 in the sense of Definition 2.6 and 2.7. Besides the analysis we have implemented the attack and the estimates and experimental results are in agreement. At this moment these are the best known plaintext attacks on RC5, which have negligible storage requirements and do not make any assumption on the plaintext distribution. These results were published in [23] by the author, Preneel and Vandewalle. We also analyze and discuss the consequences of our attack for former AES candidate RC6 and compare our results to more recently published related work.

The remainder of this chapter is organized as follows. In Sect. 4.2 we give an overview of the history of RC5 and the results concerning security analysis. We show the merits and limitations of linear cryptanalysis techniques when applied to RC5 in Sect. 4.3. In Sect. 4.4 we describe our attack on RC5 and we give experimental results on RC5-32 and RC5-64. We discuss the consequences for RC6 in Sect. 4.5, compare our results to an attack based on the recent results on RC6 in Sect. 4.6 and conclude in Sect. 4.7.

4.2 History and Results for RC5

The iterated block cipher RC5 was introduced by Rivest in [116, 117]. It has been patented by RSA in [118, 119] and was published by the IETF in RFC 2040 [5]. It was also used as basis for the former AES candidate RC6. It has a variable number of rounds denoted with r and a key size of b bytes. The design is word-oriented for word sizes $w = 32, 64$ and the block size is $2w$. The choice of parameters is usually denoted by RC5-w, RC5-w/r, or RC5-$w/r/b$. Currently RC5-32/16/16 is recommended to give sufficient resistance against linear and differential attacks [73]. However, when it was introduced in [117] the “nominal choices” for parameters were suggested to be RC5-32/12/16 and RC5-64/16/16. Hence, this is an increase of 4 rounds in 3 years.

RC5 has been analyzed intensively. Main reasons for this are its transparent structure and the reputation of its inventor (the R of RSA). RC5 has a simple structure such that its cryptographic strength would be rapidly determined. A new feature was its use of data-dependent rotations, which should increase the resistance against linear and differential analysis. The heavy use of data dependent rotations in ciphers is actually one of the items that has been patented in [118, 119]. We give an overview of the currently obtained results. The results refer to RC5 with block size 32, unless explicitly stated otherwise.

A first evaluation was published by Kaliski and Yin in [72], where its strength is analyzed against linear and differential cryptanalysis. They presented a differential attack that would require 2^{62} plaintext pairs for 12 rounds and a linear attack that would require 2^{57} plaintexts for 6 rounds. They conjectured that their linear analysis is optimal, hence LC would not improve on exhaustive key search for more than 6 rounds. In [81] Knudsen and Meier improved the differential attack by a factor up to 512. They also presented an attack on the RC5 with 64-bit words that required 2^{123} plaintexts for 24 rounds.

Biryukov and Kushilevitz developed some advanced techniques to mount a differential attack on RC5 in [11]. They presented an attack on 12 rounds that requires 2^{44} chosen plaintexts. Although the attack makes use of very sophisticated techniques, according to its inventors the required amount of chosen plaintexts for the attack on 12 rounds might be reduced to 2^{38} [11]. Furthermore, they estimate that an attack on RC5-32/12/16 is feasible and will require less than 2^{63} chosen plaintexts. The good results in differential cryptanalysis were further exploited in [12] by Biryukov and Kushilevitz to mount known ciphertext attacks on reduced round versions of RC5, that they have practically implemented. They assumed that the encrypted plaintext was English text, coded in ASCII. This assumption caused the exploitable differences to appear more often than for random plaintexts.

Linear analysis has had no impressive results until [125], where Selçuk implemented the linear attack of Kaliski and Yin and found that it did not work as expected. This was caused by the fact that the probability of the linear relation that it should exploit as well as the success of the method to exploit it were estimated under assumptions that did not hold. Selçk analyzes this effect in [125]. This was the inspiration of our work on RC5, that we describe in this chapter.

Currently the best published chosen plaintext attack is by Biryukov and Kushilevitz [11]. We summarize the complexities for different round versions of RC5-32 in the second column of Table 4.1. As this is a differential attack, it yields a known plaintext attack for a larger amount of known plaintexts according to Theorem 2.3. We give the estimated required amount of known plaintexts in the third column of Table 4.1. The known plaintext attack however needs a storage capacity for all the required plaintexts, i.e., the attack can not be mounted in a way that the attacker obtains and analyzes the plaintexts one by one. We give the estimated required storage capacity of the known plaintext-version in the fourth column. For example, to mount this attack for 4 rounds one would need to store 2^{36} plaintexts with corresponding ciphertexts, which is about 1 GByte. The attack described in this chapter requires a negligible storage capacity. We give the required amount of plaintexts in the fifth column of Table 4.1.

Table 4.1: Comparison of the complexities of the attack on RC5-32 of Biryukov and Kushilevitz and the attack as presented in this chapter. The table contains the binary logarithm (lg) of the number of known plaintexts, chosen plaintexts and memory requirements.

	Biryukov/Kushilevitz			Our attack	
Rounds	Chosen texts	Known texts	Memory (# texts)	Known texts	Memory (# texts)
4	7	36	36	28	negligible
6	16	40.5	40.5	40	negligible
8	28	46.5	46.5	52	negligible
10	36	50.5	50.5	64	negligible
12	44	54.5	54.5	–	–

Our attack is a linear attack, whose high success rate is based on a large linear hull-effect [105]. To our knowledge it is the first time that this effect has significant consequences in the evaluation of the resistance of a cipher against linear attacks. Furthermore we use techniques closely related to multiple linear approximations to set up a practical attack.

4.3 RC5 and Linear Cryptanalysis

RC5 is defined as follows. First $2r+2$ round keys $S_i \in \{0,1\}^w$, $i = 0, \ldots, 2r+1$, are derived from the user key[1]. If $(L_0, R_0) \in \{0,1\}^w \times \{0,1\}^w$ is the plaintext, then the ciphertext (L_{2r+1}, R_{2r+1}) is computed iteratively with:

$$\begin{aligned} L_1 &= L_0 + S_0 && (4.1)\\ R_1 &= R_0 + S_1 && (4.2)\\ U_i &= L_i \oplus R_i && (4.3)\\ V_i &= U_i \lll R_i && (4.4)\\ R_{i+1} &= V_i + S_{i+1} && (4.5)\\ L_{i+1} &= R_i\,, && (4.6) \end{aligned}$$

for $i = 1, \ldots, 2r$. Here $+$ denotes addition modulo 2^w and $x \lll y$ rotation of w-bit word x to the left over y mod w places. The computation of (L_{i+2}, R_{i+2}) from (L_i, R_i) with i odd is considered as one round of RC5. In Fig. 4.1 a graphical representation of one round is given.

4.3.1 Linear Approximations

We shall consider the following linear approximations for xor, data-dependent rotation and addition. We only look at approximations that consider one bit of each term of the equation. The binary vector that has a 1 on position i and is 0 everywhere else, will be denoted with e_i. Let $A = B \oplus C$. Then:

$$e_i \cdot A = e_i \cdot B \oplus e_i \cdot C, \quad \delta = 2^{-1}\,, \qquad (4.7)$$

for $i \in \{0, \ldots, w-1\}$. Let $D = E \lll F$. Then:

$$e_i \cdot D = e_j \cdot E \oplus e_k \cdot F \oplus e_k \cdot (i-j), \quad \delta = 2^{-\lg w - 1}\,, \qquad (4.8)$$

for $i, j \in \{0, \ldots, w-1\}$ and $k \in \{0, \ldots, \lg w - 1\}$. Here lg denotes the logarithm to the base 2. (Note that we have abused the "$\cdot$"-notation slightly to denote the k-th bit in the binary representation of $i-j$.) If one is only interested in the bias of (4.8), one can leave out the term $e_k \cdot (i-j)$ or even $e_k \cdot F$, see for example [73]. We use (4.7) and (4.8) to pass the xor and rotation in RC5 as follows. Let $j, k \in \{0, \ldots, \lg w - 1\}$. Then

$$\begin{aligned} e_j \cdot U_i &= e_j \cdot L_i \oplus e_j \cdot R_i, \quad \delta = 2^{-1} && (4.9)\\ e_k \cdot V_i &= e_j \cdot U_i \oplus e_j \cdot R_i \oplus e_j \cdot (k-j), \quad \delta = 2^{-\lg w - 1}\,. && (4.10) \end{aligned}$$

Chaining these two yields:

$$e_k \cdot V_i = e_j \cdot L_i \oplus e_j \cdot (k-j), \quad \delta = 2^{-\lg w - 1}\,. \qquad (4.11)$$

[1]As the key schedule of RC5 has no relevance for our analysis we refer to [117] for a description. However for our experiments we have used the key schedule.

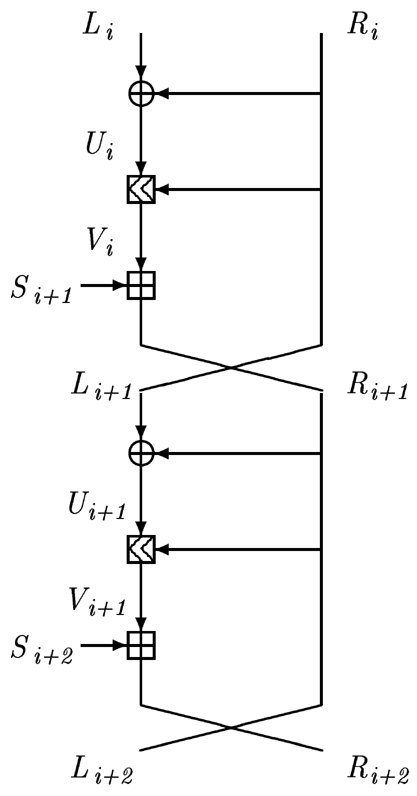

Figure 4.1: One round of RC5.

Finally, let $G = H + S$, where S is fixed. Then:

$$e_0 \cdot G = e_0 \cdot H \oplus e_0 \cdot S, \quad \delta = 2^{-1} \tag{4.12}$$

$$e_i \cdot G = e_i \cdot H \oplus e_i \cdot S, \quad \delta = 2^{-1} - 2^{-i} S[i]\,, \tag{4.13}$$

for $i \in \{1, \ldots, w-1\}$. Here $S[x] = S \bmod 2^{x+1}$, hence the x LSBs of S. Hence, depending on the key, the bias of (4.13) can vary between 0 and $\frac{1}{2}$. On the average it is $\frac{1}{4}$.

4.3.2 Key Dependency and Piling-Up

Because of the fact that (4.12) has a bigger bias than (4.13), 'traditionally' approximations on the LSB have been considered to be most useful for a linear attack (see [72, 73]). Using Approximations (4.7), (4.8) and (4.12) one can derive the following iterative approximation for one round of RC5:

$$e_0 \cdot L_i \oplus e_0 \cdot S_{i+1} = e_0 \cdot L_{i+2} \ (i \geq 1), \quad \delta = 2^{-\lg w - 1}\,. \tag{4.14}$$

This approximation can be chained to l rounds as follows.

$$e_0 \cdot L_i \oplus \bigoplus_{j=0}^{l-1} e_0 \cdot S_{i+1+2j} = e_0 \cdot L_{i+2l} \ (i \geq 1), \ \ \delta = ?\,. \qquad (4.15)$$

According to the Piling Up Lemma (2.1) this approximation would have deviation $\delta = 2^{l-1}2^{(-\lg w-1)l} = 2^{-l\lg w-1}$ if the chained approximations would be independent. This is however not the case.

To illustrate this consider (4.15) for $l = 2$. The deviation of this approximation depends on the $\lg w$ least significant bits of S_{i+2}. This can be seen as follows. The probability that (4.14), i.e., the approximation over the first round, holds depends on the value of $R_i \bmod w$. If $R_i \bmod w = 0$ it always holds, otherwise it holds with probability $\frac{1}{2}$. Hence, the deviation of (4.14) is computed under the assumption that every value of $R_i \bmod w$ is equally likely. If $R_i \bmod w = 0$ then the $\lg w$ least significant bits of L_{i+1} are known. Now consider the approximation for each possible value of R_{i+1} separately. If $R_{i+1} \bmod w \in \{0, \ldots, \lg w - 1, w - \lg w + 1, \ldots, w - 1\}$ then, depending on the value of $S_{i+2} \bmod w$, part of the $\lg w$ least significant bits of R_{i+2} can be computed. It turns out that the values of R_{i+3} are not equally likely. Hence, the Piling Up Lemma cannot be applied.

To illustrate this effect we have computed the bias of the two round approximation for RC5-32 for every value of $S_{i+2} \bmod w$. These results are given in Table 4.2. It can be seen that the bias can vary significantly between $\approx 2^{-10}$ and $\approx 2^{-16}$. On the other hand the average value is $\frac{1}{32}\sum_{x=0}^{31} |\delta_x| \approx 2^{-11}$. Since $\delta_x \neq 0$ for all x, the average amount of expected plaintexts needed based on Table 4.2 is given by $\frac{1}{32}\sum_{x=0}^{31} \delta_x^{-2} \approx 2^{22}$. Both are in accordance with the Piling Up Lemma.

Note further that for two values of $S_{i+2} \bmod w$ the deviation is negative. For those values this means that if one would mount a basic linear attack (Algorithm 1 in [96]) on four rounds using (4.15) to find $S_0 \oplus S_2 \oplus S_4$ based on the Piling Up Lemma, most likely one would find $S_0 \oplus S_2 \oplus S_4 \oplus 1$, given enough texts.

We can conclude that although the deviation can vary significantly, the Piling Up Lemma gives a good estimate for the average bias for RC5. We will show that this estimate can be used to compute the expected average success rate of a linear attack that does not use the sign of the deviation, but only its absolute value: the bias.

4.3.3 Key Dependency and Linear Hulls

The concept (approximate) *linear hull* was introduced by Nyberg [105]. We will use the term linear hull in the way it was used in [27]. A linear hull is the set of all chains of linear equations over (a part of) the cipher that produce the same linear equation. The existence of this effect for RC5 was first noticed in [27, 121] where also some preliminary work in determining the linear hull effects for RC6 (and some simplified versions) can be found.

Table 4.2: The deviation δ_x of Approximation (4.15) with $l = 2$ for RC5-32, depending on $x = S_{i+2} \bmod w$.

$x = S_{i+2} \bmod w$	$10^3 \delta_x$	$\lg \lvert\delta_x\rvert$	$x = S_{i+2} \bmod w$	$10^3 \delta_x$	$\lg \lvert\delta_x\rvert$
00	0.930786	-10.07	10	0.228882	-12.09
01	0.991821	- 9.98	11	0.656128	-10.57
02	0.473022	-11.05	12	0.015259	-16
03	0.564575	-10.79	13	-0.015259	-16
04	0.595093	-10.71	14	0.137329	-12.83
05	0.686646	-10.51	15	0.534058	-10.87
06	0.473022	-11.05	16	0.503540	-10.96
07	0.503540	-10.96	17	0.717163	-10.45
08	0.595093	-10.71	18	0.625610	-10.64
09	0.564575	-10.79	19	0.778198	-10.33
0a	0.289917	-11.75	1a	0.747681	-10.39
0b	0.381470	-11.36	1b	0.839233	-10.22
0c	0.411987	-11.25	1c	0.381470	-11.36
0d	0.289917	-11.75	1d	0.381470	-11.36
0e	0.106812	-13.19	1e	0.076294	-13.68
0f	0.228882	-12.09	1f	-0.045776	-14.42

The following linear hull for an approximation of two rounds (i till $i + 3$) of RC5 can be noticed. Let $j, l \in \{0, \dots, w-1\}$. Using (4.7), (4.8), (4.12) and (4.13) for j the following $\lg w$ approximations for two rounds can be derived.[2]

$$e_j \cdot L_i \oplus e_j \cdot (k - j) \oplus e_k \cdot S_{i+1} = e_k \cdot L_{i+2}, \quad \delta = \delta_{j,k}\,, \tag{4.16}$$

for $k = 0, \dots, \lg w - 1$. Likewise for the next two rounds and l also $\lg w$ approximations can be derived:

$$e_k \cdot L_{i+2} \oplus e_k \cdot (l - k) \oplus e_l \cdot S_{i+3} = e_l \cdot L_{i+4}, \quad \delta = \delta_{k,l}\,, \tag{4.17}$$

for $k = 0, \dots, \lg w - 1$. Hence, one can chain $\lg w$ pairs of (4.16) and (4.17) to obtain the two round approximation

$$e_j \cdot L_i \oplus e_k \cdot S_{i+1} \oplus e_l \cdot S_{i+3} \oplus e_j \cdot (k - j) \oplus e_k \cdot (l - k) = e_l \cdot L_{i+4}, \quad \delta = \delta_{j,k,l}\,. \tag{4.18}$$

Neglecting the key-dependency of chaining and using the Piling Up Lemma one

[2]Note that in Equation (4.16) and others $\delta_{j,k}$ is not the Kronecker delta, but a variable δ with indices j and k.

gets for this bias

$$\delta_{j,k,l} = \begin{cases} 2^{-2\lg w-1} & k=0, l=0 \\ 2^{-2\lg w-1}(1-2^{-k+1}S_{i+1}[k-1]) & k\neq 0, l=0 \\ 2^{-2\lg w-1}(1-2^{-l+1}S_{i+3}[l-1]) & k=0, l\neq 0 \\ 2^{-2\lg w-1}(1-2^{-k+1}S_{i+1}[k-1])(1-2^{-l+1}S_{i+3}[l-1]) & k\neq 0, l\neq 0\,. \end{cases} \tag{4.19}$$

We note two things about (4.18) and its deviation given in (4.19). Firstly it is clear from (4.19) that the deviation is key dependent, i.e., dependent on the $\lg w - 1$ LSBs of S_{i+1} and S_{i+3}. But we will show that because of the linear hull effect, this key dependency is negligible. Secondly, for each choice of the triple j, k, l the term $C_{j,k,l} = e_k \cdot S_{i+1} \oplus e_l \cdot S_{i+3} \oplus e_j \cdot (k-j) \oplus e_k \cdot (l-k)$ is constant, either 0 or 1. This constant actually determines the sign of the deviation of the following approximation, which can be derived from (4.18) by leaving out $C_{j,k,l}$.

$$e_j \cdot L_i = e_l \cdot L_{i+4},\ \delta = \tilde{\delta}_{j,l}\,. \tag{4.20}$$

For the deviation the following holds:

$$\tilde{\delta}_{j,l} = \sum_{k\in V_{j,l}} \delta_{j,k,l} - \sum_{k\notin V_{j,l}} \delta_{j,k,l}\,, \tag{4.21}$$

where $V_{j,l} = \{k \in \{0,\dots,\lg w-1\} | C_{j,k,l} = 0\}$.

One can extend approximation (4.20) to hold for r subsequent rounds. In this way one gets the following approximation for r rounds.

$$e_j \cdot L_i = e_l \cdot L_{i+2r},\ \delta = \tilde{\delta}_{j,l}\,, \tag{4.22}$$

where $i, j \in \{0,\dots,w-1\}$. The deviation can be computed/approximated by considering the parts of the different chains, for the k-th round in the chain given by

$$e_{j_k} \cdot L_{i+2k} = e_{l_k} \cdot L_{i+2k+2}\,, \tag{4.23}$$

where $k \in \{0,\dots,r-1\}$ and

$$j_k \begin{cases} = j & \text{if } k=0 \\ \in \{0,\dots,\lg w-1\} & \text{if } k\neq 0 \end{cases}$$

$$l_k \begin{cases} \in \{0,\dots,\lg w-1\} & \text{if } k\neq r-1 \\ = l & \text{if } k=r-1 \end{cases}$$

and

$$j_k = l_{k-1} \text{ for } k \in \{1,\dots,r-1\}\,.$$

It follows that Equation (4.22) is a linear hull that consists of $(\lg w)^{r-1}$ different chains, namely for all of the choices of j_k, l_k. When we take the average bias of $\frac{1}{4}$ for the approximation of addition on non-LSBs, we get for the bias of (4.22):

$$|\tilde{\delta}_{j,l}| = \begin{cases} c(r) & \text{if } l = 0 \\ \frac{1}{2}c(r) & \text{if } l \neq 0\,, \end{cases} \tag{4.24}$$

where $c(r)$ can be estimated as

$$\begin{aligned} c(r) &\approx \gamma \left(\sum_{k=0}^{r-1} \binom{r-1}{k} (\lg w - 1)^k 2^{(-\lg w - 1)r + r - 1 - k} \right) \\ &= \gamma \left(2^{(-\lg w - 1)r} \sum_{k=0}^{r-1} \binom{r-1}{k} (\lg w - 1)^k 2^{r-1-k} \right) \\ &= \gamma \left(2^{(-\lg w - 1)r} (\lg w - 1 + 2)^{r-1} \right) \\ &= \frac{\gamma}{2w} \left(\frac{\lg w + 1}{2w} \right)^{r-1}\,, \end{aligned} \tag{4.25}$$

where γ is a factor that accounts for the effect of different chains that cancel each others contribution. The increment factor $c^+(r)$ of the bias is expressed as

$$c(r+1) = c^+(r) \cdot c(r), \text{ for } r = 2, \ldots \tag{4.26}$$

Hence, if a linear attack can be mounted on RC5 with r rounds using x known plaintexts, then this attack will have the same success probability if adapted to RC5 with $(r{+}1)$ rounds if $(c^+(r))^{-2}x$ known plaintexts are available. From (4.25) and (4.26) follows $c^+ = \frac{\lg w + 1}{2w}$. Now, for RC5-32 we have $c^+(r) \approx 2^{-3.4}$ and for RC5-64 we have $c^+(r) \approx 2^{-4.2}$. In Sect. 4.4.5 we will show that the attacks that we have implemented for RC5-32 and RC5-64 approximately behave according to the previously derived approximate expectations. Hence, we can neglect the key dependency in (4.22): the bias given according to (4.24) is a sufficient practical estimate.

An alternative treatment of the linear hull effect for the case of RC6 can be found in [27, 121].

4.4 The Attack on RC5

This section presents an attack based on our findings on the properties of the internal structure of RC5 in the previous section.

4.4.1 Outline

As explained in Sect. 2.5.3 to mount a linear attack an attacker has two main tasks: finding a good propagation property, i.e., a linear approximation and exploiting it. In the previous section we have made a study of the properties of the linear approximations for RC5 and analyzed their probabilities. Based on this analysis we choose the one that can be exploited. However, the exploitation also can not be done straightforwardly with standard techniques from linear cryptanalysis. The linear approximations we use do not involve any key material, hence to exploit them an outer round trick is required. Standard outer round tricks as published in literature are not directly applicable for use with this approximation. Hence, we introduce a new concept: a non-uniformity function.

4.4.2 The Linear Approximations

We have derived and implemented a linear attack that uses approximation (4.22). Since this approximation does not involve any key bits, a basic linear attack is not possible. In particular for an r-round attack we will use (4.22) with $i = 0$ and $k = r$. The first addition, adding S_0, will be passed with an approximation on the LSB, since this has bias $\frac{1}{2}$. Therefore we take $j = 0$ in (4.22). We also choose $l = 0$ in (4.22) because then also the last key-addition in the whole approximation will be passed with bias $\frac{1}{2}$. It might be possible to further improve the attack by (also) using approximations with other values for j and l, but we have not found a method to do this.

To attack r rounds of RC5 we use the fact that each linear path that is part of the hull given by (4.22) is a chain of r 1-round approximations of the form (4.23). We will split the approximation into two parts. The first contains the approximation for the first key addition and the first round. This gives us the following $\lg w$ approximations. Each is the first part of a set of linear approximations that is contained in the linear hull of (4.22).

$$e_0 \cdot L_0 = e_k \cdot L_3 \text{ for } k = 0, \ldots, \lg w - 1 . \tag{4.27}$$

The remainder of the whole approximation can be specified by the following $\lg w$ approximations, each beginning with a different bit of L_3:

$$e_k \cdot L_3 = e_0 \cdot L_{2r+1} \text{ for } k = 0, \ldots, \lg w - 1 . \tag{4.28}$$

When for a certain plaintext encryption the intermediate value $R_1 \bmod w = k$, hence an element of $\{0, 1, \ldots, \lg w - 1\}$, then (4.27) behaves in the deviation direction. Hence, if also (4.28) behaves in the deviation direction then the whole approximation behaves in that direction. On the other hand, if $R_1 \bmod w \notin \{0, 1, \ldots, \lg w - 1\}$ then the probability that the whole approximation behaves in the direction of the deviation is much lower.

In the attack we want to check for every text if one of the approximations that correspond to (4.28) was followed. Since we have no information about any intermediate values we do not have a criterion that always holds. In the next section we will show how we nevertheless can exploit these approximations to deduce key material.

4.4.3 A Non-Uniformity Function

We will introduce the concept of a non-uniformity function as follows.

Definition 4.1 *A* non-uniformity rule h'_3 *is a function that takes as input a set of samples $\mathcal{S}$ (as defined in Vaudenay's statistical attack model) and gives as output a rule to increase or decrease the values of an array $\nu = (\nu(0), \ldots, \nu(m-1))$ of m counters:*

$$h'_3 : \mathcal{S} \to \mathbf{Z}^m .$$

We call the values of ν non-uniformity values *and ν a* non-uniformity array *or* non-uniformity function. *From the array ν a guess for target key material can be derived using a statistical measurement function.*

This extends Vaudenay's model of statistical analysis in a way that an attack can be more efficiently described. The analysis phase as given in Sect. 2.5.3 can now be described as.

Analysis phase.
1. Initialize the counter array $\nu = (\nu(0), \ldots, \nu(m-1))$ to 0^m.
2. For all i adjust the counter array ν according to the non-uniformity function.
3. For each candidate k of the l candidates for the key material: Compute the 'mark' $M_k = \Sigma(\nu(0), \ldots, \nu(m-1))$.

The 'difference' with Vaudenay's description is summarized by the following note.

Note 4.1 *The statistical attack using a non-uniformity function is a special case of Vaudenay's statistical attack in the sense that in the attack algorithm the first time that the computations depend on the guessed value of the key material is the moment that the marks are computed.*

RC5. We will derive a statistical measurement function that is expected to give higher values when one of the approximations was followed. Hence this function will have higher values for encryptions where $R_1 \bmod w \in \{0, 1, \ldots, \lg w - 1\}$ than for other R_1 values. Because $R_1 \bmod w = (R_0 - S_1) \bmod w$ and R_0 is known we can guess $S_1 \bmod w$ from this.

The non-uniformity function ν computes non-uniformity values for a given set of corresponding plaintext/ciphertext pairs. This set is divided into w subsets, each set contains plaintexts with the same $R_0 \bmod w$-value. For each set a non-uniformity value can be computed.

We look at the last round of the encryption. Suppose that one approximation of the linear hull corresponding to (4.28) is followed up to the last round. Say that in this particular case $e_k \cdot L_{2r-1}$ was biased for some $k \in \{0, \ldots, \lg w - 1\}$. Then following the approximation to the end would mean that $R_{2r-1} \bmod w = (w-k) \bmod w$ with a higher probability than other values. As seen in Sect. 4.3.2 depending on the subkeys also the other possible values of $R_{2r-1} \bmod w$ might not be uniformly distributed. But in any case certain values of $R_{2r-1} \bmod w$ will have a higher probability when (4.28) behaves in the deviation direction than when it does not behave in that way.

If we knew the value of $R_{2r-1} \bmod w$ it would be possible to compute $\lg w$ bits of S_{2r+1} or two possible values for those bits from the ciphertext with:

$$S_{2r+1} = R_{2r+1} - (R_{2r-1} \oplus L_{2r+1}) \lll L_{2r+1}\,, \tag{4.29}$$

since the values of R_{2r+1} and L_{2r+1} are known from the ciphertext. We will use $S\langle n\rangle$ to denote the $\lg w$-bit string, given by the bits $(n + \lg w - 1) \bmod w, \dots , (n + 1) \bmod w, n$ of S. The value of $L_{2r+1} \bmod w$ determines for which $\lg w$ bits of S_{2r+1} information can be computed, namely $S\langle L_{2r+1} \bmod w\rangle$. If $L_{2r+1} \bmod w = 0$ then the $\lg w$ LSBs can be computed. When $L_{2r+1} \bmod w \neq 0$ we can compute two values for $S\langle L_{2r+1} \bmod w\rangle$. The carry bit of the addition in (4.29) determines which one is the correct one.

In the attack we do not know the value of $R_{2r-1} \bmod w$ or even which value would be the most probable. Instead of trying to compute $\lg w$ bits of S_{2r+1} we make a similar computation for a value which we call S'. It is computed by taking $R_{2r-1} \bmod w = 0$ in (4.29), i.e.,

$$S' = R_{2r+1} - (L_{2r+1} \lll L_{2r+1})\,. \tag{4.30}$$

Due to the non-uniform distribution of $R_{2r-1} \bmod w$ it is expected that the distribution of $S'\langle\cdot\rangle$-values will be more non-uniform for encryptions where the approximation was followed than for others.

Hence, in the attack we use a counter array $A(i, j, k)$ for $i, j, k = 0, \dots , w-1$, where each i corresponds to a possible value of $R_0 \bmod w$, each j to a possible value of $L_{2r+1} \bmod w$ and each k to a possible value of $S'\langle L_{2r+1} \bmod w\rangle$. For each text we check if the approximation holds. If it holds, we change the counter array as follows. If $L_{2r+1} \bmod w = 0$, we increase $A(R_0 \bmod w, L_{2r+1} \bmod w, v)$ by 2, where v is the suggested $S'\langle L_{2r+1} \bmod w\rangle$-value. If $L_{2r+1} \bmod w \neq 0$, we increase $A(R_0 \bmod w, L_{2r+1} \bmod w, v_0)$ and $A(R_0 \bmod w, L_{2r+1} \bmod w, v_1)$ by 1, where v_0 and v_1 are the suggested values. If the approximation does not hold, we decrease the specific array entries accordingly.

Each $(R_0 \bmod w,\ L_{2r+1} \bmod w)$-combination gives a distribution of $S'\langle\cdot\rangle$-values. From Sect. 4.4.2 we know that the distributions corresponding to the values $R_1 \bmod w \in \{0, \dots , \lg w - 1\}$ will be the most non-uniform. To measure the non-uniformity we check for all w bits of our S' based on the $S'\langle\cdot\rangle$-values how many times 0 is suggested and how many times 1 and take the difference of these amounts. For each $R_0 \bmod w$ we take the sum over all possible $L_{2r+1} \bmod w$ of the absolute values of these differences. In this way the non-uniformity function $\nu : \{0, \dots , w - 1\} \to \mathbf{N}$ is defined:

$$\nu(r_0) = \sum_{l_{2r+1}=0}^{w-1} \sum_{x=0}^{\lg w-1} \Big| \sum_{v:e_x\cdot v=0} A(r_0, l_{2r+1} + x, v) - \sum_{v:e_x\cdot v=1} A(r_0, l_{2r+1} + x, v)\Big|\,, \tag{4.31}$$

where all indices of A are taken mod$w$. We expect that the sum of $\lg w$ non-uniformity values will be maximal for the values corresponding to texts with

$R_1 \bmod w \in \{0, \dots, \lg w - 1\}$. Hence, we define Σ to give the highest probability to the corresponding value. The final step of the algorithm guesses the value of $S_1 \bmod w$ accordingly. For our experiments we choose Σ to only rank the value first that gives the highest probability.

The observant reader will have noticed that according to the above description it is not necessary to have counters for the $S'\langle\cdot\rangle$-values. Instead of this one could use counters corresponding to the bits of S' and change these accordingly. The description above is used to emphasize that with the $S'\langle\cdot\rangle$-distribution also other non-uniformity measurements could be used. For example, one could look at all possible values for two subsequent bits of S'. Also it is probably possible to derive information about the actual value of S_{2r+1} from the $S'\langle\cdot\rangle$-distribution. This seems an open topic for further research.

Description according to Vaudenay's modified model. The attack falls in the category of statistical attacks, hence it can be described in the form of the model as described in Sect. 2.5.3. However, to give a formal description using this exact model would complicate the overview of the attack. Hence, we describe the attack formally using the modified model with the non-uniformity function as defined at the beginning of this section. Hence, instead of h_3 we have a non-uniformity rule h'_3. For the attack, we have the following functions and sets.

- $\mathcal{P} = \mathcal{C} = \{0,1\}^{2w}$ and $\mathcal{K} = \{0,1\}^{128}$.
- $\mathcal{L}$ contains all possible values of $S_1 \bmod w$, hence $\mathcal{L} = \{0, \dots, w-1\}$.
- $\mathcal{S}$ contains the tuples of all possible values of $R_{2r+1} \bmod w$, L_{2r+1}, $R_0 \bmod w$, and $e_0 \cdot L_0$, hence $\mathcal{S} = \{0,1\}^{\lg w} \times \{0,1\}^{w} \times \{0,1\}^{\lg w} \times \{0,1\}$.
- $\mathcal{Q}$ does not have to be defined since we work with a non-uniformity function.
- h_1 is defined as the part of the RC5 key schedule that derives the $\lg w$ least significant bits from the user key.
- $r = 1$ and h_2 is defined by the function that hashes a plaintext, ciphertext pair a tuple of the values $R_{2r+1} \bmod w$, L_{2r+1}, $R_0 \bmod w$, and $e_0 \cdot L_0$.
- h'_3 is the rule to create the non-uniformity function, given by (4.31).
- The measurement function Σ is defined as

$$\Sigma(q) = \sum_{i=0}^{\lg w - 1} |\nu((-q + i) \bmod w)| \,.$$

4.4.4 The Algorithm

The pseudo-code of the algorithm corresponding to the attack as described in the previous section is as follows.

```
1. Acquire $N$ known plaintext/ciphertext-pairs $(P_0, C_0), \dots, (P_{N-1}, C_{N-1})$.
2. Initialize a counter array[3] $A(i, j, k) := 0$ for $i, j, k = 0, \dots, w-1$.
3. For each plaintext/ciphertext-pair do:
   If $L_{2r+1} \bmod w = 0$ then
      (a) Compute $S'\langle 0 \rangle$-guess $v$.
      (b) If $e_0 \cdot L_0 = e_0 \cdot L_{2r+1}$ then $A(R_0, L_{2r+1}, v) += 2$.
          If $e_0 \cdot L_0 = e_0 \cdot L_{2r+1} \oplus 1$ then $A(R_0, L_{2r+1}, v) -= 2$.
   If $L_{2r+1} \bmod w \neq 0$ then
      (a) Compute $S'\langle L_{2r+1} \bmod w \rangle$-guesses $v_0$ and $v_1$.
      (b) If $e_0 \cdot L_0 = e_0 \cdot L_{2r+1}$ then $A(R_0, L_{2r+1}, v_0) += 1$
                                                and $A(R_0, L_{2r+1}, v_1) += 1$.
          If $e_0 \cdot L_0 \neq e_0 \cdot L_{2r+1}$ then $A(R_0, L_{2r+1}, v_0) -= 1$
                                                and $A(R_0, L_{2r+1}, v_1) -= 1$.
4. Compute $w$ non-uniformity values $\nu(i)$ according to (4.31), where $i$ corresponds with a value of $R_0 \bmod w$.
5. Find the value $x \in \{0, \dots, w-1\}$ for which $\sum_{i=0}^{\lg w - 1} |\nu((x+i) \bmod w)|$ is maximal.
6. Guess $S_1 \bmod w = w - x$.
```

4.4.5 The Results

We have implemented the attack on RC5-32 and RC5-64. The results are given in Table 4.3 and Table 4.4. As can be seen in the tables, we have done tests for up to 5 rounds of RC5-32 and up to 4 rounds of RC5-64. For each number of rounds tests were performed for several amounts of plaintexts. To give an indication of the practical aspects of the experiments: with our RC5-implementation carrying out the attack on the 5-round version for 10 keys with 2^{32} plaintexts took about 21 hours on a 333 MHz Pentium. To our knowledge these are the first experimentally executed known plaintext attacks on reduced versions of RC5, which require a negligible storage.

As stated at the end of Sect. 4.3.3, we would expect that an attack on r rounds of RC5 with x known plaintexts should have the same success probability as an attack on $r+1$ rounds with $(c^+)^{-2}x$ texts. For RC5-32 it holds that

[3]For clarity in the following description of the algorithm we have left out mod w when referring to indices of A.

Table 4.3: Experimental results of the attack on RC5-32.

Rounds	Known plaintexts	Success rate
2	2^{13}	28/100
	2^{14}	46/100
	2^{15}	89/100
	2^{16}	92/100
3	2^{17}	7/100
	2^{18}	15/100
	2^{19}	28/100
	2^{20}	49/100
	2^{21}	69/100
	2^{22}	81/100
4	2^{24}	7/100
	2^{25}	26/100
	2^{26}	44/100
	2^{27}	77/100
	2^{28}	82/100
5	2^{32}	4/10
	2^{33}	7/10
	2^{34}	9/10

Table 4.4: Experimental results of the attack on RC5-64.

Rounds	Known plaintexts	Success rate
2	2^{17}	39/100
	2^{18}	82/100
	2^{19}	96/100
3	2^{25}	28/50
	2^{26}	40/50
	2^{27}	47/50
4	2^{34}	9/10

$(c^+)^{-2} \approx 2^{6.8}$, for RC5-64 $(c^+)^{-2} \approx 2^{8.4}$. It can be seen from the tables that the results are better than expected, i.e., the factor to attack an extra round is $\approx 2^6$ for RC5-32 and $\approx 2^8$ for RC5-64. We conjecture that the reason for this is that the value of $R_{i+2} \bmod w$ depends significantly on the value of $R_i \bmod w$. Supporting experimental results are given in the next section, where we discuss the consequences for RC6.

Based on the experimental results and the theoretical estimation of the bias of the linear approximation we can estimate the complexity of the attack on more than 4 or 5 rounds (cf. Table 4.5). It follows that an attack can be mounted on RC5-32 with 10 rounds that has a success probability of 45% if 2^{62} plaintexts are available. An attack on RC5-64 with 15 rounds has a success probability of 90% if 2^{123} plaintexts are available.

Table 4.5: Expected number of required plaintexts for a known plaintext attack on $r(\geq 2)$ rounds of RC5-32 or RC5-64.

Success probability:	45%	90%
RC5-32	2^{2+6r}	2^{4+6r}
RC5-64	2^{1+8r}	2^{3+8r}

4.5 Consequences for RC6

The block cipher RC6 [120] has been submitted to NIST as an AES-candidate. Its design was based on RC5 and the security evaluation of RC5. To meet the block size requirement of 128 bits and to keep a 32-bit processor oriented design for this block size, RC6 was designed as two RC5-32's (with some changes) in parallel that interact.[4] Hence, cryptanalysis of RC5 can mostly be adapted for analysis of RC6.

However, the RC5-structure used in RC6 differs from the original version, which can be seen from the following description of RC6, and its visualisation given by Figure 4.2 (taken from [27]). The word size is w. First $2r + 4$ w-bit round keys are derived: $S_0 \ldots S_{2r+3}$. If $(A_0, B_0, C_0, D_0) \in \{0,1\}^w \times \{0,1\}^w \times \{0,1\}^w \times \{0,1\}^w$ is the plaintext, then the ciphertext $(A_{r+2}, B_{r+2}, C_{r+2}, D_{r+2})$

[4]Actually, the design of RC6 also is word oriented and the block size is $4w$, where w is the word size. We only discuss the 128-block size version, as it is the main object for the AES standardization process.

is computed iteratively with:

$$
\begin{aligned}
A_1 &= A_0 && (4.32)\\
B_1 &= B_1 + S_0 && (4.33)\\
C_1 &= C_0 && (4.34)\\
D_1 &= D_1 + S_1 && (4.35)\\
t_i &= (B_i(2B_i + 1)) \lll \lg w && (4.36)\\
u_i &= (D_i(2D_i + 1)) \lll \lg w && (4.37)\\
A_{i+1} &= B_i && (4.38)\\
B_{i+1} &= ((C_i \oplus u_i) \lll t_i) + S_{2i+1} && (4.39)\\
C_{i+1} &= C_i && (4.40)\\
D_{i+1} &= ((A_i \oplus t_i) \lll u_i) + S_{2i} && (4.41)\\
A_{r+2} &= A_{r+1} + S_{2r+2} && (4.42)\\
B_{r+2} &= B_{r+1} && (4.43)\\
C_{r+2} &= C_{r+1} + S_{2r+3} && (4.44)\\
D_{r+2} &= D_{r+1}\,, && (4.45)
\end{aligned}
$$

for $i = 1, \ldots, 2r$.

One of the most important differences is the following. The amount of rotation in RC5-32 was determined by taking the 5 LSBs of a 32-bit word. In RC6, the 5 bits that determine the amount of rotation depend on all 32 bits of a 32-bit word.

In the first place these changes to RC5 were made to preclude the successful differential attacks on RC5 [11, 120]. These attacks make use of the fact that only five LSBs determine the rotations. However, these changes also provide increased resistance to the attack method described in this paper.

To illustrate this we look at a transition version between RC5 and RC6. In the definition of RC5, replace (4.3) and (4.4) with

$$
\begin{aligned}
T_i &= (R_i(2R_i + 1)) \lll 5 && (4.46)\\
U_i &= L_i \oplus T_i && (4.47)\\
V_i &= U_i \lll T_i\,. && (4.48)
\end{aligned}
$$

For our theoretical analysis concerning the linear hull effect, this change makes little difference. One can still use the same linear approximations. However, the first round trick and last round trick we have used now become more complicated. To compute T_1 mod 32, one has to guess all[5] bits of S_1 and the construction of a non-uniformity function is not obvious.

We have done tests on the above described intermediate version with the last and first round replaced with a normal RC5-round. Then the first and last

[5]The online complexity can be decreased by precomputing T_1 mod 32 for all possible values of S_1 and storing these in a table. This would require $2^{32} - 2^{27}$ entries, since one could omit the values for one specific rotation amount.

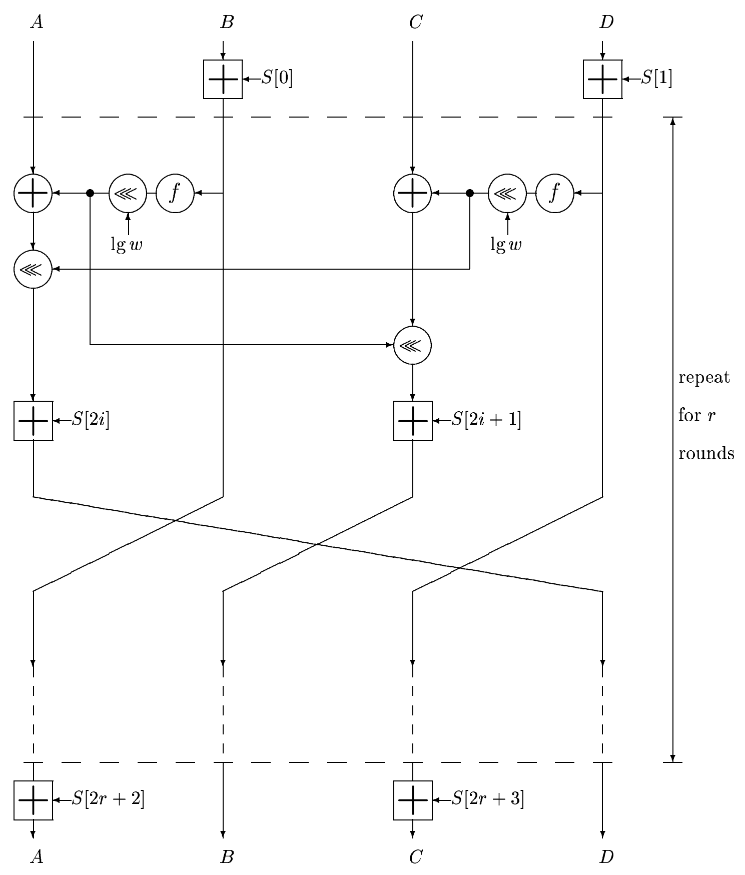

Figure 4.2: The cipher RC6. Here $f(x) = x(2x + 1)$ and the symbol $S[i]$ should be read as S_i.

round trick are straightforward. We have implemented the attack for 3 and 4 rounds of the cipher and results indicate that the increase in the number of necessary plaintexts for the same success probability was more in accordance with the theoretical results for RC5. Hence, the application of the extra function to determine the rotation amounts causes these values to be more independent.

Using similar ideas as for the described attack on RC5 several attacks on RC6 were described by Knudsen and Meier in [82], and by Gilbert, Handschuh, Joux and Vaudenay in [6, 58]. The attacks were however described in the context of generic statistical attacks based on statistical observations in reduced round versions.

In [82] the property is exploited that if for a number of texts the least significant bits of A_0 and C_0 are fixed to zero, then the values of the concatenation of A_{2i} and C_{2i} are not uniformly distributed if i is not too large. This has been measured experimentally with χ^2 tests up to 6 rounds in [82]. This resulted in a method to distinguish RC6 with up to 15 rounds from a random permutation and a key recovery attack with a lower complexity than required for exhaustive key search for up to 15 rounds. Both are chosen plaintext attacks.

In [58] it has been heuristically argued and experimentally tested that there exist probabilistic relations between input and output variables that only depend on fixed values of the key. For this they constructed statistical tests that verify this behavior. They have exploited this in a method that can distinguish RC6 with up to 13 rounds from a random permutation and a key recovery attack for up to 14 rounds. Both are known plaintext attacks.

In comparison with our attack the key recovery attacks are of interest. Both attacks use generic statistical tests to measure the non-randomness of their propagation property. However, we believe that a test that exploits the inner structure of RC6 might improve the attacks. Such a test might involve a non-uniformity function. We give some motivation in the next section.

4.6 Comparison to a Statistical Attack on RC5

In both papers [58, 82] it is mentioned that their techniques might be used to attack RC5, although no experimental evidence is given to illustrate the complexity. The approach of [82] has been applied to RC5-32 in [130] by Shimoyama, Takeuchi and Hayakawa with interesting results. In our context, their key recovery attack according to their **Algorithm 5.1** is of interest. (They also describe a chosen ciphertext attack that can distinguish RC5-32 from a random permutation for up to 10 rounds, a chosen plaintext attack that requires 2^{56} texts to break RC5-32 for up to 8.5 rounds, and some analysis of simplified versions of RC6.) This algorithm can be described as follows:

Algorithm 5.1 (Modified Knudsen, Meier) [130]

```
1. Acquire an amount of corresponding plaintext ciphertext pairs
   $(L_0, R_0), (L_x, R_x)$ with $L_0 \bmod 32 = 0$.
2. Initialize $2^{10}$ counters to 0: $A[0, \dots, 2^5 - 1][0, \dots, 2^5 - 1]$, corresponding to
   all possible values of pairs of $S_0 \bmod 32$ and $L_x \bmod 32$.
3. For each text:
   a. Compute $S_0 \bmod 32 = 32 - (R_0 \bmod 32)$.
   b. Increase the counter corresponding to $S_0, (L_x \bmod 32)$ by 1.
4. For each value of $S_0 \bmod 32$ compute the $\chi^2$ value over the set of values
   $\{A[S_0 \bmod 32][x] | x \in \{0, \dots, 2^5 - 1\}\}$.
5. Guess $S_0 \bmod 32$ has the value for which the $\chi^2$ value is maximal.
```

The results of this algorithm are summarized in Table 4.6. The third column gives the success probability that the algorithm outputs the correct value of

$S_0 \bmod 32$. The fourth column gives the probability that the output of the algorithm and the value of $S_0 \bmod 32$ only differ at most 1 bit.

Table 4.6: Experimental results of the statistical attack on RC5-32 where a χ^2 test is used as described in [130]. The third column shows the success probability to find five target key bits, the fourth column the success probability to find the value of five target key bits of which one is incorrect.

Rounds	Chosen plaintexts	Success prob. χ^2 att. [130]	Almost success χ^2 att. [130]
3	2^{14}	5 %	15 %
	2^{17}	15 %	30 %
	2^{18}	16 %	40 %
	2^{19}	18 %	52 %
	2^{20}	19 %	58 %
	2^{21}	20 %	58 %
	2^{22}	23 %	62 %
4	2^{20}	2 %	15 %
	2^{24}	18 %	46 %
	2^{25}	22 %	50 %
	2^{26}	22 %	55 %
	2^{27}	23 %	60 %
	2^{28}	25 %	65 %

When comparing the results of Table 4.3 on p. 75 with those of Table 4.6, one can see the following.

- To achieve the same success probability for an extra round, both attacks require an increase in the amount of plaintexts by the same factor of about 2^6 (extra experimental motivation for this can be found in [130]). Hence, it seems that they exploit the same propagation property.
- The success probability of the χ^2 algorithm is larger at first, but after a certain amount of plaintexts is exceeded, the success probability of our attack is the largest of the two.

Since our attack exploits a linear approximation, the first item supports the existence of a relation between the statistical effect as exploited in [130] and the propagation of the linear equation. The second item is an illustration of the advantages of having a theoretical explanation for non-random statistical behavior of a cipher.

It remains an open question if the statistical behavior as exploited for attacks on RC6 as described in [58, 82] can be explained by a more structural analysis such as linear cryptanalysis. If so, this might be exploitable by using more advanced outer round techniques. We want to encourage further research in this direction.

4.7 Conclusions

In this chapter we have evaluated the resistance of RC5 against linear attacks. Previous evaluation of Kaliski and Yin [72] had proven to be incorrect by Selçuk due to hidden assumptions under which they made their analysis [125]. We have taken into account the applicability of the Piling Up Lemma and the consequences of linear hull effects, both in combination with possible key dependency. This resulted in estimates for the complexity to mount a linear attack. This is the first contribution of this chapter.

A second and main contribution is the construction of an attack that exploits the linear hull effect that we described. A third contribution is the introduction of the concept of a non-uniformity function. This concept fits in the concept of Vaudenay's statistical attack model and extends the tools available to a cryptanalyst. We have implemented the attack on reduced versions of RC5-32 and RC5-64 and compared the results with the theoretical estimates. The results were slightly better.

In this way we estimate that our attack can theoretically break RC5-32 with 10 rounds and RC5-64 with 15 rounds. In comparison with previous attacks on RC5, our attack needs negligible storage capacity, i.e., it could be practically implemented.

We also evaluated the resistance of RC6 to this attack. The attack method has no serious consequences for the security of RC6. Apparently the precautions that the designers made to make RC6 more resistant against differential attacks, also made RC6 more resistant against more sophisticated linear attack methods.

We described more recently introduced theoretical attacks on reduced versions of RC6 [58, 82]. The attack on RC6 of Knudsen and Meier [82] has been described and implemented on RC5 by Shimoyama *et al.* [130]. The statistical properties exploited by this attack are similar to the linear approximation exploited in our attack. Experimental evidence support the advantage of using a non-uniformity function instead of a χ^2 test in this case, where it resulted in higher success probabilities.

This leads to the interesting open question if the key recovery attack of Knudsen and Meyer can be improved by a more structural analysis.

Chapter 5

Key Schedule Analysis

5.1 Introduction

An important aspect of block cipher design is the key schedule. The key schedule is a function that derives round keys from the user key. The requirements are similar to those for the data processing part: a key schedule needs to be secure and efficient. Weaknesses in the key schedule can have considerable consequences, not only for encryption but also if the block cipher is used as building block for other primitives. Such security weaknesses can appear for only a part of the set of user keys. The sets for which the keys have weak properties are called classes of *weak keys*. For example the DES has four weak keys for which $E_K(E_K(x)) = x$ and six pairs of semi-weak keys for which $E_{K_1}(E_{K_2}(x)) = x$.

This chapter describes a class of weak keys for the block cipher CRYPTON [91], one of the original 15 AES candidates. Several attacks are described that exploit the weak keys when the cipher is used in the Davies-Meyer scheme, a widely used hashing standard. From this we conclude that CRYPTON is broken in the sense of Definition 2.9. These results have been submitted to NIST by the author in [18]. A new version of CRYPTON with a revised key schedule has been introduced in [92].

The remainder of this chapter is organized as follows. In Sect. 5.2 the Wide Trail Strategy is introduced, according to which CRYPTON was designed. In Sect. 5.3 CRYPTON is described, the class of weak keys as well as methods to exploit this weakness in relevant practically used constructions. We conclude in Sect. 5.4.

5.2 The Wide Trail Strategy

The Wide Trail Strategy was introduced by Daemen in [30]. It is a design strategy for iterated block ciphers, that falls into the category of provable security as described in Sect. 2.4. The basic strategy assumes a block cipher design whose components are either linear (or have good linear properties), which means that they are 'easily' passed by linear approximations or differentials, or either (highly) non-linear S-boxes. For a further explanation, we introduce the concept *active S-box* as follows.

Definition 5.1 *An* active S-box *with respect to a differential or linear approximation is an S-box whose input/output difference is non-zero for a differential or for which input and/or output bits occur in the linear approximation.*

This definition of course can be generalized to other attacks that exploit propagation characteristics (that 'easily' pass the linear components). Since by assumption, the rest of the cipher's components are 'easily' passed, the resistance of the cipher against linear and differential depends on the possible number of active S-boxes and on the properties of the S-boxes. Daemen distinguished two possible approaches to design secure block ciphers based on this observation:

- Choose S-boxes with good resistance against linear and differential cryptanalysis. For example very large S-boxes.
- Design the round function in such a way that many S-boxes are active with respect to every differential or linear approximation.

In the Wide Trail Strategy the second approach is emphasized, although of course the choice of good S-boxes is still preferred. For efficiency on low-end platforms that use 8-bit processors usually 8×8 S-boxes are chosen.

A round function of a cipher designed according to the Wide Trail Strategy is composed of at least 3 different consecutive transformations that each contribute in a different way to provide resistance against linear and differential attacks. These transformations also are called layers and the following are always present:

- **Round key addition**: Mostly done with xor to avoid weak keys. Often denoted with σ or σ_K.
- **Non-linear layer** or **substitution layer**: the S-boxes provide resistance against linear and differential cryptanalysis at a local level. Often denoted by γ.
- **Diffusion layer** or **linear layer**: Ensures that there are always many active S-boxes. The purpose of this layer is similar to providing confusion in the sense of Shannon [127]. For 8×8 one could refer to it as diffusion at the byte level. A cipher can also have several diffusion layers per round.

Many iterated block ciphers have been designed according to this strategy: 3-Way by Daemen [30], Shark by Rijmen *et al.* [112, 111], Square, BKSQ and AES/Rijndael by Daemen and Rijmen [31, 33, 34]. The design of the AES candidate Serpent by Anderson, Biham and Knudsen seems to fit in this design strategy as well. None of these has been broken.

Resistance against linear and differential attack seems to be well provided by the Wide Trail Strategy. However, dedicated attacks have been published that should be considered for every design. During the design of Square, Knudsen introduced the dedicated Square attack [31] that was applicable to reduced round versions. Jacobsen and Knudsen described an interpolation attack on a slightly modified version of Shark in [70]. Attacks using truncated differentials also seem to be more effective against Wide Trail designs than against (most) other ciphers, amongst others illustrated in [31]. Main reason for this is the byte oriented structure of the ciphers.

Another block cipher designed according to the Wide Trail Strategy is one of the 15 original AES candidates CRYPTON. It was introduced by Lim in [91] and it achieved high performance figures on various platforms compared to the other candidates. However, it contained a flaw in the key schedule. In this section we describe a class of 2^{32} weak keys. This is mainly a consequence of the use of linear operations in the key schedule in combination with the byte-wise operations. These weak keys especially have consequences for the use of CRYPTON in certain hash function constructions. We present an attack that breaks CRYPTON, according to Definition 2.8 with respect to its use in the Davies-Meyer hash function construction. Hence, we conclude that CRYPTON is broken in the sense of Definition 2.9.

Later CRYPTON was revised by Lim in [92] into CRYPTON Version 1.0 and the original cipher was referred to as CRYPTON Version 0.5. We shall refer to Version 0.5 as CRYPTON, unless stated otherwise.

5.3 Weak Keys of CRYPTON

CRYPTON is an iterated block cipher with block size 128. Its key size is variable and can be chosen to be an integer number of bytes between 8 and 32. Hence, the key sizes 128, 192 and 256 are supported, as requested by NIST for the AES standard. The design is based on Square [31], i.e., on the Wide Trail Strategy; together with Rijndael this made for two proposals based on that strategy.

In this section we describe a class of 2^{32} weak 256-bit keys. If such a key is used for encrypting plaintexts from certain subsets of the text space, one always obtains ciphertexts in a specified subset. This is mainly caused by the use of linear transformations and symmetrical constants in the key schedule. There are 3 subsets of size 2^{64} of the text spaces that are invariant or that transform into each other by the transformation, specified by CRYPTON using a weak key. The weakness does not directly make CRYPTON unsuitable for use as an encryption algorithm, as the probability that a weak key is used is rather small. However, it can affect the security of CRYPTON in other primitives or applications, where an attacker can choose or influence the key considerably. The weakness for example can be exploited to find collisions when CRYPTON with a 256-bit key is used in the popular Davies-Meyer hash function construction (see also Sect. 2.6.3).

The rest of this section is organized as follows. In Sect. 5.3.1 we describe the features of CRYPTON that are relevant for our analysis. In Sect. 5.3.2 we describe the weak key class and in Sect. 5.3.4 its consequences. We conclude in

Sect. 5.4.

5.3.1 The Cipher

CRYPTON is defined as a 12-round block cipher preceded by an initial transformation and followed by a final output transformation. Each round transformation is composed of four different, consecutive transformations. All transformations operate on 16-byte strings, i.e., 128 bits. The transformations are designed according to the Wide Trail Strategy, hence each transformation contributes to a different security aspect, as explained in Sect. 5.2. The transformations are described as follows.

Notation. We use two distinct notations for the transformations. A transformation $\lambda : \{0,1\}^{128} \to \{0,1\}^{128}$ can be denoted as:

$$\lambda(X) = Y\,, \tag{5.1}$$

and expressing X and Y as 16-byte strings:

$$\lambda(x_{33}x_{32}x_{31}x_{30}x_{23}\cdots x_{01}x_{00}) = y_{33}y_{32}y_{31}y_{30}y_{23}\cdots y_{01}y_{00}\,, \tag{5.2}$$

where all x_{ij} and y_{ij} are byte values. The transformations of CRYPTON also can be viewed upon as operating on a 4×4 byte array. In our notation (5.1) and (5.2) are equivalent to

$$\begin{pmatrix} x_{03} & x_{02} & x_{01} & x_{00} \\ x_{13} & x_{12} & x_{11} & x_{10} \\ x_{23} & x_{22} & x_{21} & x_{20} \\ x_{33} & x_{32} & x_{31} & x_{30} \end{pmatrix} \xrightarrow{\lambda} \begin{pmatrix} y_{03} & y_{02} & y_{01} & y_{00} \\ y_{13} & y_{12} & y_{11} & y_{10} \\ y_{23} & y_{22} & y_{21} & y_{20} \\ y_{33} & y_{32} & y_{31} & y_{30} \end{pmatrix}. \tag{5.3}$$

A last useful representation is to represent the rows of the array with a variable, representing four bytes. Hence, (5.1), (5.2) and (5.3) are equivalent to

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \xrightarrow{\lambda} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}. \tag{5.4}$$

Key addition σ. This is performed by xoring the input with a 128-bit round key K_i:

$$\sigma_{K_i}(A) = A \oplus K_i, \text{ for } K_i, A \in \{0,1\}^{128}\,. \tag{5.5}$$

Non-linear substitution layer γ. In this layer two S-boxes are used: S_0 and S_1, with the property

$$S_0 = S_1^{-1}\,. \tag{5.6}$$

For the construction of these S-boxes we refer to [91]. The substitution transformation differs for odd and even rounds. For odd rounds we have $\gamma_o : \{0,1\}^{128} \to \{0,1\}^{128}$, defined as

$$\begin{pmatrix} x_{03} & x_{02} & x_{01} & x_{00} \\ x_{13} & x_{12} & x_{11} & x_{10} \\ x_{23} & x_{22} & x_{21} & x_{20} \\ x_{33} & x_{32} & x_{31} & x_{30} \end{pmatrix} \xrightarrow{\gamma_o} \begin{pmatrix} S_1(x_{03}) & S_0(x_{02}) & S_1(x_{01}) & S_0(x_{00}) \\ S_0(x_{13}) & S_1(x_{12}) & S_0(x_{11}) & S_1(x_{10}) \\ S_1(x_{23}) & S_0(x_{22}) & S_1(x_{21}) & S_0(x_{20}) \\ S_0(x_{33}) & S_1(x_{32}) & S_0(x_{31}) & S_1(x_{30}) \end{pmatrix}, \tag{5.7}$$

and for even rounds we have $\gamma_e : \{0,1\}^{128} \to \{0,1\}^{128}$, defined as

$$\begin{pmatrix} x_{03} & x_{02} & x_{01} & x_{00} \\ x_{13} & x_{12} & x_{11} & x_{10} \\ x_{23} & x_{22} & x_{21} & x_{20} \\ x_{33} & x_{32} & x_{31} & x_{30} \end{pmatrix} \xrightarrow{\gamma_o} \begin{pmatrix} S_0(x_{03}) & S_1(x_{02}) & S_0(x_{01}) & S_1(x_{00}) \\ S_1(x_{13}) & S_0(x_{12}) & S_1(x_{11}) & S_0(x_{10}) \\ S_0(x_{23}) & S_1(x_{22}) & S_0(x_{21}) & S_1(x_{20}) \\ S_1(x_{33}) & S_0(x_{32}) & S_1(x_{31}) & S_0(x_{30}) \end{pmatrix}. \tag{5.8}$$

Diffusion layers π and τ. For the diffusion CRYPTON uses three linear transformation π_o, π_e and τ. The transformations π_o and π_e are bit permutations, that specify 4 transformations that separately operate on the four columns, i.e., the values of column i of the output only depend on the values of column i of the input, $i = 0, \dots, 3$.

The permutations involve four 4-byte constants:

$$\begin{aligned} M_0 &= \texttt{0x3fcff3fc} \\ M_1 &= \texttt{0xfc3fcff3} \\ M_2 &= \texttt{0xf3fc3fcf} \\ M_3 &= \texttt{0xcff3fc3f}. \end{aligned}$$

For odd rounds we have $\pi_o : \{0,1\}^{128} \to \{0,1\}^{128}$, defined as

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \xrightarrow{\pi_o} \begin{pmatrix} (a_0 \wedge M_0) \oplus (a_1 \wedge M_1) \oplus (a_2 \wedge M_2) \oplus (a_3 \wedge M_3) \\ (a_0 \wedge M_1) \oplus (a_1 \wedge M_2) \oplus (a_2 \wedge M_3) \oplus (a_3 \wedge M_0) \\ (a_0 \wedge M_2) \oplus (a_1 \wedge M_3) \oplus (a_2 \wedge M_0) \oplus (a_3 \wedge M_1) \\ (a_0 \wedge M_3) \oplus (a_1 \wedge M_0) \oplus (a_2 \wedge M_1) \oplus (a_3 \wedge M_2) \end{pmatrix}, \tag{5.9}$$

and for even rounds we have $\pi_e : \{0,1\}^{128} \to \{0,1\}^{128}$, defined as

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \xrightarrow{\pi_e} \begin{pmatrix} (a_0 \wedge M_1) \oplus (a_1 \wedge M_2) \oplus (a_2 \wedge M_3) \oplus (a_3 \wedge M_0) \\ (a_0 \wedge M_2) \oplus (a_1 \wedge M_3) \oplus (a_2 \wedge M_0) \oplus (a_3 \wedge M_1) \\ (a_0 \wedge M_3) \oplus (a_1 \wedge M_0) \oplus (a_2 \wedge M_1) \oplus (a_3 \wedge M_2) \\ (a_0 \wedge M_0) \oplus (a_1 \wedge M_1) \oplus (a_2 \wedge M_2) \oplus (a_3 \wedge M_3) \end{pmatrix}. \tag{5.10}$$

The transformation τ is a byte transformation, that transposes the byte value at position (i,j) in the byte array to position (j,i) for all i,j. Hence,

$\tau : \{0,1\}^{128} \to \{0,1\}^{128}$ is defined as

$$\begin{pmatrix} x_{03} & x_{02} & x_{01} & x_{00} \\ x_{13} & x_{12} & x_{11} & x_{10} \\ x_{23} & x_{22} & x_{21} & x_{20} \\ x_{33} & x_{32} & x_{31} & x_{30} \end{pmatrix} \xrightarrow{\tau} \begin{pmatrix} x_{30} & x_{20} & x_{10} & x_{00} \\ x_{31} & x_{21} & x_{11} & x_{01} \\ x_{32} & x_{22} & x_{12} & x_{02} \\ x_{33} & x_{23} & x_{13} & x_{03} \end{pmatrix} . \tag{5.11}$$

Round transformation ρ. The round transformations for odd and even rounds are slightly different, though have the same structure. They are defined as

$$\begin{aligned} \rho_{oK_r}(A) &= (\sigma_{K_r} \circ \tau \circ \pi_o \circ \gamma_o)(A), \text{ odd rounds: } r = 1, 3, \dots , \\ \rho_{eK_r}(A) &= (\sigma_{K_r} \circ \tau \circ \pi_e \circ \gamma_e)(A), \text{ even rounds: } r = 2, 4, \dots , \end{aligned}$$

for $K_i, A \in \{0,1\}^{128}$. The initial transformation is defined by σ_{K_0}. The final output transformation is defined by $\phi_e = \tau \circ \pi_e \circ \tau$. Hence, an encryption with CRYPTON is given by

$$E_K = \phi_e \circ \rho_{eK_r} \circ \rho_{eK_{r-1}} \circ \dots \circ \rho_{oK_1} \circ \sigma_{K_0} . \tag{5.12}$$

Key schedule. The round keys K_i, $i = 0, \dots , r$ each consist of four 32-bit words which we denote with $(K[4i+3], K[4i+2], K[4i+1], K[4i])$. The round keys are derived from the user key as follows. Firstly, if the user key length is smaller than 32 bytes then it is extended to 256 bits by extending zeroes to the left. From this extended key the key schedule proceeds in 2 phases:

1. The first 2 round key words (hence, the first 2 round keys) $K[i], i = 0 \dots 7$, are derived from the user key via a rather involved invertible transformation, using the cipher's components.

2. After that for each $i \in \{0 \dots 7\}$ each expanded key $K[i+8j], j = 1 \dots 6$, only depends on $K[i]$.

This concludes the description of CRYPTON as far as its structure is relevant for our analysis. For further details we refer to [91].

5.3.2 The Weak Keys

We can make two observations about the second phase of the key schedule:

1. The only operations that are used in the second phase of the key scheduling are rotations with a multiple of 8 and xoring with round constants of the form *aaaa*, where *a* are byte values. This has as consequence that if for one of the first eight expanded keys holds that $K[i] = aaaa$ for some byte value a, then for all derived expanded keys $K[i+8j] = bbbb$ for some b, not necessarily the same b for all j.

2. The xoring and rotations are done in the following way. Each subkey of four 32-bit words $(K[4i+3], K[4i+2], K[4i+1], K[4i])$ is derived from a previous subkey $(K[4i+3-8], K[4i+2-8], K[4i+1-8], K[4i-8])$, either
   (a) by xoring the first and third 32-bit word with a constant *aaaa* and and rotating the second and last 32-bit word by a multiple of eight, or
   (b) by xoring the second and last 32-bit word with a constant *aaaa* and and rotating the first and third 32-bit word by a multiple of eight.

   Hence, if for a pair $(i_1, i_0) \in \{(7,5),(6,4),(3,1),(2,0)\}$ the expanded keys satisfy $K[i_1] = K[i_0] = aaaa$, then $K[i_1 + 8j] = K[i_0 + 8j] = bbbb$.

With the above we can construct 2^{32} user keys for which every round key is of the form *bbbbaaaabbbbaaaa*; it is sufficient to start with the first two round keys of such a form. These user keys can be computed by inverting the first phase of the key schedule. In this way we find 2^{32} 256-bit (or 32-byte) keys. Experimentally we have deduced that about 2^{28} of them have an equivalent 31-byte user key, 2^{24} have an equivalent 30-byte key, 2^{20} have an equivalent 29-byte and none of them have an equivalent to a key of less than 29 bytes. The set of user keys that results in weak rounds keys is denoted with W, and the set of rounds keys of the form *bbbbaaaabbbbaaaa* is denoted with W_r.

Definition 5.2

$$W_r = \{K \in \{0,1\}^{128} | K = bbbbaaaabbbbaaaa\}\,. \tag{5.13}$$

5.3.3 Special Subsets of the Text Space

We define three subsets of the text space as follows.

Definition 5.3

$$\begin{aligned} V_0 &= \{x \in \{0,1\}^{128} | x = ghghefefcdcdabab\} \\ V_1 &= \{x \in \{0,1\}^{128} | x = ghefcdabefghabcd\} \\ V_2 &= \{x \in \{0,1\}^{128} | x = efghabcdefghabcd\}\,, \end{aligned}$$

where $a, \ldots, h$ are byte values. Each of the subsets contains 2^{64} elements.

These sets have the following property.

Property 5.1 *For all i, V_i forms a closed group under the operation $\oplus$.*

We will present other special properties of these three sets: the key addition with weak round keys, $K \in W_r$, and the other transformations of CRYPTON leave these sets invariant or transform them into each other. In combination with Property 5.1 these properties can be exploited to mount practical attacks on certain constructions that use CRYPTON as a building block.

Key addition with a weak round key. For the key addition σ with a round key of the form $K = bbbbaaaabbbbaaaa$ it holds that:

Property 5.2 *If $x \in V_i$ and $K \in W_r$ then $\sigma_K(x) \in V_i$ for all i.*

The proof is trivial.

Non-linear transformations γ_o and γ_e. Here we have the following property.

Property 5.3 *If $x \in V_i$ then $\gamma_o(x) \in V_i$ and $\gamma_e(x) \in V_i$ for all i.*

For example, if $x \in V_0$ with $x = ghghefefcdcdabab$, then the computation of $\gamma_o(x)$ can be denoted by

$$\begin{pmatrix} a & b & a & b \\ c & d & c & d \\ e & f & e & f \\ g & h & g & h \end{pmatrix} \xrightarrow{\gamma_o} \begin{pmatrix} S_1(a) & S_0(b) & S_1(a) & S_0(b) \\ S_0(c) & S_1(d) & S_0(c) & S_1(d) \\ S_1(e) & S_0(f) & S_1(e) & S_0(f) \\ S_0(g) & S_1(h) & S_0(g) & S_1(h) \end{pmatrix} = \begin{pmatrix} a' & b' & a' & b' \\ c' & d' & c' & d' \\ e' & f' & e' & f' \\ g' & h' & g' & h' \end{pmatrix}. \tag{5.14}$$

Bit permutations π_o and π_e. We first derive some properties of these permutations that do not depend on the input values. Since these properties hold for both even and odd round versions of π we will omit the subscripts.

From (5.9) and (5.10) it can be seen that π consists of four separate transformations, which are transformations on the separate columns of the array: the bits of the rows a_i are selected by the constants M_i, but are not shifted from their positions. Hence, π can be written as

$$\pi(x_3, x_2, x_1, x_0) = (\pi_3(x_3), \pi_2(x_2), \pi_1(x_1), \pi_0(x_0)), \tag{5.15}$$

where the x_i's represent the columns of an array. For example, if we define

$$M_j^i = M_j \wedge (\texttt{0xff}) \ll 8i, \text{for } i, j = 0, \ldots, 3$$

then π_{oi} can be represented as

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \xrightarrow{\pi_{oi}} \begin{pmatrix} ((a \wedge M_0^i) \oplus (b \wedge M_1^i) \oplus (c \wedge M_2^i) \oplus (d \wedge M_3^i)) \gg 8i \\ ((a \wedge M_1^i) \oplus (b \wedge M_2^i) \oplus (c \wedge M_3^i) \oplus (d \wedge M_0^i)) \gg 8i \\ ((a \wedge M_2^i) \oplus (b \wedge M_3^i) \oplus (c \wedge M_0^i) \oplus (d \wedge M_1^i)) \gg 8i \\ ((a \wedge M_3^i) \oplus (b \wedge M_0^i) \oplus (c \wedge M_1^i) \oplus (d \wedge M_2^i)) \gg 8i \end{pmatrix}, \tag{5.16}$$

for $i = 0, \dots, 3$. Hence, when we substitute $\begin{pmatrix} a \\ b \\ a \\ b \end{pmatrix}$ for $\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$ in (5.16) we get

$$\begin{pmatrix} a \\ b \\ a \\ b \end{pmatrix} \xrightarrow{\pi_{oi}} \begin{pmatrix} ((a \wedge M_0^i) \oplus (b \wedge M_1^i) \oplus (a \wedge M_2^i) \oplus (b \wedge M_3^i)) \gg 8i \\ ((a \wedge M_1^i) \oplus (b \wedge M_2^i) \oplus (a \wedge M_3^i) \oplus (b \wedge M_0^i)) \gg 8i \\ ((a \wedge M_2^i) \oplus (b \wedge M_3^i) \oplus (a \wedge M_0^i) \oplus (b \wedge M_1^i)) \gg 8i \\ ((a \wedge M_3^i) \oplus (b \wedge M_0^i) \oplus (a \wedge M_1^i) \oplus (b \wedge M_2^i)) \gg 8i \end{pmatrix} = \begin{pmatrix} a' \\ b' \\ a' \\ b' \end{pmatrix} \tag{5.17}$$

for all $i = 0, \dots, 3$. The same property can be derived for all π_{ei}, hence we have:

Property 5.4

$$\begin{pmatrix} a \\ b \\ a \\ b \end{pmatrix} \xrightarrow{\pi_i} \begin{pmatrix} a' \\ b' \\ a' \\ b' \end{pmatrix}, \textit{ for } i = 0, \dots, 3 .$$

To derive the next property of π we first make the following observation about the constants M_i.

Property 5.5

$$M_{(i+l)\bmod 4} = (M_i) \gg (8 \cdot (l \bmod 4)), \textit{ for any integer } l \textit{ and } i = 0, \dots, 3 .$$

From Property 5.5 follows:

Property 5.6

$$M_j^i = M_{(j+l)\bmod 4} \wedge (\texttt{0xff}) \ll (8i - 8(l \bmod 4)), \textit{ for any integer } l \textit{ and } i, j = 0, \dots, 3 .$$

Now if we increase i in (5.16) by two modulo 4 (which we abbreviate with .4), we get

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \xrightarrow{\pi_{o((i+2)\bmod 4)}} \tag{5.18}$$

$$\begin{pmatrix} ((a \wedge M_0^{(i+2)\bmod 4}) \oplus \dots \oplus (d \wedge M_3^{(i+2)\bmod 4})) \gg 8((i+2) \bmod 4) \\ ((a \wedge M_1^{(i+2)\bmod 4}) \oplus \dots \oplus (d \wedge M_0^{(i+2)\bmod 4})) \gg 8((i+2) \bmod 4) \\ ((a \wedge M_2^{(i+2)\bmod 4}) \oplus \dots \oplus (d \wedge M_1^{(i+2)\bmod 4})) \gg 8((i+2) \bmod 4) \\ ((a \wedge M_3^{(i+2)\bmod 4}) \oplus \dots \oplus (d \wedge M_2^{(i+2)\bmod 4})) \gg 8((i+2) \bmod 4) \end{pmatrix} .$$

Using Property 5.6 we can write (5.18) as

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \xrightarrow{\pi_{o((i+2)\mathrm{mod}4)}} \begin{pmatrix} ((a \wedge M_2^i) \oplus (b \wedge M_3^i) \oplus (c \wedge M_0^i) \oplus (d \wedge M_1^i)) \ggg 8i \\ ((a \wedge M_3^i) \oplus (b \wedge M_0^i) \oplus (c \wedge M_1^i) \oplus (d \wedge M_2^i)) \ggg 8i \\ ((a \wedge M_0^i) \oplus (b \wedge M_1^i) \oplus (c \wedge M_2^i) \oplus (d \wedge M_3^i)) \ggg 8i \\ ((a \wedge M_1^i) \oplus (b \wedge M_2^i) \oplus (c \wedge M_3^i) \oplus (d \wedge M_0^i)) \ggg 8i \end{pmatrix}. \tag{5.19}$$

Comparison of (5.16) and (5.19) leads to the following property, that also can be derived for π_e.

Property 5.7

$$\textit{If } \pi_i \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} \textit{ then } \pi_{(i+2)\mathrm{mod}4} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} g \\ h \\ e \\ f \end{pmatrix}, \textit{ for } i = 0, \ldots, 3.$$

From Properties 5.4 and 5.7 the following property can be derived with respect to the special subsets.

For $ghghefefcdcdabab \in V_0$ it holds that

$$\begin{pmatrix} a & b & a & b \\ c & d & c & d \\ e & f & e & f \\ g & h & g & h \end{pmatrix} \xrightarrow{\pi} \begin{pmatrix} i & m & k & o \\ j & n & l & p \\ k & o & i & m \\ l & p & j & n \end{pmatrix} \in V_1.$$

For $ghefcdabefghabcd \in V_1$ it holds that

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ c & d & a & b \\ g & h & e & f \end{pmatrix} \xrightarrow{\pi} \begin{pmatrix} i & m & i & m \\ j & n & j & n \\ k & o & k & o \\ l & p & l & p \end{pmatrix} \in V_0.$$

For $efghabcdefghabcd \in V_2$ it holds that

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ a & b & c & d \\ e & f & g & h \end{pmatrix} \xrightarrow{\pi} \begin{pmatrix} i & k & m & o \\ j & l & n & p \\ i & k & m & o \\ j & l & n & p \end{pmatrix} \in V_2.$$

This can be summarized in the following property.

Property 5.8

- *If* $x \in V_0$ *then* $\pi(x) \in V_1$.
- *If* $x \in V_1$ *then* $\pi(x) \in V_0$.
- *If* $x \in V_2$ *then* $\pi(x) \in V_2$.

Byte permutation τ. The following property holds.

Property 5.9

- *If $x \in V_0$ then $\tau(x) \in V_2$.*
- *If $x \in V_1$ then $\tau(x) \in V_1$.*
- *If $x \in V_2$ then $\tau(x) \in V_2$.*

For example, if $x \in V_0$ with $x = ghghefefcdcdabab$, then the computation of $\tau(x)$ can be denoted by

$$\begin{pmatrix} a & b & a & b \\ c & d & c & d \\ e & f & e & f \\ g & h & g & h \end{pmatrix} \xrightarrow{\tau} \begin{pmatrix} h & f & d & b \\ g & e & c & a \\ h & f & d & b \\ g & e & c & a \end{pmatrix}. \tag{5.20}$$

CRYPTON. From the Properties 5.2, 5.3, 5.8 and 5.9, that specify the transformations' properties on the special subsets, the following property of one round of CRYPTON, used with a weak key, can be derived.

Property 5.10 *For all $K \in W_r$:*

- *If $x \in V_0$ then $\rho_K(x) \in V_1$.*
- *If $x \in V_1$ then $\rho_K(x) \in V_2$.*
- *If $x \in V_2$ then $\rho_K(x) \in V_0$.*

For the output transformation ϕ we have:

Property 5.11 *For all $K \in W_r$:*

- *If $x \in V_0$ then $\phi_K(x) \in V_0$.*
- *If $x \in V_1$ then $\phi_K(x) \in V_2$.*
- *If $x \in V_2$ then $\phi_K(x) \in V_1$.*

Hence, for each of the keys in W, CRYPTON (with 12 rounds) will encrypt the elements of the special subsets as follows.

Property 5.12 *For all $K \in W$:*

- *If $x \in V_0$ then $E_K(x) \in V_0$.*
- *If $x \in V_1$ then $E_K(x) \in V_2$.*
- *If $x \in V_2$ then $E_K(x) \in V_1$.*

A consequence of this property is:

Property 5.13 *The subspace of size* 2^{32}

$$V_3 = V_0 \cap V_1 = V_0 \cap V_2 = \{x \in \{0,1\}^{128} | x = abcdabcdabcdabcd\}$$

is invariant under all transformations defined by CRYPTON, using a key from W.

Part of the results were independently found by a team of French researchers and published in [6]. Considering special subsets of the text space they only give the set V_3.

Note 5.1 *Assuming that the key schedule permits user keys for which all round keys are elements of* W_r*, the weakness given by Property 5.12 for 12 rounds is independent of the number of rounds. Changing the number of rounds would have as consequence that the indices of* V *might permute in Property 5.12.*

5.3.4 Attacks

There is only a small probability that a weak key is used when a user key is generated randomly. Besides that only CRYPTON used with keys of length 256 (or at least 29 bytes) has a subset of weak keys that we described. On the other hand with a few chosen plaintexts it can be easily tested if a weak key of W is used for encryption. However, the weak key class certainly influences security significantly in practical applications where an attacker can choose the key. We have reported the weaknesses of CRYPTON to NIST in [18] with as an example of the consequences some attacks on the Davies-Meyer hash function scheme using CRYPTON as a building block. Besides the weakness in that particular standardized and widely used construction there is always the risk of exploiting this weakness in other constructions.

Remark: Due to the non-linear first phase of the key schedule, there is no obvious way in which the algebraic properties of this section might be used to mount an attack on the Matyas-Meyer-Oseas or the Miyaguchi-Preneel hash function constructions.

The Davies-Meyer hash function construction. Besides encryption a typical application of a block cipher is its use as a building block for a hash function. One of the most popular hash function constructions is the Davies-Meyer scheme. Also in [91] it was explicitly mentioned that CRYPTON can be used as underlying block cipher in the Davies-Meyer hash function construction. A description can be found in Sect. 2.6.3. For this scheme the hash H_m on a message $x_0 x_1 \ldots x_m$, where x_i is of the same length as the key length of the underlying block cipher, is defined iteratively by

$$H_i = H_{i-1} \oplus E_{x_i}(H_{i-1}),$$

where H_0 is an initial vector. Note that by using larger key lengths the performance of the system improves, as fewer computations of the underlying block cipher are needed, which is the bottleneck in the performance. For the case of CRYPTON using the Davies-Meyer scheme with 256-bit keys instead of 128-bit keys implies a performance improvement with a factor 2.

We describe three possible attacks on the Davies-Meyer scheme based on CRYPTON, that exploit the weak keys. The properties of hash functions and possible attacks can be found in Sect. 2.6.

1. A free-start collision attack with processing complexity $2^{16.5}$.
2. A collision attack, where the initialization vector can not be chosen by the attacker, with processing complexity $2^{64}/3$.
3. A second preimage attack for set of 2^{64} images: 2^{65}.

The third attack was found by Wagner [138].

Free-start collision attack. The basic attack finds collisions for one block messages x_0, hence for the equation:

$$H_1 = H_0 \oplus E_{x_0}(H_0)\,.$$

According to Properties 5.1 and 5.12, if $H_0 \in V_4$ and $x_0 \in W$, then $E_{x_0}(H_0) \in V_4$ and also $H_1 \in V_4$. The attack is as follows. An attacker computes the hash value H_1 for a number of combinations of values for $x_0 \in W$ and $H_0 \in V_4$ until he finds a collision for H_1.

Since V_0 has 2^{32} elements, in accordance with the birthday paradox one finds a collision for H_1 with probability ≈ 0.39 and ≈ 0.63 in respectively 2^{16} and $2^{16.5}$ hash calculations. A brute force collision attack would require a complexity of $2^{32.5}$. We have generated 1000 sets of 2^{16} hash values as described above; 427 sets contained at least one collision, which is slightly higher than expected.

General collision attack. Contrary to the free-start attack as described above, here an attacker does not have the ability to choose H_0. We describe the general collision attack as an attack to find collisions for two block messages x_0x_1, for which a hash-computation is visualized in Figure 5.1.

We now assume that H_0 has a fixed value. The attack proceeds in two stages:

1. An attacker computes H_1 for different values of x_0 until $H_1 \in V_i$ for a value of i.
2. The value of x_0, found in the previous step is set to be the first block of the message. The attacker proceeds with computing $H_2 = H_1 \oplus E_{x_1}(H_1)$ for different values of $x_1 \in W$ until he finds a collision for H_2.

The probability that an H_1 as computed according to step 1 is in one of the 3 special subsets is approximately $3 \cdot 2^{-64}$ and the total of possible values for H_1 is 2^{128}, hence the processing complexity of step 1 can be estimated as $2^{64}/3$.

If the value of H_1 found in the first step would be an element of V_4, the complexity of step 2 would be the same as for the free-start collision attack:

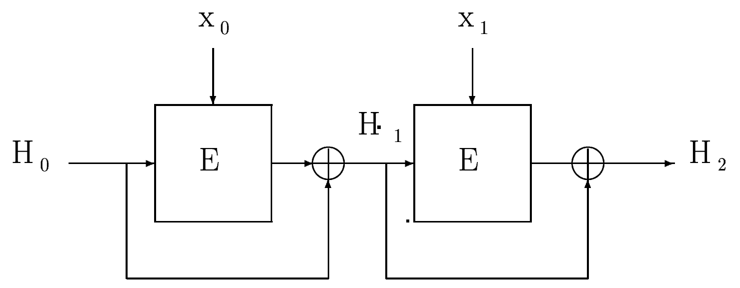

Figure 5.1: Computation of a hash on a two block message, according to the Davies-Meyer scheme.

$2^{16.5}$. Otherwise, since each set V_i contains 2^{64} elements, the complexity of step 2 would be approximately $2^{32.5}$ block cipher computations. In all cases, the complexity of step 2 would be negligible compared to that of step 1. Hence, the attack has a complexity of $2^{64}/3$, which is slightly better than a brute force collision attack, that would have a complexity of $2^{64.5}$.

Second preimage attack. For a hash value $H \in V_0$ generated with the Davies-Meyer scheme using CRYPTON, a second preimage can be found with this attack. Hence, the attack is only feasible for 2^{64} possible hash values. The preimage is a message consisting of three blocks: $x_0x_1x_2$. We presume that H_0 has a fixed value. The attack consists of 2 stages:

1. The attacker computes H_1 for different values of x_0 until $H_0 \in V_0$.
2. The value x_0 found in the first step is the first block of the message. The attacker proceeds with computing H_3 for different values of $x_1, x_2 \in W$ until he finds $H_3 = H$.

The complexity of the first step is 2^{64}. Each value for H_3 in the second step is expected to be an element of V_0. Since each value is equally likely the probability that $H_3 = H$ is 2^{-64} for each computation of H_3. Hence, the processing complexity of step 2 is approximately 2^{64}. The total complexity of the attack is $2^{64} + 2^{64} = 2^{65}$. A brute force second preimage attack would require a complexity of 2^{128}.

5.3.5 Version 1.0

CRYPTON's designer Lim made changes to his algorithm, which resulted in CRYPTON Version 1.0 [92]. The changes include a revision of the key schedule to decrease its linear and symmetric properties. The new key schedule does not have weak keys as described in this chapter.

CRYPTON was not selected as one of the last five candidates for AES.

5.4 Conclusions

This chapter showed a practical example of the relevance of key schedule cryptanalysis. Its main contributions are the description of a class of weak keys for CRYPTON, one of the original 15 AES candidates. CRYPTON has a class of 2^{32} weak 256-bit keys. Hence care has to be taken when CRYPTON with 256-bit keys is used in applications where an attacker can choose the key, such as the Davies-Meyer hash function construction. This application is typically used with large key lengths since this increases the throughput. We have given several methods to exploit the weakness in this scheme.

A way to resolve this weakness is to strengthen the key schedule. This can be done by introducing more non-linearity in the key schedule, as for example was done in Rijndael [34], the design of which was based on Square. The designer of CRYPTON has released a new version in which the problems seem to have been resolved.

Chapter 6

Power Analysis

6.1 Introduction

This chapter studies a recently introduced class of attack methods, that exploit the power consumption of a cryptographic device to deduce secret key material of cryptographic primitives. Especially smart cards, either used for identification or electronic payment systems, can be vulnerable against these kind of attacks. The existence of such a widespread target combined with the limited requirements to mount this type of attack, has resulted in a situation where intensive research is being performed to find countermeasures. However, this has also increased the temptation within the industry to opt for security through obscurity.

In this chapter we study power attacks and deduce the factors that cause their applicability and success probability. Based on this we make a classification of possible countermeasures and their advantages and disadvantages, which depend on the situation in which they are applied. This framework can be used as a tool for developers of systems that are potentially vulnerable to power attacks to select the appropriate countermeasures. In this framework we develop several specific countermeasures, amongst others for a typical payment system that is based on triple-DES.

Part of this chapter has been published by the author, Preneel and Vandewalle in [23]. The results in Sect. 6.6 are joint work with Preneel and Rijmen. Section 6.6.4 describes an idea of Shamir.

The remainder of this chapter is organized as follows. We introduce power attacks and their impact in Sect. 6.2. We give an overview of relevant properties of smart cards in Sect. 6.3. In Sect. 6.4 the different kinds of power attacks are explained. Section 6.5 elaborates on the possible countermeasures and gives a clear classification of proposals that have appeared in literature. In Sect. 6.6 methods are analyzed to implement countermeasures at the protocol level for typical applications. We conclude and summarize the contributions of this chapter in Sect. 6.7.

6.2 History

In [86] power attacks were introduced. Power attacks are side channel attacks; they exploit characteristics of the implementation of an algorithm. This is different from 'classical' cryptanalysis, where an attacker only uses information of the known input and/or output of the algorithm. Power attacks exploit differences in the power consumption of the implementation of a cryptographic algorithm during cryptographic computations. By analyzing the power consumption information about the secret key of a cryptographic device can be extracted.

Especially vulnerable against this kind of attacks are smart cards. This is caused mainly by the fact that smart cards get their power externally. To mount a power attack an attacker mainly needs the following: a smart card reader, a device (e.a. an oscilloscope) that can measure the power consumption accurately enough and sufficient computing power to analyze the measurements. He then proceeds with sending commands to perform cryptographic computations to the smart card via the reader. These commands are mostly standardized. Successful attacks have been reported that only need a few hundred of power samples and the contemporary opinion is that this can be brought down considerably.

The compromise of key material of a smart card can have several consequences, depending on the function of the card and the design of the infrastructure of which the card is a component. Possible consequences are:

- An attacker can modify the memory of the card, i.e., recharge its electronic wallet for free, increase the maximum limit of his wallet, change the program such that no recharging is required, etc.
- An attacker can make additional cards.
- An attacker can perform transactions under a false identity, for example make money transactions from the card owner's bank account to his own.

Many smart cards have been reported to be vulnerable. Paul Kocher has publicly stated that with his DPA technique he managed to break essentially all the types of smart cards deployed so far by financial institutions [128]. Note that this implies that many smart cards using implementations of triple-DES in a typical protocol setting are vulnerable. The choice of the algorithm, its implementation and the design of the protocols were done before power attacks were known. Hence, not only the choice of hardware is an issue in providing security against power attacks.

Since a few years much research is being done to find countermeasures. This research meets resistance from several factors. At first smart cards are limited devices, i.e., they have a limited amount of memory and relatively slow processors. Hence there is little possibility for major changes to the software. Secondly, several countermeasures have been proposed that involve using new techniques (new algorithms, new hardware). But the development and testing of these techniques will undoubtedly require much time.

The difficulty of this issue has been illustrated during the AES selection process: a procedure initiated by NIST to replace the DES as block cipher standard [2, 56]. There, discussions around the theme of DPA resistance and the difficulties of this resulted in a shift to put lower priority on the smart card suitability of the candidate algorithms.

In this chapter we give an overview of the developments in the area of power analysis. Furthermore we analyze, classify and introduce several possible countermeasures.

6.3 Smart Cards

In this section we give a brief introduction to smart cards. For an elaborate overview of cryptography on smart cards, we refer to the article by the author, Preneel and Rijmen [21].

Smart cards are small, portable, tamper-resistant devices containing a central processing unit and storage capability. They are widely used in applications such as electronic purses, GSM SIM cards, identification and ecommerce. To provide the services offered by these applications, many security objectives have to be met. To achieve this in an efficient way cryptographic tools are used: cryptographic primitives and protocols are implemented on the card and the card contains key material that has to remain secret.

Most smart cards have several constraints that have a considerable influence on the design and programming of its software architecture in an efficient and secure way. The most important factors are the limited capabilities of the CPU and the limited amount of memory. We summarize them as follows.

- **8-bit CPU:** Typically a smart card contains an 8-bit CPU, that is clocked at 3.57 MHz. Currently there are also smart cards on the market on which a 32-bit CPU is implemented and smart cards with optional dedicated hardware to perform efficiently the mathematical computations required for public key cryptography, but their cost is considerably higher. Hence, the majority of used smart cards contain 8-bit CPU's.
- **ROM:** Typically the ROM can contain between 16 and 32 Kbyte. It is used to store the code of an algorithm.
- **EEPROM:** Typically EEPROM can contain between 4 and 32 Kbyte. It is used to store long term keys.
- **RAM:** Typically the RAM can contain between 256 byte and 2 Kbyte. RAM is used to store variables, intermediate values and session keys.

Tradeoffs between the use of memory are often made to enhance the efficiency of cryptographic algorithms or to store them in a compact way on a card on which also other applications need to be implemented. RAM usage can be traded in for time. Typically RAM usage for intermediate values and variables increases the performance of an algorithm. However, it is possible not to store those values in RAM by recalculating them each time they are needed. The EEPROM can as well be used for the storage of intermediate values, but I/O operations in EEPROM are considerably slower. Furthermore, ROM usage can be traded in for time. Typically, an efficient implementation of an algorithm requires more ROM usage.

The 8-bit CPU puts restrictions on the design and implementation of an algorithm: the algorithms used should have a good performance on the 8-bit architecture and should be implementable with a limited code size. This can be achieved efficiently by using or designing algorithms that make use of 8-bit operations. Examples of such algorithms are Rijndael and CRYPTON.

Many aspects of the architecture of a smart card and smart card reader, as size, contact location, voltage and current values, file structure, etc., are standardized in ISO 7810 and ISO/IEC 7816. The format of the communication protocols between smart card and terminal are described in part 4 [65]. Furthermore EMV has published standards concerning payment protocols that can be acquired via [45].

6.4 Power Attacks

Power attacks exploit differences in the power consumption of the implementation of a cryptographic algorithm during encryption or decryption. The power consumption P (or $P(t)$) at each moment (time t) during operation of an algorithm can be written as

$$P = P_{\text{instruction}} + P_{\text{data}} + P_{\text{noise}}, \tag{6.1}$$

i.e., the sum of the power consumptions caused by the performed instructions, the manipulation and storage of data, and some noise. Typically, it holds that

$$P_{\text{instruction}} > P_{\text{noise}} > P_{\text{data}}. \tag{6.2}$$

Most of the difference in P_{data} is caused by a difference in power consumption between the manipulation and/or storage of ones or zeroes. This is caused by several effects. Most cryptographic devices that may be vulnerable to power attacks only consume power when some change occurs in the logic state. Examples of locations where these changes occur are: contents of the RAM, internal registers, bus-lines, states of logic gates and transistors, etc. Such changes occur every clock cycle. Their occurrence, timing and power consumption depend on the physical structure of the device, and may be influenced by external factors. It seems very difficult to construct a general and efficiently usable model for all different situations, that in practice might occur, although an attempt has been made by Chari *et al.* in [26].

The following kinds of power attacks can be distinguished.

Simple Power Analysis (SPA). From the measured power consumption $P(t), 0 \leq t \leq T$ of a cryptographic operation, the execution time of individual or groups of instructions are determined. Hence, this attack exploits the differences in $P_{\text{instruction}}$ and thus typically needs few samples.

An example of this is determining the sixteen DES rounds in a non SPA resistant DES implementation. The DES round function consists of an expansion (of 32 to 48 bits), 8 table look ups (which map 6 to 4 bits) and a bit permutation on 32 bits. In a graph of the power consumption given as a function of time the sixteen rounds of DES can be distinguished. This can even be done visually. This is not an attack in the sense that it deduces key material, but it can be used as a method to find out where the cryptographic computations are performed in the time period between the beginning (send command to card) and end of a transaction (receive response).

Another simple example concerns cryptographic algorithms that use a 'multiply and square'-algorithm to perform an exponentiation with key material in a group operation. This is the case with RSA and algorithms based on the (elliptic curves) discrete logarithm problem. A simple multiply and square algorithm performs either a multiply or a square based on the value of a bit of the secret exponent. Hence, if multiplying and squaring have a different power consumption pattern one can easily deduce the exponent.

Differential Power Analysis (DPA). This is a statistical attack that needs several samples of the power consumption of a cryptographic operation. These samples are combined and analyzed in order to average out $P_{\text{instruction}}$ and P_{noise} from Equation (6.1) and deduce key information from the remaining P_{data}. Differential power attacks on DES implementations are described by Kocher *et al.* in [86] and by Chari *et al.* in [26].

To illustrate how these attacks work, we describe the following attack on DES. The input of an S-box in the first round can be described as $P \oplus K$ where P and K are 6 bits of respectively the plaintext and the key. Suppose we have obtained N samples. There are $2^6 = 64$ possible values for K. For each of these values we can compute the output of the S-box for all N samples. Thus in 64 ways we can divide up the N samples in two sets, depending on the value (0 or 1) of a certain output bit of the S-box (this is possible for each of the 4 S-box output bits). For the right guess for K the power consumption for the two sets can be distinguished at the places where this bit is manipulated or stored. Hence, in this way 6 bits of the key can be determined. Note that this attack can also be mounted if the ciphertext is known; then one analyzes computations in the last round.

In [86] also the term higher order differential power attacks was introduced. In this scenario an attacker would exploit a combination of different parts of the algorithm, occurring at different times of the execution, where the same key material is used. If such an attack takes into account n times in the execution time, it is called an n-th order differential power attack.

These attacks typically need considerably more samples than ordinary DPA attacks. In [32] Daemen, Peeters and Van Assche made the following estimation: If z power consumption patterns are needed for normal DPA, then about $2z^2$

samples are needed for second order DPA.

Inferential Power Analysis (IPA). This kind of attack was introduced by Biham and Shamir in [10], and generalized as well as given its name by Fahn and Pearson in [48]. An interesting property is that in principle such an attack can be mounted without knowledge about plaintext or ciphertext. An attack consists of two phases:

1. An attacker first performs measurements of cryptographic computations that use different key material; for example on different cards (of the same type) or after key updates. Analyzing these samples one finds that on the places where they differ the most the key material is used in the operations.
2. After Stage 1 the attacker has knowledge where to find the key related operations. That can be used to obtain information about the key.

For example, Biham and Shamir describe an attack on DES in [10] which works as follows. In step 1 the places are determined where all subkey bytes are stored. For the target implementation the power consumption that is used for storage of a byte depends on its Hamming weight. Hence, one finds the Hamming weight of all subkey bytes. Due to the key schedule of DES it is then possible to deduce the whole key. Note that from this also follows that the key schedule is a relevant factor in the resistance against these attacks.

First order DPA can be viewed as the most powerful type of attack. Higher order DPA requires significantly more samples and seems to be adequately addressed by some countermeasures that also address first order DPA. Often SPA does not necessarily have to be prevented and IPA does not reveal sufficient key information. In the next section we discuss possible countermeasures, with an emphasis on countermeasures against DPA.

6.5 Countermeasures

If no countermeasures are taken, power analysis may be very effective against many applications. Smart cards are a particularly suited target, because they obtain their power externally and implementing countermeasures is difficult due to the small memory size that is available for the code.

In order to mount successfully a differential power attack on an implementation of a cryptographic primitive all of the following assumptions should hold.

Assumption 6.1 *A necessary condition for DPA to be feasible is that the platform on which the cryptographic primitive is implemented can perform different operations that have different power consumption patterns that depend on the data on which they are performed. We call these operations* DPA-weak.

Daemen and Rijmen noticed in [34] that there are gradations in the weakness or DPA-resistance of DPA-weak operations. Here, they also classified the original

15 candidates concerning the possibility to implement their DPA-weak operations in a sufficiently power resistant way. Probably similar classifications can be created for resistance against other implementation-based weaknesses (or might be created for newly discovered weaknesses).

Assumption 6.2 *A necessary condition for DPA to be feasible is that the implementation of the cryptographic primitive makes use of DPA-weak operations that involve both*

- *key material or values that can be directly deduced from key material, and*
- *known input or output values of the primitive or values that can be directly deduced from these known values. We call these values* DPA-exploitable values.

Assumption 6.3 *A necessary condition for DPA to be feasible is that an attacker can acquire* sufficient *and* sufficiently accurate *measurements of the part of the computation that involve key material and known values. The number of measurements that is sufficient depends on the platform and the accurateness of the measurements.*

Assumption 6.4 *A necessary condition for DPA to be feasible is that the measurements that are to be used for the attack involve computations of the cryptographic primitive for which the following holds.*

- *The DPA-exploitable values can be deduced from the information about the input or output of the computations. Furthermore, the DPA-exploitable values need to take on sufficiently different values. The attack is also feasible if input and output are not known, but variable according to a known scheme, such that the difference between any two inputs (outputs) is known.*
- *All computations have been done with the same key value or key values that vary according to a known scheme, such that the difference between any two key values is known and exploitable.*

Taking countermeasures means making at least one of the above assumptions practically infeasible.

Countermeasures against DPA are possible at the following levels. The first three are mostly mentioned in the open literature. In [23] the author, Preneel and Vandewalle introduced a fourth method, which previously received little attention.

- **At the hardware level:** Here one wants to make Assumption 6.1 possibly in combination with 6.3 infeasible. The following countermeasures have been proposed.

- It is possible to increase P_{noise}, for example by implementing a random noise generator. The purpose is to make it more difficult to perform accurate measurements, hence this countermeasure also involves Assumption 6.3. Such a countermeasure typically increases the amount of samples needed for a successful attack; it may also have as effect that the attack needs to be performed with more accurate measurement equipment.
- Another countermeasure, *circuit balancing*, introduced by Daemen and Rijmen in [35] is modifying the hardware that performs the instructions to give a balanced power consumption, independent of the data. This seems to imply some disadvantages, i.e., increase of the chip area, increase of the total power consumption, and decrease of the performance.
- One can add a capacitor across the power supply to uniformize the power consumption. However, a capacitor that fits on a smart card does not uniformize the power consumption totally [128]. However, using extra countermeasures this approach seems viable.
- An idea is to add a sensor that measures fluctuations in the power consumption and tries to control it accordingly. However, the delay between fluctuation and reaction still leaves a (short) period in which fluctuations appear that can be exploited [128].
- A natural possibility is to use smart cards with their own power supply, i.e., on chip batteries. However, this is an expensive solution since batteries that fit in the card have a too limited life span. The use of rechargable batteries has as (additional) disadvantage that a rechargable battery that is not used for a while drains quickly. The required recharging period would make these cards impractical to use. In [128] Shamir introduced a similar idea that involved the use of two capacitors in a switching mode: one capacitor supplies power to the chip, the other is recharged by the external power supply. All methods have as disadvantage that they do not protect against attacks where an attacker can measure the internal power consumption or can somehow bypass the battery or capacitors and provide the chip externally with power.

The consequences of all strategies need to be scrutinized by analyzing and doing measurements on an actual implementation in hardware.

- **At the software level:** There are several ways to secure an implementation at the software level.
  - One can for example desynchronize the instructions randomly in such a way that power samples of different executions are hard to combine. The sequence of the instructions can be determined by a pseudo-random number generator in software or a hardware random generator. In the latter case we still classify it as a countermeasure at the

software level since software is involved to determine the sequence. Hence, this countermeasure concerns Assumption 6.3. However, one has to take care that with extra analysis of the power consumption of the instructions, it might be possible to synchronize them again. This seems an adequate method to counter higher order differential power attacks [32].

– One can also implement some key dependent instructions in such a way that key material is not directly combined with known values to counteract Assumption 6.2. An example of this is the **r-way split** as described and analyzed by Chari *et al.* in [26], which is also described and applied to DES by Goubin and Patarin in [60] under the name *The Duplication Method.* Here an alternative implementation for the algorithm is constructed, where the key material is not directly combined with known values. To achieve this the output of a cryptographic key dependent algorithm is computed in several 'streams'. The input of each stream corresponds to a randomly masked input value and its masks. The output of the streams can be combined to give the correct output with knowledge of the random mask. Hence, in the case of two streams the computation of a function f can be done using two functions f_1' and f_2' as follows:

$$f(P) = C \longrightarrow \langle f_1'(P \oplus r_1), f_2'(r_1) \rangle = \langle C \oplus r_2, r_2 \rangle,$$

where r_1, r_2 are random masks, that are either secret values stored on the card, i.e., extra secret keys, or are generated on the card by a random number generator. Note that for the last possibility an extra secret value is required to initialize the random number generator. The latter possibility seems to be preferable. Computing the xor of a secret value and a known value for each computation of the cryptographic primitive requires additional protection against power analysis.

This method typically increases the number of required samples. It was shown by Chari *et al.* in [26] that it provably increases the number of required samples for a differential power attack under two assumptions:

* An attacker can not gather information or take sufficient power samples from the moment that the input or output of the function is masked.
* No information about the random masks can be obtained by an attacker. Note that if the algorithm producing the random numbers is implemented on the card it should be resistant against (power) attacks.

• **At the algorithm level:** Choose, design or make changes to a cryptographic algorithm so that it is particularly suitable for power analysis resistant implementation. The design increases the barrier for Assumption 6.2 and 6.3 to hold.

Resistance against power attacks is based on the implementation. However, based on the algorithm specification, it is possible to determine the suitability of an algorithm for a power analysis resistant implementation. Which countermeasures are suitable for implementation depends on the algorithm and the implementation platform. Examples of this approach are:

  - Daemen, Peeters and Van Assche showed in [32] that bitslice ciphers as 3-Way, BaseKing and Serpent are particularly suited for an r-way split implementation.
  - The key schedule of Skipjack seems to fall into the category of favorable schedules with respect to the attacks as described in [10].
  - In Chapter 7 the block cipher design strategy of multiple layered security is described to construct ciphers that have an increased security against differential power analysis and side channel attacks in general.

- **At the protocol level:** At the protocol level one can put restrictions on the amount of executions of an algorithm to prevent an attacker from gathering sufficient power samples, hence a countermeasure against Assumption 6.4. The most obvious ways to do this are to change the key regularly or to update the key material such that it is difficult to combine different power samples with related keys. Contemporary protocol standards offer possibilities for efficient introduction of these changes, as they were not designed with resistance to power attacks in mind.

Which method is the best suited approach for a given application depends on the following practical issues.

- **Security** should be the first objective to meet. However, based on risk and loss analysis of the application as a whole, tradeoffs can be made with other objectives. Keeping this in mind, we shall take a conservative approach as motivated in Sect. 2.4.1.
- **Efficiency:** Since smart cards and other applications that are potentially vulnerable to this kind of attacks have severe restrictions considering performance and code size, this has to be taken into account.
- **Design process:** Here we make a distinction between two factors:
  - **Evaluation.** The security evaluation of a new countermeasure requires time and involvement of experts or the cryptographic community, before the trust in its effectiveness is established.
  - **Production process.** This is a factor that especially concerns hardware. Since the existing theoretic models to evaluate hardware on

its vulnerabilities for side channels are not sufficiently general, implementation based attacks need to be tested on the hardware. Hence, this can induce considerable impact on duration and cost of the design.

- **Cost** is partly influenced by the design process. Besides that the introduction of a countermeasure in an existing infrastructure can be costly if it cannot be done efficiently.

All methods have advantages and disadvantages. The introduction of new hardware seems very time consuming. For example the construction and implementation of a new processor could easily take a few months. Compared to the other countermeasures it probably would also be more expensive to replace existing hardware. Besides that it seems to bear considerable risks for the occurrence of information leakage via other side channels.

Changes at the software level usually result in a performance degradation and increased memory requirements, even to an extent that the code may become too large to fit in the ROM of a card.

The introduction of a new algorithm seems also time consuming. This is mainly due to the process of scrutinizing such an algorithm for its resistance against other cryptanalytic attacks, such as linear and differential cryptanalysis.

At the protocol level it is often possible to take sufficient countermeasures, that can be efficiently implemented and that are conservative (hence based on a considerable amount of theory and experience) from a security point of view.

We summarize the comparison of the four approaches in Table 6.1. It follows from the table that countermeasures at the protocol level deserve to be considered if possible. However, strong resistance against power attacks will certainly involve taking countermeasures at all levels if possible. And besides that it is very important to evaluate the implementation on the target platform.

Table 6.1: Comparison of the four approaches for countermeasures against power attacks.

Level	Security	Efficiency	Design process	Cost
Hardware	?	?	−	−
Software	+	−	−	+
Algorithm	?	+	−	+
Protocol	+	?	+	+

6.6 Countermeasures at the Protocol Level

In Sect. 6.5 we have seen that at the protocol level one can take countermeasures to achieve the following goals. In order to obtain DPA resistant applications, it cannot be tolerated that the hardware performs 'many' encryptions of (partially) known texts with the same key (Assumption 6.2). Furthermore, not 'too many' encryptions should occur of texts that vary according to a known scheme, with keys that vary according to a known scheme (Assumption 6.4). In this section we demonstrate how to take countermeasures at the protocol level for a popular type of scheme.

The remaining of this section is organized as follows. In Sect. 6.6.1 we describe a typical scheme where encryption on smart cards is used. In Sect. 6.6.2 we describe how to increase the resistance of this scheme against differential power attacks. Some performance issues are addressed in Sect. 6.6.3. Sections 6.6.4 and 6.6.5 describe variations on the scheme to resolve some performance issues.

6.6.1 Session Keys

In a typical application a smart card communicates with a terminal. Part of this communication, where for example a financial transaction is performed, is encrypted during transmission. This part is called a session. Usually, during a session symmetric encryption is used and usually in each session a different session key is used. Before each session there is a protocol in which the session key is established. Other features of this protocol mostly include identification from card to terminal, etc.

We describe a common basic scenario of session key establishment, which is used in most cryptographic smart card applications. A session key is derived from two values. The first is a master key MK, which is stored in a tamper resistant way on the smart card, such that an attacker can not read it. The other value is derived from some data R_i that is different for each session i, i.e., it usually includes a session counter. The derivation of the ith session key can be denoted as follows:

$$SK_i = F(MK, R_i). \tag{6.3}$$

Here, the function F should have the following property.

Property 6.1 *If the master key MK is not known, the function $F(MK,.)$ should be a one-way function.*

This property has as consequence that if one session key is compromised, the master key and other session keys are not directly compromised. Due to memory constraints of smart cards F is usually a one-way function based on a block cipher, such as the constructions in Sect. 2.6.3. Hence, MK is used as key or plaintext for encryption.

Due to the use of session keys the MK is less used as input for a cipher, compared to a situation where all encryption in all sessions would be performed with the MK. Hence, the conditions in assumptions 6.2 and 6.4 are less applicable, which means increased resistance against differential power attacks.

However, it still could be possible to mount attacks on the session key derivation. In the following sections we describe possibilities to increase the resistance of this scheme against differential power attacks, without making too many modifications.

6.6.2 Extra Key Levels

The scheme described in Sect. 6.6.1 uses two key levels. We describe how to introduce extra key levels for increasing resistance against differential power attacks.

According to (6.3) a certain number N of session keys SK is derived from the master key MK. Here, the master key represents one level and the session keys represent the other level. Between these two levels other levels can be inserted, containing keys that we call intermediate keys or IKs. Each of the intermediate keys is derived from an intermediate key at the previous level, except for the first level of intermediate keys that are derived from the MK. In this way we get a tree structure, where the session keys are derived from the last levels of IKs.

From each IK and from the MK a number of m IKs are derived. We denote the jth intermediate key at level i with $IK_{i,j}$. From each IK and from the MK a number of m IKs are derived. Hence, level i contains m^i intermediate keys and with N levels m^N session keys can be derived.

Generalizing 6.3 for the derivation of intermediate keys we get:

$$
\begin{aligned}
IK_{0,0} &= MK & (6.4)\\
IK_{i+1,j} &= F'(IK_{i,\lceil \frac{j}{m}\rceil}, R_{i,j}), \text{ for } i = 0,\ldots,N-1,\ j = 0,\ldots,m^i-1\,. & (6.5)\\
SK_j &= F(IK_{N,j}, R_{N,j}), \text{ for } j = 0,\ldots,m^N-1\,. & (6.6)
\end{aligned}
$$

Ideally, in this scheme, both F' and F should be one-way functions. If this would not be feasible, then at least F should be one-way, since typically more encryptions are performed with one session key than with an intermediate key, hence session keys are (at least in theory) more vulnerable to compromise.

Scheme using 3-DES. In most smart card applications for symmetric encryption 3-DES is implemented. This cipher has a 64-bit block length and uses 112-bit keys $K = K_L|K_R$, where K_L and K_R denote the left and right 56 bits of K. In this case we can denote the a possible derivation of $F(K,R) = K_L^+||K_R^+$ (or F') as

$$
\begin{aligned}
K_L^+ &= \text{trunc}_{56}(\text{3-DES}_K(f(R))) & (6.7)\\
K_R^+ &= \text{trunc}_{56}(\text{3-DES}_K(g(R)))\,, & (6.8)
\end{aligned}
$$

where trunc_{56} truncates its input by taking the leftmost 56 bits and f and g are different functions that give as output a 64-bit value. As an alternative representation, that will be useful later on we will leave out the truncation part in (6.7) and (6.8) and assume that 3-DES_K takes as input for K two 64-bit values, from which two 56-bit values are chosen as keys in the DES operations.

Hence, in this notation we have

$$K_L^+ = \text{3-DES}_K(f(R)) \quad (6.9)$$
$$K_R^+ = \text{3-DES}_K(g(R))\,, \quad (6.10)$$

where K_L^+ and K_R^+ are 64-bit values. It is also possible to xor the right halves of (6.9) and (6.10) with respectively $f(R)$ and $g(R)$. In this way a Davies-Meyer construction is obtained. However, if R is a known value, this is superfluous.

6.6.3 Performance Issues

For resistance against DPA, the optimal way to implement the scheme is to choose the number m of derived *IK*s from one *IK* equal to 2, and the level of N should be chosen large enough, such that the number of session keys m^N is large enough for the lifetime of a card.

The following remarks can be made about the performance.

- The terminal either needs to derive the required session key from the master key, or needs to make an on-line connection with another machine that can compute the required session key from the master key. This other machine usually would be part of a centralized server that performs these request for distributed terminals. Hence, the terminal or another machine needs to perform the derivation algorithm $n + 1$ times for each transaction. This should not pose a problem for a terminal, since usually more computing power is available than on a smart card. However, it could have consequences when a centralized machine is used in a widely implemented system: waiting times for transactions could be significantly longer in busy periods.

  Note that DPA resistance is less of an issue here, because it is difficult to perform power measurements at a terminal or at a centralized machine.

- The smart card also has to derive the required session key. If this whole derivation should begin with the master key the card needs to perform $n + 1$ derivations; this may result in too slow a performance.

A possible solution to the performance issues is that the smart card stores the most recent values of the intermediate keys together with the master key. Then the number of key derivations that the smart card has to perform, can still be $n + 1$, but is usually only 1 or 2. Hence n extra key slots are needed for every master key that is protected in this way. It is also possible to store only a subset of the intermediate keys, but then the expected number of derivations to be performed increases. However, this solution increases the amount of memory that is required on the smart card.

In Sect. 6.6.4 we describe a method that was proposed by Shamir to decrease the amount of required memory. In Sect. 6.6.5 we describe a method to decrease the amount of required key levels, keeping the same level of resistance against differential power attacks.

6.6.4 Key Text Alternation

The following scheme was proposed by Shamir as a modification to the scheme as described in Sect. 6.6.2. It decreases the required amount of intermediate key slots to two for a tree with any amount of key levels.

The basic idea is that 2 subsequent intermediate keys at level i and $i+1$ (for which one is derived from the other) are used to derive an intermediate key at the next level $i+2$ with an invertible derivation function. Hence, Equation (6.5) would be specified as

$$IK_{i+1,j} = F'(IK_{i,\lceil \frac{j}{m} \rceil}, IK_{i,\lceil \frac{j}{m^2} \rceil}, R_{i,j}), \text{ for } i = 0, \dots, N-1 \text{ and } j = 0, \dots, m^i - 1, \tag{6.11}$$

where $IK_{-1,0}$ is some given value. A similar adjustment can be done with (6.6) for the derivation of the session keys.

Scheme using 3-DES. In this notation 3-DES$_K$ has as key parameter K a 64-bit value, from which the 8 bits, that are used as parity bits, are ignored. Hence, the pair of equations (6.9) and (6.10) are completed as

$$\begin{aligned}
IK_{-1,0} &= IV \\
IK_{0,0} &= MK \\
IK_{i+1,j:L} &= \text{3-DES}_{IK_{i,\lceil \frac{j}{m} \rceil}}(IK_{i,\lceil \frac{j}{m^2} \rceil:L} \oplus j \bmod m)) \\
IK_{i+1,j:R} &= \text{3-DES}_{IK_{i,\lceil \frac{j}{m} \rceil}}(IK_{i,\lceil \frac{j}{m^2} \rceil:R} \oplus j \bmod m \oplus \texttt{0xf0}) \\
SK_{j:L} &= \text{3-DES}_{IK_{N,\lceil \frac{j}{m} \rceil}}(IK_{N-1,\lceil \frac{j}{m^2} \rceil:L} \oplus j \bmod m)) \oplus IK_{N-1,\lceil \frac{j}{m^2} \rceil:L} \\
SK_{j:R} &= \text{3-DES}_{IK_{N,\lceil \frac{j}{m} \rceil}}(IK_{N-1,\lceil \frac{j}{m^2} \rceil:R} \oplus j \bmod m \oplus \texttt{0xf0})) \oplus IK_{N-1,\lceil \frac{j}{m^2} \rceil:R}
\end{aligned}$$

where i and j can take on the values as given in the general scheme.

6.6.5 Key Bit Shuffling

The idea behind this method is to bring more diversity in the derivation of the values of $IK_{i+1,j}, IK_{i+1,j+1}, \dots$ from $IK_{i,j}$. This in turn will allow to reduce the number of intermediate key levels while keeping the same security level against power attacks.

We propose to replace the key derivation (6.5) with

$$IK_{i+1,j} = F'(\phi^{j \bmod m_i}(IK_{i,\lceil \frac{j}{m} \rceil}), R_{i,j}), \text{ for } i = 0, \dots, N-1,\ j = 0, \dots, m^i - 1. \tag{6.12}$$

The function ϕ gives a k-bit result, where k is the key length and has the following properties:

1. For all i, j with $i \neq j$ it is difficult to combine power measurements of encryptions calculated with $\phi^i(IK)$ and $\phi^j(IK)$ to perform DPA attacks, that find (information about) IK.

2. It is not possible to perform DPA on ϕ to find MK or any SK or IK.
3. ϕ can be implemented efficiently in the ROM of a smart card.

We shall give an example how to construct such a function for a scheme that uses 3-DES.

A scheme using 3-DES. Since the derivation now includes two 3-DES encryptions, it is even better to shuffle the key bits half-way the key derivation. Hence, using key shuffling as in equation (6.12) we get

$$K_L^+ = \text{3-DES}_{\phi^{2(j \bmod m_i)}(K)}(f(R)) \tag{6.13}$$

$$K_R^+ = \text{3-DES}_{\text{3-DES}(\phi^{2(j \bmod m_i)+1}(K))}(g(R)), \tag{6.14}$$

Hence ϕ should give a 112-bit output and should have the required properties. There are several candidates for ϕ. An obvious candidate would be $\phi = \text{trunc}_{112}(\text{SHA-1}(MK, i))$, however this function as well as any other 'complex' function would probably not meet property 2.

It turns out that the presence of PC-1 in the key schedule of DES ensures that simple choices for ϕ, such as a rotation, already provide a satisfactory solution against straightforward DPA-attacks. However, permutations might be vulnerable against slightly more advanced attacks, where an attacker uses his knowledge of the position of identical key bits in different keys.

We propose to shuffle the key bits of subsequent keys with an LFSR. For performance and security we choose it to be a byte-wise LFSR. A key contains 14 bytes, hence the polynomial that defines the LFSR needs to be of degree 14. Furthermore, we choose it to be primitive, since then the state of the LFSR only cycles after $2^{14} - 1$ iterations. In order to change all bytes of the key with every update, we iterate the LFSR 14 times with each update. We choose the polynomial that defines the LFSR to have 9 terms. In this way each bit of a key depends on at least 8 bits of the previous key. A possible polynomial is

$$x^{14} + x^{13} + x^{12} + x^{11} + x^8 + x^4 + x^2 + x + 1 \ . \tag{6.15}$$

Denote the 14 byte LFSR-register with $Y[13] \ldots Y[0]$. The LFSR-register can be updated as follows:

```
Do 14 times
    $t = Y[0] \oplus Y[1] \oplus Y[2] \oplus Y[3] \oplus Y[6] \oplus Y[10] \oplus Y[12] \oplus Y[13]$
    For $i = 0$ to 12 do $Y[i] = Y[i+1]$
    $Y[13] = t$
```

This choice of ϕ satisfies the required properties. Considering them point by point we get the following.

- **DPA on subsequently derived keys.** As long as the LFSR does not cycle, a DPA attack would be difficult to perform, since an attacker has insufficient information about the bits of subsequent keys. Since the LFSR cycles after $2^{14} - 1$ iterations and 14 iterations are made each update after $\lfloor \frac{2^{14}-1}{14} \rfloor = 1171$ updates one can point out two keys that have the same byte. Here $\lfloor x \rfloor$ denotes the largest integer smaller than or equal to x. Hence, after a few more than 1171 updates, a DPA attack can exploit similarities of the derived keys. Before 1171 updates the success probability of DPA is negligible.

- **DPA on ϕ.** An LFSR is due to its simple structure very well suited for DPA-resistant implementation. One can for example use an r-way split [26, 60] to implement the xor.
- **Performance.** The byte-wise character of ϕ makes it excellently implementable on a smart card. Furthermore, only $14 \cdot 7$ xors have to be performed for each key derivation.

Note that the particular choice for ϕ has not yet been implemented and neither have tests been done on a practical implementation. Hence, this could be an interesting topic for further research.

6.7 Conclusions

Power attacks exploit the variance in power consumption of the implementation of a cryptographic algorithm. Especially vulnerable to this kind of attacks are smart cards. Since their introduction by Kocher many implementations have found to be vulnerable. Hence, a substantial amount of research has been performed in order to develop methods that prevent power analysis or increase resistance against power analysis.

The resistance against differential power cryptanalysis can be improved at different levels of a system. This chapter presents a classification of countermeasures, which would be of assistance to an implementor of cryptographic algorithms or systems in determining the suitable implementation and/or platform. This is a first contribution of this chapter.

Also a conservative method is described, where power attacks are countered at the protocol level. Several approaches are described to implement countermeasures at the protocol level for a typical smart card application. This can be seen as the main contribution of this chapter.

Chapter 7

Multiple Layered Security

7.1 Introduction

This chapter introduces a new conservative block cipher design strategy and proposes a block cipher designed according to this strategy. The idea behind this strategy is to have a structure that contains several different strong block ciphers. Each cipher uses a set of different subkeys. The knowledge of one set of subkeys does not reveal any (useful) information about the other key sets. Hence, an attacker would be forced to use different attack methods. This strategy can be used to divide possible weaknesses over different components. The most important benefit of this strategy is that it can be used to enhance security against side-channel attacks. This chapter shows that it is possible to design an efficient cipher according to this strategy. A new block cipher based on Rijndael is proposed and its security analyzed. This block cipher proposal has been submitted to the NESSIE call for cryptographic primitives [104] under the name GRAND CRU.

The remainder of this chapter is organized as follows. In Sect. 7.2 we motivate the design strategy and introduce the concept of multiple layered security. In Sect. 7.3 the strategy is formalized. A concrete proposal of the design of a block cipher based on Rijndael is made in Sect. 7.4. Its implementation and security aspects are studied respectively in Sect. 7.5 and 7.6. An overview of advantages and limitations of the strategy is made in Sect. 7.7 and we conclude and summarize our contributions in Sect. 7.8.

7.2 Motivation

In the last few years block cipher design has received increased attention. This was caused by the initiative from NIST to replace DES by AES as a US federal standard [2, 56]. Fifteen algorithms were accepted and after a first selection process there remained five possible candidates: MARS, RC6$^{\text{TM}}$, Rijndael, Twofish and Serpent. NIST's final selection for AES was Rijndael.

The AES predecessor DES has survived more than 20 years of cryptanalysis and a conservative approach would have been to design the new AES according to similar principles, bearing in mind the progress in cryptanalytic research. However, DES was designed by IBM and NSA and its design criteria were never completely published. This has made the AES selection process an interesting stage for the evaluation of different design strategies. As this process is nearing its end it is interesting to look back at the selection process and arguments that led to the remaining candidates. One can say about the five remaining candidates that they are sufficiently efficient on many platforms and sufficiently secure against currently known attacks. The same can actually be said about several other ciphers from the group of original proposals. For example CAST-256, DFC, and E2 would be sufficiently secure and sufficiently efficient for many applications on most platforms. On the other hand, except for resistance against exhaustive key search, few things can (and have been) be said about possible attacks that are the result of evolving cryptanalytic research. However, one statement can be made, which we use as a working assumption.

Assumption 7.1 *The probability that one of a group of x (for example 5 ciphers, as in the case of 5 AES finalists) ciphers gets broken in the future is larger than the probability that all x would get broken.*

This assumption has had an impact on the AES selection process. Although the original purpose was to use one standard, soon comments were addressed to NIST about choosing multiple standards. At the third AES conference there was even a forum dedicated to this issue. We use Assumption 7.1 as the basis of a new design strategy: block cipher design with *multiple layered security.*

If one has a set of x unbroken (but not provably secure) ciphers, one can choose one and use it for encryption. However, there is always the possibility that the selected cipher gets broken and a large amount of encrypted information gets compromised. To decrease the probability of compromise of the encrypted information, one could use a chain of the x secure ciphers with independent keys (see for example Maurer and Massey [100]). However, this reduces the performance by a factor of about x or it would result in the need for about x times more hardware resources. In this chapter we suggest a similar approach from the security point of view, that still gives opportunity for efficient cipher design. *In particular it gives rise to much more efficient designs than compared to the use of a sequence of several ciphers.*

The basic idea is to fit x secure ciphers (*subciphers*) into a similar design, such that if $x - 1$ cipher keys are known still a secure cipher would remain. We call this notion of security *multiple layered security.* The user key would either have to be sufficiently long to contain the user key of each subcipher or the keys of the subciphers would need to be derived from the user key with one or several one-way functions, such that these keys would not be distinguishable from independently chosen keys. For the cipher to be implementable on platforms where limited memory for code is available, such as smart cards it would be preferable that the one-way function(s) use the same structure as the cipher.

Another feature of this design strategy is that it can be used to design block ciphers that are particularly suitable for implementations that resist side-channel attacks on a wide variety of platforms. In order to achieve this the subciphers should have different operations for involving the keys in the computation, since these are the operations whose information leakage via most practical side-channels is exploited.

In this chapter we formalize this design approach, discuss the different design strategies that could be combined and propose a design, based on Rijndael, NIST's final selection for AES.

7.3 Strategy

The concept of creating a more secure block cipher by chaining x secure ciphers is well known [127]. The most important idea behind it is that if $x - 1$ ciphers would get broken, one still would be left with a secure cipher. Another idea is that in this way one can create ciphers with larger key length, without having to redesign the key schedule. The most prominent example of this is triple-DES. The general acceptance of triple-DES as replacement for DES in applications where a 56-bit key does not suffice illustrates the confidence in this method. The main disadvantage of this strategy is the performance drop, i.e., a chain of x ciphers is approximately x times as slow as one cipher.

One does not have to chain x ciphers to combine them such that one would be left with a secure cipher if the keys of $x-1$ would be known. An example of this is DESX, which was proposed by Rivest (in 1984) to strengthen DES against exhaustive key search. It is described and analyzed by Kilian and Rogaway in [74]. The cipher is a DES, where 64-bit keys are xored to the input and output:

$$\mathrm{DESX}_{K_1||K_2||K_3}(X) = \mathrm{DES}_{K_1}(X \oplus K_2) \oplus K_3. \tag{7.1}$$

One can see this as an interleaved combination of DES and a cipher consisting out of two outer key mixings with a (known) confusion layer in between. The latter was already proposed by Shannon in [127]. Based on the method of outer key mixing there are some attacks possible; in the case of DESX see the analysis by Kilian and Rogaway in [74] and the attacks by Biryukov and Wagner in [14].

Another related strategy is the design strategy of the family of COCONUT ciphers as described by Vaudenay in [136]. A COCONUT cipher is a chain of ciphers $E_1(E_2(E_3))$, where E_2 (and hence the whole cipher) is perfectly decorrelated to the order two, hence provably secure against a large class of attacks.

The case of DESX is interesting since the keyed DES in the middle would represent an unkeyed confusion and diffusion layer, if its key was known. This leads to the following:

Design strategy.

1. Start with a secure block cipher (with well-defined designing strategy and analysis).

2. Change unkeyed operations in that cipher into keyed operations or add (keyed) operations such that:

   (a) The security of the original cipher is not compromised, i.e., the design principles of the original cipher and the motivations or proofs for security against certain attacks should still hold for the remaining cipher and should still hold if the keys of the extra operations are known. Note that in this case there can be different attack scenarios: *known subcipher key attacks* and *chosen subcipher key attacks.* The latter is only possible if the subcipher keys are independent (or can not be distinguished from independent keys).

   (b) One can divide the keys of the keyed operations into $x - 1$ sets. Together with the subkeys of the original cipher one has x sets. In this way x ciphers are defined. Ideally if any $x - 1$ sets where known, then the remaining cipher should still be secure (note that this implies the item 2.(a)). We say that the cipher provides *multiple layered security.* But if this condition is not met optimally, the designed cipher still can be useful. Again one can distinguish known and chosen subcipher key attacks.

   (c) There are no non-trivial alternative representations of the cipher in less than x subciphers.

3. Design efficient and secure key schedules for the $x - 1$ ciphers that had none.

4. The cipher is keyed by a set of keys $K_0, \ldots, K_{x-1}$, where

   - this set forms the user key; or
   - this set is derived from the user key with one or several one-way functions. This derivation should be done in such a way that if $x - 1$ derived keys are known, the user key should not be deducible.

5. The remaining cipher is still (sufficiently) efficient. In particular it should be at least as efficient as a sequence of x ciphers that provide approximately the same level of security.

6. The remaining cipher is secure against ad-hoc analysis.

The subciphers ideally complement each other from a security point of view. It is not wanted that a successful attack on one subcipher implies a successful attack on another subcipher. This means:

In a classical cryptanalytic sense. The security of the different subciphers should rely on different strategies. For example, one does not want two subciphers, for which the security against LC and DC is mainly based on the properties of data-dependent rotations.

With respect to side-channel analysis. This strategy can increase the resistance against side-channel attacks. Indeed, all practical side-channel attacks exploit side-channel information that is leaked by key dependent operations. Hence, it is important to implement or choose the key dependent operations such that they do not leak side-channel information. For the other operations this is of less importance: for example SPA is possible because all operations leak information through their power consumption, but it does not have to be exploitable, as was shown in Chapter 6. Different operations can give rise to exploitation by different side-channels. For example, the xors with keys in DES can be vulnerable to power analysis, while the data dependent rotations in RC5 for some implementations can leak information through timing analysis as shown by Handschuh and Heys in [61].

If necessary the operations that leak side-channel information can usually be implemented in a way such that there is no exploitable leakage. The techniques used may depend on the platform. On a specific platform, some operations might be more suitable for implementations resistant against side-channel weaknesses than others.

Hence, there are several advantages in choosing the key dependent operations in the different subciphers different from each other. Firstly, the resistance against side-channel attacks is guaranteed by the subcipher whose potential vulnerability can be most efficiently countered on the specific platform. Secondly, using several layers of different key dependent operations gives a higher probability against side-channel attacks that might evolve in the future. Hence, this would make it more improbable that a situation would occur where the smart card industry is forced to make a substantial investment due to a discovery such as power analysis.

Instead of choosing different key dependent operations one might allow key mixing with the same operations but use different code (or hardware) to implement these operations.

The above described design strategy is very conservative and intuitively comprehensible. However, to apply it in practice several hurdles have to be taken. Suppose we should choose DES in step 1. Since the design principles of DES are not known, there is little motivation to change or add operations to the internals. Hence, it is not possible to do much more than a DESX-like construction. Fortunately, not in the last place caused by the AES selection process, there are several alternative block ciphers, that have received much analysis and that are designed (or at least analyzed) with a clear design strategy. A little unfortunate, but not unexpected, is that none of these ciphers can be efficiently combined with each other. So a construction with one or more subciphers that is less analyzed will probably result.

Efficiency. The items of the design strategy mainly concern the security aspects of the design. Next we list some miscellaneous items that concern the efficiency of the design.

- The addition of extra operations should be minimized, since every extra operation decreases the performance.
- In many applications it would be beneficial if the extra-designed key derivation procedures and the one-way functions would use the same structure of operations as used in the algorithm. This would reduce the code size in applications with memory restrictions and the implementation cost for hardware applications.
- If the application also contains a secure one-way function such as SHA-1 [54] SHA-256, SHA-384, SHA-512 (currently in development, see [55]) or RIPEMD-160 [43], then these can be used for subcipher key derivation if there are no performance restrictions.
- The use of several subciphers leads to an increase of the amount of subkeys and hence will typically require more memory. The comparative high memory requirements seem to be inherent to this design strategy.

In the next section we will describe a combination of 4 subciphers based on the block cipher Rijndael, which is to become the new standard AES.

7.4 A Proposal

In this section we use the design strategy to construct a four-layered cipher, based on Rijndael. Hence, step 1 in the design strategy is to choose Rijndael.

7.4.1 Design Goals

We explicitly state the following principle, in which our design strategy differs from conventional design. *The efficiency of the design should be compared to ciphers that offer an equivalent level of security, i.e., multiple layered security. In our case: sequences of ciphers, that have different security motivations. With efficiency we mean: performance, code size, gate count, etc.* Concerning security and efficiency the design should have the following properties.

- The block size of the cipher is 128 bits: the nominal choice to resist dictionary type attacks,
- It can be used with a 128-bit user key, which is the minimal specified key length. Alternative key sizes (higher than 128 bits) can be defined.
- Each subcipher has a 128-bit key. These keys are derived from the user key using a one-way function. If the user key size is greater than $4 \cdot 128 = 512$ bits the one-way function does not have to be applied.
- The cipher is sufficiently resistant against all known shortcut attacks.

- Each subcipher is sufficiently resistant against all known shortcut attacks.
- The design should be efficient on platforms that are most vulnerable to side-channel analysis. The most suitable targets for this are smart cards. These mainly use 8-bit processors. Hence, for efficiency we keep the operations as much byte-oriented as possible.
- The code of the implementation of the cipher should fit in the memory a typical smart card. The code which would be needed to implement the inverse cipher does not have as much priority, since typical smart card applications only need an implementation of the cipher in encryption mode.
- For software implementation on other platforms the efficiency should be acceptable. In particular we do not emphasize possible optimization on general purpose 32-bit (or 64-bit) architectures since on these platforms, as Moore's law seems to be more applicable than on low end smart cards.
- The security goals of our design strategy also can be achieved by using several ciphers in sequence. Hence, a four-layered cipher should be more efficient as a sequence of 4 ciphers.

7.4.2 Rijndael

Rijndael, designed by Daemen and Rijmen [34], is selected by NIST for AES. Its design is based on the Wide Trail Strategy as described in Chapter 4. Rijndael consists of four transformations. All transformations operate on 8-byte strings, that we call *states*. The transformations each contribute to one of three layers as described in Chapter 5. The layers can be described as follows. We use the notations for the transformations as given by equations (5.1), (5.2) and (5.3). For Rijndael, an extra useful notation is to express a transformation $\lambda : \{0,1\}^{128} \rightarrow \{0,1\}^{128}$ as a transformation on the separate columns:

$$(x_3|x_2|x_1|x_0) \xrightarrow{\lambda} (y_3|y_2|y_1|y_0)\,. \tag{7.2}$$

Round key addition σ. This is performed by xoring the input with a 128-bit round key K_i:

$$\sigma(A)_{K_i} = A \oplus K_i, \text{ for } K_i, A \in \{0,1\}^{128}\,. \tag{7.3}$$

Non-linear substitution layer γ. This is called the ByteSub transformation in [34]. It uses an 8×8 S-box S, which operates on each of the bytes of the state independently:

$$\begin{pmatrix} x_{03} & x_{02} & x_{01} & x_{00} \\ x_{13} & x_{12} & x_{11} & x_{10} \\ x_{23} & x_{22} & x_{21} & x_{20} \\ x_{33} & x_{32} & x_{31} & x_{30} \end{pmatrix} \xrightarrow{\gamma} \begin{pmatrix} S(x_{03}) & S(x_{02}) & S(x_{01}) & S(x_{00}) \\ S(x_{13}) & S(x_{12}) & S(x_{11}) & S(x_{10}) \\ S(x_{23}) & S(x_{22}) & S(x_{21}) & S(x_{20}) \\ S(x_{33}) & S(x_{32}) & S(x_{31}) & S(x_{30}) \end{pmatrix}\,. \tag{7.4}$$

The S-box is constructed such that it has optimal properties against LC and DC. This is summarized by the following property.

Property 7.1 *With respect to linear and differential cryptanalysis, the Rijndael S-box has the following properties.*

- *The linear approximation with maximal bias for S has bias* 2^{-3}.
- *The differential with maximal probability for S has probability* 2^{-6}.

Furthermore the S-box is constructed such that its properties are hard to exploit in algebraic attacks like interpolation attacks.

Diffusion layers π and θ. The transformation π or ShiftRow transformation is a permutation of the bytes of a state. It operates on the rows of a state. It rotates the second row one, the third row two, and the fourth row three bytes. Hence, $\pi :\rightarrow \{0,1\}^{128}$ is defined as

$$\begin{pmatrix} x_{03} & x_{02} & x_{01} & x_{00} \\ x_{13} & x_{12} & x_{11} & x_{10} \\ x_{23} & x_{22} & x_{21} & x_{20} \\ x_{33} & x_{32} & x_{31} & x_{30} \end{pmatrix} \xrightarrow{\pi} \begin{pmatrix} x_{03} & x_{02} & x_{01} & x_{00} \\ x_{10} & x_{13} & x_{12} & x_{11} \\ x_{21} & x_{20} & x_{23} & x_{22} \\ x_{32} & x_{31} & x_{30} & x_{33} \end{pmatrix}. \tag{7.5}$$

For security its most important property is:

Property 7.2 *For π the four rotation amounts of the rows are different.*

This property ensures resistance against attacks using truncated differentials and resistance against the dedicated Square attack.

The transformation θ or MixColumn transformation operates separately on each column of a state:

$$(x_3|x_2|x_1|x_0) \xrightarrow{\theta} (\theta(x_3)|\theta(x_2)|\theta(x_1)|\theta(x_0))\,. \tag{7.6}$$

The columns of a state are considered as polynomials over GF(2^8) and multiplied modulo x^4+1 with a fixed polynomial $c(x)$. This can be represented as the following matrix multiplication:

$$\begin{pmatrix} x_{03} & x_{02} & x_{01} & x_{00} \\ x_{13} & x_{12} & x_{11} & x_{10} \\ x_{23} & x_{22} & x_{21} & x_{20} \\ x_{33} & x_{32} & x_{31} & x_{30} \end{pmatrix} \xrightarrow{\theta} \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_{03} & x_{02} & x_{01} & x_{00} \\ x_{13} & x_{12} & x_{11} & x_{10} \\ x_{23} & x_{22} & x_{21} & x_{20} \\ x_{33} & x_{32} & x_{31} & x_{30} \end{pmatrix}. \tag{7.7}$$

This transformation has a very important role with respect to the Wide Trail Strategy. It provides diffusion in the sense that it guarantees a high number of active S-boxes. For this the following property is important.

Property 7.3 *For each $i \in \{0,1,2,3\}$, θ_i has* branch number *5. This means:*

- *Each pair of corresponding four tuples of input bytes and output bytes of θ_i either contains at least 5 non-zero bytes or all are zero.*
- *Since θ_i is linear also for the difference of each two pairs of corresponding four tuples of input bytes and output bytes of θ_i it holds that either at least 5 bytes are non-zero or all bytes are zero.*

This property ensures that a linear approximation or a differential of one round always involves at least five active S-boxes.

Round transformation. One round ρ of Rijndael can be defined as

$$\rho_K = \theta \circ \pi \circ \gamma \circ \sigma_K\,. \tag{7.8}$$

There is a different final round ω:

$$\omega_{K_9,K_{10}} = \sigma_{K_{10}} \circ \pi \circ \gamma \circ \sigma_{K_9}\,. \tag{7.9}$$

A Rijndael encryption with a 128-bit key consists of 9 rounds ρ_{K_i} followed by one round $\omega_{K_9,K_{10}}$, where $K_0^0, \ldots, K_{10}^0$ are 128-bit round keys derived from the user key K^0:

$$\text{Rijndael}_{K^0} = \sigma_{K_{10}^0} \circ \pi \circ \gamma \circ \sigma_{K_9} \circ \prod_{i=8}^{0} (\theta \circ \pi \circ \gamma \circ \sigma_{K_i^0})\,, \tag{7.10}$$

where $\prod$ stands for concatenation of transformations. Rijndael decryption has the same structure It can be given by

$$\text{Rijndael}_{K^0}^{-1} = \sigma_{K_0^0} \circ \pi^{-1} \circ \gamma^{-1} \circ \sigma_{K_1} \circ \prod_{i=2}^{10} (\theta \circ \pi^{-1} \circ \gamma^{-1} \circ \sigma_{K_i^0})\,, \tag{7.11}$$

Key schedule. The key schedule is defined by a transformation $f_{\text{rk}} : \{0,1\}^k \to \{0,1\}^k$ that derives as many k-bit strings as necessary to form all subkeys. Each k-bit string is derived by applying ψ on the previous one. This procedure has the advantage that during encryption with Rijndael only one k-bit string has to be stored for key material; the needed key subkeys can be efficiently computed *on-the-fly*. For a 128-bit key the first subkey is the user key and the subkey i is derived from subkey $i-1$.

We describe the key schedule for a user key of 128 bits. For a description for other key lengths we refer to [33]. Similar as for the cipher states such 128-bit subkey can also be represented by a 4×4-array:

$$k_{33}k_{32}k_{31}k_{30}k_{23}\ldots k_{01}k_{00} = \begin{pmatrix} k_{03} & k_{02} & k_{01} & k_{00} \\ k_{13} & k_{12} & k_{11} & k_{10} \\ k_{23} & k_{22} & k_{21} & k_{20} \\ k_{33} & k_{32} & k_{31} & k_{30} \end{pmatrix} = (k_3|k_2|k_1|k_0)\,. \tag{7.12}$$

We define the transformation Rotl, which operates on columns of a key state. It rotates the bytes of a column as follows:

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \xrightarrow{\text{Rotl}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_0 \end{pmatrix} . \tag{7.13}$$

Furthermore the transformation Γ or SubByte also operates on the bytes of a column and is defined as:

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \xrightarrow{\Gamma} \begin{pmatrix} S(x_0) \\ S(x_1) \\ S(x_2) \\ S(x_3) \end{pmatrix} . \tag{7.14}$$

Rijndael with a 128-bit key length requires 11 128-bit subkeys that we denote with

$$K_{10} \ldots K_1, K_0 = (K'_{43}|K'_{42}|K'_{41}|K'_{40}), \ldots , (K'_7|K'_6|K'_5|K'_4), (K'_3|K'_2|K'_1|K'_0) . \tag{7.15}$$

If the user key is denoted by K^u then

$$K_0 = K^u . \tag{7.16}$$

The remaining key material is derived by

$$K'_i = K'_{i-4} \oplus \Gamma(\text{Rotl}(K'_{i-1}) \oplus c(i/4), \text{ if } 4|i, \tag{7.17}$$

$$K'_i = K'_{i-4} \oplus K'_{i-1} \text{ else}, \tag{7.18}$$

where $c(i)$ are the round constants of the key schedule as defined in Table 7.1. In the following we shall use a shortened notation to denote the ith subkey that is derived from the user key K^u:

$$K_i = f_{\text{rk}}(K^u, i) . \tag{7.19}$$

The key schedule has the following property which is of importance in applications where the memory available for subkeys is restricted.

Property 7.4 *The Rijndael key schedule permits* on-the-fly subkey computation. *In each round only a 128-bit subkey needs to be stored. The next subkey can be computed from the previous one.*

7.4.3 Security Motivations

The formal arguments for the security of Rijndael are based on Properties 7.1, 7.2 and 7.3. Combined with the structure of Rijndael, i.e., the specified sequence of transformations, they provide arguments for its resistance against linear and differential cryptanalysis and similar attacks, as well as more dedicated

Table 7.1: Round constants of the Rijndael key schedule, which are defined for up to 30 rounds.

i	$c(i)$	i	$c(i)$	i	$c(i)$
1	0x01	11	0x6c	21	0x97
2	0x02	12	0xd8	22	0x35
3	0x04	13	0xab	23	0x6a
4	0x08	14	0x4d	24	0xd4
5	0x10	15	0x9a	25	0xb3
6	0x20	16	0x2f	26	0x7d
7	0x40	17	0x5e	27	0xfa
8	0x80	18	0xbc	28	0xef
9	0x1b	19	0x63	29	0xc5
10	0x36	20	0xc6	30	0x91

structural attacks such as interpolation attacks and attacks using truncated differentials, which would exploit the byte oriented design.

Step 2 in the design strategy is to change unkeyed permutations into keyed operations and/or add keyed operations without violating the original security motivations, which here are the properties 7.1, 7.2, and 7.3. We make and motivate the changes in the following sections.

7.4.4 Keyed Permutations

This security motivations still hold when we make the following changes.

- Replace π with a keyed byte permutation π_K^1. This is a permutation that rotates three rows of the state with respectively one, two or three bytes. Which row is rotated by which amount is determined by the value of K. This key can take on $4! = 24$ different possible values. The permutation π_K^1 has the same property as given by Property 7.2: all the rotation amounts of the rows are different. It does not interfere with the other security motivations.

- Add before each permutation π_K^1 a keyed byte permutation π_K^2, that operates column-wise on a state and permutes the bytes of each column, depending on the value of K. This key can take on $(4!)^4$ possible values. This addition does not interfere with Property 7.1 since it only permutes bytes. It does not interfere with Property 7.3: rearranging the bytes in a column does not change the branch number, i.e., the number of non-zero bytes stays the same; they only might be at a different position. Finally it does not interfere with Property 7.2. In the equation representing the cipher each π_K^2 is the neighbor of a θ or can be shifted to be one, without altering the outcome of the cipher. Hence, one can view this addition as changing θ for $\theta \circ \pi_K^2$, which does not interfere with π_K^2 or its (relevant) properties.

Both permutations can be combined in one keyed permutation π_K, where K can take on $(4!)^5 \approx 2^{22}$ possible values. We propose to replace π with π_K. As motivated, this can be done without compromising the security arguments of Rijndael.

The permutation can be implemented by representing each of the five separate permutations on four bytes or rows by 5 key bits that represent a value smaller than 24. A table of 24 entries (that could have a length of 6 bits, but 8 bits seems more practical) can be used to represent all possible permutations. We call this table t and its ith element t_i. Each element is denoted by:

$$t_i = t_i[3]t_i[2]t_i[1]t_i[0], \text{ where } \{t_i[3], t_i[2], t_i[1], t_i[0]\} = \{0, 1, 2, 3\}.$$

The hexadecimal representation of all 24 elements is given in Table 7.2. Now π_K can be defined as a permutation that consists of 5 stages. Each stage depends on the value of one of the five key bytes of $K = k_4 \dots k_0$. The first stage is the rotation of the rows:

$$\begin{pmatrix} x_{03} & x_{02} & x_{01} & x_{00} \\ x_{13} & x_{12} & x_{11} & x_{10} \\ x_{23} & x_{22} & x_{21} & x_{20} \\ x_{33} & x_{32} & x_{31} & x_{30} \end{pmatrix} \xrightarrow{\pi_{k_0}^1} \begin{pmatrix} x_{t_{k_0}[0]3} & x_{t_{k_0}[0]2} & x_{t_{k_0}[0]1} & x_{t_{k_0}[0]0} \\ x_{t_{k_0}[1]0} & x_{t_{k_0}[1]3} & x_{t_{k_0}[1]2} & x_{t_{k_0}[1]1} \\ x_{t_{k_0}[2]1} & x_{t_{k_0}[2]0} & x_{t_{k_0}[2]3} & x_{t_{k_0}[2]2} \\ x_{t_{k_0}[3]2} & x_{t_{k_0}[3]1} & x_{t_{k_0}[3]0} & x_{t_{k_0}[3]3} \end{pmatrix}. \tag{7.20}$$

In the following four stages each column is permuted. For example, column 0 is permuted according to the value of k_1:

$$\begin{pmatrix} x_{03} & x_{02} & x_{01} & x_{00} \\ x_{13} & x_{12} & x_{11} & x_{10} \\ x_{23} & x_{22} & x_{21} & x_{20} \\ x_{33} & x_{32} & x_{31} & x_{30} \end{pmatrix} \xrightarrow{\pi_K} \begin{pmatrix} x_{03} & x_{02} & x_{01} & x_{t_{k_0}[1]0} \\ x_{13} & x_{12} & x_{11} & x_{t_{k_0}[1]0} \\ x_{23} & x_{22} & x_{21} & x_{t_{k_0}[1]0} \\ x_{33} & x_{32} & x_{31} & x_{t_{k_0}[1]0} \end{pmatrix}. \tag{7.21}$$

Table 7.2: Hexadecimal representation of the table for the keyed permutations.

i	t_i	i	t_i	i	t_i
1	0x78	9	0x87	17	0x6c
2	0x36	10	0x63	18	0xd8
3	0x2d	11	0xd2	19	0xe1
4	0xc9	12	0x39	20	0xc6
5	0x93	13	0x8d	21	0x27
6	0x72	14	0x1e	22	0xb1
7	0x4b	15	0xe4	23	0x4e
8	0x9c	16	0xb4	24	0x1b

With this change we have the cipher

$$\mathrm{E}_{K^0||K^1} = \sigma_{K_9^0} \circ \pi_{K_9^1} \circ \gamma \circ \prod_{i=8}^{0} (\sigma_{K_i^0} \circ \theta \circ \pi_{K_i^1} \circ \gamma)\,. \tag{7.22}$$

7.4.5 Keyed Byte-Wise Rotations

Another change that does not violate the security motivations is the addition of a (keyed) transformation that operates on each byte and does not compromise the properties of the S-box. Since it operates only on bytes it does not interfere with Property 7.2 and 7.3. To keep Property 7.1 intact a sufficient condition is that it only permutes bits. We propose to insert the keyed transformation β_K that rotates each byte of a state with an amount that depends on the value of K. There are $8^{16} = 2^{48}$ possible values for K. We propose to insert β_K between π and γ in (7.8) and (7.9).

The rotations are determined by the value of the 48-bit key K as follows. Let $X = x_{15}||\ldots||x_0$ denote a 16-byte state and $K[x+l..x]$ the radix-2 value formed by the bits $x+l$ to x of K, then β_K is defined as

$$\beta_K(X) = (x_{15} \lll K[47..45])||\ldots||(x_0 \lll K[2..0])\,. \tag{7.23}$$

With this change we have the cipher

$$\mathrm{E}_{K^0||K^1||K^2} = \sigma_{K_9^0} \circ \beta_{K_9^2} \circ \pi_{K_9^1} \circ \gamma \circ \prod_{i=8}^{0} (\sigma_{K_i^0} \circ \beta_{K_i^2} \circ \theta \circ \pi_{K_i^1} \circ \gamma)\,. \tag{7.24}$$

Note that for reasons of efficiency it also might have been possible to perform key-dependent rotations only on a subset of the 16 bytes.

The change of the diffusion layer also implies that the main concerns of Murphy and Robshaw as described in [103] are resolved. They described a slightly modified version of this layer for which sixteen subsequent applications are equivalent to the identity transformation. Since for our design the diffusion layer depends on the key (of each round) such simplification would be key-dependent and only applicable for a negligible amount of keys.

7.4.6 Shannon's Construction

As a (final) addition to the cipher we propose a DESX-like 'simple' key mixing as inner and outer layer. This does not interfere with any of the security motivations of Rijndael. We do not use the DESX construction of xoring a key of block length before and after the Rijndael-like cipher for the following reasons:

- Rijndael already mixes the key in the cipher with xors. As motivated in Sect. 7.3 it is preferable to have diverse methods of key mixing.
- Just xoring keys would give opportunity to mount advanced slide attacks as described by Biryukov and Wagner in [14]. See also Sect. 7.6.3.

Instead of that we choose in the outer rounds to mix a 128-bit key byte-wise using addition modulo 2^8. We denote it with ψ_K:

$$\begin{pmatrix} x_{03} & x_{02} & x_{01} & x_{00} \\ x_{13} & x_{12} & x_{11} & x_{10} \\ x_{23} & x_{22} & x_{21} & x_{20} \\ x_{33} & x_{32} & x_{31} & x_{30} \end{pmatrix} \xrightarrow{\psi_K} \begin{pmatrix} x_{03}+k_{03} & x_{02}+k_{02} & x_{01}+k_{01} & x_{00}+k_{00} \\ x_{13}+k_{13} & x_{12}+k_{12} & x_{11}+k_{11} & x_{10}+k_{10} \\ x_{23}+k_{23} & x_{22}+k_{22} & x_{21}+k_{21} & x_{20}+k_{20} \\ x_{33}+k_{33} & x_{32}+k_{32} & x_{31}+k_{31} & x_{30}+k_{30} \end{pmatrix}, \tag{7.25}$$

where $+$ denotes addition modulo 2^8.

Because this mixing is byte-oriented reduced versions of the remaining cipher would be equally vulnerable as Rijndael to Square-like attacks, that are described in [2, 6, 49, 59, 93]. These kind of attacks exploit the slow diffusion of a cipher. In our construction these attacks can be made infeasible by adding a diffusion layer between the outer round key mixing and the first transformation of the Rijndael structure γ.

A similar approach was used in the design of former AES candidate MARS. We propose the following transformation $\nu : \{0,1\}^{128} \to \{0,1\}^{128}$. Let the input of ν be denoted by $x_{15} \dots x_0$, where each x_i is a byte. Then the output $x_{15} \dots x_0$ is derived iteratively by:

```
1. For $i = 0$ to 15 do
       $x_{i+1 \bmod 16} = x_{i+1 \bmod 16} \oplus S(x_i)$.
2. For $i = 0$ to 15 do
       For $j = 0$ to 15, $j \neq i$, do
           $x_j = x_j \oplus S(x_i)$.
```

The inverse transformation ν^{-1} is given by:

```
1. For $i = 15$ to 0 do
       For $j = 0$ to 15, $j \neq i$, do
           $x_j = x_j \oplus S(x_i)$.
       $x_{i+1 \bmod 16} = x_{i+1 \bmod 16} \oplus S(x_i)$.
2. For $i = 15$ to 0 do
       $x_{i+1 \bmod 16} = x_{i+1 \bmod 16} \oplus S(x_i)$.
```

This transformation has the following properties.

- Each byte of the output depends on the value of every input byte and every output byte. Hence, one could say it provides good diffusion and a fast avalanche of key bits. This makes the application of outer round tricks more difficult, since they would involve guessing the entire 128-bit outer round key.
- Changing one input byte causes a change of all output bytes with high probability. This is caused by the previous item and the involved use of

S-boxes and xors. It increases resistance against the dedicated Square attack and differential attacks.

- Due to the extensive use of the Rijndael S-box there are no predictable changes of small inputs with high probability or low probability. This increases resistance against attacks using truncated differentials.

7.4.7 The Cipher

After this change we have the final cipher

$$\begin{aligned} \mathrm{E}_K &= \psi_{K^3_1} \circ \nu^{-1} \circ \sigma_{K^0_{10}} \circ \beta_{K^2_9} \circ \pi_{K^1_9} \circ \gamma \circ \sigma_{K^0_9} \circ \\ &\quad \prod_{i=8}^{0} (\beta_{K^2_i} \circ \theta \circ \pi_{K^1_i} \circ \gamma \circ \sigma_{K^0_i}) \circ \nu \circ \psi_{K^3_0} \, . \qquad (7.26) \end{aligned}$$

Note that, contrary to Rijndael, it is not possible to have an equivalent inverse cipher structure.

7.4.8 The One-Way Functions

The cipher requires four 128-bit keys to derive the subkeys for the transformations ψ, σ, π and β. We denote these keys with K^0, K^1, K^2 and K^3 for respectively ψ, σ, π and β, hence in the order of which they are applied in the encryption. These keys should be indistinguishable from independent keys as motivated in Sect. 7.2. To derive the subcipher keys from the user key, when the key length is smaller than 512 bits, we propose to use the following simple construction.

$$K^i = \mathrm{trunc}_{128}(F(K, i)) \text{ for } i = 0, \ldots, 3 \, , \qquad (7.27)$$

where F is a suitable one-way function and trunc_{128} truncates the result to 128 bits, by taking the leftmost 128 bits, if necessary. For F we propose two possibilities.

A dedicated one-way function. It is very reasonable to choose a dedicated one-way function f in the construction of F:

$$F(K, i) = f(K \| i), \text{ for } i = 0, \ldots, 3 \, , \qquad (7.28)$$

where f can be chosen from MD5, SHA-1, RIPEMD-160, or other proposals. The reason to do this is that for many applications also a dedicated hash function is implemented.

The cipher in one-way function mode. To save on memory by implementing an extra one-way function and to have a one-way function based on the same security principles as the block cipher one can use the cipher itself in a one-way function mode. We propose the use of a one-way function mode inspired by the Matyas-Meyer-Oseas hash function construction, as described in Sect. 2.6.3. First an alternative method has to be described to specify the subcipher keys. If one uses the cipher with a 512-bit key, one can derive the 4 subcipher keys by taking the subsequent 128 bits of the user key. Hence, we propose the use of a 512-bit key that depends on i. Define the 8-byte constant

$$c = \texttt{0x0123456789abcdef}.$$

Note that c is a string of 128 bits. Now we define the function F as

$$F(K, i) = E_{(c|c|c|c)}(K \ll 4 * i) \oplus (K \ll 4 * i) \tag{7.29}$$

if K has 128 bits. This function has the properties of the Matyas-Meyer-Oseas construction for a given IV.

For the case that the user key K is not a multiple of 128-bit values padding is necessary. We propose to concatenate zeroes to the left of K until $0 \ldots 0K = \tilde{K}_0 \ldots \tilde{K}_{l-1}$ has a length of $l \cdot 128$ bits and every $\tilde{K}$ has a length of 128 bits. Then K_i can be computed according to the Matyas-Meyer-Oseas construction.

$$\begin{aligned} F_0 &= E_{(c|c|c|c)}(\tilde{K}_0 \ll 4 * i) && (7.30)\\ F_{j+1} &= E_{(c|c|c|c)}(\tilde{K}_j \ll 4 * i) \oplus (\tilde{K}_j \ll 4 * i) \quad \text{for } j = 0 \ldots l-2 && (7.31)\\ F(K, i) &= F_{l-1}\,. && (7.32) \end{aligned}$$

7.4.9 Round Key Derivation

For the subciphers defined in Sect. 7.4.4 and 7.4.5 we need to define a key schedule to derive their round keys from the K^i's, that were derived with the one-way function. We propose the following. Use the key schedule of Rijndael for all subciphers. Rijndael is analyzed with this key schedule and hence can be trusted. The choice of key scheduling for the other subciphers is less important from a security point of view since the changes to the cipher increase the diffusion and confusion [127] of the structure, such that attacks that exploit properties of the key schedule are less likely.

We use the notation for f_{rk} as defined in (7.19). For each keyed transformation the round keys are derived as follows.

- ψ needs two 128-bit subkeys K_0^3 and K_1^3. They are derived from K^3 as

$$K_0^3, K_1^3 = f_{\text{rk}}(K^3, 0), f_{\text{rk}}(K^3, 1)\,.$$

- σ is the original Rijndael key xoring and needs eleven 128-bit subkeys. These keys are derived from K^0 as

$$K_0^0, \ldots, K_{10}^0 = f_{\text{rk}}(K^1, 0), \ldots, f_{\text{rk}}(K^1, 10)\,.$$

- A round key for π can be represented by 5 bytes with a value smaller than 24. A 16 byte round key $k_{15} \dots k_0$ is derived from K^1 for each round. From this key the first 5 bytes that are smaller than 24 are taken as subkey bytes for π. If $5 - s$ key bytes are smaller than 24, then the values of the first s bytes modulo 16 are chosen:

```
1. For $i = 0$ to 9 do
     a. Compute $k_{15} \dots k_0 = f_{\mathrm{rk}}(K^1, i)$.
     b. Set counter $c = 0$.
     c. For $j = 0$ to 15 do
          If $k_j < 24$ then $K_i^1[c] = k_j$, $c{+}{+}$.
     d. If $c < 5$
          For $j = c$ to 5 do
            $K_i^1[c] = k_{j-c} \bmod 16$.
```

- β needs per round sixteen 3-bit values. These 3-bit values are the three least significant bits of each key byte of a round key that can be derived from K^3. Hence, for $i = 0, \dots, 9$

$$k_{15} \dots k_0 = f_{\mathrm{rk}}(K^2, i) \tag{7.33}$$
$$K_i^2 = \mathrm{LSB}_3(k_{15}) \dots \mathrm{LSB}_3(k_0) \,. \tag{7.34}$$

7.4.10 Miscellaneous

Ideally, we would also have liked to insert a decorrelation module, like in the COCONUT construction [136]. Unfortunately, we do not know of a perfect decorrelation module that is efficiently implementable on an 8-bit architecture and would add to the diversity of key mixing operations.

Rijndael seems remarkably suitable for addition and changes without violation of the original design criteria. There is still some room for incorporating more layers. For example, the Rijndael S-box was optimized for resistance against LC and DC. However the Wide Trail Strategy permits the use of other S-boxes with 'worse' properties. For example, in [34, p.23] the designers state:

Even an S-box that is "average" in this respect is likely to provide enough resistance against differential and linear cryptanalysis.

Hence, in principle it would be possible to replace the Rijndael S-box with one or several key dependent S-boxes. One motivation against doing this is the occurrence of weak keys. For, example in the case of a chosen subkey attack an attacker could choose a key for this subcipher for which the S-box would be the identity transformation, which would make the whole Rijndael-structure linear. Another disadvantage is that it would also be very performance demanding to use different subkeys in every round, since this would mean that each round an S-box (or 16 S-box entries) would need to be derived.

7.5 Implementation

One of the most important motivations in our design was resistance against side-channel attacks. Most suitable targets for these kind of attacks are smart cards with 8-bit processors. On these kind of processors the cipher would have a slightly lower performance than Rijndael, due to the extra operations. We believe that this performance drop is compensated by the added security in the proposed cipher.

Furthermore, a healthy strategy to make a smart card application (more) resistant against statistical side-channel attacks would be to change the key regularly. Keeping in mind the devastating effect of differential power analysis [86] it is strongly recommended to use the same key at most 10 times. This implies that the key schedule should be performed many times. For our cipher the need for fast key updates is not as stringent. Hence the overall performance in this type of applications would be satisfactory.

Note that due to the increased amount of subkeys the memory requirements of an implementation increase. However, since our design uses the Rijndael key schedule it also benefits from its feature of on-the-fly key computation, as given in Property 7.4. Without on-the-fly key computation for our proposal 402 bytes of round key material need to be stored. However, using Property 7.4 only 512 bits of memory are needed for subkey storage.

Due to the insertion of many key dependent operations, the cipher Rijndael will have a better performance than our design on other platforms. Moreover, the mathematical optimization which caused Rijndael to have a very good performance on 32-bit and 64-bit processors is not possible anymore. Furthermore, in hardware extra resources will be needed to implement all the extra operations. Possibly another application of the design strategy, beginning with Rijndael could result in a cipher with better performance. It still is an open question if efficient design over many platforms is possible with this strategy.

7.6 Security Analysis

In Sect. 7.4 we have shown that the insertion and adding of operations did not decrease the security of the Rijndael-like subcipher. Rijndael also is secure if none of the other subciphers is added. We make the following conjecture

Conjecture 7.1 *Each method that would successfully break our proposal would also break Rijndael.*

This would certainly hold if all the subkeys for π and β were chosen such that Rijndael would be the core in between the outer round permutations. The only case in which Conjecture 7.1 would not hold is if weak keys would exist. However, the existence of such weak keys is very unlikely. Besides that, it is not even clear which weakness or special property these keys should induce. However, since we have no proof, Statement 7.1 is formulated as a conjecture.

One could also extract each subcipher in a similar way from the structure. However, *we certainly do not claim that any subcipher that is extracted from the structure in such a way is a secure block cipher.* For example: the cipher defined by the two outer key mixings needs a good diffusion layer in between. Otherwise, it could be 'easily' broken by several techniques. Hence we analyze the security of the other subciphers in a structure, where all but one subcipher key are known or chosen. Note that if the outer round keys are known, these outer rounds could as well be omitted. Hence, if these keys are known the outer round does not contribute to the security of any other subcipher. We omit these transformations in the analysis models in Sect. 7.6.1 and 7.6.2.

7.6.1 The Permutation Cipher

The cipher we analyze can be modeled as:

$$\mathrm{E}_{K^1} = \pi_{K^1_9} \circ \gamma'_9 \circ \prod_{i=8}^{0} (\theta \circ \pi_{K^1_i} \circ \gamma'_i) , \tag{7.35}$$

where γ'_i is a transformation that performs a known S-box transformation on each byte of a state. The S-boxes are with high probability different for each pair of byte position and value of i (the only exception may occur if a chosen subcipher key attack is mounted).

The keyed permutation as described in Sect. 7.4.4 is determined by just 22 key bits per round. This is very small for a cipher that only consists of 10 rounds. It is possible to mount a simple meet-in-the-middle attack [101] that needs 2^{110} half encryptions/decryptions and storage for 2^{110} plaintext/'ciphertext'-pairs. This is more efficient than exhaustive key search. Time-memory tradeoffs are possible. However, it is not very likely that this attack could be modified to be feasible in practice.

Furthermore, due to the diffusion and confusion provided by the cipher structure we have found no method to exploit the seemingly simple key dependent permutations. Hence, we see no reason to judge this subcipher as being superfluous or to increase the number of rounds.

7.6.2 The Rotation Cipher

The cipher we analyze can be modeled as:

$$\mathrm{E}_{K^2} = \beta_{K^3_9} \circ \pi'_9 \circ \prod_{i=8}^{1} (\gamma'_i \circ \beta_{K^3_i} \circ \theta \circ \pi'_i) \circ \sigma'_0 \circ \beta_{K^3_0} , \tag{7.36}$$

where π'_i is a known byte permutation for each i, γ'_i is a transformation that performs a known S-box transformation on each byte of a state, and σ'_0 is an xor with a known constant. Note that γ'_i still has optimal properties for LC and DC resistance and that the diffusion properties due to the $\theta \circ \pi'_i$-transformations still hold.

Its security is based on key-dependent rotations. Block ciphers for which the security is (heavily) based on data-dependent rotations are RC5 and its successor and former AES candidate RC6. Due to the extensive analysis of these ciphers in [2, 11, 23, 82] much is known about methods to exploit data-dependent rotations. Due to the diffusion and confusion properties of the structure the cipher seems to be sufficiently secure.

7.6.3 The Outer Round Cipher

Here the model is:

$$\mathrm{E}_{K^3} = \psi_{K_1^3} \circ \chi \circ \psi_{K_0^3} \,, \tag{7.37}$$

where χ is a known transformation with very good diffusion and confusion.

In [14] advanced slide attacks were introduced to attack DESX. The fact that χ is not keyed makes such an attack in this case even more efficient. This attack is illustrated as follows. Suppose that in (7.37) the ψ transformation would xor its input with a key. If an attacker obtains $2^{64.5}$ plaintext/ciphertext pairs (P, C), then with high probability for one pair $(P, C), (P', C')$ of those holds

$$P \oplus K_0^3 = \chi^{-1}(C') \text{ and } P' \oplus K_0^3 = \chi^{-1}(C) \,. \tag{7.38}$$

This implies that $C \oplus C' = K_1^3$ and hence one can easily deduce K_0^3. From (7.38) also follows that $P \oplus P' = E^{-1}(C) \oplus E^{-1}(C')$. Hence, if the attacker finds such a pair, with high probability he can compute K_1^3 and K_0^3.

Because ψ does its key mixing with additions modulo 2^8 this kind of attack is not applicable.

7.7 Advantages and Limitations

In this section we resume the main advantages and limitations of the design strategy of multiple layered security. The limitations are mainly in the area of implementation where high speed is required or where few memory is available.

Advantages.

- The security can be based on different security motivations.
- The security can be based on a well evaluated cipher or several well evaluated ciphers, in our case: Rijndael.
- The use of different operations to mix in the keys implies that an attacker would need different techniques.
- Enhanced security against implementation based attacks.
- Enhanced security against developments in cryptanalytic attack methods.

- Better performance is possible compared to ciphers that offer a similar level of security, i.e., compared to sequences of ciphers that are designed on different design principles.
- Computer code for a cipher can be written on the basis of known and available code if the cipher is based on a well-known and widely implemented cipher as in our case Rijndael.
- The result of the design strategy might seem to lack simplicity and elegance, but this can be easily resolved by taking also notice of the design process.

Limitations.

- A design that is based on one motivation would probably have better encryption/decryption speed and key-setup time.
- With this strategy it seems difficult to design a cipher that is very efficient over a large variety of platforms.
- It is inherent to the strategy that a rather large memory is required for subkeys. For example, for our design 512 bits of subkey material would need to be stored during encryption or decryption, even though it uses the on-the-fly key schedule of Rijndael.
- The use of different key mixing operations increases the cost for hardware implementation and makes optimizing code less trivial.

7.8 Conclusions

In this chapter we have described a conservative block cipher design strategy, which fits in the approach of provable and practical security. This is the main contribution of this chapter. In this strategy security is divided over several ciphers in a structure. Breaking all but one should still result in a secure cipher.

We have worked out and analyzed a more concrete proposal with four subciphers. This is another contribution. The proposal is based on Rijndael, which turned out to be very suitable for this design strategy. It has been submitted to the NESSIE project for evaluation.

The main advantage of the proposed strategy is that it can be used to assure that a cipher does not get broken by newly developed techniques, either classical or side-channel attacks. In this aspect the purpose is to give a comparable level of security as can be achieved by using a sequence of several ciphers. Hence, an efficiency comparison should take place taking into account this concept of multiple layered security. In this context the strategy is a valuable asset in the toolbox of a designer of security systems that have to provide security over a long period of time and/or might be vulnerable to side channel leakage.

The down side of the strategy is that it seems a very difficult task to design a concrete proposal that would have a similar efficiency across a large number of different platforms as a classically designed cipher. It is an interesting question to which extent such a design is practically feasible.

Chapter 8

General Conclusions

In this chapter we summarize the results of our work, our most important conclusions and the main contributions of this thesis.

In current society the role of information security is increasing. For the possibility and success of information based technologies such as ecommerce, electronic payments, secure storage and the transmission of confidential or private information and the use of smart cards, it is essential that specific security objectives are addressed adequately. Information security studies methods that address these security objectives. In Chapter 2 we gave an overview of the role of information security and indicated methods that can be used. A class of methods that can be used to address security objectives are cryptographic primitives. They typically function as building blocks of a larger system.

In this thesis we described the results and consequences of several security evaluations of cryptographic primitives. These include evaluations of generic primitives and specific designs. The analyzed security objectives include generic objectives and objectives required for use in a specific system, application or platform. These evaluations resulted in the estimation of the security level of several primitives, pointing out design flaws in specific proposals, restrictions of standard analysis methods, the development of attack methods, suggestions for increasing security of practically used schemes and a new design strategy with its specific advantages and limitations.

Brute force attacks are attacks that can be applied to any specific primitive and for which the complexity and success probability depends on certain parameters of the primitive. In Chapter 3 we described a brute force attack that can be applied to block ciphers, stream ciphers, one-way functions and hash functions. This method improves a method of Hellman, using an idea of Rivest, in the sense that the amount of required memory accesses was reduced significantly. This makes practical and efficient implementation of the attack possible in dedicated hardware, as well as for distributed computation on general purpose machines. We analyzed the complexity and success probability, such that the most efficient approach to implement the attack can be determined, based on resources and the specific goal of an attacker. The main conclusion of this chapter is that this attack should be taken into consideration when determining the parameters for the relevant primitives.

Shortcut attacks exploit properties of the inner structure of a primitive. In Chapter 4 we study the applicability of a shortcut attack to the block cipher RC5. For this, we use a statistical attack technique: linear cryptanalysis. Previous research claimed that RC5 should have high resistance to this kind of attack. However, this claim was based on standard assumptions that do not hold for RC5. We constructed a method specific for RC5 to estimate its resistance against linear attacks. Furthermore, we introduced a new tool, the non-uniformity function, with which we constructed an attack that breaks reduced versions of RC5. This showed that the earlier claims about the resistance of RC5 against linear attacks were far from valid, although resistance of the full round version is sufficient. The same holds for the AES finalist RC6. The main conclusion of this chapter is that security claims based on standard assumption should be scrutinized before they can be trusted and that it is recommended to have a considerable safety margin built in during design.

Key schedule analysis studies consequences of properties of a part of a block cipher: the key schedule. We analyzed the key schedule of the block cipher CRYPTON, which was one of the original 15 AES candidates. The security of CRYPTON was based on the same design strategy as the winner of the AES competition Rijndael. In Chapter 5 we describe several weaknesses in the key schedule of CRYPTON and methods to exploit them when CRYPTON is used in a popular hash function construction, from which the designer had claimed that this could be done securely. Except for the conclusion that NIST had drawn from these results, the main conclusion that can be drawn from this chapter is that a design based on a sound strategy is still no guarantee for security.

Side channel attacks exploit characteristics of the implementation of a cryptographic algorithm on a specific platform that can be measured during cryptographic computations. Examples of such characteristics are power consumption, execution time and electromagnetic radiation. Chapter 6 studies power attacks; here the side channel is power consumption. This attack method has been introduced in the late 90s and many applications have been reported to be vulnerable. Particularly suitable targets are smart cards, since they get their power consumption externally. We have given an overview of the possible kinds of power attacks. Furthermore, we have classified methods that can be applied to increase security against these attacks. This classification is an important tool for the designer of a secure system, in the sense that it gives insight in the possible countermeasures, their advantages and their limitations. We have also determined a specific level for countermeasures, the protocol level, that can very efficiently be used to introduce secure countermeasures in typical existing architectures and systems. We can conclude from this chapter that security requirements that only seem to apply to a certain aspect of a system can be addressed efficiently by making adjustments or design decisions on aspects of the whole design.

Chapter 7 introduces a new block cipher design strategy and describes a specific block cipher that was designed according to this strategy. The philosophy behind the strategy is to base the security on different complementary motivations. We call this concept: multiple layered security. In this way one can enhance security against classical shortcut attacks and implementation based attacks. Also a better security against developments in cryptanalytic attacks is provided. More specifically the strategy aims to insert several block ciphers, whose security motivations are complementary, in one structure. The specific design proposal is amongst others based on Rijndael, the winner of the AES competition. With the design we showed that this strategy can be adopted to design secure and efficient block ciphers, which is the main conclusion of this chapter.

8.1 Further Research

During the work for this thesis we have encountered several interesting subjects in which further research might be performed. We summarize some specific items:

- Determining the values of α and β as defined in Chapter 3 theoretically.
- More theoretical research to find a general approach to the treatment of linear hull effects, the applicability of the Piling-Up Lemma and key dependence in statistical cryptanalysis.
- Applying the non-uniformity function to other block ciphers.
- Using the techniques of our attack on RC5 to improve on the current attacks on RC6.
- Finding a general approach to determining the suitable countermeasures against power analysis on all levels specified in Chapter 6.
- Applying the design strategy of multiple layered security for the design of other cryptographic primitives.
- Analysis of the block cipher proposal GRAND CRU and other primitives submitted to NESSIE.

Bibliography

[1] 3GPP, The 3GPP homepage, `http://www.3gpp.org/`.

[2] National Institute For Standards and Technology (NIST), "AES, A Crypto Algorithm for the Twenty-first Century," `http://www.nist.gov/aes`.

[3] ANSI X3.92, *American National Standard – Data Encryption Algorithm,* American National Standards Institute, 1981.

[4] ANSI X9.17 *American National Standard – Financial Institution Key Management,* American National Standards Institute, 1985.

[5] R. Baldwin, R. Rivest, *RFC 2040: The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms,* October 30, 1996.

[6] 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 Second AES Candidate Conference*, 1999, pp. 1–12.

[7] E. Biham, A. Biryukov, A. Shamir, "Miss in the Middle Attacks on IDEA and Khufu," *Fast Software Encryption, LNCS 1636*, L. Knudsen, Ed., Springer-Verlag, 1999, pp. 124–138.

[8] E. Biham, A. Biryukov, A. Shamir, "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials," *Advances in Cryptology, Proc. Eurocrypt'99, LNCS 1592*, J. Stern, Ed., Springer-Verlag, 1999, pp. 12–23.

[9] E. Biham, A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.

[10] E. Biham, A. Shamir, "Power Analysis of the AES Candidates," *Proc. of the Second AES Candidate Conference*, 1999, pp. 115–121.

[11] A. Biryukov, E. Kushilevitz, "Improved Cryptanalysis of RC5," *Advances in Cryptology, Proc. Eurocrypt'98, LNCS 1403*, K. Nyberg, Ed., Springer-Verlag, 1998, pp. 85–99.

[12] A. Biryukov, E. Kushilevitz, "From Differential Cryptanalysis to Ciphertext-Only Attacks," *Advances in Cryptology, Proc. Crypto'98, LNCS 1462*, H. Krawczyk, Ed., Springer-Verlag, 1998, pp. 72–88.

[13] A. Biryukov, A. Shamir, D. Wagner, "Real Time Cryptanalysis of A5/1 on a PC," *Fast Software Encryption, LNCS 1978*, B. Schneier, Ed., Springer-Verlag, 2001, pp. 1–18.

[14] A. Biryukov, D. Wagner, "Advanced Slide Attacks," *Advances in Cryptology, Proc. Eurocrypt 2000, LNCS 1807*, B. Preneel, Ed., Springer-Verlag, 2000, pp. 589–606.

[15] B. den Boer, A. Bosselaers, "An Attack on the Last Two Rounds of MD4," *Advances in Cryptology, Proc. Crypto'91, LNCS 576*, S. Vanstone, Ed., Springer-Verlag, 1992, pp. 194–203.

[16] M. Blaze, W. Diffie, R.L. Rivest, B. Schneier, T. Shimomura, E. Thompson, M. Wiener, *Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security,* January 1996.

[17] J. Borst, *Public Key Cryptosystems using Elliptic Curves*, M. Thesis, Faculteit Wiskunde & Informatica, Technische Universiteit Eindhoven, 1997.

[18] J. Borst, "Weak Keys of CRYPTON," comment to NIST, August 1998. *Presented at the Rump Session of the 2nd AES conference.*

[19] J. Borst, L.R. Knudsen, V. Rijmen, "Two Attacks on Reduced IDEA," *Advances in Cryptology, Proc. Eurocrypt'97, LNCS 1233*, U. Maurer, Ed., Springer-Verlag, 1997, pp. 1–13.

[20] J. Borst, B. Preneel, J.-J. Quisquater, J. Stern, "A Time-Memory Tradeoff Using Distinguished Points," submitted.

[21] J. Borst, B. Preneel, V. Rijmen, "Cryptography on Smart Cards," *Computer Networks,* Vol. 36, 2001, pp. 423–435.

[22] J. Borst, B. Preneel, J. Vandewalle, "On the Time-Memory Tradeoff Between Exhaustive Key Search and Table Precomputation," *Proc. of the 19th Symposium on Information Theory in the Benelux*, WIC, 1998, pp. 111–118. Also available from `http://www.esat.kuleuven.ac.be/~borst`.

[23] J. Borst, B. Preneel, J. Vandewalle, "Linear Cryptanalysis of RC5 and RC6," *Fast Software Encryption, LNCS 1636*, L. Knudsen, Ed., Springer-Verlag, 1999, pp. 16–30.

[24] J. Borst, B. Preneel, J. Vandewalle, "An Adaptive Chosen Ciphertext Attack on a Variation of the Cramer-Shoup Public-Key Encryption Scheme," *Electronics Letters,* Vol. 36, No. 1, 2000, p. 32.

[25] S. Chari, C. Jutla, J. Rao, P. Rohatgi, "A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards," *Proc. of the Second AES Candidate Conference*, 1999, pp. 133–147.

[26] S. Chari, C. Jutla, J. Rao, P. Rohatgi, "Towards Sound Approaches to Counteract Power-Analysis Attacks," *Advances in Cryptology, Proc. Crypto'99, LNCS 1666*, M.J. Wiener, Ed., Springer-Verlag, 1998, pp. 398–412.

[27] S. Contini, R.L. Rivest, M.J.B. Robshaw, Y.L. Yin, *The Security of the RC6 Block Cipher. v1.0,* 1998, available via `http://www.rsa.com/rsalabs/aes`.

[28] F. Chabaud, A. Joux, "Differential Collisions in SHA-0," *Advances in Cryptology, Proc. Crypto'98, LNCS 1462*, H. Krawczyk, Ed., Springer-Verlag, 1998, pp. 56–71.

[29] D. Coppersmith, "The Data Encryption Standard (DES) and Its Strength Against Attacks," *IBM Journal of Research and Development,* Vol. 38, No. 3, May 1994, pp. 243–250.

[30] J. Daemen, *Cipher and Hash Function Design,* PhD. Thesis, Departement Elektrotechniek–ESAT/COSIC, Katholieke Universiteit Leuven, March 1995.

[31] J. Daemen, L.R. Knudsen, V. Rijmen, "The Block Cipher SQUARE," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 149–165.

[32] J. Daemen, M. Peeters, G. Van Assche, "Bitslice Ciphers and Power Analysis Attacks," *Fast Software Encryption, LNCS 1978*, B. Schneier, Ed., Springer-Verlag, 2001, pp. 134–149.

[33] J. Daemen, V. Rijmen, "The Block Cipher BKSQ," *Smart Card Research and Applications, Proc. CARDIS'98, LNCS 1820*, J.-J. Quisquater and B. Schneier, Eds., Springer-Verlag, 2000, pp. 288–296.

[34] J. Daemen, V. Rijmen, *AES Proposal: Rijndael,* available via [2].

[35] J. Daemen, V. Rijmen, "Resistance Against Implementation Attacks, A Comparative Study of the AES Proposals," *Proc. of the Second AES Candidate Conference*, 1999, pp. 122–132.

[36] D.E. Denning, *Cryptography and Data Security*, Addison-Wesley, 1982.

[37] C. D'Halluin, G. Bijnens, V. Rijmen, B. Preneel, "Attack on Six Rounds of CRYPTON," *Fast Software Encryption, LNCS 1636*, L. Knudsen, Ed., Springer-Verlag, 1999, pp. 46–59.

[38] W. Diffie, M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory, Vol. IT-22*, 1976, pp. 644–654.

[39] W. Diffie, M. E. Hellman, "Exhaustive Cryptanalysis of the NBS Data Encryption Standard," *Computer*, Vol. 10, 1977, pp. 397–427.

[40] H. Dobbertin, "Cryptanalysis of MD4," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 53–69.

[41] H. Dobbertin, "The Status of MD5 after a Recent Attack," *CryptoBytes*, Vol. 2, No. 2, Summer 1996, pp. 1–6.

[42] H. Dobbertin, "The First Two Rounds of MD4 are Not One-Way," *Fast Software Encryption, LNCS 1372*, S. Vaudenay, Ed., Springer-Verlag, 1998, pp. 284–292.

[43] H. Dobbertin, A. Bosselaers, B. Preneel, “RIPEMD-160: A Strengthened Version of RIPEMD,” *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 71–82.

[44] Electronic Frontier Foundation, *Cracking DES, Secrets of Encryption Research, Wiretap Politics & Chip Design,* (O’Reilly & Associates, Sebastopol, 1998). Source code of the implementation described in the book can be downloaded from `https://www.cosic.esat.kuleuven.ac.be/des/`.

[45] *EMVCo, LLC,* `http://www.emvco.com`.

[46] *ETSI/SAGE Specification – Specification of the 3GPP Confidentiality and Integrity Algorithms – Document 1: $f8$ and $f9$ Specification,* Version 1.2, September 2000, available via `http://www.etsi.org/dvbandca/3GPP/3gppspecs.htm`.

[47] *ETSI/SAGE Specification – Specification of the 3GPP Confidentiality and Integrity Algorithms – Document 2: KASUMI Specification,* Version 1.0, December 1999, available via `http://www.etsi.org/dvbandca/3GPP/3gppspecs.htm`.

[48] P. Fahn, P. Pearson, “IPA: A New Class of Power Attacks,” *Cryptographic Hardware and Embedded Systems, CHES’99, LNCS 1717*, Ç. K. Koç and C. Paar, Eds., Springer-Verlag, 1999, pp. 173–186.

[49] N. Ferguson, J. Kelsey, B. Schneier, M. Stay, D. Wagner, D. Whiting, “Improved Cryptanalysis of Rijndael,” *Fast Software Encryption, LNCS 1978*, B. Schneier, Ed., Springer-Verlag, 2001, pp. 213–230.

[50] A. Fiat, M. Naor, “Rigorous Time/Space Tradeoffs for Inverting Functions,” *Proc. of the 23rd Annual ACM Symposium on Theory of Computing*, 1991, pp. 534–541.

[51] P. Flajolet, A. Odlyzko, “Random Mapping Statistics,” *Advances in Cryptology, Proc. Eurocrypt’89, LNCS 434*, J.-J. Quisquater and J. Vandewalle, Eds., Springer-Verlag, 1990, pp. 329–354.

[52] FIPS 46, *Data Encryption Standard,* Federal Information Processing Standard (FIPS), National Bureau of Standards, U.S. Department of Commerce, Washington D.C., January 1977 (revised as FIPS 46-1: 1988; FIPS 46-2: 1993).

[53] FIPS 81, *DES Modes of Operation,* Federal Information Processing Standard (FIPS), National Bureau of Standards, U.S. Department of Commerce, Washington D.C., December 1980.

[54] FIPS 180-1, *Secure Hash Standard,* Federal Information Processing Standard (FIPS), National Bureau of Standards, U.S. Department of Commerce, Washington D.C., April 1995.

[55] FIPS 180-2, *Secure Hash Standard,* Federal Information Processing Standard (FIPS), National Bureau of Standards, U.S. Department of Commerce, Washington D.C., Draft, May 2001.

[56] FIPS xxx, *Advanced Encryption Standard,* Federal Information Processing Standard (FIPS), National Bureau of Standards, U.S. Department of Commerce, Washington D.C., Draft, February 2001.

[57] "GSM Cloning," `http://www.isaac.cs.berkeley.edu/isaac/gsm.html`.

[58] H. Gilbert, H. Handschuh, A. Joux, S. Vaudenay, "A Statistical Attack on RC6," *Fast Software Encryption, LNCS 1978*, B. Schneier, Ed., Springer-Verlag, 2001, pp. 64–74.

[59] H. Gilbert, M. Minier, "A Collision Attack on 7 Rounds of Rijndael," *Proc. of the Third AES Candidate Conference*, 2000, pp. 230–241.

[60] L. Goubin, J. Patarin, "DES and Differential Power Analysis. The "Duplication" Method," *Cryptographic Hardware and Embedded Systems, CHES'99, LNCS 1717*, Ç. K. Koç and C. Paar, Eds., Springer-Verlag, 1999, pp. 158–172.

[61] H. Handschuh, H.M. Heys, "A Timing Attack on RC5," *Proc. Workshop on Selected Areas of Cryptography, SAC '98, LNCS 1556*, Springer-Verlag, 1998.

[62] C. Harpes, J.L. Massey, "Partitioning Cryptanalysis," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 13–27.

[63] M. Hellman, "A Cryptanalytic Time-Memory Tradeoff," *IEEE Transactions of Information Theory*, Vol. 26, 1980, pp. 401–406.

[64] M. Hellman, S. Langford, "Differential-Linear Cryptanalysis," *Advances in Cryptology, Proc. Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 26–36.

[65] ISO/IEC 7816, *Information technology – Identification cards – Integrated circuit(s) cards with contacts – Part 4: Interindustry commands for interchange,* International Organization for Standardization, Geneva, Switzerland, 1997.

[66] ISO 7832, *Banking – Key Management (Wholesale),* International Organization for Standardization, Geneva, Switzerland, 1988.

[67] ISO/IEC 10118-2, *Information technology – Security techniques – Hash-functions – Part 3: Hash-functions using an n-bit block cipher algorithm,* International Organization for Standardization, Geneva, Switzerland, 1994.

[68] ISO/IEC 10118-3, *Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions,* International Organization for Standardization, Geneva, Switzerland, 1998.

[69] 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.

[70] T. Jakobsen, L.R. Knudsen, "The Interpolation Attack on Block Ciphers," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 28–40.

[71] B.S. Kaliski, M.J.B. Robshaw, "Linear Cryptanalysis Using Multiple Approximations," *Advances in Cryptology, Proc. Eurocrypt'94, LNCS 950*, A. De Santis, Ed., Springer-Verlag, 1995, pp. 26–39.

[72] B.S. Kaliski, Y.L. Yin, "On Differential and Linear Cryptanalysis of the RC5 Encryption Algorithm," *Advances in Cryptology, Proc. Crypto'95, LNCS 963*, D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 171–184.

[73] B.S. Kaliski, Y.L. Yin, *On the Security of the RC5 Encryption Algorithm,* RSA Laboratories Technical Report TR-602, Version 1.0, September 1998, available via `http://www.rsa.com/rsalabs/aes`.

[74] J. Kilian, P. Rogaway, "How to Protect DES Against Exhaustive Key Search," *Advances in Cryptology, Proc. Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 252–267. (Final version to appear in *Journal of Cryptology*).

[75] I.-J. Kim, T. Matsumoto, "Achieving Higher Success Probability in Time-Memory Trade-Off Cryptanalysis without Increasing Memory Size," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science*, E82-A, 1999, pp. 123–129.

[76] L. Knudsen, *Block Ciphers – Analysis, Design and Applications,* PhD. Thesis, Computer Science Department, Aarhus University, 1994.

[77] L. Knudsen, "Truncated and Higher Order Differentials," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 196–211.

[78] L. Knudsen, "Why SAFER K Changed Its Name," *Technical Report LIENS 96-13,* Département de Mathématiques et Informatique, ENS, Paris, 1996.

[79] L. Knudsen, T. Berson, "Truncated Differentials of SAFER," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 15–26.

[80] L. Knudsen, J.E. Mathiassen, "A Chosen-Plaintext Linear Attack on DES," *Fast Software Encryption, LNCS 1978*, B. Schneier, Ed., Springer-Verlag, 2001, pp. 262–272.

[81] L. Knudsen, W. Meier, "Improved Differential Attacks on RC5," *Advances in Cryptology, Proc. Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 216–228.

[82] L. Knudsen, W. Meier, "Correlations in RC6 with a Reduced Number of Rounds," *Fast Software Encryption, LNCS 1978*, B. Schneier, Ed., Springer-Verlag, 2001, pp. 94–108.

[83] L. Knudsen, V. Rijmen, R. Rivest, M. Robshaw, "On the Design and Security of RC2," *Fast Software Encryption, LNCS 1372*, S. Vaudenay, Ed., Springer-Verlag, 1998, pp. 206–221.

[84] L. Knudsen, M. Robshaw, D. Wagner, "Truncated Differentials and Skipjack," *Advances in Cryptology, Proc. Crypto'99, LNCS 1666*, M.J. Wiener, Ed., Springer-Verlag, 1998, pp. 165–180.

[85] 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.

[86] P. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis," *Advances in Cryptology, Proc. Crypto'99, LNCS 1666*, M.J. Wiener, Ed., Springer-Verlag, 1998, pp. 388–397.

[87] K. Kusuda, T. Matsumoto, "Optimization of Time-Memory Trade-Off Cryptanalysis and Its Application to DES, FEAL-32, and Skipjack," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science*, E79-A, 1996, pp. 35–48.

[88] X. Lai, *On the Design and Security of Block Ciphers,* PhD Thesis, ETH, Zürich, Switzerland, 1992.

[89] X. Lai, J.L. Massey, "A Proposal for a New Block Encryption Standard," *Advances in Cryptology, Proc. Eurocrypt'90, LNCS 473*, I.B. Damgård, Ed., Springer-Verlag, 1991, pp. 17–38.

[90] 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.

[91] C. H. Lim, *CRYPTON: A New 128-bit Block Cipher,* AES Proposal, 1998, available via `http://crypt.future.co.kr/~chlim`.

[92] C. H. Lim, "A Revised Version of CRYPTON – CRYPTON V1.0 –," *Fast Software Encryption, LNCS 1636*, L. Knudsen, Ed., Springer-Verlag, 1999, pp. 31–45.

[93] S. Lucks, "Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys," *Proc. of the Third AES Candidate Conference*, 2000, pp. 215–229.

[94] J.L. Massey, "SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm," *Fast Software Encryption, LNCS 809*, R. Anderson, Ed., Springer-Verlag, 1994, pp. 1–17.

[95] J.L. Massey, "SAFER K-64: One Year Later," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 212–241.

[96] 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.

[97] 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.

[98] M. Matsui, "New Block Encryption Algorithm MISTY," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 54–68.

[99] M. Matsui, A. Yamagishi, "A New Method for Known Plaintext Attack of FEAL Cipher," *Advances in Cryptology, Proc. Eurocrypt'92, LNCS 658*, Springer-Verlag, 1993, pp. 81–91.

[100] U. Maurer, J.L. Massey, "Cascade Ciphers: the Importance of Being First," *Journal of Cryptology*, Vol. 6, No. 1, 1993, pp. 55–61.

[101] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[102] R. Motwani, P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

[103] S. Murphy, M. Robshaw, "New Observations on Rijndael," *comment to NIST,* August 2000.

[104] *New European Schemes for Signatures, Integrity, and Encryption,* project within the Information Societies Technology (IST) Programme of the European Commission, `http://www.cryptonessie.org`.

[105] K. Nyberg, "Linear Approximations of Block Ciphers," *Advances in Cryptology, Proc. Eurocrypt'94, LNCS 950*, A. De Santis, Ed., Springer-Verlag, 1995, pp. 439–444.

[106] K. Nyberg, L. Knudsen, "Provable Security Against a Differential Attack," *Journal of Cryptology*, Vol. 8, No. 1, 1995, pp. 27–38.

[107] B. Preneel, *Analysis and Design of Cryptographic Hash Functions,* PhD. Thesis, Departement Elektrotechniek–ESAT/COSIC, Katholieke Universiteit Leuven, January 1993.

[108] J.-J. Quisquater, J.-P. Delescaille, "How Easy is Collision Search. Applications to DES," *Advances in Cryptology, Proc. Eurocrypt'89, LNCS 434*, J.-J. Quisquater and J. Vandewalle, Eds., Springer-Verlag, 1990, pp. 429–433.

[109] J.-J. Quisquater, J.-P. Delescaille, "How Easy is Collision Search. New Results and Applications to DES," *Advances in Cryptology, Proc. Crypto'89, LNCS 435*, G. Brassard, Ed., Springer-Verlag, 1990, pp. 408–413.

[110] J.-J. Quisquater, J. Stern, "Time-Memory Tradeoff Revisited," unpublished, December 1998.

[111] V. Rijmen, *Cryptanalysis and Design of Iterated Block Ciphers,* PhD. Thesis, Departement Elektrotechniek–ESAT/COSIC, Katholieke Universiteit Leuven, October 1997.

[112] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, E. De Win "The Cipher SHARK," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 99–111.

[113] R. Rivest, "The MD4 Message Digest Algorithm," *Advances in Cryptology, Proc. Crypto'90, LNCS 537*, Springer-Verlag, 1991, pp. 303–311.

[114] R. Rivest, *RFC 1320: The MD4 Message-Digest Algorithm,* April 1992.

[115] R. Rivest, *RFC 1321: The MD5 Message-Digest Algorithm,* April 1992. Presented at the Rump Session of Crypto'91.

[116] R. Rivest, "The RC5 Encryption Algorithm," *Dr. Dobb's Journal,* Jan. 1995, pp. 146-148.

[117] R. Rivest, "The RC5 Encryption Algorithm," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 86–96.

[118] R. Rivest, *Block Encryption Algorithm with Data-Dependent Rotations,* U.S. Patent 5,724,428, Mar. 3, 1998. (Patent assigned to RSA Data Security, Inc.)

[119] R. Rivest, *Block Encryption Algorithm with Data-Dependent Rotations,* U.S. Patent 5,835,600, Nov. 10, 1998. (Patent assigned to RSA Data Security, Inc.)

[120] R.L. Rivest, M.J.B. Robshaw, R. Sidney, Y.L. Yin, *The RC6 Block Cipher. v1.1,* AES Proposal, 1998, available via `http://www.rsa.com/rsalabs/aes`.

[121] M.J.B. Robshaw, *Linear Hulls and RC6 (DRAFT),* September, 1998, available from the author.

[122] *The RSA Laboratories Secret-Key Challenge,* `http://www.rsasecurity.com/rsalabs/challenges/secretkey/index.html`.

[123] B. Schneier, "Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish," *Fast Software Encryption, LNCS 809*, R. Anderson, Ed., Springer-Verlag, 1994, pp. 191–204.

[124] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C,* Wiley & sons, New York, 2nd edition, 1996.

[125] A. Selçuk, "New Results in Linear Cryptanalysis of RC5," *Fast Software Encryption, LNCS 1372*, S. Vaudenay, Ed., Springer-Verlag, 1998, pp. 1–16.

[126] A. Selçuk, "Bias Estimation in Linear Cryptanalysis," *Proc. Indocrypt 2000, LNCS 1977*, B. Roy and E. Okamoto, Eds., Springer-Verlag, 2000, pp. 52–66.

[127] C.E. Shannon, "Communication Theory of Secrecy Systems," *Bell System Technical Journal,* Vol. 28, Oct. 1949, pp. 656–715.

[128] A. Shamir, "Protecting Smart Cards from Passive Power Analysis with Detached Power Supplies," *Cryptographic Hardware and Embedded Systems, CHES 2000, LNCS 1965*, Ç. K. Koç and C. Paar, Eds., Springer-Verlag, 2000, pp. 71–77.

[129] T. Shimoyama, T. Kaneko, "Quadratic Relation of S-box and Its Application to the Linear Attack of Full Round DES," *Advances in Cryptology, Proc. Crypto'98, LNCS 1462*, H. Krawczyk, Ed., Springer-Verlag, 1998, pp. 200–211.

[130] T. Shimoyama, K. Takeuchi, J. Hayakawa, "Correlation Attack to the Block Cipher RC5 and the Simplified Variants of RC6," *Presented at the Third AES Candidate Conference*, 2000 (unpublished).

[131] G. Simmons (Ed.), *Contemporary Cryptology, The Science of Information Integrity*, IEEE Press, 1992.

[132] D. Stinson, *Cryptography: Theory and Practice,* CRC Press, 1995.

[133] B. Van Rompay, J. Verelst, *Analysis of Blowfish,* Master's Thesis (in Dutch), Departement Elektrotechniek–ESAT/COSIC, Katholieke Universiteit Leuven, 1996.

[134] S. Vaudenay, "On the Weak Keys of Blowfish," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 27–32.

[135] S. Vaudenay, "An Experiment on DES – Statistical Cryptanalysis," *Proc. of the 3rd ACM Conference on Computer Security*, ACM Press, 1996, pp. 139–147.

[136] S. Vaudenay, "Provable Security for Block Ciphers by Decorrelation," *STACS 98*, LNCS 1373, Springer-Verlag, 1998, pp. 249–275.

[137] G. Vernam, Cipher Printing Telegraph Systems for Secret Wire and Radio Telegraphic Communications, *J. Am. Inst. Elec. Eng.,* Vol. 55, 1926, pp. 7–10.

[138] D. Wagner, email communication, 1999.

[139] M. Wiener, "Efficient DES Key Search – An Update," *CryptoBytes*, Vol. 3, No. 2, pp. 6–8, 1998.