Christophe DE CANNIÈRE

ANALYSIS AND DESIGN OF SYMMETRIC ENCRYPTION ALGORITHMS

Mei 2007

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK
Kasteelpark Arenberg 10, B–3001 Heverlee

ANALYSIS AND DESIGN OF SYMMETRIC ENCRYPTION ALGORITHMS

Promotor:
Prof. dr. ir. Bart Preneel

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de toegepaste wetenschappen

door

Christophe DE CANNIÈRE

Mei 2007

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK
Kasteelpark Arenberg 10, B–3001 Heverlee

ANALYSIS AND DESIGN OF SYMMETRIC ENCRYPTION ALGORITHMS

Jury:
Prof. Y. Willems, voorzitter
Prof. B. Preneel, promotor
Prof. T. Johansson (Lunds Tekniska Högskola)
Prof. V. Rijmen (Technische Universität Graz)
Prof. M. Van Barel
Prof. J. Vandewalle
Prof. P. Wambacq

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de toegepaste wetenschappen

door

Christophe DE CANNIÈRE

U.D.C. 681.3*D46 Mei 2007

D/2007/7515/75

ISBN 978-90-5682-841-7

Acknowledgments

The years leading to this doctoral dissertation have been for me an exciting period of discovery and growth, made possible by the help and support of numerous people who accompanied me on my journey. I wish to gratefully acknowledge their essential contribution to the results presented here.

First of all, I would like to thank my supervisor, Prof. Bart Preneel. Not only did he direct my research all along, he also introduced me to a large cross section of the worldwide cryptographic community, of which he is such a distinguished member. The participation in the ECRYPT projects which he leads has given me the opportunity to meet and work with a lot of eminent cryptographers; it has also been the direct motivation for part of my work. My very first steps in cryptanalysis, though, were undertaken in the group of Prof. Thomas Johansson of Lunds Tekniska Högskola, where I spent a few months as an undergraduate Erasmus student. I am very grateful for the warm welcome and the precious counseling I received during this memorable stay in Sweden. Prof. Joos Vandewalle, head of the ESAT-SCD group, has been a very appreciated mentor for me, giving advice when important choices had to be made, and providing encouragement when needed. Prof. Marc Van Barel is the fourth member of the reading committee. I would like to thank the four of them for their patience while waiting for the "just in time" delivery of the draft of this thesis, and for their useful suggestions to improve the final version.

I also would like to express my gratitude to the other members of the jury: Prof. Yves Willems, chairman; Prof. Patrick Wambacq; and Prof. Vincent Rijmen. Prof. Rijmen was my host during my stay at the Technische Universität Graz in 2005–2006. This year yielded a particularly rich experience, both at a professional and a personal level, in great part thanks to Vincent and the other members of the Krypto team: Florian Mendel, Jorge Munilla, Norbert Pramstaller, and Christian Rechberger.

I was very lucky that the start of my PhD work coincided with the arrival in Leuven of Alex Biryukov. I am grateful to have had this experienced researcher as my office mate and sparring partner for four years, and our endless discussions have led to quite a few of the ideas discussed in this thesis.

There are many more fine crypto professionals with whom I have had fruitful interactions during the past years. I cannot mention them all, but I do want to thank the people with whom I have co-authored papers: Steve Babbage, Alex Biryukov, An Braeken, Gustaf Dellkrantz, Joe Lano, Florian Mendel, Sıddıka Berna Örs, Bart Preneel, Michaël Quisquater, Christian Rechberger, Joos Vandewalle, Dai Watanabe, and Hirotaka Yoshida.

I spent most of my time during this period in the COSIC group of the K.U.Leuven. It provided a high quality, challenging yet pleasant environment for research in cryptography. This was the merit of the individual COSIC members, all colleagues that I highly appreciate. One person deserves a special mention for the unique role she played, and still plays, at COSIC: Péla

Noë, our valued secretary. She would address all practical questions except financial ones: these were competently handled by Elvira Wouters.

I would also like to acknowledge the Fund for Scientific Research – Flanders in Belgium (FWO), and the Austrian Science Fund (FWF), whose financial support has made this research possible.

Finally, I would like to thank my parents and sisters for their support and encouragement throughout all these years. I am especially grateful to my father, for his increasingly frequent reminders of the fact that it was about time to finish this PhD, and for his invaluable help in proofreading large parts of this thesis.

Christophe De Cannière
May 2007

Abstract

Symmetric encryption is the oldest branch in the field of cryptology, and is still one of the most important ones today. This thesis covers several aspects of the analysis and design of modern symmetric encryption algorithms.

The thesis starts with an overview of the different types of encryption algorithms, pointing out their security requirements and their fundamental differences in approach. We then focus on block ciphers and explain the basic strategies and techniques used in modern cryptanalysis. In order to illustrate this, we present some concrete attacks, which are based on a number of new ideas, but still serve as nice examples of the basic approach. One of the most generic and powerful analysis techniques is linear cryptanalysis, which is why we devote a separate chapter to it. We derive a new theoretical framework which allows to rigorously and accurately analyze the performance of this attack, and at the same time extend the attack to simultaneously exploit multiple linear approximations.

In the second part of this thesis, we concentrate on design aspects of symmetric encryption algorithms. We first present tools to analyze and classify substitutions boxes, components which play an important role in the design of block ciphers which resist the attacks discussed in the first part. We then present a new design strategy for stream ciphers based on techniques similar to those used to strengthen block ciphers against linear cryptanalysis. This eventually leads to the specification of a compact and elegant new stream cipher called TRIVIUM.

Beknopte Samenvatting

Symmetrische encryptie is de oudste discipline in het domein van de cryptologie, en ze is tot op vandaag nog steeds een van de belangrijkste. Deze thesis behandelt verschillende aspecten die een rol spelen in de analyse en het ontwerp van moderne symmetrische encryptie-algoritmen.

De thesis start met een overzicht van de verschillende soorten encryptie-algoritmen. We bestuderen wat hun veiligheidsvereisten zijn, en op welke verschillende manieren ze die trachten te bereiken. Vervolgens spitsen we ons toe op blokcijfers en lichten we de basistechnieken en strategieën toe die gebruikt worden in de moderne cryptanalyse. Om dit te illustreren, stellen we twee concrete aanvallen voor die steunen op een aantal nieuwe ideeën, en tegelijkertijd een goed beeld geven van de gangbare algemene aanpak. Een van de meest algemeen toepasbare en krachtige analysetechnieken is lineaire cryptanalyse, en om die reden besteden we er een afzonderlijk hoofdstuk aan. We ontwikkelen een nieuw theoretisch kader dat toelaat om de efficiëntie van deze aanvallen nauwkeurig te berekenen, en breiden de aanvallen tegelijk ook uit om simultaan gebruik te maken van meerder lineaire benaderingen.

In het tweede gedeelte van deze thesis, richten we ons op ontwerpaspecten van symmetrische encryptie-algoritmen. Eerst ontwikkelen we algoritmen om substitutie-boxen (S-boxen) te analyseren en te classificeren. S-boxen spelen een belangrijke rol in het ontwerp van blokcijfers die bestand zijn tegen de aanvallen besproken in het eerste gedeelte van deze thesis. Vervolgens stellen we een nieuwe ontwerpmethode voor om stroomcijfers te ontwikkelen. De methode is gebaseerd op technieken die gelijkenissen vertonen met degene die gebruikt worden om blokcijfers te wapenen tegen lineaire cryptanalyse. Dit leidt uiteindelijk tot de ontwikkeling van een compact en elegant nieuw stroomcijfer, TRIVIUM.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Cryptology

Over the past decades, information technology has infiltrated more and more areas of our society. The development of digital information and telecommunication systems opened a wide range of new possibilities which were seized to improve the efficiency of different sorts of processes. Today, an ever-increasing number of interactions between end users, organizations such as banks, and governments is carried out electronically. Two of the most remarkable breakthroughs were the world-wide expansion of the Internet and the spectacular growth of digital mobile networks (e.g., GSM). The number of users of both systems, which were nonexistent or confined to research communities until the early 1990s, is now in the order of a billion.

The success of these new technologies can be attributed to a number of intrinsic advantages of digital systems: digital information is nearly insensitive to noise, it can be sent over long distances, copied or modified without any loss of quality. Moreover, the link between the information and its carrier has disappeared. The exact same piece of information can be transmitted over a wireless link, sent over an optical fiber, stored on a hard disk, and printed on a barcode. This allows a large number of very different devices to interact seamlessly.

However, the same properties which make digital information systems so attractive render them particularly vulnerable to a broad range of abuses. In a traditional mail system, the receiver of a letter can perform some simple tests to assure himself that the message was not compromised. He can check that the (sealed) envelope was not opened, study whether the handwriting or signature matches, and search for anomalies which might indicate that parts have been rewritten. These tests are all built on the assumption that any manipulation of the message will necessarily leave some traces on its physical carrier. Unfortunately, this is exactly what digital systems have tried to avoid. As was soon understood, the only way to secure digital systems without sac-

rificing their advantages, is to transform the information in such a way that it protects itself, independently of how it is transferred or stored. The science which studies this problem is called *cryptology*, and an excellent (but maybe slightly outdated) overview of this field can be found in Menezes et al. [82].

It should be noted that many of the security issues raised by modern information technology are not new. Still, owing to the large scale of current information systems and the unprecedented impact they have on our daily live, cryptology has never been more important than today. Rapidly expanding networks are interconnecting more and more devices all over the globe, increasing both the number of interesting targets and the number of potential attackers. Moreover, eavesdropping on these networks has become much easier with the proliferation of wireless access points. Finally, the growing complexity of communication and information systems makes their security much harder to control, giving rise to some rather unexpected new problems such as computer viruses and worms.

1.2 Symmetric Encryption

The protection of digital information typically involves at least two distinct problems: *secrecy protection* (preventing information from being disclosed to unintended recipients) and *authentication* (ensuring that received messages originate from the intended sender, and were not modified on their way). This thesis is entirely devoted to the first problem, with the exception of the appendix, which discusses some specific aspects of the second one.

In cryptology, intended senders and recipients are distinguished from unintended ones by assuming that they know some secret pieces of information, called *keys*. These keys can be shared between the sender and the receiver, or they can be different, in which case the sender and receiver are also prevented from impersonating each other. In this thesis, we will concentrate on the first case, called *symmetric* cryptography.

Symmetric cryptography addresses the problem of secrecy protection by using the shared secret key to transform the message in such a way that it cannot be recovered anymore without this key. This process is called *symmetric encryption*. Algorithms which perform symmetric encryption are known as *ciphers*. Based on the paradigm used to process the message, these ciphers are typically categorized into one of two classes: *block ciphers* and *stream ciphers*.

The security of symmetric encryption algorithms can in general not be proved (the notable exception being the one-time pad). Instead, the trust in a cipher is merely based on the fact that no weaknesses have been found after a long and thorough evaluation phase. This explains the importance of a strong interaction between *cryptography*, the field which studies techniques to protect information, and *cryptanalysis*, which focuses on methods to defeat this protection. In this thesis, we will promote simplicity as an effective catalyst to enhance this interaction. While simple designs may be more likely to

be vulnerable to simple, and possibly devastating, attacks, they certainly inspire more confidence than complex schemes, if they survive a long period of public scrutiny despite their simplicity.

1.3 About this thesis

In this thesis, we will consider several aspects of symmetric encryption, both from a cryptanalyst's point of view, and from the perspective of a designer. The general outline of the thesis, and the main contributions presented in the different chapters, are summarized below:

- In Chap. 1, we outline the structure of this thesis. (← you are here)
- In Chap. 2, we provide a general introduction to symmetric encryption. We point out the fundamentally different approach taken by block encryption schemes and stream encryption schemes, review several block cipher modes and stream cipher constructions, and explain how their security is assessed.
- In Chap. 3, we concentrate on the cryptanalysis of block ciphers. We explain the basic attack strategy of building a distinguisher and guessing round keys, and describe three widely used attack techniques based on this approach: differential cryptanalysis, linear cryptanalysis, and multiset cryptanalysis. We then present two dedicated attacks (against SAFER++ and ARIA) which introduce some novel ideas, but still nicely illustrate the basic attack strategy. Parts of this chapter are based on [34], and some of the results are published in [19, 20].
- In Chap. 4, we provide a much more rigorous analysis of linear cryptanalysis. We introduce a general statistical framework, which allows to analyze and devise attacks exploiting multiple linear approximations. We show that this extension of linear cryptanalysis can lead to more efficient attacks, and that it sheds some new light on the relation between two previously known attacks. The results of this chapter are published in [21].
- In Chap. 5, we focus on substitution boxes (S-boxes) as an important building block in the design of ciphers which resist the linear attacks described above. We argue that their most relevant properties are invariant under affine transformations, and present efficient algorithms to detect the affine equivalence of two S-boxes. We introduce the concept of a linear representative, and show how this can be used to effectively reduce the set of all 2×10^{13} 4×4-bit S-boxes to a list of 302 candidates. Parts of this chapter are published in [18].

- In Chap. 6, we present a new design strategy for stream ciphers based on the same techniques that are used to protect block ciphers against linear cryptanalysis. We show that a carefully designed linear structure can be surprisingly efficient in diffusing the nonlinearity of very few small nonlinear components. Based on this strategy, we develop a compact and efficient hardware oriented stream cipher called TRIVIUM. The design ideas presented in this chapter are published in [30], and TRIVIUM itself has been submitted to the eSTREAM Stream Cipher Project [31]. At the time of writing, it has successfully passed two selection rounds, and is amongst the 8 hardware candidates retained for the third evaluation phase.
- In Chap. 7, we conclude this thesis with some ideas for further research.
- As a supplement to this thesis, we present in App. A some results of our recent research on hash functions [32]. While these results do not involve symmetric encryption, and are therefore not included in the main part of this thesis, they do have some relevance in the area of symmetric encryption because of the duality between differential characteristics in hash functions on the one hand, and linear characteristics in stream ciphers on the other hand.

As indicated above, the main results presented in this thesis have been published in [18, 19, 20, 21, 30, 32, 34]. Other results obtained during this doctoral research, which are not considered in this thesis, can be found in the publications [5, 16, 33, 35, 109, 114].

Chapter 2

Symmetric Encryption

This chapter introduces what will be the main objects of study in this thesis: symmetric encryption algorithms. We explain what they are, what properties they are supposed to have, and what they typically look like.

2.1 Encryption Algorithms

The purpose of an encryption algorithm is to protect the secrecy of messages which are sent over an insecure channel. A general encryption algorithm consists of two mathematical transformations: an encryption function E, and a decryption function $D = E^{-1}$. In order to communicate in a secure way, the sender (traditionally called Alice) will apply the encryption function to the original message P (the *plaintext*), and transmit the resulting *ciphertext* $C = E(P)$ over the insecure channel. Once C is received by the intended recipient (called Bob), the plaintext is recovered by computing $D(C) = P$.

In order for this scheme to meet its purpose, i.e., to ensure that Alice's message will only be read by Bob, a number of conditions need to be fulfilled. First, the decryption function D must be known to Bob but kept secret from anybody else, with the possible exception of Alice. Secondly, the transformation E must be designed in such a way that an eavesdropper intercepting the ciphertext (often called Eve) cannot, at least in practice, extract any information about the plaintext, except maybe its length. Finally, the implementations of the transformations E and D should not require a prohibitive amount of computational resources.

2.1.1 Symmetric vs. Asymmetric Encryption

Until the 1970s, it was intuitively assumed that the previous conditions immediately implied that the encryption function E had to be secret as well. The reasoning was that if Eve were given E, it would suffice for her to reverse

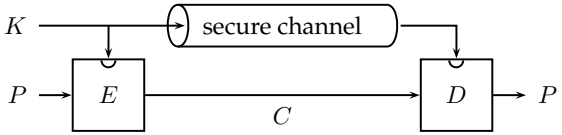


Figure 2.1: Model for symmetric encryption

this transformation to recover D. In the mid-1970s Diffie and Hellman [36] realized that the secrecy of the encryption function was in fact not required, at least in theory, provided that one could construct so-called trapdoor one-way functions. These are functions which are easy to evaluate, but cannot be inverted efficiently, unless some extra information (the trapdoor) is given. Examples of trapdoor one-way functions were soon found, and allowed the development of practical public key encryption algorithms such as RSA [95].

While public key cryptography has the major advantage that Bob does not need to exchange any secret information with Alice before she can start encrypting, schemes which do rely on the secrecy of their encryption function still play a vital role in practical systems. The reason is that implementations of *secret key* or *symmetric* encryption algorithms, as they are called nowadays, are orders of magnitude more efficient than their public key (or asymmetric) counterparts. As its title suggests, this thesis will exclusively deal with symmetric encryption algorithms.

2.1.2 Kerckhoffs' Principle

In most situations, it is fairly hard to keep an encryption or decryption algorithm completely secret: either Alice and Bob have to design and implement their own algorithm, or they have to trust a designer not to disclose the algorithm to others. Moreover, for each correspondent Alice wants to communicate with, she will need a different algorithm. The solution to this problem is to introduce a secret parameter K as in Fig. 2.1, and to construct parametrized encryption and decryption functions, in such a way that $D_{K'}(E_K(P))$ does not reveal anything about P as long as $K' \neq K$. Instead of repeatedly having to design new secret algorithms, it now suffices to agree on a secret value for K, called the *key*. Typically, this key is a short binary string of 80 to a few hundred bits. Since the security of the resulting system only relies on the secrecy of the key, the functions E and D can now be shared and even made public. The principle that full disclosure of the underlying algorithms should never affect the security of a good encryption scheme, as long as the key is kept secret, is known as *Kerckhoffs' principle*.

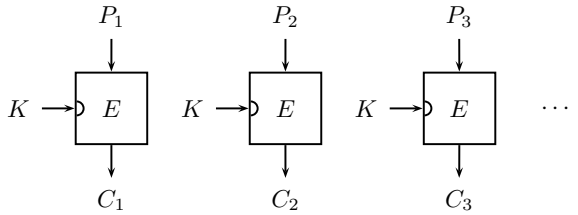


Figure 2.2: A block cipher in so-called ECB mode (see p. 11)

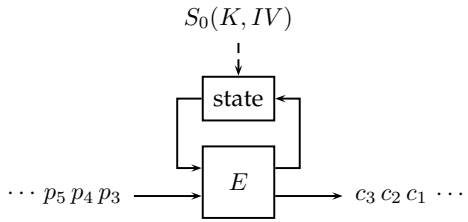


Figure 2.3: A stream cipher

2.1.3 Stream and Block Encryption

Let us now take a closer look at the boxes E and D in Fig. 2.1. In practice, these boxes will not take a complete text as input and then transform it at once, but rather operate in a sequential fashion. With this respect, symmetric encryption algorithms are traditionally divided into two categories: *stream ciphers* and *block ciphers*. A block cipher divides the plaintext into separate blocks of fixed size (e.g., 64 or 128 bits), and encrypts each of them independently using the same key-dependent transformation. A stream cipher, on the other hand, takes as input a continuous stream of plaintext symbols, typically bits, and encrypts them according to an internal state which evolves during the process. The initialization of this state is controlled by the secret key K and a public initial value IV. The differences between both systems are illustrated in Fig. 2.2 and Fig. 2.3.

While the definitions above would at first sight allow to draw a clear theoretical distinction between stream ciphers and block ciphers based on the presence of a state, the situation is a bit more blurred in practice. The fact is that block ciphers are rarely used in the way shown in Fig. 2.2. Instead, the output of the key-dependent transformation is typically kept in memory and used as a parameter when encrypting the next block. While this approach, known as cipher block chaining (see Sect. 2.1.5), would fall into the category of stream encryption according to the previous definition, it is still commonly called block encryption.

In order to resolve this apparent inconsistency, we will follow Daemen's [26]

suggestion, and make a distinction between a block cipher, which is just an invertible key-dependent transformation acting on blocks of fixed length, and a *block encryption scheme*, which encrypts plaintexts of arbitrary length and will typically use a block cipher as a component. While a block cipher is stateless by definition, this is rarely the case for a block encryption scheme.

The need for a state in a block encryption scheme arises from a fundamental difference in approach between block ciphers and stream ciphers. A stream cipher tries to defeat the adversary by making the encryption of a plaintext symbol depend in an unpredictable way on the *position* in the stream. A block cipher, on the other hand, aims to make its output depend in an unpredictable way on the *value* of the plaintext block. A consequence of the latter approach is that repeated blocks in a plaintext message are easily detected at the output of a block cipher, thus providing the adversary with possibly useful information. The purpose of the state in a block encryption scheme is precisely to make such repetitions unlikely, by first "randomizing" the plaintext blocks before they are fed into the block cipher.

The different roles played by the internal states of block and stream encryption schemes are reflected in the following informal definitions:

Definition 2.1 (block encryption)**.** A block encryption scheme is an encryption scheme whose state, if it has one, can be kept fixed without significantly reducing its security, provided that the plaintext symbols are independent and uniformly distributed.

Definition 2.2 (stream encryption)**.** A stream encryption scheme is an encryption scheme whose state cannot be kept fixed without severely reducing its security, even if the plaintext symbols are independent and uniformly distributed.

It is interesting to note that the two branches in symmetric cryptology have evolved in rather different circumstances. Block ciphers owe much of their popularity to a few successful designs (such as DES [86] and its successor, AES [85]) which are standardized, freely available, and can be deployed in many different applications. The most widely used stream ciphers, on the contrary, are proprietary designs (e.g., RC4, A5/1), closely tied to a particular application (e.g., GSM). Many of these designs were kept secret, until they eventually leaked out, or were reverse-engineered. This explains why, in the 1990s, stream ciphers have tended to receive less attention from the open research community than block ciphers.

2.1.4 Anatomy of a Block Cipher

Whereas stream ciphers are based on a variety of principles, most block cipher designs follow the same general approach. They typically consist of a short sequence of simple operations, referred to as the *round function*, which is repeated r times (called *rounds*). The first round takes an n-bit plaintext block

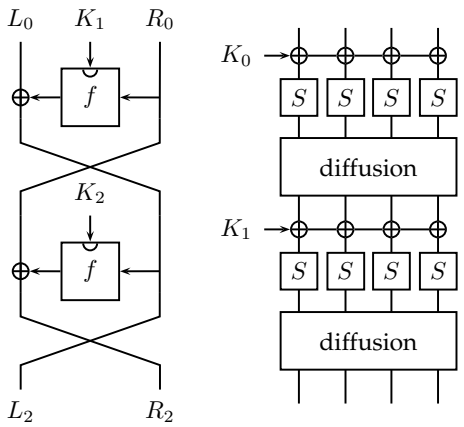


Figure 2.4: Feistel cipher vs. SP network

as input, and the last round outputs the ciphertext. Additionally, each round depends on a *subkey* (or *round key*) which is derived from a k-bit secret key (this derivation process is called the *key schedule*). Since the receiver must be able to uniquely decrypt the ciphertext, the round function has to be bijective for any value of the secret key. This is usually achieved in one of the following ways:

Feistel Ciphers. The round function of a Feistel cipher (named after H. Feistel, one of the IBM researchers who designed LUCIFER and DES) splits the input block into two parts L_{i-1} and R_{i-1}. The right part R_{i-1} is left unchanged and forms the left part of the output L_i. The right part of the output is constructed by adding a modified copy of R_{i-1} to the left part of the input L_{i-1}, i.e.,

$$L_i = R_{i-1}\,,$$
$$R_i = L_{i-1} + f(R_{i-1}, K_i)\,.$$

It is not hard to see that this operation can be inverted by subtracting $f(L_i, K_i)$ from R_i, no matter how the function f is constructed. Many block ciphers are based on this structure, including DES.

SP Networks. Another approach consists in building a round function by combining layers of simple invertible functions: substitutions (called *S-boxes*) and permutations. The substitution layers act on small units of data (rarely more than 8 consecutive bits), and their highly nonlinear properties introduce local *confusion* into the cipher. The permutation layers, on the other hand, are simple linear transformations, but they

operate on the complete block, and thus *diffuse* the effect of the substitutions. The terms confusion and diffusion, which were introduced by Shannon [100], will be a recurrent theme in this thesis.

The most prominent block cipher based on an SP network is the Advanced Encryption Standard (AES). Notice also that the f-functions of many Feistel ciphers consist of a small SP network.

2.1.5 Modes of Operation

As we already mentioned in Sect. 2.1.3, a block cipher in itself is just a component which describes a set of invertible transformations on blocks of fixed length n. Before Alice can actually start encrypting data, she needs to turn this block cipher into an encryption scheme. The different ways in which this can be achieved are called *modes of operation*. The purpose of a mode of operation is to extend the cryptographic properties of a block cipher to larger messages. The property which this thesis mainly focuses on is confidentiality, but modes providing message integrity and authenticity, possibly in addition to confidentiality, exist as well.

Although security obviously remains the primary criterion, other (non-cryptographic) considerations often play an equally important role in the selection of a mode of operation:

Data expansion. Some constructions require the plaintext length to be an exact multiple of the block length n. This implies that the original message may have to be expanded with extra padding bits, which is usually undesirable.

Error propagation. Single bit transmission errors may have different effects on the decrypted ciphertext. Either the error only affects a single bit or block of the recovered plaintext, or it might propagate to one, a few or all subsequent blocks.

Random access. A number of modes allow ciphertext blocks to be decrypted (or even modified) at arbitrary positions without first having to process all preceding blocks. This is particularly useful for storage encryption.

Parallel processing. Some modes allow different blocks to be processed simultaneously, which may be an interesting way to increase the throughput in certain applications.

We now shortly review some of the most common confidentiality modes. Except for the OCB mode, all of these modes have been standardized by NIST in the first part of Special Publication 800-38 [87]. The remaining three parts of SP 800-38, which describe authentication and authenticated encryption modes, are definitely a recommended read, but lie beyond the scope of this thesis.

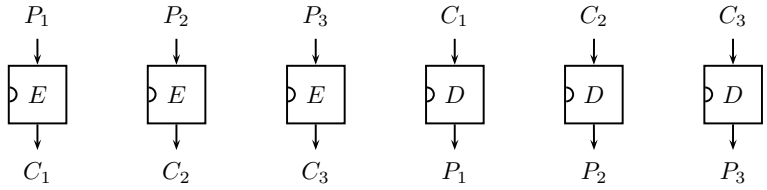


Figure 2.5: A block cipher in ECB mode

Block Encryption Modes

The first three modes in this section turn a block cipher into a block encryption scheme as defined in Sect. 2.1.3.

Electronic Codebook Mode (ECB). The ECB mode is without doubt the most straightforward way to encrypt messages whose length exceed the block length: the message is simply partitioned into n-bit blocks, each of which is encrypted independently. The scheme is depicted in Fig. 2.5 and can be described as follows:

Encryption:	Decryption:
$C_i = E_K(P_i)\,.$	$P_i = D_K(C_i)\,.$

The advantages of this mode are its simplicity and its suitability for parallel processing. Blocks at arbitrary positions can be encrypted or decrypted separately and errors do not propagate from one block to another. As mentioned in Sect. 2.1.3, however, the major problem of this approach is that it does not hide all patterns in the plaintext: i.e., whenever the plaintext contains identical blocks, so will the ciphertext. This limits the applications of the ECB mode to those (rare) cases where all blocks encrypted with a single key are guaranteed to be different.

Cipher-Block Chaining Mode (CBC). The CBC mode, which is presently the most widely used mode of operation, masks each plaintext block with the previous ciphertext block before applying the block cipher (see Fig. 2.6):

Encryption:	Decryption:
$C_0 = IV\,,$	$C_0 = IV\,,$
$C_i = E_K(C_{i-1} \oplus P_i)\,.$	$P_i = D_K(C_i) \oplus C_{i-1}\,.$

Since the output of a good block cipher is supposed to be completely unpredictable for anyone who does not know the key, all consecutive values of $C_{i-1} \oplus P_i$ will appear to be independent and uniformly distributed,[1] and this regardless of the plaintext (assuming that the text

[1] Note that this is not automatically the case for $i = 0$, and hence, when different messages are encrypted under the same key, care should be taken that the initial values (IV's) are sufficiently unpredictable as well.

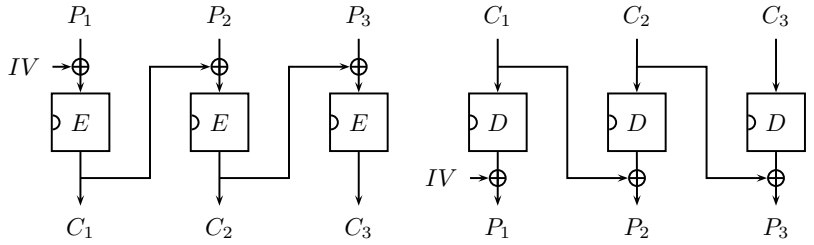


Figure 2.6: A block cipher in CBC mode

itself does not depend on the key). Repetitions at the input of the block cipher are therefore unlikely to occur, which remedies the main shortcoming of the ECB mode. However, when the message length exceeds $2^{n/2}$ blocks, repeated values start to be unavoidable because of the birthday paradox. For this reason, the CBC mode (and in fact all modes in this section) should never be used to encrypt more than $2^{n/2}$ blocks with the same key.

The cost of masking the plaintext in CBC is that the ciphertext feedback in the encryption part prevents the blocks from being processed in parallel. The decryption, on the other hand, depends only on two consecutive ciphertext blocks, and can still be performed independently for each block. This has the additional benefit that a bit error in the ciphertext can only affect the decryption of two blocks.

Offset Codebook Mode (OCB). The masking in CBC effectively destroys all dependencies in the plaintext. However, if the only purpose is to prevent repeated input blocks, then it suffices to require that the blocks are *pairwise* independent, or even weaker, just pairwise differentially uniform, i.e., that the difference between any two input blocks is uniformly distributed. The IAPM mode by Jutla [57] and, derived from it, the OCB mode by Rogaway et al. [96] are block encryption modes based on this observation. Both modes are variants of the ECB mode in which a stream of words Z_i is added before and after the encryption (see Fig. 2.7):

Encryption:	Decryption:
$C_i = E_K(P_i \oplus Z_i) \oplus Z_i\,.$	$P_i = D_K(C_i \oplus Z_i) \oplus Z_i\,.$

In the case of OCB, the words Z_i are distinct multiples of a secret parameter L, which is derived from the key K by encrypting a nonce (i.e., a value used only once).

A first advantage of OCB compared to CBC is that it is completely parallelizable, both for encryption and decryption. But what makes the mode particularly attractive is the fact that it provides message authenticity at a very small additional cost: it suffices to compute a simple (non-

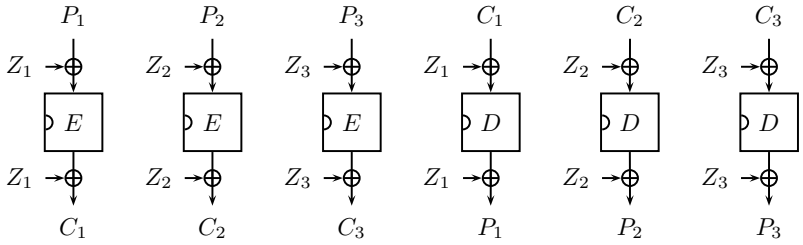


Figure 2.7: A block cipher in OCB mode

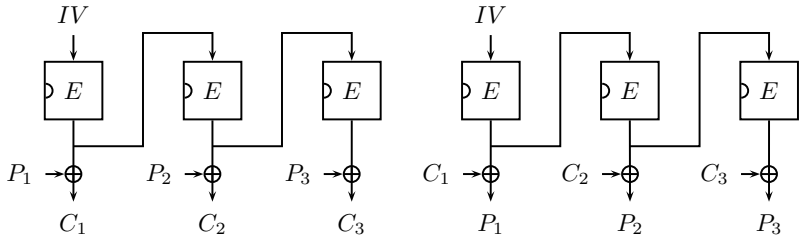


Figure 2.8: A block cipher in OFB mode

cryptographic) checksum of the plaintext, and to encrypt it with the same key K.

A drawback for the deployment of OCB is its intellectual property situation, which is somewhat unclear. This explains why the OCB mode was relegated to optional status in the IEEE 802.11i standard, in favor of the mandatory CCM mode, which is not encumbered by patents.

Stream Encryption Modes

Block ciphers can also be used to perform stream encryption, as illustrated by the three modes below. A noteworthy feature of these modes is that they only use the encryption function of the block cipher.

Output Feedback Mode (OFB). The OFB mode, depicted in Fig. 2.8, encrypts plaintext blocks by combining them with a stream of blocks called *key stream*, which is generated by iterating the block cipher:

Encryption:	Decryption:
$Z_0 = IV\,,$	$Z_0 = IV\,,$
$Z_i = E_K(Z_{i-1})\,,$	$Z_i = E_K(Z_{i-1})\,,$
$C_i = P_i \oplus Z_i\,.$	$P_i = C_i \oplus Z_i\,.$

The generation of key stream blocks in OFB is independent of the plaintext. This means that the stream can be precomputed as soon as the IV

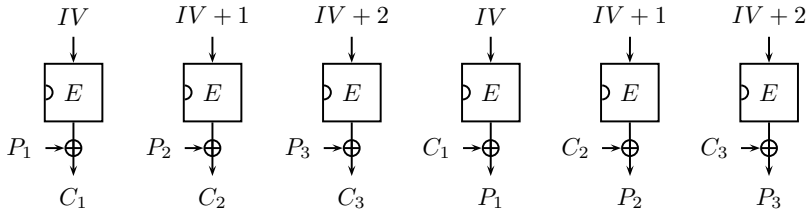


Figure 2.9: A block cipher in CTR mode

is known, a feature which may be useful in real-time applications. The mode is strictly sequential, though: the decryption of a single block at an arbitrary position in the ciphertext requires all preceding key stream blocks to be computed first.

Owing to the invertibility of E_K, all Z_i will necessarily be different, until one of them hits the value of Z_0 again, at which point the sequence will start repeating itself. A secure n-bit block cipher is not expected to cycle in much less than 2^{n-1} blocks, which implies that this periodicity has no practical consequences for a typical 128-bit block cipher. The mere fact that all Z_i within a cycle are different leaks some information as well, though. As a consequence, it is not recommended to encrypt much more than $2^{n/2}$ blocks with a single key.

Counter Mode (CTR). The CTR mode takes a similar approach as the OFB mode, but this time the key stream is generated by encrypting a counter:

Encryption:	Decryption:
$Z_0 = IV\,,$	$Z_0 = IV\,,$
$C_i = P_i \oplus E_K(Z_i)\,,$	$P_i = C_i \oplus E_K(Z_i)\,,$
$Z_{i+1} = Z_i + 1\,.$	$Z_{i+1} = Z_i + 1\,.$

As opposed to the OFB mode, the CTR mode allows data blocks at arbitrary positions to be processed independently, both during encryption and decryption. This also allows pipelining in hardware, which can result in significant efficiency gains. Apart from this feature, the OFB and the CTR mode have very similar properties.

Cipher Feedback Mode (CFB). Both OFB and CTR (or OCB for that matter) require perfect synchronization during decryption, i.e., in order to decrypt a ciphertext block, the receiver needs to know the block's exact position in the stream. The CFB mode eliminates this requirement, and is similar to CBC in this respect.

The CFB mode is designed to process messages in r-bit segments, with $1 \le r \le n$ (typically $r = 1$, $r = 8$, or, as in Fig. 2.10, $r = n$). The encryption mode consists in shifting successive r-bit ciphertext segments back into an internal state block S_i, and combining the leftmost bits of $E_K(S_i)$ with the plaintext:

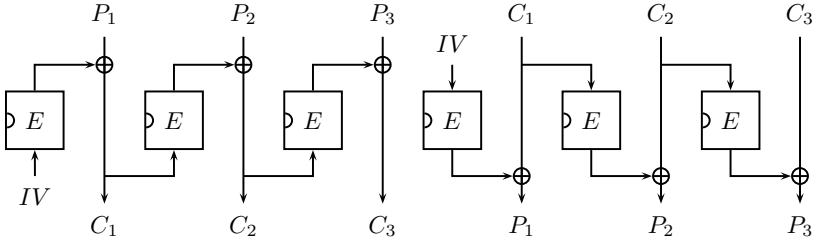


Figure 2.10: A block cipher in CFB mode

Encryption:	Decryption:
$S_1 = IV\,,$	$S_1 = IV\,,$
$C_i = P_i \oplus E_K(S_i)[1 \cdots r]\,,$	$P_i = C_i \oplus E_K(S_i)[1 \cdots r]\,,$
$S_{i+1} = (S_i \ll r) + C_i\,.$	$S_{i+1} = (S_i \ll r) + C_i\,.$

The feedback in CFB prevents the parallel encryption of plaintext blocks. Still, arbitrary ciphertext blocks can be decrypted independently, provided that the $\lceil n/r \rceil$ preceding blocks are available. As a direct consequence, single bit errors in the ciphertext cannot propagate over more than $\lceil n/r \rceil$ successive blocks.

Again, and for similar reasons as in CBC, a single key should not be used to encrypt more than $2^{n/2}$ blocks. For small values of r, additional precautions should be taken in order to avoid weak IV values. In particular, if the bits of the IV were to form a periodic sequence, then this would considerably increase the probability of repeated values at the input of the block cipher.

2.1.6 Dedicated Stream Ciphers

In its most general form, a stream cipher consists of a transformation which takes as input a plaintext symbol and the current state, and combines both to produce two outputs: a ciphertext symbol and the next state. This general construction was depicted in Fig. 2.3. In the vast majority of practical ciphers, however, the transformation E can be further decomposed into separate functions in either of the three ways shown in Fig. 2.11.

Synchronous Stream Ciphers

The first and by far most common approach is to update the state independently from the plaintext, in which case the stream cipher is said to be *syn-*

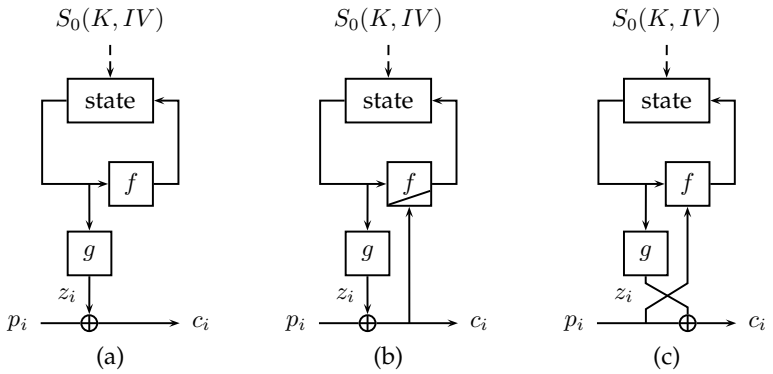


Figure 2.11: Three types of stream ciphers
(a) binary additive, (b) self-synchronizing, and (c) accumulating

chronous. The computations performed by a synchronous stream cipher can be described by three functions f, g, and h. The function f computes the next state, the function g produces key stream symbols z_i, and the function h outputs ciphertext symbols by combining plaintext and key stream symbols:

Encryption:	Decryption:
$z_i = g(S_i)\,,$	$z_i = g(S_i)\,,$
$c_i = h(p_i, z_i)\,,$	$p_i = h^{-1}(c_i, z_i)\,,$
$S_{i+1} = f(S_i)\,.$	$S_{i+1} = f(S_i)\,.$

In addition to being synchronous, most modern stream ciphers are also *binary additive*, which means that their combining function h is simply defined as $c_i = p_i \oplus z_i$, as in Fig. 2.11 (a).

Note that we already saw two special examples of synchronous stream ciphers in the previous section, namely the OFB mode and the CTR mode. A notable common feature of these block cipher based constructions is the way in which they use their state: one part of the state (the key K) is kept secret, but stays constant during the whole encryption process; the other part is continuously updated, but is either known to the attacker, or directly used as key stream and thus easily derived from fragments of known plaintext. With respect to the functions f and g, OFB and CTR take diametrical approaches. The OFB mode uses an extremely simple h-function, and relies completely on the strength of f for its security. The CTR mode works the other way around.

In order to keep deriving unpredictable key stream bits from a state whose bits are all either constant or known, the OFB and CTR constructions have to place heavy demands on either f or g, and this justifies the need for a relatively complex component such as a block cipher. Of course, nothing forces a synchronous stream cipher to use its state in this particular way. In fact, most

dedicated synchronous stream ciphers will make sure that none of the state bits stay constant for more than a few iterations, and that only a fraction of these bits are revealed to the attacker at any time. This strategy, as opposed to the OFB and CTR constructions, allows the cipher to gradually accumulate unpredictable bits in its state. The advantage of this approach is that its security depends more on the interaction between f and g, than on the individual functions themselves. In principle, neither of these functions is required to be particularly strong on its own, which explains why dedicated synchronous stream ciphers have the potential to be significantly more efficient and compact than block cipher based schemes, as will be illustrated in Chapt. 6.

Examples of synchronous stream ciphers include modern stream ciphers, such as the widely used RC4 [94] and A5/1 [1], but also historical rotor-based machines such as Enigma (the latter is an illustration of a non-additive synchronous stream cipher). Note that even though all these ciphers share the same high-level structure, the design of their components is often based on very different principles.

Self-Synchronizing Stream Ciphers

The synchronous stream ciphers in the previous section derive their name from the fact that their state must be perfectly synchronized with the incoming ciphertext stream in order to recover the plaintext. On unreliable channels, this requires an external synchronization mechanism, which can be impractical in certain applications. In these cases, self-synchronizing stream ciphers can come in handy. The underlying idea of self-synchronizing stream ciphers is to use the ciphertext stream itself to synchronize the state. The encryption and decryption operations are defined as follows:

Encryption:	Decryption:
$z_i = g(S_i)\,,$	$z_i = g(S_i)\,,$
$c_i = h(p_i, z_i)\,,$	$p_i = h^{-1}(c_i, z_i)\,,$
$S_{i+1} = f(S_i, c_i)\,.$	$S_{i+1} = f(S_i, c_i)\,.$

The function f is non-injective with respect to S_i, and defined in such a way that the state S_i can always be computed as a function of the initial state and the last t ciphertext symbols, i.e.,

$$S_i = f(f(\cdots f(S_0, c_{i-t})\cdots), c_{i-1})\,, \quad \forall i \geq t\,. \tag{2.1}$$

Fig. 2.11 (b) shows the encryption mode of a self-synchronizing stream cipher where $h(p_i, z_i) = p_i \oplus z_i$. The narrowing shape of f symbolizes the non-injective nature of the function.

Block ciphers in CFB mode (see p. 14) are the most widely used self-synchronizing stream ciphers. The state in this case consists of a constant part which stores the secret key, and a variable part which contains the last t =

$\lceil n/r \rceil$ ciphertext symbols. The state update function f is just a shift of this second part. As in CTR mode, the security of CFB completely relies on the strength of g.

Dedicated (as opposed to block ciphers based) self-synchronizing stream ciphers are relatively rare. The reason for this is that the switch to a dedicated self-synchronizing stream cipher is not likely to result in the same efficiency gain as in the synchronous case. A first limitation is that self-synchronizing stream ciphers cannot accumulate unpredictability in their state to the same extent as synchronous stream ciphers, simply because (2.1) does not allow f to perform any computation that would affect the state for more than t iterations. The second complication is that the state update depends on ciphertext symbols, which means that an adversary who can influence the ciphertext (either by corrupting the communication channel, or by controlling parts of the plaintext), can also influence how the state is updated. In particular, the adversary can force the decrypting cipher to return to any previous state by replaying fragments of the ciphertext. This allows chosen-ciphertext (or -plaintext) attack scenarios which do not apply to synchronous stream ciphers (see Sect. 2.2.1). Because of these two structural properties, self-synchronizing designs still have to rely to a large extent on the individual strength of the combined function $g \circ f$, which limits the potential for large efficiency gains.

Accumulating Stream Ciphers

A third class of stream ciphers, which has only started to appear very recently, follows the structure of Fig. 2.11 (c), and can be described by the following equations:

Encryption:	Decryption:
$z_i = g(S_i)\,,$	$z_i = g(S_i)\,,$
$c_i = h(p_i, z_i)\,,$	$p_i = h^{-1}(c_i, z_i)\,,$
$S_{i+1} = f(S_i, p_i)\,.$	$S_{i+1} = f(S_i, p_i)\,.$

The expressions above resemble somewhat the equations of a self-synchronizing stream cipher, but in fact, the construction serves the opposite purpose. Whereas a self-synchronizing stream cipher aims to limit the effect of communication errors (bit flips, insertions, or deletions), the goal of the current construction is to make sure that any modification in the ciphertext (be it accidental or malicious) would have a very high probability to permanently disturb the state. In order to achieve this, condition (2.1) is dropped, allowing the function f to be invertible with respect to both S_i and p_i. The class of ciphers based on this construction has not been given a special name in the literature so far, but in this thesis we will refer to them as *accumulating stream ciphers*.

The main use of accumulating stream ciphers is to perform authenticated encryption. When used for this purpose, the state has to undergo some additional processing at the end of the encryption process, and the result is used as a message authentication code (MAC). Phelix [110] and Shannon [50] are two ciphers based on this paradigm.

An issue that complicates the design of efficient and secure accumulating stream ciphers is the fact that the scheme provides the adversary with a means to influence the state, just as in self-synchronizing stream ciphers. This time, however, there is no obvious way for the adversary to reset the state. That is, unless she can force a reinitialization of the cipher with the same key and IV. If this possibility cannot be excluded, $g \circ f$ would need the same kind of strength as in a self-synchronizing stream cipher in order to resist attacks such as those presented by Wu and Preneel [112], and there would be little hope for the cipher to attain the same efficiency as a synchronous cipher.

2.2 Security Considerations

In the previous sections, we have introduced different types of symmetric encryption schemes. In several places, we have pointed out, in a rather informal way, which issues need to be considered in order for these schemes to be secure. However, although it was to some extent implied in the earlier discussion, we have not yet explicitly defined what we understand by a "secure" encryption scheme. Before we address this point, let us first identify possible attack scenarios.

2.2.1 Attack Scenarios

In the case of encryption, the task of the adversary Eve consists in recovering unknown parts of the plaintext, or better yet, recovering the secret key. Different attack scenarios can be distinguished depending on what information Eve can obtain, and to what extent she can interfere in the communication between Alice and Bob.

Ciphertext-only attack. This type of attack only assumes that Eve is capable of capturing encrypted text. As this is likely to be the case (otherwise there would be little reason to encrypt the messages in the first place), encryption schemes succumbing to cipher-text only attacks are considered to be particularly insecure.

Known-plaintext attack. A known-plaintext attack requires Eve to have access to (parts of) the plaintext corresponding to the captured ciphertext. This additional requirement is typically rather easy to fulfill. A good example is an online payment on the Internet: while the browser and the server will exchange several kilobytes of encrypted data, it is likely that the only unknown part is a 16-digit credit card number.

Chosen-plaintext attack. Some attacks only succeed when the plaintexts have a specific form. In order to mount such attacks, Eve must find a way to

influence the encrypted plaintexts. A practical example is a secure connection between Alice and her mail server. By sending carefully crafted mails to Alice, Eve can get the server to encrypt the plaintexts she needs.

Chosen-ciphertext attack. This attack requires Eve to have control over the ciphertexts sent to Bob, and to be capable of monitoring how they are decrypted. For example, Eve could try to attack a pay TV decoder by feeding it with special ciphertexts and analyzing its output. Notice that such attacks will not work if the receiver has a means to check the integrity of the ciphertexts.

Chosen-IV attacks. If the encryption scheme takes as input an IV, Eve might have means to control this value as well. At the receiver side, the IV typically needs to be derived from a header which is prefixed to the ciphertext. Hence, if Eve can corrupt ciphertexts, she can most likely modify the IV used during decryption as well.

Adaptively chosen-plaintext/ciphertext/IV attack. In order to mount one of the attacks described above, Eve will typically need to obtain the encryptions or decryptions of a whole series of chosen data blocks. When the choice of a certain block depends on the results obtained from previous blocks, the attack is called adaptive.

One could still imagine other attack scenarios. Related key attacks, for instance, where an attacker manages to obtain pairs of plaintexts and ciphertexts encrypted with different but related secret keys. Such attacks are worth studying when block ciphers are used to construct hash algorithms, for example. However, in the context of encryption schemes, these attacks are of limited relevance, since they can easily be prevented by choosing an appropriate key generation procedure, without affecting the efficiency of the scheme in any significant way.

2.2.2 Measuring Security

The security of symmetric encryption schemes has been analyzed from several perspectives in the literature.

Perfect Secrecy

The first rigorous treatment is given by Shannon [100], who introduces the concept of *perfect secrecy*.

Definition 2.3 (perfect secrecy)**.** An encryption scheme is called perfectly (or unconditionally) secure if the observation of the ciphertext does not change the a priori probability distribution of possible plaintexts, i.e., the plaintext and the ciphertext are statistically independent.

Shannon shows that this property is realized by the Vernam scheme (a.k.a. the *one-time pad*), but also proves that perfect secrecy can only be achieved by encryption schemes whose secret key is at least as long as the message. Clearly, such a requirement is highly inconvenient in practice, and this suggests the need for a more relaxed notion of security.

Computational Security

A concession, which is perfectly reasonable given the practical reality, is to require protection only against adversaries with bounded computational resources. The amount of resources that would be required to break this protection can then be used as a measure for the so-called *computational security* of an encryption scheme.

Definition 2.4 (computational security)**.** An encryption scheme is said to provide n bits of (computational) security if the most efficient attack requires a computational effort equivalent to an exhaustive search over 2^n values.

As opposed to perfect secrecy, which can typically easily be proved or (more likely) disproved, the computational security of a scheme is in general very hard to evaluate. Although an increasing number of encryption schemes follow design principles which allow to derive lower bounds on the computational cost of a successful attack, each of these bounds eventually relies either on an unproven assumption or on a restricted attack model. Based on the nature of these assumptions or restrictions, we can distinguish different types of lower bounds:

- The first type of bounds are those that consider only a restricted, well defined subset of attacks. This is a common practice in the design of symmetric primitives. Many recent block ciphers, for instance, come with provable bounds on the expected linear and differential probabilities of linear approximations and differentials. Provided that a number of additional assumptions are valid, these bounds can be translated into bounds on the efficiency of attacks based on standard differential and linear cryptanalysis (see Chap. 3). Of course, and as has been demonstrated more than once [29, 54, 104], this does not rule out the existence of efficient attacks based on different principles.

- A second type of bounds relies on the assumed security properties of one of the components of the scheme. They are derived by first modeling an attack scenario (preferably giving the adversary as much freedom as possible), and then proving that any efficient attack within this model would imply a violation of the component's assumed properties. This approach can be used to analyze the strength of general constructions such as the Feistel structure [73], or to reduce the security of modes of operation to the security of the underlying block cipher [9]. Reductions of this type provide interesting insights in how to use secure primitives

in a secure way; however, when it comes to designing these secure primitives, they are of relatively little help.

- The third type of bounds is based on the conjectured hardness of well-studied mathematical problems, such as factoring large composite number, computing discrete logarithms, or solving systems of multivariate quadratic equations. While such bounds are still not to be considered as absolute and conclusive security proofs, they do provide the assurance that no adversary will break the scheme without a major breakthrough in mathematics. This type of proofs has become an essential element in the design of asymmetric primitives. In the world of symmetric cryptography, they remain notoriously rare, though. The reason is that primitives which allow this sort of security reductions tend to be significantly slower than symmetric schemes designed in the traditional way. This effect is often reinforced by the need for relatively large security margins (see for instance [113]) to compensate for the fact that the exact complexity of several of these mathematical problems is difficult to evaluate, even though they are widely agreed to be asymptotically hard.

Encryption schemes which allow these sorts of reasoning are sometimes called "provably secure". However, as should be clear from the discussion above, such security proofs should always be interpreted with caution, and certainly they should not be seen as suggesting that there is no need for further analysis. The literature contains several examples of "provably secure" schemes which were eventually broken. In some cases, the discovered weaknesses can be traced back to errors in the proofs (which are sometimes rather complex), but more often the attacks do not invalidate the proofs themselves, but rather the assumptions which they rely on. Although security proofs can be very useful guidelines during the design stage, the most convincing indication of the security of a scheme remains the fact that it has survived a long period of scrutiny by the cryptographic community.

2.2.3 How Secure Should a Scheme Be?

In the previous section, we showed how the security of encryption schemes is measured by estimating how much computational effort it would take to break them. The next question is: how large should this estimated effort really be for a scheme to be considered secure?

From a practical point of view, it suffices for Alice and Bob to ensure that the computational effort exceeds, by some safety margin, what can be afforded by Eve. Estimates for the minimum level of security required to defend against different types of adversaries are listed in Table 2.2. The figures were derived by the ECRYPT NoE in 2007 [37] by estimating the amount of computation that can be carried out within a given budget over a limited period of a few months. Based on these numbers, it seems that a security level

of 80 bits should be considered a minimum requirement for achieving an acceptable degree of protection. Clearly, this level of security does not provide long-term protection against intelligence agencies or large organizations, but it is still amply sufficient for those applications where a single plaintext has relatively little value, and the purpose of the encryption is not so much to make plaintext recovery attacks absolutely infeasible, but rather to make the cost of performing such an attack completely unjustifiable.

For applications which do require long-term protection, it is recommended to aim for 128-bit or 256-bit security. A 128-bit security level is expected to resist for a period of at least 30 years. Schemes providing 256 bits of security should remain secure in the "foreseeable future", even in the advent of a breakthrough in quantum computing.

In general, it takes only a linear increase in implementation cost to increase the security level of a symmetric encryption scheme from 80 bits to 128 bits (or from 128 bits to 256 bits). Hence, unless efficiency is absolutely crucial, there is often little reason not to opt for the higher security levels.

2.2.4 Generic Attacks and Ideal Ciphers

In addition to the more pragmatic approach of matching the security of an encryption scheme against the computational capabilities of the adversary, there exists a tradition in symmetric cryptography to link the security requirements of a cipher to its external dimensions. The ambition is to design ciphers which are optimal in the sense that they cannot be attacked in a significantly more efficient way than any other conceivable cipher with the same external dimensions. Or with other words, that their security can only be increased by increasing at least one external dimension. If a cipher fails to satisfy this requirement, then this is considered to be a *certificational weakness*, regardless of the practical implications.

The discussion above raises two questions: (1) what exactly do we consider to be the external dimensions of a cipher? And (2) what is the maximal

Table 2.1: Some reference numbers

Reference	Magnitude
Seconds in a year	2^{25}
Age of solar system in years	2^{32}
Clock cycles per year on 2 GHz computer	2^{56}
Floating point operations per year on BOINC[a]	2^{74}
Capacity of 120 GB hard disk in bits	2^{40}
Information generated or replicated in 2006 in bits[b]	2^{70}

[a] http://www.boincstats.com/
[b] http://www.emc.com/about/destination/digital_universe/

Table 2.2: Minimum security for various adversaries in 2007 [37]

Attacker	Budget	Hardware	Min. security
Hacker	0	PC	52 bit
	$< \$400$	PC(s)/FPGA	57 bit
	0	Malware	60 bit
Small organization	\$10k	PC(s)/FPGA	62 bit
Medium organization	\$300k	FPGA/ASIC	67 bit
Large organization	\$10M	FPGA/ASIC	77 bit
Intelligence agency	\$300M	ASIC	88 bit

security level that can be expected from a cipher with given dimensions? In the next two sections we briefly address these questions for the two most common types of symmetric primitives.

The Ideal Block Cipher

There is little discussion possible about what constitute the external dimensions of a block cipher: clearly, the two relevant parameters are the block length n and the key length k. As explained earlier, the purpose of a block cipher is to realize an unpredictable invertible mapping from n-bit plaintext blocks to n-bit ciphertext blocks, controlled by a k-bit secret key. Considering this, and disregarding all practical implementation issues, it appears that the ideal block cipher would simply consist of a collection of 2^k permutations, picked at random from the set of all possible permutations over n-bit blocks.

Despite the fact that this hypothetical cipher would lack any exploitable structure, it is still susceptible to a number of generic attacks. The most obvious attack, which applies to any cipher with a k-bit secret key, is *exhaustive key search*. If Eve is given a small number of plaintext/ciphertext pairs, she could encrypt the plaintexts with all possible keys and compare the result with the previously observed ciphertext. On average, the correct key will reveal itself after $2^k/2$ trials. In certain circumstances, it will be possible to reduce this workload. For example, if a single plaintext is encrypted under 2^t different keys, Eve can attack all keys simultaneously, and is expected to find the first match after only 2^{k-t} trials.

A second possible improvement, proposed by M. E. Hellman [51], is a time-memory trade-off based on a precomputed table of $2^{2 \cdot k/3}$ entries. The precomputation itself still takes 2^k steps, but once the table is completed, any subsequent key can be recovered within $2^{2 \cdot k/3}$ steps. It is important to note here that the large amount of memory required for this approach has a significant impact on the cost of the attack. A rigorous analysis by M. J. Wiener [111] shows that even without taking into consideration the precomputation, the full cost of recovering a specific key is in fact not significantly lower than the

cost of a regular exhaustive search. On the other hand, a parallel implementation of this trade-off allows to recover a large number of keys simultaneously without increasing the total cost, in which case the cost per key is effectively reduced (see also [10]).

Another generic attack is related to the block length n. If Eve manages to capture the ciphertexts of all 2^n possible plaintext blocks, she can construct a dictionary which allows her to decrypt any future message encrypted with the same secret key. In fact, Eve does not necessarily need a complete dictionary: whenever a block cipher outputs the same ciphertext block twice, it leaks information about the plaintexts. Since repetitions in a random set of n-bit blocks start to occur frequently when the number of blocks exceeds $2^{n/2}$ (a consequence of the birthday paradox, as mentioned earlier), it is not advisable for Alice and Bob to encrypt more than $2^{n/2}$ blocks with the same secret key.

The Ideal Synchronous Stream Cipher

It is maybe slightly less obvious which parameters should be considered as external dimensions in a synchronous stream cipher. Clearly, the key length k and the IV length v should be included. However, if we would only consider these two parameters, then our ideal synchronous stream cipher should logically consist of a collection of 2^{k+v} infinite streams of independent and uniformly distributed bits. The problem is that there is no way to implement such an ideal cipher on a finite device, which makes it an unfair reference for practical stream ciphers. In order to avoid this problem, we choose to introduce a third parameter d, which restricts the maximum allowed key stream length to 2^d symbols.

One might wonder why the state size was not taken into consideration in the previous discussion. Until recently, it was indeed common practice to link the security of a stream cipher to its state size (which was often equal to the key length). The main argument against this approach is that the state size does not directly affect the external interface of a cipher (as opposed to the key length, the IV length, or the maximum key stream length), and hence it is better left as an internal implementation property. This reasoning is supported by the fact that the most efficient stream ciphers often use states which are considerably larger than what would be expected from their security level. Artificially restricting the state sizes of these ciphers would seriously hurt their performance. On the other hand, it is clear that the parameters k, v, and d do affect the minimal state size of a cipher. An ideal synchronous stream cipher, implemented as a large static table of independent and uniformly distributed bits, would have to store at least the key, the IV, and the current position in the stream, which suggests that stream ciphers should have a minimal state size of $k + v + d$ bits.

The generic key recovery attacks discussed in the previous section also apply to synchronous stream ciphers. Several other trade-offs are possible, and we refer the interested reader to [53] for an extensive overview. However, if

the state size satisfies the bound given above, none of these attacks are significantly more efficient than the parallel trade-off attack mentioned in the previous section.

2.3 Conclusions

In this chapter, we have reviewed several types of symmetric encryption algorithms. We have clarified the distinction between block ciphers and stream ciphers, and have tried to explain the intuition behind the different designs. We have identified different possible attack scenarios, and have discussed criteria to evaluate the security of encryption algorithms.

Chapter 3

Block Cipher Cryptanalysis

At the end of the previous chapter, we saw a number of generic attacks which apply to any cipher, and are solely based on the cipher's limited dimensions. In this chapter, we concentrate on block ciphers, and consider attack techniques which try to exploit the specific structure of practical block cipher constructions.

3.1 General Attack Strategy

Once our adversary Eve has convinced herself that the block and key lengths of a block cipher make all generic attacks infeasible, she will search for special properties in the cipher's internal structure. Attacks which reduce the complexity of exhaustive search by exploiting internal properties are called *shortcut attacks*. In the last two decades, cryptanalysts have started to develop systematic methods to search for shortcut attacks. Many of the successful techniques boil down to the same two-step strategy:

Step 1: Build a distinguisher. Given a sequence of plaintext-ciphertexts pairs, Eve will always be able to tell from the encryption of an unknown plaintext block whether or not it is equal to one of the previous plaintexts. In order to make sure that this is also the only information that Eve can extract, a secure block cipher must present itself as a completely random permutation to any adversary which does not know the key and has a limited amount of computational resources. Conversely, any property which allows Eve to distinguish the block cipher from a random permutation is an interesting weakness, which, as explained below, is typically only one step away from a key recovery attack.

Step 2: Recover round keys. Let us consider a reduced encryption function constructed by omitting the last round of the block cipher, and let us suppose that Eve is able to efficiently distinguish whether or not a given

sequence of input/output blocks could have been produced by this reduced function for some secret key. If Eve is now given plaintext-ciphertext pairs from the original cipher, she can guess (parts of) the last round key, (partly) decrypt the last round, and use her distinguisher on the first $r-1$ rounds to check whether the guess could have been correct. Once she has obtained a correct round key, she can proceed with an exhaustive search for the remaining key bits, or peel off one round and start again.

In the next three sections, we first discuss three general analysis techniques based on these ideas. As a further illustration, we will then present two concrete attacks against the block ciphers SAFER++ and ARIA. Although these dedicated attacks use some new ideas and hence deviate somewhat from the general techniques that will be presented in the first part of this chapter, we will see that they still follow the same basic two-step strategy.

3.2 Differential Cryptanalysis

Differential cryptanalysis has been, and still is, one of the most influential techniques in block cipher cryptanalysis. It was developed by E. Biham and A. Shamir in the late 1980s and was originally used to demonstrate weaknesses in the block cipher FEAL. The technique was first published in a generalized form in 1990, and illustrated with attacks on reduced-round versions of DES [12, 14]. After a few additional improvements it eventually led, in 1991, to the first attack on the full 16-round DES which was faster than exhaustive search [13].

3.2.1 A Differential Distinguisher

Constructing an efficient distinguisher essentially consists in finding a distinctive property in the input and output blocks which would indicate the use of the block cipher regardless of the value of the secret key.[1] This suggests that the attacker should somehow eliminate the effect of the unknown key. Differential cryptanalysis attempts to do exactly that, by studying differences of input and output blocks encrypted with the same key.

In many block ciphers (including FEAL, DES, and AES), the secret key bits are injected in the encryption function by XORing them to intermediate data blocks at different stages in the computation. Let X_1 and X_2 be the values of such an intermediate data block for two different plaintexts P_1 and P_2. Assuming that both plaintexts are encrypted with the same key, we can write

$$\begin{cases} Y_1 = X_1 \oplus K_i \\ Y_2 = X_2 \oplus K_i \end{cases} \quad \Rightarrow \quad \begin{aligned} \Delta Y &= Y_1 \oplus Y_2 \\ &= (X_1 \oplus K_i) \oplus (X_2 \oplus K_i) \\ &= X_1 \oplus X_2 = \Delta X\,. \end{aligned}$$

This simple observation illustrates the purpose of a differential approach: while the adversary cannot compute the values Y_1 and Y_2 without knowing the round key K_i, she can easily determine their difference ΔY, given ΔX. The idea of differential cryptanalysis is to try to extend this property over multiple rounds. If Eve manages to predict the output difference ΔC by tracing how the input difference ΔP evolves through the cipher, then this obviously distinguishes the cipher from a random permutation.

3.2.2 Differential Characteristics

In practice, a cipher does not only consist of key additions (which, as shown above, are completely transparent to differences); it also contains diffusion components and nonlinear S-boxes. Linear diffusion layers do not pose a serious problem. Although they do not preserve differences, they do transform them in a predictable way:

$$\begin{cases} Y_1 = A \cdot X_1 \\ Y_2 = A \cdot X_2 \end{cases} \quad \Rightarrow \quad \Delta Y = A \cdot \Delta X\,. \tag{3.1}$$

Unfortunately for Eve, this is not true for S-boxes (or any other nonlinear component the cipher may have). Unless the difference ΔX at the input of the S-box is 0, Eve can typically not determine the output difference ΔY without knowing the actual value of X_1. However, given ΔX and assuming that X_1 is uniformly chosen, she can compute the statistical distribution of possible output differences (we will see an example later). In order to proceed, Eve will simply pick one of these output differences, compute the probability that her choice was correct, and continue her analysis. Eventually, she will reach the output of the cipher, and will have described one of the possible ways in which the difference ΔP at the input could have propagated through the cipher. This is called a differential *characteristic*. The probability p that a given pair of plaintexts actually follows this characteristic is the product of the probabilities of all choices that Eve had to make (assuming that these probabilities are independent).

3.2.3 Minimizing the Data Requirements

In order to use the probability p to distinguish the block cipher from a random permutation, Eve will need the encryptions of a sufficient amount of plaintext pairs with a fixed difference ΔP. Notice that this assumes that she can *choose*

[1] Efficient distinguishers which only work for specific values of the key (called *weak* keys), are also useful, provided that the fraction of these keys is sufficiently large. We will see an example later in this chapter.

the plaintexts. Eve will then count the number of pairs which produce the output difference predicted by her characteristic. For N pairs of plaintexts encrypted with the block cipher, this number is expected to be at least[2] $p \cdot N$. In the random case, Eve expects the predicted output difference to appear only $p_R \cdot N$ times, with p_R in the order of $1/2^n$. In order to clearly distinguish both cases, the numbers must differ by at least a few standard deviations:

$$|p \cdot N - p_R \cdot N| \propto \sqrt{N \cdot p(1-p) + N \cdot p_R(1-p_R)}\,. \tag{3.2}$$

Hence, assuming that $p_R \ll p \ll 1$, we obtain the condition

$$N \propto \frac{1}{p}\,.$$

This clearly shows that the larger the probability of Eve's characteristic, the more efficient the distinguisher will be. Searching for the most probable characteristic typically involves a tradeoff between two objectives: Eve's first goal is to select the differences at the inputs and outputs of the diffusion layers in such a way that they affect as few S-boxes in the neighboring layers as possible. Whenever an S-box is kept *inactive* this way, its output difference does not need to be guessed (it can only be 0). Secondly, in all places where the differences do affect an S-box (called an *active* S-box), Eve will try to choose the pair of input and output differences that has the largest possible probability. To facilitate this task, she will construct a *difference distribution table*, which lists the probabilities of all possible pairs of differences at the input and the output of the given S-box.

Table 3.1 gives an example for a 3-bit S-box. For each input difference ΔX, the table contains a row showing the distribution of possible output differences ΔY. Notice that in this specific example, $\Delta X = 2$ results in $\Delta Y = 6$ with probability 1. Such a weakness is not likely to exist in a larger S-box.

3.2.4 Applications and Extensions

In their original attack, E. Biham and A. Shamir used a 13-round distinguisher to recover key bits from the last two rounds of a DES variant reduced to 15 rounds. The distinguisher was based on a 13-round differential characteristic with probability 2^{-47}. In 1991, the two researchers realized that they could allow an extra round at the input of the distinguisher by imposing additional restrictions on the plaintexts. This observation, together with an improved procedure to eliminate wrong key candidates, eventually resulted in the first theoretical break of the full 16-round DES cipher. The attack required an impractical amount of data (2^{47} chosen plaintexts), but was significantly more

[2] As an input difference might propagate to the same output difference in multiple ways, this number is sometimes significantly higher. The set of all characteristics with the same input and output differences is called a *differential*.

Table 3.1: Difference distribution table for a 3-bit S-box $Y = S(X)$

X	Y
0	7
1	5
2	1
3	3
4	2
5	6
6	4
7	0

$\Delta X \backslash \Delta Y$	0	1	2	3	4	5	6	7
0	1	0	0	0	0	0	0	0
1	0	0	1/2	0	1/2	0	0	0
2	0	0	0	0	0	0	1	0
3	0	0	1/2	0	1/2	0	0	0
4	0	0	0	1/2	0	1/2	0	0
5	0	1/2	0	0	0	0	0	1/2
6	0	0	0	1/2	0	1/2	0	0
7	0	1/2	0	0	0	0	0	1/2

efficient than exhaustive search (i.e., trying out about half of all 2^{56} possible keys).

After the publication of these first differential attacks, various improvements and extensions have been proposed. Techniques have been developed to exploit *truncated differences* [64] (differences which leave a number of bits undetermined), *impossible differentials* [15] (combinations of input and output differences that can never occur), and *higher-order differences* [69] (differences of differences). Another interesting development is the *boomerang attack* [104], which builds an adaptive attack using two separate differential characteristics, each covering half of the cipher.

3.3 Linear Cryptanalysis

The second powerful technique developed in the early 1990s is linear cryptanalysis. The attack in its current form was introduced by M. Matsui in 1993 [76], and was first applied to DES. However, as was the case with differential cryptanalysis, early variants of the attack were already used in 1992 to break FEAL [79]. Linear cryptanalysis will be the main topic in the next chapter, but we already outline the basic ideas here.

3.3.1 Linear Approximations

Whereas differential cryptanalysis focuses on differences in data blocks, linear cryptanalysis studies the relation between linear combinations of plaintext and ciphertext bits. The attack relies on the existence of a *linear approximation* of the cipher. This is a linear expression of the form

$$\Gamma_P^{\mathsf{T}} \cdot P \oplus \Gamma_C^{\mathsf{T}} \cdot C = \Gamma_K^{\mathsf{T}} \cdot K\,, \tag{3.3}$$

which holds with probability $p \neq 1/2$, where C is the encryption of P under the key K. The column vectors Γ_P, Γ_C, and Γ_K are called *linear masks* and

represent a particular linear combination of bits.

The motivation to study differential properties in the previous section was that it allowed to eliminate the secret key. In linear cryptanalysis this goal is only partly achieved: the secret key is reduced to a single unknown bit, $\Gamma_K^\mathsf{T} \cdot K$. As a result, three cases can be distinguished. Assuming that Eve is given the encryptions of N arbitrary plaintexts, let T be the number of texts such that the left-hand side of (3.3) is 0. If Eve finds that T is close to $p \cdot N$, she will conclude that the block cipher was used with a secret key K satisfying $\Gamma_K^\mathsf{T} \cdot K = 0$. On the other hand, if T converges to $(1-p) \cdot N$, she will assume that $\Gamma_K^\mathsf{T} \cdot K = 1$. Finally, a value of T close to $\frac{1}{2} \cdot N$ (which is what Eve would expect in the random case) indicates that the plaintext-ciphertext pairs were probably not generated by the block cipher. The number of texts required to accurately distinguish these three cases can be computed using (3.2). This time, we have $p \approx p_R = 1/2$, resulting in the condition

$$N \propto \frac{1}{(p-1/2)^2}\,.$$

As can be noticed in the previous paragraph, an important advantage of linear cryptanalysis over differential cryptanalysis, is that it does not impose restrictions on the plaintexts: it is a *known* instead of a *chosen* plaintext attack. Moreover, the procedure described above does not only allow Eve to distinguish the block cipher from a random permutation, it also immediately provides her with (the equivalent of) one secret key bit. Nevertheless, we will see in Chap. 4 that, in order to recover the complete key efficiently, the distinguisher will still have to be used as explained in Sect. 3.1.

3.3.2 Linear Characteristics

When mounting a linear attack, Eve's first task consists in finding a useful linear approximation. As deduced above, the more the probability of the approximation differs from $1/2$, the lower the number of plaintexts required by the attack. Finding the best linear approximation for an arbitrary cipher is in general not a trivial task. However, if the cipher is composed of simple components (as is mostly the case), one could try to approximate the complete cipher by combining linear approximations for individual components.

Finding linear relations between the input and the output bits of components which are linear already, is obviously very easy. For example, in order to write a linear expression which holds with probability 1 for a key addition, it suffices to choose masks Γ_X, Γ_Y, and Γ_K in the following way:

$$Y = X \oplus K_i \quad \Rightarrow \quad \begin{array}{c} \Gamma_Y^\mathsf{T} \cdot Y = \Gamma_Y^\mathsf{T} \cdot X \oplus \Gamma_Y^\mathsf{T} \cdot K_i \\ \Uparrow \\ \Gamma_X = \Gamma_K = \Gamma_Y\,. \end{array}$$

Table 3.2: Linear approximation table for a 3-bit S-box $Y = S(X)$

X	Y
0	7
1	5
2	1
3	3
4	2
5	6
6	4
7	0

$\Gamma_X \backslash \Gamma_Y$	0	1	2	3	4	5	6	7
0	$1/2$	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	$-1/2$
2	0	0	$-1/4$	$-1/4$	$-1/4$	$1/4$	0	0
3	0	0	$-1/4$	$1/4$	$1/4$	$1/4$	0	0
4	0	$-1/2$	0	0	0	0	0	0
5	0	0	0	0	0	0	$1/2$	0
6	0	0	$1/4$	$1/4$	$-1/4$	$1/4$	0	0
7	0	0	$-1/4$	$1/4$	$-1/4$	$-1/4$	0	0

Similarly, if Eve wants to construct a linear relation for a linear diffusion layer, she can choose the masks as follows:

$$Y = A \cdot X \quad \Rightarrow \quad \begin{array}{c} \Gamma_Y^\mathsf{T} \cdot Y = \Gamma_Y^\mathsf{T} \cdot A \cdot X \\ = (A^\mathsf{T} \cdot \Gamma_Y)^\mathsf{T} \cdot X \\ \Uparrow \\ \Gamma_X = A^\mathsf{T} \cdot \Gamma_Y\,. \end{array} \tag{3.4}$$

S-boxes, which are designed to be highly non-linear, can typically not be approximated very accurately with a linear expression. The linear approximation with the best correlation can be found by constructing a *linear approximation table*, which is the equivalent of the difference distribution table used in differential cryptanalysis. An example is given in Table 3.2. For all pairs (Γ_X, Γ_Y), the table lists the value of $(p-1/2)$, with p the probability that $\Gamma_X^\mathsf{T} \cdot X = \Gamma_Y^\mathsf{T} \cdot Y$. The more this value differs from zero, the better the approximation.

When comparing equations (3.1) and (3.4), we notice that there is a certain duality between differential and linear cryptanalysis: the first equation describes how differences propagate from the input to the output of a diffusion layer; the second equation shows a similar property for linear masks, but this time the masks propagate from the output to the input, and they are multiplied with A^T instead of A. A result of this duality is that chains of approximations, called *linear characteristics* can be constructed in exactly the same way as differential characteristics. This is illustrated in Fig. 3.1.

3.3.3 Piling-up Lemma

The only missing link in the construction of linear characteristics is a rule for computing the total probability of a chain of approximations. This is where the so-called *Piling-up Lemma* comes into play.

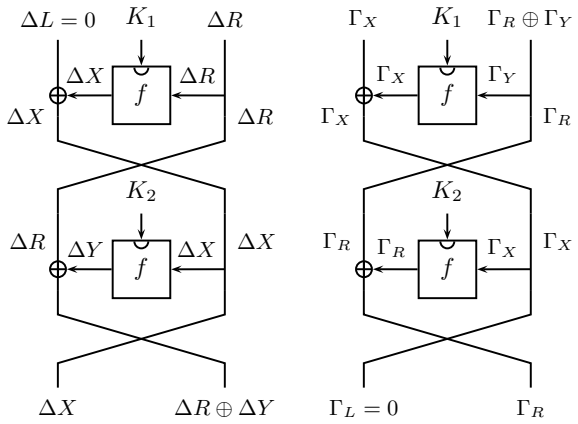


Figure 3.1: Propagation of differential and linear characteristics in a Feistel cipher. The difference ΔR propagates from the input to the output; the linear mask Γ_R takes the opposite direction

Lemma 3.1. *If n* independent *linear approximations of the form $\Gamma_i^\mathsf{T} \cdot X_i = \Gamma_{i-1}^\mathsf{T} \cdot X_{i-1}$ are each satisfied with a probability $p_i = 1/2 + \epsilon_i$, then the combined probability of the approximation $\Gamma_n^\mathsf{T} \cdot X_n = \Gamma_0^\mathsf{T} \cdot X_0$ is given by $p = 1/2 + \epsilon$ with*

$$\epsilon = 2^{n-1} \prod_{i=1}^{n} \epsilon_i \,. \tag{3.5}$$

The values ϵ_i used above are called the *biases* of the linear approximations. The lemma can be further simplified by defining $c_i = 2 \cdot \epsilon_i$, known as the *correlation* or the *imbalance*. With this notation, (3.5) reduces to $c = \prod c_i$. The square of the correlation, appropriately called the *linear probability*, makes the similarity between linear and differential cryptanalysis even more apparent: linear probabilities can be multiplied as before, and just as in differential cryptanalysis, the inverse of their product is proportional to the number of plaintexts required by the distinguisher.

3.3.4 Applications

In his original paper, M. Matsui presented two different attack algorithms for DES. The first, called Algorithm 1, used one large characteristic covering all 16 rounds, and allowed to recover the value of $\Gamma_K^\mathsf{T} \cdot K$. The second algorithm, Algorithm 2, was a key recovery attack based on a 15-round linear distinguisher. In 1994, Matsui proposed an improved variant of Algorithm 2, using a 14-round linear characteristic. The attack required 2^{43} known plaintexts and was the first attack on DES that was verified experimentally.

Today, Matsui's attack is still considered to be amongst the most efficient attacks on DES. A number of interesting variants of linear cryptanalysis have been proposed in the last decade, including attacks using chosen plaintexts [65], non-linear approximations [66, 101], or, as we will see in the next chapter, multiple linear approximations [21, 59]. When applied to DES, however, none of these approaches improves Matsui's attack with more than a factor 4.

3.4 Multiset attacks

After the discovery of linear and differential cryptanalysis, cryptographers started to design ciphers which minimized both the maximum probability of differential characteristics and the maximum correlation of linear characteristics. One of these ciphers was SQUARE, designed by J. Daemen and V. Rijmen in 1997. However, during the analysis of a preliminary version of this block cipher, L. Knudsen discovered that it was vulnerable to a new type of attack. This forced the designers to increase the number of rounds, and the resulting cipher was published in [29], together with the new attack, which was from then on referred to as the "SQUARE attack."

Differential and linear attacks are in general very sensitive to the exact specification of each component in the cipher. This is much less the case for the type of cryptanalysis described in this section: the SQUARE attack is not affected by specific design choices for individual components, but relies only on how these components, which are considered as black boxes, are interconnected. Another interesting feature of the attack is that it is not probabilistic: if Eve does not detect the special property which the distinguisher relies on, then she knows for sure that the plaintext-ciphertext pairs were not generated by the block cipher.

The general technique used in the SQUARE attack has been given different names in the last few years. S. Lucks proposed the name *saturation attack* [74], A. Biryukov and A. Shamir treated the technique as a special case of *structural cryptanalysis* [17], and L. Knudsen and D. Wagner referred to it as *integral cryptanalysis* [67].

3.4.1 Multisets

The key idea behind the SQUARE attack is somewhat similar to the differential approach in Sect. 3.2. However, instead of analyzing pairs of related plaintexts, the attacker will now study the behavior of complete sets of carefully chosen plaintexts. In order to analyze these sets, the text blocks are first split into m-bit words whose size matches the internal structure of the cipher. The different values taken by each individual word are then treated as multisets. A multiset is a list of values, each of which can appear multiple times, but the

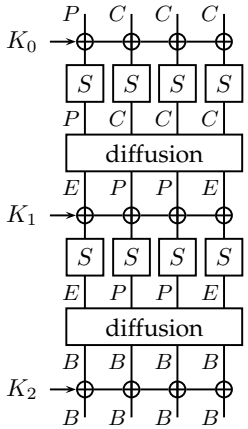


Figure 3.2: An example of how multisets might propagate through an SP network. The labels C, P, E, and B respectively stand for constant, permutation, even, and balanced

order of which is irrelevant. The goal of the attack is to keep track of multisets with special properties. The following properties are of particular interest:

Constant multiset. A multiset consisting of a single value repeated an arbitrary number of times.

Permutation or saturated multiset. A multiset which contains all 2^m possible values for the word exactly once.

Even multiset. A multiset in which each value, if present, occurs an even number of times.

Balanced multiset. A multiset such that the XOR of all values (taking into account their multiplicity) is zero.

Notice that some of these properties are implied by others. For example, a constant multiset with an even number of elements is also an even multisets, and any saturated or even multiset is automatically balanced.

3.4.2 How Multisets Propagate

In order to distinguish a block cipher from a random permutation, the adversary will first construct a set of plaintexts such that the values at different positions form special multisets. For example, when analyzing an SP network consisting of 8×8-bit S-boxes, Eve might choose 256 plaintexts which take on

all possible values in the first byte (a saturated multiset), but are constant in the others. Her task is then to trace how these multisets are transformed as the plaintexts are encrypted. Interestingly, most of the transformations commonly found in a block cipher preserve or translate at least some of the multiset properties described above. A constant or an even multiset, for example, retains its special properties after having been transformed by an arbitrary function over m-bit values (e.g., an S-box). Similarly, a saturated multiset is preserved by any bijective function, and a balanced multiset by any linear transformation. Finally, if a saturated and a constant multiset are combined in a linear way, the result will be either saturated or even.

Using the propagation rules described above, Eve can typically keep track of the multiset properties over 2 to 4 rounds (see for example Fig. 3.2). If the multisets at the output of the last round do not exhibit the predicted properties, then this indicates that the output texts were generated in a different way.

3.4.3 Applications

Multiset attacks are of particular significance today because of their applicability to RIJNDAEL. The RIJNDAEL cipher, designed by J. Daemen and V. Rijmen in 1998 [28], is a successor of SQUARE. It was submitted to the U.S. National Institute of Standards and Technology[3] (NIST) in response to an open call for 128-bit block ciphers. It was, together with 14 other candidates, extensively evaluated during two years, before NIST announced in 2000 that RIJNDAEL would replace DES and become the new Advanced Encryption Standard (AES).

Just as its predecessor SQUARE, RIJNDAEL was specifically designed to resist differential and linear cryptanalysis. As of today, multiset attacks have shown to be the most effective in breaking reduced versions of RIJNDAEL. The SQUARE attack, which was also applicable to the RIJNDAEL structure, allowed to break 6 rounds out of 10. It recovered the 128-bit key using a set of 2^{32} special plaintexts, and it required a computational effort of 2^{72} steps. The work factor was later reduced to 2^{44} by performing the calculations in a more efficient way [40]. Ferguson et al. [40], as well as H. Gilbert and M. Minier [43], have developed more sophisticated multiset attacks that could be applied to 7 rounds. However, when RIJNDAEL is used with a 128-bit secret key, both attacks are only marginally faster than generic attacks. If the reduced cipher is used with a larger key, it takes one or two more rounds before the complexities of the currently best attacks exceed the complexity of exhaustive search.

[3] Previously called the National Bureau of Standards (NBS).

3.5 Example 1: Attacking 3 Rounds of SAFER++

In the remaining two sections, we try to convey a sense of what it takes to devise an attack against a practical block cipher by considering two concrete examples of dedicated attacks. Both attacks nicely illustrate the concepts of Sect. 3.1, while at the same time introducing some new ideas.

The first attack targets a 3-round reduced variant of the block cipher SAFER++. Although we demonstrate considerably more powerful attacks covering up to 5.5 rounds in [19], we choose to concentrate on this 3-round attack because of its instructive simplicity.

3.5.1 Description of SAFER++

SAFER++ [75] is a block cipher which was submitted to the European NESSIE competition in 2000, where it made it to the second phase. It is the latest member of a family of block ciphers designed by J. Massey which includes SAFER-K, SAFER-SK, and SAFER+. The latter is an AES candidate which is currently widely used owing to its important role in the authentication and key generation algorithms specified in the Bluetooth standard.

The standard version of SAFER++ operates on 128-bit blocks and is based on a 7-round SP network. The round function, which consists exclusively of byte operations, is depicted in Fig. 3.3. It can be subdivided into four layers: a first key addition layer, an S-box layer, another key addition layer, and finally a linear diffusion layer. As can be seen in the figure, the key addition layers alternate XORs ('$\oplus$') and integer additions ('$\boxplus$'), and the S-box layer uses two different S-boxes X and L defined as follows:

$$X(a) = (45^a \bmod 257) \bmod 256\,,$$
$$L(a) = X^{-1}(a)\,.$$

The linear diffusion layer consists of a composition of two identical linear transforms, which successively reorder the bytes, split them into segments of four consecutive bytes, and feed each of those into a linear component called 4-point Pseudo Hadamard Transform (4PHT). The complete diffusion layer is designed to make sure that a change in a single input byte propagates to at least ten output bytes. It is interesting to note that this does not hold anymore during decryption: when the diffusion layer is used in reverse direction, and a single byte difference is applied, only five bytes are guaranteed to be affected.

The round keys are derived from a 128-bit key according to a simple key scheduling algorithm. Since the exact details of this algorithm are irrelevant to our attack, we will not discuss them here.

3.5.2 A Two-round Distinguisher

As mentioned earlier, changes propagate relatively slowly through the linear layer in decryption direction. Most byte positions at the input of a decryption

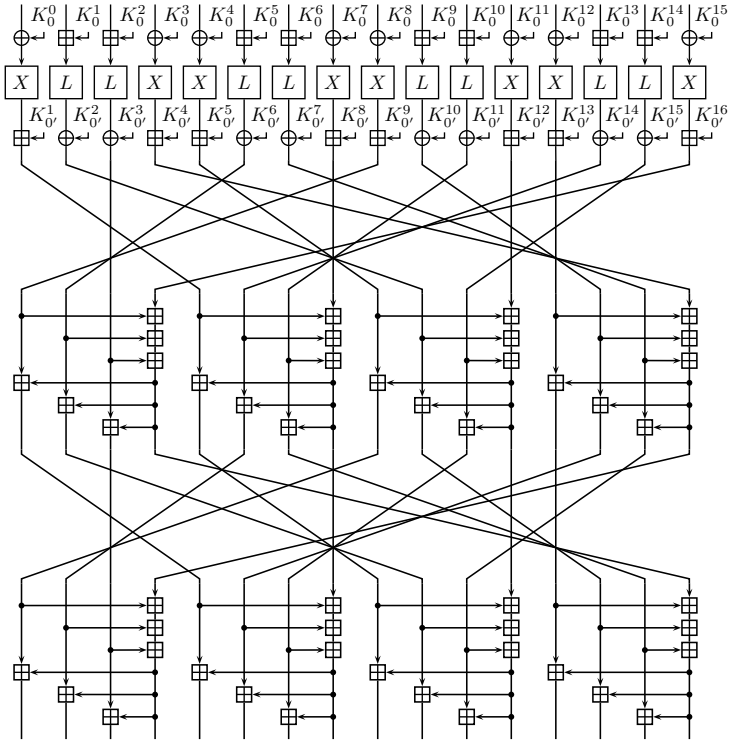


Figure 3.3: One round of SAFER++

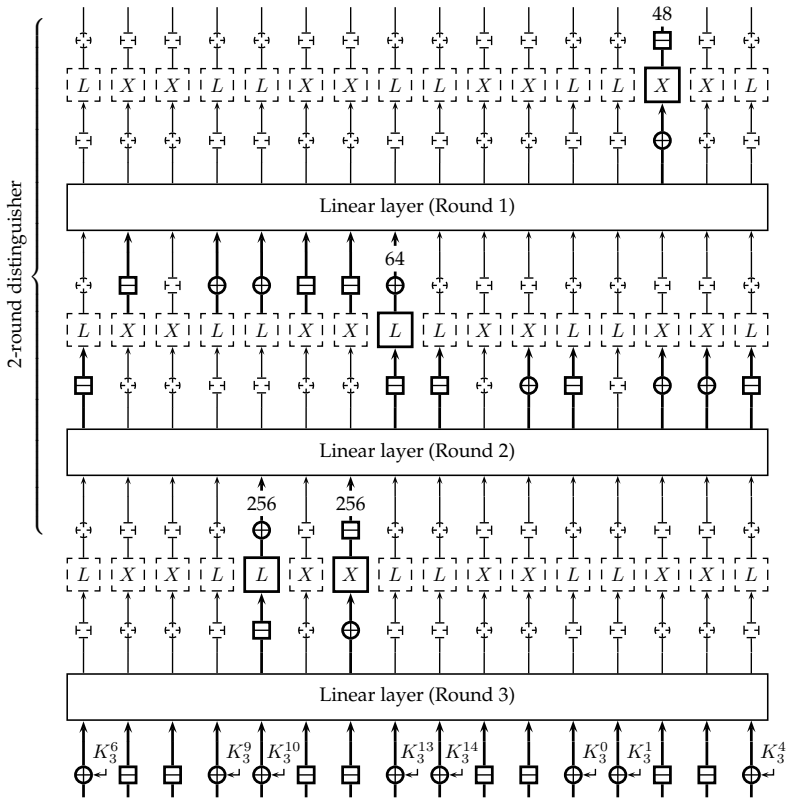


Figure 3.4: A 3-round attack

round affect only 6 bytes at the output, and conversely, most byte positions at the output are affected by only 6 bytes at the input. After one more round, this property is destroyed, i.e., a change in any input byte can induce changes in any output byte after two rounds (complete diffusion on byte level). The number of different paths through which the changes propagate is still small, however, and this property allows us to build an efficient distinguisher.

The structure of a 2-round distinguisher is shown in Fig. 3.4. First, a multiset of 2^{16} texts is constructed, constant in all bytes except in byte 4 and 6, in which it takes all possible values. After one decryption round, these texts are still constant in 8 positions, due to the weak diffusion properties of the linear layer. We now focus on byte 13 at the top of the distinguisher. This byte is affected by 6 bytes of the preceding round, 5 of which are constant. This implies that the changes in the two bytes at the input essentially propagate through a single path.

As a closer look reveals, this path has some additional properties: in Round 2, the two input bytes are first summed and then multiplied by 4 (modulo 256) before they reach S-box 7. The byte at this position is then again multiplied by 4 in Round 1. The effect of the first multiplication is that the two least significant bits at the input of S-box 7 remain unaffected; the second multiplication, on its turn, causes the two most significant bits at the output of this same S-box to be discarded. Due to the special algebraic structure of the S-box, this results in a reduction of the number of different values that can be observed at the top to exactly 48. Clearly, this is extremely unlikely to happen if the cipher were a random permutation, in which case all 256 possible values would almost certainly be observed.

3.5.3 Adding a Round at the Bottom

The distinguisher above is very strong, but is unfortunately hard to use as such when a round is added at the bottom. Initiating the desired multiset at the input of the distinguisher (see Fig. 3.4) would require half of the key bytes (those that are XORed) to be guessed in the final key addition layer.

Instead, we use a better approach and consider small multisets of 2^4 texts that are constant except in the two most significant bits of the fifth and the seventh byte. In order to control such multisets from the bottom, we only need to guess the second most significant bit of the 8 key bytes that are XORed in the final key addition layer and generate sets of 2^4 ciphertexts of the form

$$(x \cdot A \boxplus y \cdot B \boxplus C) \oplus K \tag{3.6}$$

with $(x, y) \in \{0, 64, 128, 192\}^2$, C any fixed 128-bit word, and

$$\begin{aligned} A &= (4, 2, 2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1)\,, \\ B &= (1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 4, 2, 2, 1, 1, 1)\,, \\ K &= (K_3^6, 0, 0, K_3^9, K_3^{10}, 0, 0, K_3^{13}, K_3^{14}, 0, 0, K_3^0, K_3^1, 0, 0, K_3^4)\,. \end{aligned}$$

Note that only the second most significant bits of the K_3^i are relevant (because of carries) and that the effect of the other bits can be absorbed in the constant C.

Now the question arises whether we can still distinguish the set of 16 values obtained in byte 13 of the plaintexts from a random set. An interesting characteristic that can be measured is the number of collisions in the sets, should they appear at all. If this number depends in a significant way on whether the key bits were correctly guessed or not, then this would allow us to recover the key. This question is easily answered by deriving a rough estimation of the number of collisions.

First, we deduce the expected number of collisions μ_0 in case the guess

was correct. Assuming that all 48 possible values are equally likely, we obtain

$$\mu_0 = \binom{16}{2} \cdot \frac{1}{48} \approx 2.50 \qquad \text{and} \qquad \sigma_0 = \sqrt{\binom{16}{2} \cdot \frac{1}{48}\left(1 - \frac{1}{48}\right)} \approx 1.56\,,$$

with σ_0 the expected deviation. In order to estimate the number of collisions given a wrong guess, one could be tempted to assume that the sets at the output of the distinguisher are random. This is not the case however.[4] One can easily see, for example, that whether K_3^6 is correctly guessed or not matters only for half of the (x, y) pairs, i.e., when the second most significant bit of $4 \cdot x \boxplus y$ is set. In all these (and only these) cases an incorrect carry bit will appear in the most significant bit. If we absorb this wrong bit in a new constant C', we obtain two subsets of 8 texts, both of which still satisfy (3.6), but with two different fixed values C and C'. Hence we find

$$\mu_1 = 2 \cdot \binom{8}{2} \cdot \frac{1}{48} + 8^2 \cdot \frac{1}{256} \approx 1.42\,,$$

$$\sigma_1 = \sqrt{2 \cdot \binom{8}{2} \cdot \frac{1}{48}\left(1 - \frac{1}{48}\right) + 8^2 \cdot \frac{1}{256}\left(1 - \frac{1}{256}\right)} \approx 1.19\,.$$

The value of μ_1 is indeed considerably higher than what we would expect for a random set (about 0.47 collisions). Exactly the same situation occurs for 17 other combinations of wrong key bits, and similar, but less pronounced effects can be expected for other guesses.

The result above can now be used to predict the complexity and success probability of our 3-round attack. The attack consists in running through all 2^8 possible partial key guesses and accumulating the total number of collisions observed after decrypting α sets of 16 texts. The maximum number of collisions is then assumed to correspond to the correct key. Taking this into consideration, we obtain the estimations below:

$$\text{Time and data complexity} \approx 2^8 \cdot 16 \cdot \alpha\,,$$

$$\text{Success probability} \approx 1 - 17 \cdot \Phi\left(\sqrt{\alpha} \cdot \frac{\mu_0 - \mu_1}{\sqrt{\sigma_0^2 + \sigma_1^2}}\right)\,,$$

where $\Phi(\cdot)$ represents the cumulative normal distribution function. Evaluating these expressions for $\alpha = 16$, we find a complexity of 2^{16} and a corresponding success probability of 77%. Similarly, for $\alpha = 32$ we get a complexity of 2^{17} and an expected probability as high as 98%. In order to verify these results, we performed a series of simulations and found slightly lower success probabilities: 70% and 89% for $\alpha = 16$ and $\alpha = 32$ respectively. This difference is due to the fact that our estimation only considers wrong guesses that have a high probability of producing many collisions.

[4] Note that this property is useful if one does not care about recovering the key, but just needs a 3-round distinguisher.

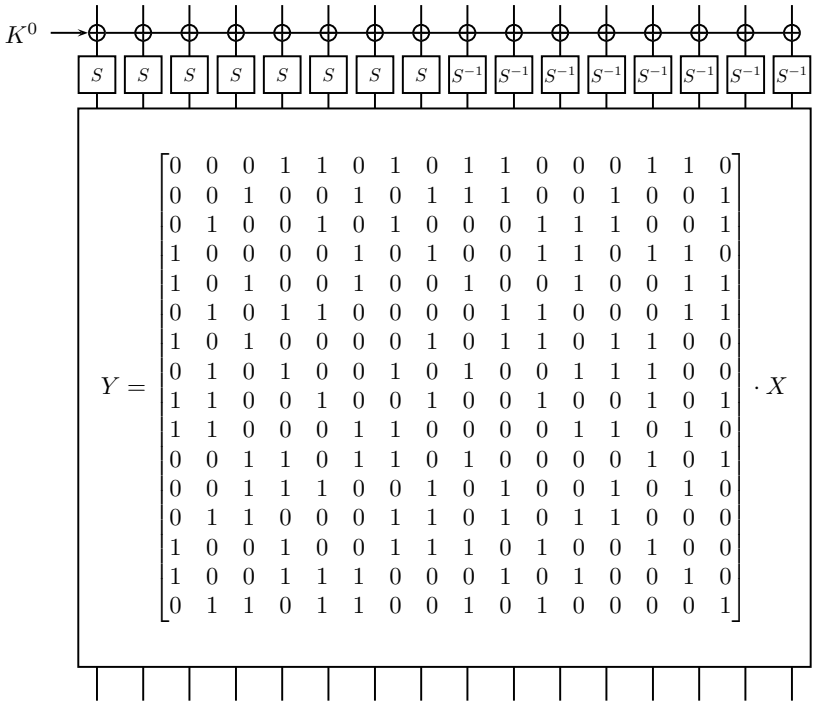


Figure 3.5: A type-1 round of ARIA version 0.8

3.6 Example 2: Weak Key Attacks Against ARIA

The target in this second example is the block cipher ARIA. This time, we will pursue an attack strategy which bears some similarities with linear cryptanalysis, but instead of considering biases of linear combinations of bits, we will concentrate on linear combinations of *bytes*.

3.6.1 Description of ARIA

ARIA is a 128-bit block cipher designed by a team of South Korean researchers. The first version (later called ARIA version 0.8) was published in 2003 [3, 88]. After the attack described in the next section was presented, the algorithm was slightly modified by first increasing the number of different S-boxes [68], and then augmenting the number of rounds [89]. In 2004, the ARIA block cipher was adopted as a standard by the South Korean Agency for Technology and Standards (ATS).

ARIA version 0.8 comes in three flavors which only differ in their key

lengths (128, 192, or 256 bits), their number of rounds (10, 12, or 14), and their key schedule. The latter does not play a significant role in our attack, and hence will not be considered here. The round structure is depicted in Fig. 3.5. It is a classical SP network consisting of a key addition layer, a layer of sixteen 8×8-bit S-boxes similar to the ones used in AES, and a diffusion layer which is described as a 16×16-byte binary matrix. As can be seen in Fig. 3.5, the S-box layers consist of two halves which are each others inverse. The odd (type-1) rounds use the S-box layer shown in figure. In the even (type-2) rounds, the roles of S and S^{-1} are reversed.

3.6.2 A Dedicated Linear Distinguisher

As mentioned earlier, the attack presented in this section will be based on the analysis of linear combinations of bytes. More in particular, we will focus on the sum (in $GF(2^8)$) of the first four bytes of the state. The core of the attack is a distinguisher based on the following observation: if we denote the bytes at the input and the output of an S-box layer by x_i and y_i, respectively, then

$$\Pr\left[\sum_{i=1}^{4} y_i = 0 \mid \sum_{i=1}^{4} x_i = 0\right] \approx 3 \cdot 2^{-8} = 2^{-8} + 2^{-7}\,. \tag{3.7}$$

This property does not depend on the choice of the S-boxes and holds as long as the 4 S-boxes involved in the expression above are equal. This can easily be explained by noting that, out of the 2^{24} combinations (x_0, x_1, x_2, x_3) which satisfy $x_0 + x_1 + x_2 + x_3 = 0$, about $3 \cdot 2^{16}$ are of the form (a, a, b, b), (a, b, a, b), or (a, b, b, a). These special patterns are preserved by the S-box layer, such that the y_i sum to 0 as well.

In order to build an r-round distinguisher based on the observation above, we also need to cross diffusion layers and key additions. Analyzing the diffusion matrix used in ARIA, one can easily see that this transformation preserves the sum of the first four bytes, but in order to cross the key additions, we need the first four bytes of the round keys to sum to zero. This implies that the attack will only succeed for a limited number of weak keys.

We first analyze the strength of a distinguisher spanning r S-box layers and $r-1$ key additions. The input bytes of the first S-box layer are denoted by x_i^1, and the outputs after the last S-box layer by y_i^r. If the first 4 bytes of each of the $r-1$ round keys sum to zero — this happens with probability $2^{-(r-1)\cdot 8}$ for random round keys — one can recursively derive that

$$\Pr\left[\sum_{i=1}^{4} y_i^r = 0 \mid \sum_{i=1}^{4} x_i^1 = 0\right] \approx 2^{-8} + 2^{-r\cdot 7}\,. \tag{3.8}$$

3.6.3 Adding Rounds at the Top and at the Bottom

In a next step we append additional key layers at the top and the bottom of the distinguisher. The bytes of the round keys of these two layers are denoted by k_i^0 and k_i^r respectively. In order to distinguish the cipher from a random permutation, an attacker could calculate $s_0 = \sum_{i=1}^{4} (x_i^1 + k_i^0)$ and $s_r = \sum_{i=1}^{4} (y_i^r + k_i^r)$ for n known plaintext-ciphertext pairs, and construct a distribution table of (s_0, s_r). In the random case, we would expect each pair (s_0, s_r) to occur with probability 2^{-16}, but for ARIA we can deduce from (3.8) that

$$\Pr\left[\sum_{i=1}^{4} y_i^r = 0 \text{ and } \sum_{i=1}^{4} x_i^1 = 0\right] \approx 2^{-16} + 2^{-8} \cdot 2^{-r\cdot 7}\,, \tag{3.9}$$

which directly implies that the pair $(\sum_{i=1}^{4} k_i^0, \sum_{i=1}^{4} k_i^r)$ has a slightly higher probability. This causes a peak in the distribution table which will stick out of the surrounding noise as soon as

$$2^{-8} \cdot 2^{-r\cdot 7} > Q^{-1}(2^{-16}) \cdot \sqrt{\frac{2^{-16}}{n}}\,, \tag{3.10}$$

where Q^{-1} denotes the inverse complementary cumulative normal distribution function, or

$$n > 18 \cdot 2^{2\cdot r\cdot 7}\,. \tag{3.11}$$

Since n is limited by the block size of 128 bits, we conclude that the distinguisher is in principle effective up to $r = 8$ rounds. However, in order to be useful, the time complexity of the attack cannot exceed the size of the weak key class (if it does, a simple exhaustive search would be more efficient). Since the time complexity includes at least the time required to encrypt the necessary data, we find the following additional condition for the 128-bit key version:

$$18 \cdot 2^{2\cdot r\cdot 7} < 2^{128} \cdot 2^{-(r-1)\cdot 8}\,, \tag{3.12}$$

or $r \leq 5$. Similarly, for the 192-bit and 256-bit version we obtain $r \leq 8$ and $r \leq 11$ (note however that r cannot exceed 8 because of the block size).

3.6.4 Appending Two More Rounds

We can now append two more rounds to the current distinguisher by guessing the first 4 key bytes of two additional key layers at the top and the bottom of the cipher (64 bits in total). For each guess, we can apply a partial encryption at the top and a partial decryption at the bottom, and compute the distribution table described above. If no peak is observed, we know that the guess was probably wrong. In order to filter out all wrong guesses, we need to strengthen the distinguisher. Considering the fact that the distinguisher is applied 2^{64} times and that $Q^{-1}(2^{-16} \cdot 2^{-64}) \approx 10$, we find

$$n > 100 \cdot 2^{2\cdot r\cdot 7} \approx 2^{2\cdot r\cdot 7+7}\,. \tag{3.13}$$

Table 3.3: Complexities of a dedicated linear attack

Key length:	128	192	256
Rounds:	7	10	10
Weak keys:	2^{96}	2^{136}	2^{200}
Data:	2^{77}	2^{119}	2^{119}
Memory[a]:	2^{64}	2^{64}	2^{64}
Counting complexity:	2^{88}	2^{88}	2^{88}
Recovered round key bits:	112	136	136

[a]Measured in number of counters. For the 128-bit version, 16-bit counters should suffice. The 192-bit and 256-bit versions require 64-bit counters.

Hence, for attacking a 128-bit key version of ARIA reduced to 7 rounds, using a 5-round distinguisher, 2^{77} known P/C pairs should suffice.

In order to avoid having to process all n plaintexts for each of the 2^{64} key guesses, we will build tables of counters and gradually transform them into distribution tables by guessing the key bytes one by one and partially evaluating the sums s_0 and s_r. The first table contains 2^{64} entries and is constructed by running through the n data pairs and increasing the counter corresponding to the value of the first 4 bytes of the plaintext and the ciphertext (64 bits in total). In the next step, we guess the 2 first key bytes at the top, apply a partial encryption, compute the sum of the first 2 terms of s_0, and create a table containing 2^{56} counters corresponding to the value of this sum, the 2 remaining bytes of the plaintext, and the first 4 bytes of the ciphertext (56 bits in total). In the next step, one more key byte is guessed and the previous table is used to compute a smaller table of 2^{48} counters, and so on. It can be shown that the total computational complexity of this approach is about $n + 2^{64} \cdot 2^{24}$ (in stead of $n \cdot 2^{64}$). The attack requires 2^{64} counters to be stored in memory.

The 7-round attack on the 128-bit key version recovers the values of 2×4 round key bytes at the top and the bottom and the sum of the first 4 key bytes used in the 6 inner key layers (112 bits in total). Table 3.3 summarizes the complexities for different key sizes.

3.7 Conclusions

In this chapter, we have discussed several general techniques to mount attacks against block ciphers. We have seen that differential cryptanalysis, linear cryptanalysis, and multiset attacks all take the same fundamental approach of devising a distinguisher for the inner rounds, which can then be used to recover round keys in the outer rounds. As an additional illustration of this idea, we have presented two new attacks on practical block ciphers. The first one, which attacks a round-reduced variant of SAFER++, is based on multisets, but uses them in a rather unconventional way. The second attack targets ARIA, and can be seen as a new variant of linear cryptanalysis which analyzes biases on byte level instead of on bit level.

Chapter 4

Linear Cryptanalysis Revisited

In Sect. 3.3 we already introduced the main concepts behind linear cryptanalysis. In this chapter, we try to generalize these basic ideas by approaching them from a slightly different angle.

4.1 Background and Objectives

Developed in 1993, linear cryptanalysis quickly became one of the most powerful attacks against modern cryptosystems. In 1994, Kaliski and Robshaw [59] proposed the idea of generalizing this attack using multiple linear approximations (the previous approach considered only the best linear approximation). However, their technique was mainly limited to cases where all approximations derive the same parity bit of the key. Unfortunately, this approach imposes a very strong restriction on the approximations that can be considered, and the additional information gained by the few surviving approximations is often not significant. In a subsequent paper [60], the same authors present some preliminary ideas for a more general approach, but leave its analysis as an open problem. Another, more recent treatment of the same problem is given by Choi et al. [24].

In this chapter, we will start by developing a theoretical framework for dealing with multiple linear approximations. We will first generalize Matsui's Algorithm 1 based on this framework, and then reuse these results to generalize Matsui's Algorithm 2. This approach will allow to derive compact expressions for the performance of the attacks in terms of the biases of the approximations and the amount of data available to the attacker. The relevance of these theoretical expressions is twofold. Not only will they provide an answer on whether the use of multiple approximations can significantly improve classical linear attacks, they will also shed a new light on the relations between

Algorithm 1 and Algorithm 2.

Our main objective is to derive a generally applicable cryptanalytic tool, which performs strictly better than standard linear cryptanalysis. In order to illustrate the potential of this new approach, we will apply our ideas to reduced-round versions of DES, using this cipher as a well established benchmark for linear cryptanalysis. We will see that the experimental results, discussed in the second part of this chapter, are in almost perfect correspondence with our theoretical predictions.

4.2 General Framework

As we saw in the previous chapter, many of the most powerful attacks against block ciphers, including differential and linear cryptanalysis, are largely based on statistics. In this section, we try to identify the main principles of what could be subsumed under the term *statistical cryptanalysis*, and set up a generalized framework for analyzing block ciphers based on maximum likelihood. This framework can be seen as an adaptation or extension of earlier frameworks for statistical attacks proposed by Murphy et al. [83], Junod and Vaudenay [55, 56, 103] and Selçuk [99].

4.2.1 Attack Model

Let us consider a block cipher E_K which maps a plaintext $P \in \mathcal{P}$ to a ciphertext $C = E_K(P) \in \mathcal{C}$. The mapping is by definition invertible and depends on a secret key $K \in \mathcal{K}$. We now assume that an adversary is given N different plaintext-ciphertext pairs (P_i, C_i) encrypted with a particular secret key K^* (a known plaintext scenario), and his task is to recover the key from this data. A general statistical approach—also followed by Matsui's original linear cryptanalysis—consists in performing the following three steps:

Distillation phase. In a typical statistical attack, only a fraction of the information contained in the N plaintext-ciphertext pairs is exploited. A first step therefore consists in extracting the relevant parts of the data, and discarding all information which is not used by the attack. In our framework, the distillation operation is denoted by a function $\psi : \mathcal{P} \times \mathcal{C} \to \mathcal{X}$ which is applied to each plaintext–ciphertext pair. The result is a vector $\mathbf{x} = (x_1, \ldots, x_N)$ with $x_i = \psi(P_i, C_i)$, which contains all relevant information. If $|\mathcal{X}| \ll N$, which is usually the case, we can further reduce the data by counting the occurrence of each element of $\mathcal{X}$ in $\mathbf{x}$ and only storing a vector of counters $\mathbf{t} = (t_1, \ldots, t_{|\mathcal{X}|})$. In this thesis we will not restrict ourselves to a single function ψ, but consider m separate functions ψ_j, each of which maps the text pairs into different sets $\mathcal{X}_j$ and generates a separate vector of counters $\mathbf{t}_j$.

Analysis phase. This phase is the core of the attack and consists in generating a list of key candidates from the information extracted in the previous step. Usually, candidates can only be determined up to a certain equivalence relation. In the concrete examples below, the keys will be determined up to a fixed number of bits, resulting in a set $\mathcal{Z}$ of equivalence classes, each with equal cardinality. In general, the attack defines a function $\sigma : \mathcal{K} \to \mathcal{Z}$ which maps each key k onto an equivalent key class $z = \sigma(K)$. The purpose of the analysis phase is to determine which of these classes are the most likely to contain the true key K^* given the particular values of the counters $\mathbf{t}_j$.

Search phase. In the last stage of the attack, the attacker exhaustively tries all keys in the classes suggested by the previous step, until the correct key is found. Note that the analysis and the searching phase may be intermixed: the attacker might first generate a short list of candidates, try them out, and then dynamically extend the list as long as none of the candidates turns out to be correct.

4.2.2 Attack Complexities

When evaluating the performance of the general attack described above, we need to consider both the data complexity and the computational complexity. The data complexity is directly determined by N, the number of plaintext-ciphertext pairs required by the attack. The computational complexity depends on the total number of operations performed in the three phases of the attack. In order to compare different types of attacks, we define a measure called the *gain* of the attack:

Definition 4.1 (Gain)**.** If an attack is used to recover a k-bit key and is expected to return the correct key after having checked on the average M candidates, then the *gain* of the attack, expressed in bits, is defined as:

$$\gamma = -\log_2 \frac{2 \cdot M - 1}{2^k} \,. \tag{4.1}$$

Let us illustrate this with an example where an attacker wants to recover a k-bit key. If he does an exhaustive search, the number of trials before hitting the correct key can be anywhere from 1 to 2^k. The average number M is $(2^k + 1)/2$, and the gain according to the definition is 0 bits. On the other hand, if the attack immediately derives the correct candidate, M equals 1 and the gain is $\gamma = k$ bits. One should observe that γ does not fully characterize the practical applicability of an attack. Let us consider two attacks which both require a single plaintext-ciphertext pair. The first deterministically recovers one bit of the key, while the second recovers the complete key, but with a probability of $1/2$. In this second attack, if the key is wrong and only one plaintext-ciphertext pair is available, the attacker has no other choice than to

perform an exhaustive search. According to the definition, both attacks have a gain of 1 bit in this case. Of course, by repeating the second attack for different pairs, the gain can be made arbitrary close to k bits, while this is not the case for the first attack.

4.2.3 Maximum Likelihood Approach

The design of a statistical attack consists of two important parts. First, we need to decide on how to process the N plaintext-ciphertext pairs in the distillation phase. We want the counters $\mathbf{t}_j$ to be constructed in such a way that they concentrate as much information as possible about a specific part of the secret key in a minimal amount of data. Once this decision has been made, we can proceed to the next stage and try to design an algorithm which efficiently transforms this information into a list of key candidates. In this section, we discuss a general technique to optimize this second step. Notice that throughout this chapter, we will denote random variables by capital letters.

In order to minimize the amount of trials in the search phase, we want the candidate classes which have the largest probability of being correct to be tried first. If we consider the correct key class as a random variable Z and denote the complete set of counters extracted from the observed data by $\mathbf{t}$, then the ideal output of the analysis phase would consist of a list of classes z, sorted according to the conditional probability $\Pr[Z = z \mid \mathbf{t}]$. Taking a Bayesian approach, we express this probability as follows:

$$\Pr[Z = z \mid \mathbf{t}] = \frac{\Pr[\mathbf{T} = \mathbf{t} \mid z] \cdot \Pr[Z = z]}{\Pr[\mathbf{T} = \mathbf{t}]}. \tag{4.2}$$

The factor $\Pr[Z = z]$ denotes the a priori probability that the class z contains the correct key K^*, and is equal to the constant $1/|\mathcal{Z}|$, with $|\mathcal{Z}|$ the total number of classes, provided that all classes have the same size, and the key was chosen at random. The denominator is the probability that the specific set of counters $\mathbf{t}$ is observed, taken over all possible keys and plaintexts. The only expression in (4.2) that depends on z, and thus affects the sorting, is the factor $\Pr[\mathbf{T} = \mathbf{t} \mid z]$, compactly written as $P_z(\mathbf{t})$. This quantity denotes the probability, taken over all possible plaintexts, that a key from a given class z produces a set of counters $\mathbf{t}$. When viewed as a function of z for a fixed set $\mathbf{t}$, the expression $\Pr[\mathbf{T} = \mathbf{t} \mid z]$ is also called the *likelihood* of z given $\mathbf{t}$, and denoted by $L_{\mathbf{t}}(z)$, i.e.,

$$L_{\mathbf{t}}(z) = P_z(\mathbf{t}) = \Pr[\mathbf{T} = \mathbf{t} \mid z].$$

This likelihood and the actual probability $\Pr[Z = z \mid \mathbf{t}]$ have distinct values, but they are proportional for a fixed $\mathbf{t}$, as follows from (4.2). Typically, the likelihood expression is simplified by applying a logarithmic transformation. The result is denoted by

$$\mathcal{L}_{\mathbf{t}}(z) = \log L_{\mathbf{t}}(z)$$

and called the *log-likelihood*. Note that this transformation does not affect the sorting, since the logarithm is a monotonously increasing function.

Assuming that we can construct an efficient algorithm that accurately estimates the likelihood of the key classes and returns a list sorted accordingly, we are now ready to derive a general expression for the gain of the attack.

Let us assume that the plaintexts are encrypted with a k-bit secret key K^*, contained in the equivalence class z^*, and let $\mathcal{Z}^* = \mathcal{Z} \setminus \{z^*\}$ be the set of classes *different* from z^*. The average number of classes checked during the searching phase before the correct key is found, is given by the expression

$$\sum_{z \in \mathcal{Z}} \Pr[\mathcal{L}_{\mathbf{T}}(z) \geq \mathcal{L}_{\mathbf{T}}(z^*) \mid z^*] = 1 + \sum_{z \in \mathcal{Z}^*} \Pr[\mathcal{L}_{\mathbf{T}}(z) \geq \mathcal{L}_{\mathbf{T}}(z^*) \mid z^*],$$

where the random variable $\mathbf{T}$ represents the set of counters generated by a key from the class z^*, given N random plaintexts. The reason why we do not include the correct key class in the sum, is that this class will have to be treated differently later on. In order to compute the probabilities in this expression, we define the sets $\mathcal{T}_z = \{\mathbf{t} \mid \mathcal{L}_{\mathbf{t}}(z) \geq \mathcal{L}_{\mathbf{t}}(z^*)\}$. Using this notation, we can write

$$\Pr[\mathcal{L}_{\mathbf{T}}(z) \geq \mathcal{L}_{\mathbf{T}}(z^*) \mid z^*] = \sum_{\mathbf{t} \in \mathcal{T}_z} P_{z^*}(\mathbf{t}).$$

Knowing that each class z contains $2^k/|\mathcal{Z}|$ different keys, we can now derive the expected number of trials M^*, given a secret key K^*. Note that the number of keys that need to be checked in the correct equivalence class z^* is only $(2^n/|\mathcal{Z}| + 1)/2$ on the average, yielding

$$M^* = \frac{2^k}{|\mathcal{Z}|} \cdot \left[\frac{1}{2} + \sum_{z \in \mathcal{Z}^*} \sum_{\mathbf{t} \in \mathcal{T}_z} P_{z^*}(\mathbf{t})\right] + \frac{1}{2}. \tag{4.3}$$

This expression needs to be averaged over all possible secret keys K^* in order to find the expected value M, but in many cases[1] we will find that M^* does not depend on the actual value of K^*, such that $M = M^*$. Finally, the gain of the attack is computed by substituting this value of M into (4.1).

At this point, the reader might be somewhat confused by the several sets, vectors, and sums introduced in this section. Their concrete meaning will hopefully become clear in the next section.

4.3 Application to Multiple Approximations

We now apply the ideas discussed above to construct a general framework for analyzing block ciphers using multiple linear approximations.

[1] In some cases the variance of the gain over different keys would be very significant. In these cases it might be worth to exploit this phenomenon in a weak-key attack scenario, like we did in Sect. 3.6.

As we saw in Sect. 3.3, the starting point in linear cryptanalysis is the existence of unbalanced linear expressions involving plaintext bits, ciphertext bits, and key bits. In this paper we assume that we can use m such expressions (we will show how to find them in practice in Sect. 4.6):

$$\Pr\left[P^{\mathsf{T}} \cdot \Gamma_P^j \oplus C^{\mathsf{T}} \cdot \Gamma_C^j \oplus K^{\mathsf{T}} \cdot \Gamma_K^j = 0\right] = \frac{1}{2} + \epsilon_j\,, \quad j = 1, \ldots, m\,, \qquad (4.4)$$

with (P, C) a random plaintext-ciphertext pair encrypted with a random key K.[2] As mentioned earlier, the deviation ϵ_j is called the bias of the linear expression.

We now use the framework of Sect. 4.2.1 to design an attack which exploits the information contained in (4.4). The first phase of the cryptanalysis consists in extracting the relevant parts from the N plaintext-ciphertext pairs. The linear expressions in (4.4) immediately suggest the following functions ψ_j:

$$x_{i,j} = \psi_j(P_i, C_i) = P_i^{\mathsf{T}} \cdot \Gamma_P^j \oplus C_i^{\mathsf{T}} \cdot \Gamma_C^j\,, \quad i = 1, \ldots, N\,,$$

with $x_{i,j} \in \mathcal{X}_j = \{0, 1\}$. These values are then used to construct m counter vectors $\mathbf{t_j} = (t_j, N - t_j)$, where t_j and $N - t_j$ reflect the number of plaintext-ciphertext pairs for which $x_{i,j}$ equals 0 and 1, respectively.[3]

In the second step of the framework, a list of candidate key classes needs to be generated. We represent the equivalent key classes induced by the m linear expressions in (4.4) by an m-bit word $z = (z_1, \ldots, z_m)$ with $z_j = K^{\mathsf{T}} \cdot \Gamma_K^j$. Note that m might possibly be much larger than k, the length of the key K. In this case, only a subspace of all possible m-bit words corresponds to a valid key class. The exact number of classes $|\mathcal{Z}|$ depends on the number of *independent* linear approximations (i.e., the rank of the corresponding linear system).

4.3.1 Computing the Likelihoods of the Key Classes

We will for now assume that the linear expressions in (4.4) are statistically independent for different plaintext-ciphertext pairs and for different values of j (in the next section we will discuss this important point in more details). This allows us to apply the maximum likelihood approach described earlier in a very straightforward way. In order to simplify notations, we define the probabilities p_j and q_j, and the correlations or imbalances c_j of the linear expressions as

$$p_j = 1 - q_j = \frac{1 + c_j}{2} = \frac{1}{2} + \epsilon_j\,.$$

We start by deriving a convenient expression for the probability $P_z(\mathbf{t})$. To simplify the calculation, we first give a derivation for the special key class $z' = (0, \ldots, 0)$. Assuming independence of different approximations and of

[2] For notational convenience, we have switched the order of the scalar products compared to Sect. 3.3.

[3] The vectors $\mathbf{t_j}$ are only constructed to be consistent with the framework described earlier. In practice of course, the attacker will only calculate t_j (this is a minimal sufficient statistic).

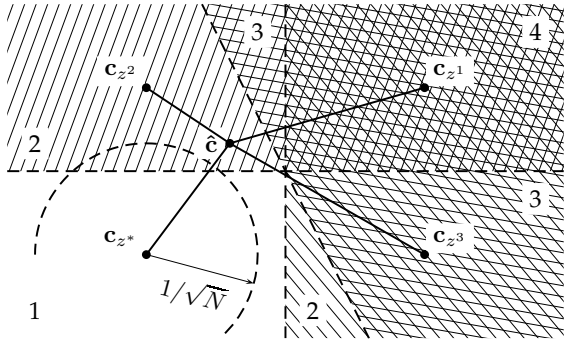


Figure 4.1: Geometrical interpretation for $m = 2$. The correct key class z^* has the second largest likelihood in this example. The numbers in the picture represent the number of trials M^* when $\hat{\mathbf{c}}$ falls in the associated area

different (P_i, C_i) pairs, the probability that this key generates the counters t_j is given by the product

$$P_{z'}(\mathbf{t}) = \prod_{j=1}^{m} \binom{N}{t_j} \cdot p_j^{t_j} \cdot q_j^{N - t_j}\,. \qquad (4.5)$$

In practice, p_j and q_j will be very close to $1/2$, and N very large. Taking this into account, we approximate the m-dimensional binomial distribution above by an m-dimensional Gaussian distribution:

$$P_{z'}(\mathbf{t}) \approx \prod_{j=1}^{m} \frac{e^{-\frac{(t_j - p_j \cdot N)^2}{N/2}}}{\sqrt{\pi \cdot N/2}} = \prod_{j=1}^{m} \frac{e^{-\frac{N}{2}(\hat{c}_j - c_j)^2}}{\sqrt{\pi \cdot N/2}} = \frac{e^{-\frac{N}{2}\sum(\hat{c}_j - c_j)^2}}{\left(\sqrt{\pi \cdot N/2}\right)^m}\,.$$

The variable $\hat{c}_j$ is called the *estimated imbalance* and is derived from the counters t_j according to the relation $N \cdot (1 + \hat{c}_j)/2 = t_j$. For any key class z, we can repeat the reasoning above, yielding the following general expression:

$$P_z(\mathbf{t}) \approx \frac{e^{-\frac{N}{2}\sum(\hat{c}_j - (-1)^{z_j} \cdot c_j)^2}}{\left(\sqrt{\pi \cdot N/2}\right)^m}\,. \qquad (4.6)$$

This formula has a useful geometrical interpretation: if we take a key from a fixed key class z^* and construct an m-dimensional vector $\hat{\mathbf{c}} = (\hat{c}_1, \ldots, \hat{c}_m)$ by encrypting N random plaintexts, then $\hat{\mathbf{c}}$ will be distributed around the vector $\mathbf{c_{z^*}} = ((-1)^{z_1^*} c_1, \ldots, (-1)^{z_m^*} c_m)$ according to a Gaussian distribution with a diagonal variance-covariance matrix $1/\sqrt{N} \cdot I_m$, where I_m is an $m \times m$ identity matrix. This is illustrated in Fig. 4.1. From (4.6) we can now directly compute

the log-likelihood:

$$\mathcal{L}_{\mathbf{t}}(z) = \log L_{\mathbf{t}}(z) = \log P_z(\mathbf{t}) \approx C - \frac{N}{2}\sum_{j=1}^{m}(\hat{c}_j - (-1)^{z_j}\cdot c_j)^2\,. \tag{4.7}$$

The constant C depends on m and N only, and is irrelevant to the attack. From this formula we immediately derive the following property.

Lemma 4.1. *The relative likelihood of a key class z is completely determined by the Euclidean distance $|\hat{\mathbf{c}} - \mathbf{c_z}|$, where $\hat{\mathbf{c}}$ is an m-dimensional vector containing the estimated imbalances derived from the known texts, and $\mathbf{c_z} = ((-1)^{z_1}c_1, \ldots, (-1)^{z_m}c_m)$.*

The lemma implies that $\mathcal{L}_{\mathbf{T}}(z) > \mathcal{L}_{\mathbf{T}}(z^*)$ if and only if $|\hat{\mathbf{c}} - \mathbf{c_z}| < |\hat{\mathbf{c}} - \mathbf{c_{z^*}}|$. This type of result is common in coding theory.

4.3.2 Estimating the Gain of the Attack

Based on the geometrical interpretation given above, and using the results from Sect. 4.2.3, we can now easily derive the gain of the attack.

Theorem 4.2. *Given m approximations and N independent pairs (P_i, C_i), an adversary can mount a linear attack with a gain equal to:*

$$\gamma = -\log_2\left[2\cdot\frac{1}{|\mathcal{Z}|}\sum_{z\in\mathcal{Z}^*}\Phi\left(-\sqrt{N}\cdot\frac{|\mathbf{c}_z - \mathbf{c}_{z^*}|}{2}\right) + \frac{1}{|\mathcal{Z}|}\right], \tag{4.8}$$

where $\Phi(\cdot)$ is the cumulative normal distribution function, $|\mathcal{Z}|$ is the number of key classes induced by the approximations, and $\mathbf{c_z} = ((-1)^{z_1}c_1, \ldots, (-1)^{z_m}c_m)$.

Proof. The probability that the likelihood of a key class z exceeds the likelihood of the correct key class z^* is given by the probability that the vector $\hat{\mathbf{c}}$ falls into the half space $\mathcal{T}_c = \{\mathbf{c} \mid |\mathbf{c} - \mathbf{c_z}| \le |\mathbf{c} - \mathbf{c_{z^*}}|\}$. We know that $\hat{\mathbf{c}}$ describes a Gaussian distribution around $\mathbf{c}_{z^*}$ with a variance-covariance matrix $1/\sqrt{N}\cdot I_m$, and by integrating this Gaussian over the half plane $\mathcal{T}_c$, we can easily compute the desired probability. Due to the zero covariances, we immediately find:

$$\Pr\left[\mathcal{L}_{\mathbf{T}}(z) \ge \mathcal{L}_{\mathbf{T}}(z^*) \mid z^*\right] = \Phi\left(-\sqrt{N}\cdot\frac{|\mathbf{c}_z - \mathbf{c}_{z^*}|}{2}\right).$$

By summing these probabilities as in (4.3) we find the expected number of trials:

$$M^* = \frac{2^k}{|\mathcal{Z}|}\cdot\left[\frac{1}{2} + \sum_{z\in\mathcal{Z}^*}\Phi\left(-\sqrt{N}\cdot\frac{|\mathbf{c}_z - \mathbf{c}_{z^*}|}{2}\right)\right] + \frac{1}{2}\,. \tag{4.9}$$

The gain is obtained by substituting this expression for M^* in equation (4.1). □

The formula derived in the previous theorem can easily be evaluated as long as $|\mathcal{Z}|$ is not too large. In order to estimate the gain in the other cases as well, we need to make a few approximations.

Corollary 4.3. *If $|\mathcal{Z}|$ is sufficiently large, the gain derived in Theorem 4.2 can accurately be approximated by*

$$\gamma \approx -\log_2\left[2\cdot\frac{|\mathcal{Z}|-1}{|\mathcal{Z}|}\cdot\Phi\left(-\sqrt{\frac{N\cdot\bar{c}^2}{2}}\right) + \frac{1}{|\mathcal{Z}|}\right], \tag{4.10}$$

where $\bar{c}^2 = \sum_{j=1}^{m} c_j^2$.

Proof. In order to show how (4.10) is derived from (4.8), we just need to construct an approximation for the expression

$$\frac{1}{|\mathcal{Z}^*|}\sum_{z\in\mathcal{Z}^*}\Phi\left(-\sqrt{N}\cdot\frac{|\mathbf{c}_z - \mathbf{c}_{z^*}|}{2}\right) = \frac{1}{|\mathcal{Z}^*|}\sum_{z\in\mathcal{Z}^*}\Phi\left(-\sqrt{N/4\cdot|\mathbf{c}_z - \mathbf{c}_{z^*}|^2}\right). \tag{4.11}$$

We first define the function $f(x) = \Phi(-\sqrt{N/4\cdot x})$. Denoting the average value of a set of variables by $E[\cdot] = \widehat{\cdot}$, we can reduce (4.11) to the compact expression $E[f(x)]$, with $x = |\mathbf{c}_z - \mathbf{c}_{z^*}|^2$. By expanding $f(x)$ into a Taylor series around the average value $\widehat{x}$, we find

$$E[f(x)] = f(\widehat{x}) + 0 + f''(\widehat{x})\cdot E[(x-\widehat{x})^2] + \ldots\,.$$

Provided that the higher order moments of x are sufficiently small, we can use the approximation $E[f(x)] \approx f(\widehat{x})$. Exploiting the fact that the jth coordinate of each vector $\mathbf{c}_z$ is either c_j or $-c_j$, we can easily calculate the average value $\widehat{x}$:

$$\widehat{x} = \frac{1}{|\mathcal{Z}^*|}\sum_{z\in\mathcal{Z}^*}|\mathbf{c}_z - \mathbf{c}_{z^*}|^2 = 2\cdot\frac{|\mathcal{Z}|}{|\mathcal{Z}^*|}\sum_{j=1}^{m}c_j^2\,.$$

When $|\mathcal{Z}|$ is sufficiently large (say $|\mathcal{Z}| > 2^8$), the right hand side can be approximated by $2\cdot\sum_{j=1}^{m}c_j^2 = 2\cdot\bar{c}^2$ (remember that $\mathcal{Z}^* = \mathcal{Z}\setminus\{z^*\}$, and thus $|\mathcal{Z}^*| = |\mathcal{Z}| - 1$). Substituting this into the relation $E[f(x)] \approx f(\widehat{x})$, we find

$$\frac{1}{|\mathcal{Z}^*|}\sum_{z\in\mathcal{Z}^*}\Phi\left(-\sqrt{N}\cdot\frac{|\mathbf{c}_z - \mathbf{c}_{z^*}|}{2}\right) \approx \Phi\left(-\sqrt{\frac{N\cdot\bar{c}^2}{2}}\right).$$

By applying this approximation to the gain formula derived in Theorem 4.2, we directly obtain expression (4.10). □

An interesting conclusion that can be drawn from the corollary above is that the gain of the attack is mainly determined by the product $N\cdot\bar{c}^2$. As a result, if we manage to increase $\bar{c}^2$ by using more linear characteristics, then the required number of known plaintext-ciphertext pairs N can be decreased by the same factor, without affecting the gain. Since the quantity $\bar{c}^2$ plays a very important role in the attacks, we give it a name and define it explicitly.

Definition 4.2. The *capacity* $\bar{c}^2$ of a system of m approximations is defined as

$$\bar{c}^2 = \sum_{j=1}^{m} c_j^2 = 4 \cdot \sum_{j=1}^{m} \epsilon_j^2 \,.$$

4.3.3 Multiple Approximations and Matsui's Algorithm 2

The approach taken in the previous section can be seen as an extension of Matsui's Algorithm 1. Just as in Algorithm 1, the adversary analyses parity bits of the known plaintext-ciphertext pairs and then tries to determine parity bits of internal round keys. An alternative approach, which is called Algorithm 2 and yields much more efficient attacks in practice, consists in guessing parts of the round keys in the first and the last round, and determining the probability that the guess was correct by exploiting linear characteristics over the remaining rounds. In this section we will show that the results derived above can still be applied in this situation, provided that we modify some definitions.

Let us denote by $\mathcal{Z}_O$ the set of possible guesses for the targeted subkeys of the outer rounds (round 1 and round r). For each guess z_O and for all N plaintext-ciphertext pairs, the adversary does a partial encryption and decryption at the top and bottom of the block cipher, and recovers the parity bits of the intermediate data blocks involved in m different $(r-2)$-round linear characteristics. Using this data, he constructs $m' = |\mathcal{Z}_O| \cdot m$ counters t_j, which can be transformed into a m'-dimensional vector $\hat{\mathbf{c}}$ containing the estimated imbalances.

As explained in the previous section, the m linear characteristics involve m parity bits of the key, and thus induce a set of equivalent key classes, which we will here denote by $\mathcal{Z}_I$ (I from *inner*). Although not strictly necessary, we will for simplicity assume that the sets $\mathcal{Z}_O$ and $\mathcal{Z}_I$ are independent, such that each guess $z_O \in \mathcal{Z}_O$ can be combined with any class $z_I \in \mathcal{Z}_I$, thereby determining a subclass of keys $z = (z_O, z_I) \in \mathcal{Z}$ with $|\mathcal{Z}| = |\mathcal{Z}_O| \cdot |\mathcal{Z}_I|$.

At this point, the situation is very similar to the one described in the previous section, the main difference being a higher dimension m'. The only remaining question is how to construct the m'-dimensional vectors $\mathbf{c_z}$ for each key class $z = (z_O, z_I)$. To solve this problem, we will need to make some assumptions. Remember that the coordinates of $\mathbf{c_z}$ are determined by the expected imbalances of the corresponding linear expressions, given that the data is encrypted with a key from class z. For the m counters that are constructed after guessing the correct subkey z_O, the expected imbalances are determined by z_I and equal to $(-1)^{z_{I,1}} c_1, \ldots, (-1)^{z_{I,m}} c_m$. For each of the $m' - m$ other counters, however, we will assume that the wrong guesses result in independent random-looking parity bits, showing no imbalance at all.[4] Accordingly,

[4] Note that for some ciphers, other assumptions may be more appropriate. The reasoning in this section can be applied to these cases just as well, yielding very similar results.

the vector $\mathbf{c_z}$ has the following form:

$$\mathbf{c_z} = (0, \ldots, 0, (-1)^{z_{I,1}} c_1, \ldots, (-1)^{z_{I,m}} c_m, 0, \ldots, 0)$$

With the modified definitions of $\mathcal{Z}$ and c_z given above, Theorem 4.2 can immediately be applied. This results in the following corollary.

Corollary 4.4. *Given m approximations and N independent pairs (P_i, C_i), an adversary can mount an Algorithm 2 style linear attack with a gain equal to:*

$$\gamma = -\log_2 \left[2 \cdot \frac{1}{|\mathcal{Z}|} \sum_{z \in \mathcal{Z}^*} \Phi\left(-\sqrt{N} \cdot \frac{|\mathbf{c}_z - \mathbf{c}_{z^*}|}{2} \right) + \frac{1}{|\mathcal{Z}|} \right] . \tag{4.12}$$

The formula above involves a summation over all elements of $\mathcal{Z}^*$. Motivated by the fact that $|\mathcal{Z}^*| = |\mathcal{Z}_O| \cdot |\mathcal{Z}_I| - 1$ is typically very large, we now derive a more convenient approximated expression similar to Corollary 4.3. In order to do this, we split the sum into two parts. The first part considers only keys $z \in \mathcal{Z}_1^* = \mathcal{Z}_1 \setminus \{z^*\}$ where $\mathcal{Z}_1 = \{z \mid z_O = z_O^*\}$; the second part sums over all remaining keys $z \in \mathcal{Z}_2 = \{z \mid z_O \neq z_O^*\}$. In this second case, we have that $|\mathbf{c}_z - \mathbf{c}_{z^*}|^2 = 2 \cdot \sum_{j=1}^{m} c_j^2 = 2 \cdot \bar{c}^2$ for all $z \in \mathcal{Z}_2$, such that

$$\sum_{z \in \mathcal{Z}_2} \Phi\left(-\sqrt{N} \cdot \frac{|\mathbf{c}_z - \mathbf{c}_{z^*}|}{2} \right) = |\mathcal{Z}_2| \cdot \Phi\left(-\sqrt{\frac{N \cdot \bar{c}^2}{2}} \right) .$$

For the first part of the sum, we apply the approximation used to derive Corollary 4.3 and obtain a very similar expression:

$$\sum_{z \in \mathcal{Z}_1^*} \Phi\left(-\sqrt{N} \cdot \frac{|\mathbf{c}_z - \mathbf{c}_{z^*}|}{2} \right) \approx |\mathcal{Z}_1^*| \cdot \Phi\left(-\sqrt{\frac{N \cdot \bar{c}^2}{2}} \right) .$$

Combining both result we find the counterpart of Corollary 4.3 for an Algorithm 2 style linear attack.

Corollary 4.5. *If $|\mathcal{Z}|$ is sufficiently large, the gain derived in Theorem 4.4 can accurately be approximated by*

$$\gamma \approx -\log_2 \left[2 \cdot \frac{|\mathcal{Z}| - 1}{|\mathcal{Z}|} \cdot \Phi\left(-\sqrt{\frac{N \cdot \bar{c}^2}{2}} \right) + \frac{1}{|\mathcal{Z}|} \right] , \tag{4.13}$$

where $\bar{c}^2 = \sum_{j=1}^{m} c_j^2$ is the total capacity of the m linear characteristics.

Notice that although Corollary 4.3 and 4.5 contain identical formulas, the gain of the Algorithm 2 style linear attack will be significantly larger because it depends on the capacity of linear characteristics over $r-2$ rounds instead of r rounds.

4.3.4 Influence of Dependencies

When deriving (4.5) in Sect. 4.3, we assumed statistical independence. This assumption is not always fulfilled, however. In this section we discuss different potential sources of dependencies and estimate how they might influence the cryptanalysis.

Dependent plaintext-ciphertext pairs.

A first assumption made by equation (4.5) concerns the dependency of the parity bits $x_{i,j}$ with $1 \leq i \leq N$, computed with a single linear approximation for different plaintext-ciphertext pairs. The equation assumes that the probability that the approximation holds for a single pair equals $p_j = 1/2 + \epsilon_j$, regardless of what is observed for other pairs. This is a very reasonable assumption if the N plaintexts are chosen randomly, but even if they are picked in a systematic way, we can still safely assume that the corresponding ciphertexts are sufficiently unrelated as to prevent statistical dependencies.

Dependent text masks.

The next source of dependencies is more fundamental and is related to dependent text masks. Suppose for example that we want to use three linear approximations with plaintext-ciphertext masks (Γ_P^1, Γ_C^1), (Γ_P^2, Γ_C^2), (Γ_P^3, Γ_C^3), and that $\Gamma_P^1 \oplus \Gamma_P^2 \oplus \Gamma_P^3 = \Gamma_C^1 \oplus \Gamma_C^2 \oplus \Gamma_C^3 = 0$. It is immediately clear that the parity bits computed for these three approximations cannot possibly be independent: for all (P_i, C_i) pairs, the bit computed for the 3rd approximation $x_{i,3}$ is equal to $x_{i,1} \oplus x_{i,2}$.

Even in such cases, however, we believe that the results derived in the previous section are still quite reasonable. In order to show this, we consider the probability that a single random plaintext encrypted with an equivalent key z yields a vector[5] of parity bits $\mathbf{x} = (x_1, \ldots, x_m)$. Let us denote by Γ_T^j the concatenation of both text masks Γ_P^j and Γ_C^j. Without loss of generality, we can assume that the m masks Γ_T^j are linearly independent for $1 \leq j \leq l$ and linearly dependent (but different) for $l < j \leq m$. This implies that $\mathbf{x}$ is restricted to a l-dimensional subspace $\mathcal{R}$. We will only consider the key class $z' = (0, \ldots, 0)$ in order to simplify the equations. The probability we want to evaluate is:

$$P_{z'}(\mathbf{x}) = \Pr\left[X_j = x_j \text{ for } 1 \leq j \leq m \mid z'\right] .$$

These (unknown) probabilities determine the (known) imbalances c_j of the linear approximations through the following expression:

$$c_j = \sum_{\mathbf{x} \in \mathcal{R}} P_{z'}(\mathbf{x}) \cdot (-1)^{x_j} .$$

[5]Note a small abuse of notation here: the definition of $\mathbf{x}$ differs from the one used in Sect. 4.2.1.

We now make the (in many cases reasonable) assumption that all $2^l - m$ masks Γ_T, which depend linearly on the masks Γ_T^j, but which differ from the ones considered by the attack, have negligible imbalances (except for $\Gamma_T = 0$, which has imbalance 1). In this case, the equation above can be reversed (note the similarity with the Walsh-Hadamard transform), and we find that:

$$P_{z'}(\mathbf{x}) = \frac{1}{2^l} \left[1 + \sum_{j=1}^{m} c_j \cdot (-1)^{x_j} \right] .$$

Assuming that $m \cdot c_j \ll 1$ we can make the following approximation:

$$P_{z'}(\mathbf{x}) \approx \frac{2^m}{2^l} \prod_{j=1}^{m} \frac{1 + c_j \cdot (-1)^{x_j}}{2} .$$

Apart from an irrelevant constant factor $2^m/2^l$, this is exactly what we need: it implies that, even with dependent masks, we can still multiply probabilities as we did in order to derive (4.5). This is an important conclusion, because it indicates that the capacity of the approximations continues to grow, even when m exceeds twice the block size, in which case the masks are necessarily linearly dependent.

Dependent trails.

A third type of dependencies might be caused by merging linear trails. When analyzing the best linear approximations for DES, for example, we notice that most of the good linear approximations follow a very limited number of trails through the inner rounds of the cipher, which might result in dependencies. Although this effect did not appear to have any influence on our experiments (with up to 100 different approximations), we cannot exclude at this point that they will affect attacks using much more approximations.

Dependent key masks.

We finally note that we did not require the independence of key masks in the previous sections. This implies that all results derived above remain valid for dependent key masks.

4.4 Discussion – Practical Aspects

When attempting to calculate the optimal estimators derived in Sect. 4.3, the attacker might be confronted with some practical limitations, which are often cipher-dependent. In this section we discuss possible problems and propose ways to deal with them.

Table 4.1: Attack Algorithm MK 1: complexities

	Data compl.	Time compl.	Memory compl.
Distillation:	$\mathcal{O}(1/\bar{c}^2)$	$\mathcal{O}(m/\bar{c}^2)$	$\mathcal{O}(m)$
Analysis:	-	$\mathcal{O}(m \cdot \lvert\mathcal{Z}\rvert)$	$\mathcal{O}(\lvert\mathcal{Z}\rvert)$
Search:	-	$\mathcal{O}(2^{k-\gamma})$	$\mathcal{O}(\lvert\mathcal{Z}\rvert)$

4.4.1 Attack Algorithm MK 1

First, let us summarize the attack algorithm presented in Sect. 4.2. We will call this algorithm Attack Algorithm MK 1 ('M' for multiple approximations, "K' for key recovery). Given m approximations (see Sect. 4.6), the algorithm proceeds as follows:

Distillation phase. Obtain N plaintext-ciphertext pairs (P_i, C_i). For $1 \leq j \leq m$, count the number t_j of pairs satisfying $P_i^{\mathsf{T}} \cdot \Gamma_P^j \oplus C_i^{\mathsf{T}} \cdot \Gamma_C^j = 0$ and compute the estimated imbalance $\hat{c}_j = 2 \cdot t_j/N - 1$.

Analysis phase. For each equivalent key class $z \in \mathcal{Z}$, determine the distance

$$\left|\hat{\mathbf{c}} - \mathbf{c_z}\right|^2 = \sum_{j=1}^{m} (\hat{c}_j - (-1)^{z_j} \cdot c_j)^2$$

and use these values to construct a sorted list, starting with the class with the smallest distance.

Search phase. Run through the sorted list and exhaustively try all n-bit keys contained in the equivalence classes until the correct key is found.

Table 4.1 lists the complexities of these different phases. A few practical remarks are in order.

When estimating the potential gain in Sect. 4.3, we did not impose any restrictions on the number of approximations m. However, while it does reduce the complexity of the search phase (since it increases the gain), having an excessively high number m increases both the time and the space complexity of the distillation and the analysis phase. At some point this complexity will dominate, cancelling out any improvement made in the search phase.

Analyzing the complexities in the table, we can make a few observations. We first note that the time complexity of the distillation phase should be compared to the time needed to encrypt $N \propto 1/\bar{c}^2$ plaintext-ciphertext pairs. Given that a single counting operation is much faster than an encryption, we expect the complexity of the distillation to remain negligible compared to the encryption time as long as m is only a few orders of magnitude (say $m < 100$).

The second observation is that the number of different key classes $|\mathcal{Z}|$ clearly plays an important role, both for the time and the memory complexities of the algorithm. In a practical situation, the memory is expected to be the strongest limitation. Different approaches can be taken to deal with this problem:

Straightforward, but inefficient approach. Since the number of different key classes $|\mathcal{Z}|$ is bounded by 2^m, the most straightforward solution is to limit the number of approximations. A realistic upper bound would be $m < 32$. The obvious drawback of this approach is that it will not allow to attain very high capacities.

Exploiting dependent key masks. A better approach is to impose a bound on the number l of *linearly independent* key masks Γ_K^j. This way, we limit the memory requirements to $|\mathcal{Z}| = 2^l$, but still allow a large number of approximations (for ex. a few thousands). This approach restricts the choice of approximations, however, and thus reduces the maximum attainable capacity. This is the approach taken in Sect. 4.5.1. Note also that the attack described in [59] can be seen as a special case of this approach, with $l = 1$.

Merging separate lists. A third strategy consists in constructing separate lists and merging them dynamically. Suppose for simplicity that the m key masks Γ_K^j considered in the attack are all independent. In this case, we can apply the analysis phase twice, each time using $m/2$ approximations. This will result in two sorted lists of intermediate key classes, both containing $2^{m/2}$ classes. We can then dynamically compute a sorted sequence of final key classes constructed by taking the product of both lists. The ranking of the sequence is determined by the likelihood of these final classes, which is just the sum of the likelihoods of the elements in the separate lists. This approach slightly increases[6] the time complexity of the analysis phase, but will considerably reduce the memory requirements. Note that this approach can be generalized in order to allow some dependencies in the key masks.

4.4.2 Attack Algorithm MK 2

We now briefly discuss some practical aspects of the Algorithm 2 style multiple linear attack, called Attack Algorithm MK 2. As discussed earlier, the ideas of the attack are very similar to Attack Algorithm MK 1, but there are a number of additional issues. In the following paragraphs, we denote the number of rounds of the cipher by r.

[6] In cases where the gain of the attack is several bits, this approach will actually decrease the complexity, since we expect that only a fraction of the final sequence will need to be computed.

Choice of characteristics. In order to limit the amount of guesses in rounds 1 and r, only parts of the subkeys in these rounds will be guessed. This restricts the set of useful $(r-2)$-round characteristics to those that only depend on bits which can be derived from the plaintext, the ciphertext, and the partial subkeys. This obviously reduces the maximum attainable capacity.

Efficiency of the distillation phase. During the distillation phase, all N plaintexts need to be analyzed for all $|\mathcal{Z}_O|$ guesses z_O. Since $|\mathcal{Z}_O|$ is rather large in practice, this could be very computation intensive. For example, a naive implementation would require $\mathcal{O}(N \cdot |\mathcal{Z}_O|)$ steps and even Matsui's counting trick [77] would use $\mathcal{O}(N+|\mathcal{Z}_O|^2)$ steps. However, the distillation can be performed in $\mathcal{O}(N+|\mathcal{Z}_O|)$ steps by gradually guessing parts of z_O and re-processing the counters.

Merging Separate lists. The idea of working with separate lists can be applied here just as for MK 1.

Computing distances. In order to compare the likelihoods of different keys, we need to evaluate the distance $|\hat{\mathbf{c}} - \mathbf{c_z}|^2$ for all classes $z \in \mathcal{Z}$. The vectors $\hat{\mathbf{c}}$ and $\mathbf{c_z}$ are both $|\mathcal{Z}_O| \cdot m$-dimensional. When calculating this distance as a sum of squares, most terms do not depend on z, however. This allows the distance to be computed very efficiently, by summing only m terms.

4.4.3 Attack Algorithm MD (distinguishing/key-recovery)

The main limitation of Algorithm MK 1 and MK 2 is the bound on the number of key classes $|\mathcal{Z}|$. In this section, we show that this limitation disappears if our sole purpose is to distinguish an encryption algorithm E_k from a random permutation R. As usual, the distinguisher can be extended into a key-recovery attack by adding rounds at the top and at the bottom.

If we observe N plaintext-ciphertext pairs and assume for simplicity that the a priori probability that they were constructed using the encryption algorithm is $1/2$, we can construct a distinguishing attack using the maximum likelihood approach in a similar way as in Sect. 4.3. Assuming that all secret keys k are equally probable, one can easily derive the likelihood that the encryption algorithm was used, given the values of the counters $\mathbf{t}$:

$$L_E(\mathbf{t}) \approx \frac{1}{2^m} \prod_{j=1}^{m} \binom{N}{t_j} \cdot \left(p_j^{t_j} \cdot q_j^{N-t_j} + q_j^{t_j} \cdot p_j^{N-t_j} \right).$$

This expression is correct if all text masks and key masks are independent, but is still expected to be a good approximation, if this assumption does not hold (for the reasons discussed in Sect. 4.3.4). A similar likelihood can be calculated for the random permutation:

$$L_R(\mathbf{t}) = \prod_{j=1}^{m} \binom{N}{t_j} \cdot \left(\frac{1}{2}\right)^N.$$

Contrary to what was found for Algorithm MK 1, both likelihoods can be computed in time proportional to m, i.e., independent of $|\mathcal{Z}|$. The complete distinguishing algorithm, called *Attack Algorithm MD* consists of two steps:

Distillation phase. Obtain N plaintext-ciphertext pairs (P_i, C_i). For $1 \le j \le m$, count the number t_j of pairs satisfying $P_i^{\mathsf{T}} \cdot \Gamma_P^j \oplus C_i^{\mathsf{T}} \cdot \Gamma_C^j = 0$.

Analysis phase. Compute $L_E(\mathbf{t})$ and $L_R(\mathbf{t})$. If $L_E(\mathbf{t}) > L_R(\mathbf{t})$, decide that the plaintexts were encrypted with the algorithm E_k (using some unknown key k).

The analysis of this algorithm is a matter of further research.

4.5 Experimental Results

In Sect. 4.3, we derived an optimal approach for cryptanalyzing block ciphers using multiple linear approximations. In this section, we discuss the performance of a number of practical implementations of attacks based on this approach. Our target block cipher is DES, the standard benchmark for linear cryptanalysis. The experiments show that the attack complexities are in perfect correspondence with the theoretical results derived in the previous sections.

4.5.1 Attack Algorithm MK 1

In our first experiment, we apply Algorithm MK 1 to 8 rounds of DES, using 86 linear approximations with a total capacity $\bar{c}^2 = 2^{-15.6}$ (see Definition 4.2). In order to speed up the simulation, the approximations are picked to contain 10 linearly independent key masks, such that $|\mathcal{Z}| = 1024$. Fig. 4.2 shows the simulated gain for Algorithm MK 1 using these 86 approximations, and compares it to the gain of Matsui's Algorithm 1, which uses the best one only ($\bar{c}^2 = 2^{-19.4}$). We clearly see a significant improvement. While Matsui's algorithm requires about 2^{21} pairs to attain a gain close to 1 bit, only 2^{16} pairs suffice for Algorithm MK 1. The theoretical curves shown in the figure are plotted by computing the gain using the exact expression for M^* derived in Theorem 4.2 and using the approximation from Corollary 4.3. Both fit nicely with the experimental results.

Note that the attack presented in this section is just a proof of concept, even higher gains would be possible with more optimized attacks. For a more detailed discussion of the technical aspects playing a role in the implementation of Algorithm MK 1, we refer to Sect. 4.4.

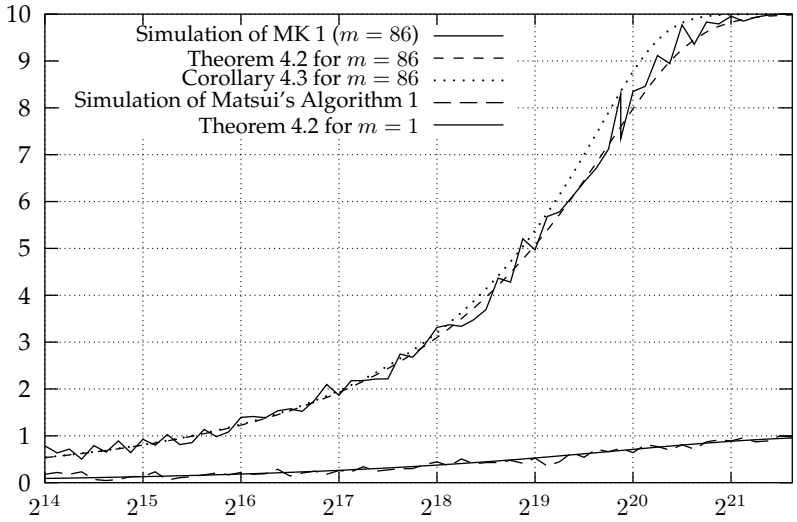


Figure 4.2: Gain in function of data for 8-round DES

4.5.2 Attack Algorithm MK 2

Let us now take a look at Attack Algorithm MK 2. Again, we run our experiments on 8 rounds of DES, and this time we compare the results to the gain of the corresponding Algorithm 2 attack described in Matsui's paper [77].

Our attack uses eight linear approximations spanning six rounds with a total capacity $\bar{c}^2 = 2^{-11.9}$. In order to compute the parity bits of these equations, eight 6-bit subkeys need to be guessed in the first and the last rounds (how this is done in practice has been discussed in Sect. 4.4.2). Fig. 4.3 plots the gain of the attack against that of Matsui's Algorithm 2, which only uses the two best approximations ($\bar{c}^2 = 2^{-13.2}$). For the same amount of data, the multiple linear attack clearly achieves a much higher gain. This reduces the complexity of the search phase by multiple orders of magnitude. On the other hand, for the same gain, the adversary can reduce the amount of data by a (modest) factor 2. For example, for a gain of 12 bits, the data complexity is reduced from $2^{17.8}$ to $2^{16.6}$. This is in close correspondence with the ratio between the capacities. Note that both simulations were carried out under the assumption of independent subkeys (this was also the case for the simulations presented in [77]). Without this assumption, the gain will closely follow the graphs on the figure, but stop increasing as soon as the gain equals the number of independent key bits involved in the attack.

As in Sect. 4.5.1 our goal is not to provide the best attack on 8-round DES, but to show that Algorithm 2 style attacks do gain from the use of multiple linear approximations, with a data reduction proportional to the increase in

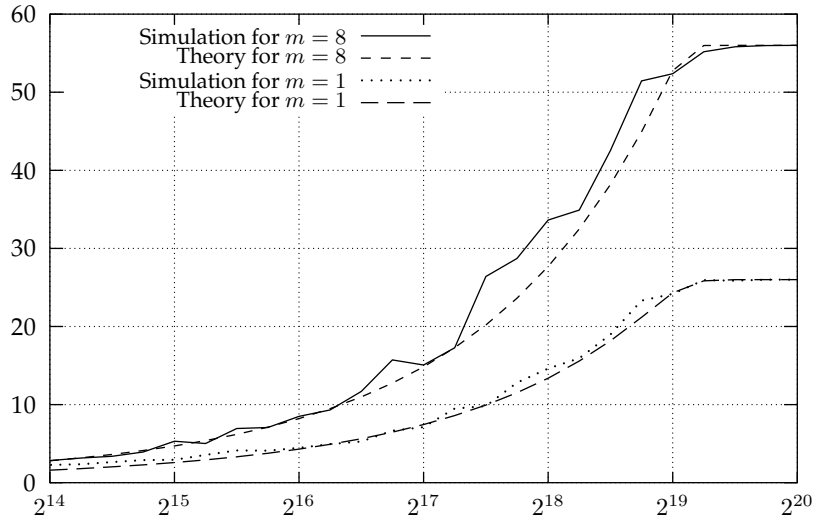


Figure 4.3: Gain in function of data for 8-round DES

the joint capacity. We refer to Sect. 4.4 for the technical aspects of the implementation of Algorithm MK 2.

4.5.3 Capacity – DES Case Study

In Sect. 4.3 we argued that the minimal amount of data needed to obtain a certain gain compared to exhaustive search is determined by the capacity $\bar{c}^2$ of the linear approximations. In order to get a first estimate of the potential improvement of using multiple approximations, we calculate the total capacity of the best m linear approximations of DES for $1 \leq m \leq 2^{16}$. The capacities are computed using an adapted version of yet another algorithm by [78] (see Sect. 4.6). The results, plotted for different number of rounds, are shown in Fig. 4.4 and 4.5, both for approximations restricted to a single S-box per round and for the general case. Note that the single best approximation is not visible on these figures due to the scale of the graphs.

Kaliski and Robshaw [59] showed that the first 10 006 approximations with a single active S-box per round have a joint capacity of $4.92 \cdot 10^{-11}$ for 14 rounds of DES.[7] Fig. 4.4 shows that this capacity can be increased to $4 \cdot 10^{-10}$ when multiple S-boxes are allowed. Comparing this to the capacity of Matsui's best approximation ($\bar{c}^2 = 1.29 \cdot 10^{-12}$), the factor 38 gained by Kaliski and Robshaw is increased to 304 in our case. Practical techniques to turn this increased capacity into an effective reduction of the data complexity are pre-

[7] Note that Kaliski and Robshaw calculated the sum of squared biases: $\sum \epsilon_j^2 = \bar{c}^2/4$.

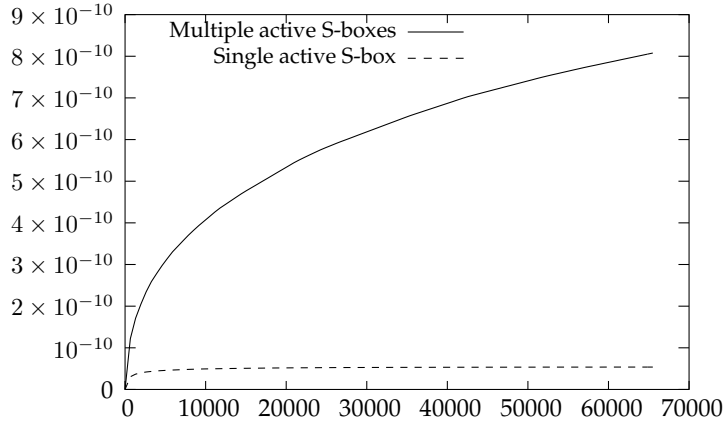


Figure 4.4: Capacity (14 rounds)

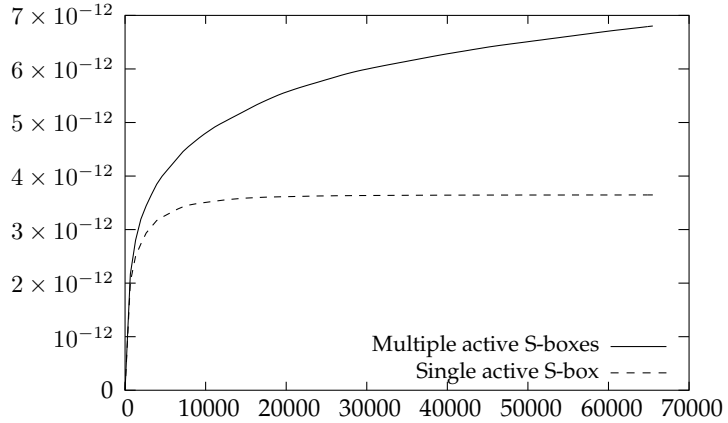


Figure 4.5: Capacity (16 rounds)

sented in this paper, but exploiting the full gain of 10 000 unrestricted approximations will require additional techniques. In theory, however, it would be possible to reduce the data complexity form 2^{43} (in Matsui's case, using two approximations) to about 2^{36} (using 10 000 approximations).

In order to provide a more conservative (and probably rather realistic) estimation of the implications of our new attacks on full DES, we search for 14-round approximations which only require three 6-bit subkeys to be guessed simultaneously in the first and the last rounds. The capacity of the 108 best approximations satisfying this restriction is $9.83 \cdot 10^{-12}$. This suggests that an MK 2 attack exploiting these 108 approximations might reduce the data complexity by a factor 4 compared to Matsui's Algorithm 2 (i.e., 2^{41} instead of 2^{43}). This is comparable to the Knudsen-Mathiassen reduction [65], but would preserve the advantage of being a known-plaintext attack rather than a chosen-plaintext one.

Using very high numbers of approximations is somewhat easier in practice for MK 1 because we do not have to impose restrictions on the plaintext and ciphertext masks (see Sect. 4.4). Analyzing the capacity for the 10 000 best 16-round approximations, we now find a capacity of $5 \cdot 10^{-12}$. If we restrict the complexity of the search phase to an average of 2^{43} trials (i.e., a gain of 12 bits), we expect that the attack will require 2^{41} known plaintexts. As expected, this theoretical number is larger than for the MK 2 attack using the same amount of approximations.

4.6 Finding characteristics

In the beginning of Sect. 4.3, we started from the assumption that we could construct m approximations of the form $P^{\mathsf{T}} \cdot \Gamma_P^j \oplus C^{\mathsf{T}} \cdot \Gamma_C^j \oplus K^{\mathsf{T}} \cdot \Gamma_K^j$, each with an imbalance $c_j \neq 0$. In this last section, we explain how these approximations are found in practice. The algorithm presented below is directly based on an algorithm by Matsui [78]. However, instead of computing the single best approximation, our algorithm returns a list of the m most imbalanced approximations. This will at the same time allow for some additional optimizations.

4.6.1 Some Notation

As explained in Sect. 3.3.2, the key idea is to build sequences of approximations called linear characteristics. We assume in this section that the block cipher consists of r identical rounds, each of which depends on a round key K_i, and we denote the intermediate values at the input and the output of the different rounds by $P = X_0, X_1, \ldots, X_r = C$. Our goal now is to construct linear characteristics of the form

$$(\Gamma_{X_0}, \Gamma_{X_1}, \ldots, \Gamma_{X_r}),$$

for which there exist key masks Γ_{K_i} such that all linear expressions

$$\Gamma_{X_{i-1}}^{\mathsf{T}} \cdot X_{i-1} \oplus \Gamma_{X_i}^{\mathsf{T}} \cdot X_i \oplus \Gamma_{K_i}^{\mathsf{T}} \cdot K_i\,, \quad \text{with } 1 \le i \le r,$$

are as biased as possible.[8] In order to simplify notations, we write the corresponding imbalances as $C(\Gamma_{X_{i-1}}, \Gamma_{X_i})$. We will assume that these r approximations are sufficiently independent such that the Piling-up Lemma of Sect. 3.3.3 applies. The imbalance of a complete characteristic is then given by the product

$$C(\Gamma_{X_0}, \Gamma_{X_1}, \ldots, \Gamma_{X_r}) = \prod_{i=1}^{r} C(\Gamma_{X_{i-1}}, \Gamma_{X_i})\,.$$

Finally, we also introduce the notation $C(S_r)$, which returns the set of imbalances corresponding to a set of characteristics S_r.

4.6.2 A Naive Search Algorithm

In order to explain the underlying ideas of the algorithm, we start with a very naive version, which we will improve in a step-wise manner.

All algorithms described in this section rely on the assumption that we can easily construct approximations for a single round of the block cipher, and more in particular, that we can efficiently enumerate pairs of masks $(\Gamma_{X_{i-1}}, \Gamma_{X_i})$ in decreasing order of $C(\Gamma_{X_{i-1}}, \Gamma_{X_i})$, with $\Gamma_{X_{i-1}}$ either fixed or not. Considering the fact that a single round of a block cipher is typically rather simple, this is a very reasonable assumption. This leads to the following straightforward depth-first search algorithm:

fill S_r with m arbitrary r-round characteristics $(\Gamma_{X_0}, \ldots, \Gamma_{X_r})$
for all $(\Gamma_{X_0}, \Gamma_{X_1})$ in decreasing order of $C(\Gamma_{X_0}, \Gamma_{X_1})$ **do**
  call naive-search(r, $(\Gamma_{X_0}, \Gamma_{X_1})$)
  if this call returns 'abort', **break** the loop
end for

The algorithm calls a procedure 'naive-search', which is defined as follows:

[8] In a typical block cipher where keys are XORed to the data in each round, there exists at most one mask Γ_{K_i} which results in a non-zero imbalance for given $\Gamma_{X_{i-1}}$ and Γ_{X_i}. Hence, there is no need to include key masks in the characteristic.

procedure naive-search(r, $(\Gamma_{X_0}, \ldots, \Gamma_{X_i})$)
  if $C(\Gamma_{X_0}, \ldots, \Gamma_{X_i}) \le \min C(S_r)$ **then**
    return 'abort'
  else
    if $i = r$ **then**
      replace smallest element of S_r by $(\Gamma_{X_0}, \ldots, \Gamma_{X_r})$
    else
      for all $\Gamma_{X_{i+1}}$ in decreasing order of $C(\Gamma_{X_i}, \Gamma_{X_{i+1}})$ **do**
        call naive-search(r, $(\Gamma_{X_0}, \ldots, \Gamma_{X_{i+1}})$)
        if this call returns 'abort', **break** the loop
      end for
    end if
    return 'continue'
  end if

When the algorithm (eventually) returns, the m arbitrary characteristics of S_r will have been replaced by the m most biased characteristics.

4.6.3 A Much Faster Algorithm

The naive algorithm described above needs to search deeply into a large tree before it can conclude that the current sequence of masks will not improve on any of the characteristics already in the set S_r. The main idea of Matsui's search algorithm is to first compute the best n-round characteristics for $n < r$, and then to use this information when searching for r-round characteristics to decide much earlier whether or not it is worth to continue following a certain branch in the three. Based on this idea, we modify the main loop as follows:

for $n = 2$ to r **do**
  fill S_n with m arbitrary n-round characteristics $(\Gamma_{X_0}, \ldots, \Gamma_{X_n})$
  for all $(\Gamma_{X_0}, \Gamma_{X_1})$ in decreasing order of $C(\Gamma_{X_0}, \Gamma_{X_1})$ **do**
    call faster-search(n, $(\Gamma_{X_0}, \Gamma_{X_1})$)
    if this call returns 'abort', **break** the loop
  end for
end for

Using the previous sets of characteristics, the 'naive-search' procedure can now be improved in two ways. First, the search can be aborted much earlier based on the observation that the imbalance of an r-round characteristic, starting with a sequence of masks $\Gamma_{X_0}, \ldots, \Gamma_{X_i}$, can never exceed $C(\Gamma_{X_0}, \ldots, \Gamma_{X_i}) \cdot \max C(S_{r-i})$. Secondly, we can exploit the fact that any $(r-i)$-round characteristic with an imbalance larger than $\min C(S_{r-i})$ must necessarily have been included in S_{r-i}. The following pseudo-code shows how this is done:

procedure faster-search($r, (\Gamma_{X_0}, \ldots, \Gamma_{X_i})$)
  if $C(\Gamma_{X_0}, \ldots, \Gamma_{X_i}) \cdot \max C(S_{r-i}) \leq \min C(S_r)$ **then**
    return 'abort'
  else
    if $i = r$ **then**
      replace smallest element of S_r by $(\Gamma_{X_0}, \ldots, \Gamma_{X_r})$
    else
      if $C(\Gamma_{X_0}, \ldots, \Gamma_{X_i}) \cdot \min C(S_{r-i}) \leq \min C(S_r)$ **then**
        for all $(\Gamma_{Y_0}, \ldots, \Gamma_{Y_{r-i}}) \in S_{r-i}$ in decr. ord. with $\Gamma_{Y_0} = \Gamma_{X_i}$ **do**
          call faster-search($s, (\Gamma_{X_0}, \ldots, \Gamma_{X_i}, \Gamma_{Y_1}, \ldots, \Gamma_{Y_{r-i}})$)
          if this call returns 'abort', **break** the loop
        end for
      else
        for all $\Gamma_{X_{i+1}}$ in decreasing order of $C(\Gamma_{X_i}, \Gamma_{X_{i+1}})$ **do**
          call faster-search($r, (\Gamma_{X_0}, \ldots, \Gamma_{X_{i+1}})$)
          if this call returns 'abort', **break** the loop
        end for
      end if
    end if
    return 'continue'
  end if

4.6.4 Exploiting Symmetries

In order to be able to prune the search tree as early as possible, it is important to quickly populate the set S_r with relatively good candidates such that $\min C(S_r)$ increases rapidly. This can be done by first trying to extend the best characteristics found for smaller number of rounds, before constructing them from scratch:

for $n = 2$ to r **do**
  fill S_n with m arbitrary characteristics $(\Gamma_{X_0}, \ldots, \Gamma_{X_n})$
  for $j = n - 1$ down to 2 **do**
    for all $(\Gamma_{X_0}, \ldots, \Gamma_{X_j}) \in S_j$ in decreasing order **do**
      call symmetric-search($n, j, (\Gamma_{X_0}, \ldots, \Gamma_{X_j})$)
      if this call returns 'abort', **break** the loop
    end for
  end for
  for all $(\Gamma_{X_0}, \Gamma_{X_1})$ in decreasing order of $C(\Gamma_{X_0}, \Gamma_{X_1})$ **do**
    call symmetric-search($n, j, (\Gamma_{X_0}, \Gamma_{X_1})$)
    if this call returns 'abort', **break** the loop
  end for
end for

Note that in order to avoid checking a lot of characteristics twice, we must add an additional condition in the second loop of 'faster-search', as done in the code below.

Finally, we can make some additional optimizations if the block cipher has a symmetric structure (e.g., a Feistel structure), such that each characteristic $(\Gamma_{X_0}, \ldots, \Gamma_{X_r})$ has its reversed counterpart $(\Gamma_{X_r}, \ldots, \Gamma_{X_0})$ with the exact same imbalance. The reasoning behind the optimization is that there is no need to construct characteristics ending with an element of S_{r-i} in the first loop of 'faster-search', if we have already checked all characteristics *starting* with those elements before. This is implemented in the procedure 'symmetric-search' below.

procedure symmetric-search($s, j, (\Gamma_{X_0}, \ldots, \Gamma_{X_i})$)
  if $C(\Gamma_{X_0}, \ldots, \Gamma_{X_i}) \cdot \max C(S_{r-i}) \leq \min C(S_r)$ **then**
    return 'abort'
  else
    if $i = r$ **then**
      replace smallest element(s) of S_r by $(\Gamma_{X_0}, \ldots, \Gamma_{X_r})$ and $(\Gamma_{X_r}, \ldots, \Gamma_{X_0})$
    else
      if $C(\Gamma_{X_0}, \ldots, \Gamma_{X_i}) \cdot \min C(S_{r-i}) \leq \min C(S_r)$ **then**
        if $r - i \leq j$ **then**
          for all $(\Gamma_{Y_0}, \ldots, \Gamma_{Y_{r-i}}) \in S_{r-i}$ in decr. ord. with $\Gamma_{Y_0} = \Gamma_{X_i}$ **do**
            call symmetric-search($s, j, (\Gamma_{X_0}, \ldots, \Gamma_{X_i}, \Gamma_{Y_1}, \ldots, \Gamma_{Y_{r-i}})$)
            if this call returns 'abort', **break** the loop
          end for
        end if
      else
        for all $\Gamma_{X_{i+1}}$ in decreasing order of $C(\Gamma_{X_i}, \Gamma_{X_{i+1}})$ **do**
          if $C(\Gamma_{X_0}, \ldots, \Gamma_{X_{i+1}}) \leq \min C(S_{i+1})$ **then**
            call symmetric-search($r, j, (\Gamma_{X_0}, \ldots, \Gamma_{X_{i+1}})$)
            if this call returns 'abort', **break** the loop
          end if
        end for
      end if
    end if
    return 'continue'
  end if

4.7 Conclusions

In this chapter, we have shown that a Maximum Likelihood approach provides an intuitive framework to deal with multiple linear approximations in an optimal way. A theoretically interesting feature of this framework is that

it allows to treat Matsui's Algorithm 1 and 2 in a very similar fashion. We have extended both attacks and shown that their behavior can easily and accurately be predicted. In order to illustrate this, we have conducted experiment on round-reduced variants of DES, and have discussed possibilities for an improvement of the best attacks on full (16-round) DES.

Chapter 5

Classification of S-Boxes

In Chaps. 3 and 4, we discussed the importance of characteristics in differential and linear cryptanalysis. We saw in Sect. 3.2 that the probabilities of these characteristics are typically determined by two factors: (1) the number of so-called *active* S-boxes, and (2) the differential or linear properties of these S-boxes. In this chapter, we concentrate on the second point, and propose a tool for classifying S-boxes which, while being of independent interest, is also useful for selecting good S-boxes,

5.1 Motivation

When discussing the structure of block ciphers in Sect. 2.1.4, we pointed out that they can typically be decomposed into building blocks of two different types: small non-linear components, called S-boxes, which *confuse* by introducing non-linearity on word level, and large linear diffusion layers, which *diffuse* this non-linearity over the complete block. Both components are interleaved with key addition layers, which mix in secret round keys, and the result would typically look like the SP network depicted in Fig. 2.4.

An interesting observation that could be made from this figure, is that the decomposition of a given block cipher into this fixed structure of non-linear S-boxes and linear diffusion layers is not unique. Suppose for instance that the output word of an S-box would be linearly transformed before it is fed into the diffusion layer. In that case, we could consider this linear transform to be part of the S-box, but we could just as well incorporate it in the diffusion layer. Similarly, additions with constants can be moved back and forth between S-boxes and round keys (by modifying the key schedule). In general, it is easy to see that any affine mapping at the input or the output of an S-box can be moved across the borders of the S-box.

The conclusion we draw from the observation above, is that if we intend to study S-boxes independently of the diffusion layer, then it only makes sense

to consider them up to an affine mapping at their input and output. This is exactly what we will do in this chapter. We will first present different algorithms to detect linear and affine equivalences between S-boxes, and then apply these tools to classify the complete set of 4×4-bit S-boxes into equivalence classes.

5.2 The Linear Equivalence Algorithm (LE)

The problem we try to solve in this section is the following: given two $n \times n$-bit invertible S-boxes S_1 and S_2, decide whether or not there exists a pair of *linear* mappings A and B such that

$$S_1(x) = B^{-1} \cdot S_2(A \cdot x)\,, \quad \forall x \in \{0,1\}^n\,.$$

If such mappings exist, the S-boxes are said to be *linearly equivalent*.

A naive approach would be to guess one of the mappings, say A, and then to compute the other one using the equation $B = S_2 \circ A \circ S_1^{-1}$. If B turns out to be linear as well, then we have found a solution. If not, we try again with a different guess. In the worst case, S_1 and S_2 are not equivalent, and we will have to try out all $\mathcal{O}(2^{n^2})$ possible linear mappings A to come to this conclusion.

We now present a simple and much more efficient algorithm, which, as turned out after we completed our work, was also considered by Patarin et al. [90] in the context of isomorphisms of polynomials. The idea of the algorithm is to guess the linear mapping A for as few input points as possible, and then use the linearity of A and B to follow the implications of these guesses as far as possible. But before we describe the algorithm, let us first introduce some notation to conveniently represent operations on a set of n-bit words W:

$$f(W) = \{f(x) \mid x \in W\}\,,$$
$$W \oplus c = \{x \oplus c \mid x \in W\}\,.$$

The linear equivalence algorithm can be expressed in terms of a number of special sets which are defined below and are updated in the successive steps of the algorithm. The relations between these sets are also illustrated in Fig. 5.1.

Sets C_A and C_B. At any given stage, these sets contain points whose images under the linear mapping A (or B, respectively) have been determined and are known not to violate the linearity of B (or A). By construction, they will always form a closed linear space, i.e., all linear combinations of points in these sets are again elements of the sets. The goal is to expand these sets until they cover the complete set $\{0,1\}^n$.

Sets N_A and N_B. These sets contain new points whose images under A (or B) could be derived using the expression $A = S_2^{-1} \circ B \circ S_1$ (or $B = S_2 \circ A \circ S_1^{-1}$), but which have not been verified to be consistent yet. The sets can be defined as follows:

$$N_A = S_1^{-1}(C_B) \setminus C_A\,,$$
$$N_B = S_1(C_A) \setminus C_B\,.$$

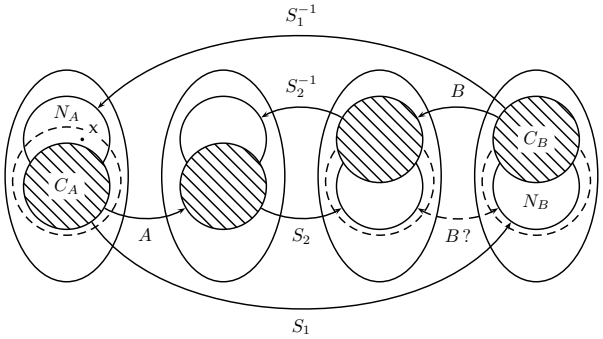


Figure 5.1: The relations between the different sets for the LE algorithm

In order to verify that the images of the points of N_A under A do not violate the linearity of B, and can therefore be moved to C_A, we apply the procedure described by the pseudo-code below:

procedure check-N_A
  while $N_A \neq \varnothing$ **do**
    pick $x \in N_A$
    for all $z \in (C_A \oplus x) \setminus N_A$ **do**
      if $B[S_1(z)] \leftarrow S_2 \cdot A(z)$ violates the linearity of B **then**
        return 'abort'
      end if
    end for
    $C_A \leftarrow C_A \cup (C_A \oplus x)$
    update N_A and N_B according to their definitions
  end while

Each time the image of a new point x is added to the partially determined linear mapping B (we write this as $B[x] \leftarrow y$), we need to verify, by a simple Gaussian elimination, whether it is consistent with the previous points.

Unless the procedure aborts because of an inconsistency, it will only return when all points of N_A have been verified and moved to C_A. During the process, however, many new points will have been added to N_B. For these points we can run a similar procedure 'check-N_B', which will on its turn generate

many new elements in N_A, etc. Eventually, assuming that the S-boxes are indeed equivalent, all points will have been covered, and the mappings A and B will be completely recovered.

In order to initiate this amplification process, we need either $N_A \neq \varnothing$ or $N_B \neq \varnothing$, and at least one non-zero element in C_A or C_B, respectively. When the algorithm starts, we have $C_A = C_B = \{0\}$, and, unless the S-boxes map 0 to itself, $|N_A| = |N_B| = 1$. In this case, guessing the image of one additional point will usually suffice to bootstrap the process, and hence the algorithm will terminate within $\mathcal{O}(2^n)$ steps. If the S-boxes do map 0 on itself, we will need to make at least two guesses and the complexity increases to $\mathcal{O}(2^{2n})$.

The complete linear equivalence algorithm is summarized, in a recursive form, in the code below. In order to illustrate the amplification process described above, we also give a concrete example in Fig. 5.2 for two 4×4-bit S-boxes.

procedure LE
  while $N_A \neq \varnothing$ or $N_A \neq \varnothing$ **do**
    call check-N_A
    call check-N_B
    if any of these calls returns 'abort' **then**
      return
    end if
  end while
  if $C_A \neq \{0,1\}^n$ **then**
    pick $x \notin C_A$ and set $N_A \leftarrow \{x\}$
    for all $y \notin A(C_A)$ **do**
      $A[x] \leftarrow y$
      call LE
    end for
  else
    return A and B and **exit**
  end if

5.3 The Affine Equivalence Algorithm (AE)

In this section, we generalize the equivalence problem to the affine case. More specifically, we search for an algorithm, which again takes two $n \times n$-bit S-boxes S_1 and S_2 as input, but this time checks whether there exists a pair of invertible *affine* mappings A_a and B_a such that $B_a^{-1} \circ S_1 \circ A_a = S_2$. Each of these affine mappings can be expressed as a linear transform followed by an addition, which leads to an *affine equivalence* relation of the form

$$S_1(x) = B^{-1} \cdot S_2(A \cdot x \oplus a) \oplus b\,, \quad \forall x \in \{0,1\}^n\,,$$

with A and B invertible $n \times n$-bit linear mappings, and a and b n-bit constants.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_1(x)$	1	B	9	C	D	6	F	3	E	8	7	4	A	2	5	0
x'	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_2(x')$	3	D	A	4	8	F	E	B	0	1	5	9	2	6	C	7

$$x \xrightarrow{A} x' \xrightarrow{S_1,S_2} y' \xleftarrow{B} y$$

$$0 \to 0 \xrightarrow{S_1,S_2} 3 \leftrightarrow 1$$

$$\mathtt{F} \leftrightarrow 8 \xleftarrow{S_1^{-1},S_2^{-1}} 0 \leftarrow 0$$

$$\text{guess} \to 1 \to \underline{1} \xrightarrow{S_1,S_2} \mathtt{D} \leftrightarrow \mathtt{B}$$

$$\mathtt{E} \to 9 \xrightarrow{S_1,S_2} 1 \leftrightarrow 5$$

$$\mathtt{C} \leftrightarrow 6 \xleftarrow{S_1^{-1},S_2^{-1}} \mathtt{E} \leftarrow \mathtt{A}$$

$$\mathtt{B} \leftrightarrow \mathtt{C} \xleftarrow{S_1^{-1},S_2^{-1}} 2 \leftarrow 4$$

$$\text{inconsistent} \to 8 \leftrightarrow \mathtt{E} \xleftarrow{S_1^{-1},S_2^{-1}} \mathtt{C} \leftarrow \mathtt{E}$$

Figure 5.2: The LE algorithm in action

5.3.1 Basic Algorithm

Considering that the problem is very similar to the linear equivalence problem, it seems natural to try to reuse the linear algorithm described above as a subroutine. A straightforward solution would be:

for all a **do**
  for all b **do**
    check whether $S_1(\cdot) \oplus b$ and $S_2(\cdot \oplus a)$ are linearly equivalent
  end for
end for

This approach adds a factor 2^{2n} to the complexity of the linear algorithm, bringing the total to $\mathcal{O}(2^{3n})$. The reason why this algorithm is rather inefficient is clearly the fact that the linear equivalence needs to be checked for each individual pair (a, b). The alternative approach presented in this section will try to avoid this by assigning a unique representative to each linear equivalence class. Indeed, if we could find an efficient method to identify such a representative for a given permutation, then we could check for affine equivalence using the following, much more efficient, algorithm:

for all b **do**
  insert the linear representative of $S_1(\cdot) \oplus b$ in table T_1
end for
for all a **do**
  insert the linear representative of $S_2(\cdot \oplus a)$ in table T_2
end for
if $T_1 \cap T_2 \neq \varnothing$ **then**
  conclude that S_1 and S_2 are affine equivalent
end if

The complexity of this alternative algorithm is about 2^n times the work needed for finding the linear representative. If the latter requires less than $\mathcal{O}(2^{2n})$, then the second approach will outperform the first. In the next section, we will see that this is indeed the case, and present an algorithm which constructs representatives in $\mathcal{O}(2^n)$. As a result, the total complexity of finding affine equivalences is brought down to $\mathcal{O}(2^{2n})$.

A particularly interesting property of this approach is that it can efficiently solve the problem of finding mutual equivalences in a large set of S-boxes. Due to the fact that the main part of the computation is performed separately for each S-box, the complexity will grow only linearly with the number of S-boxes (and not with the number of possible pairs). We will exploit this property in Sect. 5.4.2.

5.3.2 Finding the Linear Representative

The efficiency of an algorithm that finds the linear representative R_S for an S-box S depends on how this unique representative is chosen. In this chapter, we decide to define it as follows:

Definition 5.1. The representative R_S of an S-box S is the lexicographically smallest S-box in the linear equivalence class containing S.

The lexicographically ordering in this definition refers to the lookup tables of the S-boxes, i.e., the smallest S-box is the identity, and for example, S-box $[0, 1, 3, 4, 7, 2, 6, 5]$ is smaller than the S-box $[0, 2, 1, 6, 7, 4, 3, 5]$.

In order to construct the representative R_S of the linear class containing a given S-box S, we use an algorithm which is based on the same principles as the algorithm in Sect. 5.2: after making an initial guess, we incrementally build the linear mappings A and B such that $R'_S = B^{-1} \circ S \circ A$ is as small as possible. This is repeated for each guess, and the representative R_S is obtained by taking the smallest R'_S over all guesses.

In the description of the algorithm below, we will refer to the same sets C_A, C_B, N_A, and N_B as in Sect. 5.2. We define them slightly differently, though, and we also introduce a new pair of sets D_A and D_B. Fig. 5.3 illustrates the new relations between the different sets.

Sets D_A and D_B. These sets contain points for which the linear mapping A (or B, respectively) is determined. By construction, they will always be of the form $\{x \mid 0 \leq x < 2^m\}$.

Sets C_A and C_B. These sets contain the points of D_A which have a corresponding point in D_B and vice versa, i.e., $S \circ A\,(C_A) = B\,(C_B)$. The sets are constructed as follows:

$$C_A = A^{-1}[A(D_A) \cap S^{-1} \circ B(D_B)]\,,$$
$$C_B = B^{-1}[B(D_B) \cap S \circ A(D_A)]\,.$$

Note that for all points in C_A and C_B, we can compute $R'_S(x) = B^{-1} \circ S \circ A(x)$ and ${R'_S}^{-1}(y) = A^{-1} \circ S^{-1} \circ B(y)$, respectively.

Sets N_A and N_B. These sets contain the remaining points of D_A and D_B, i.e.,

$$N_A = D_A \setminus C_A\,,$$
$$N_B = D_B \setminus C_B\,.$$

The main part of the algorithm for finding a candidate R'_S consists in repeatedly picking the smallest input x for which R'_S is not known, and trying to assign it to the smallest available output y. As long as N_A contains elements, this can be done by following the procedure below:

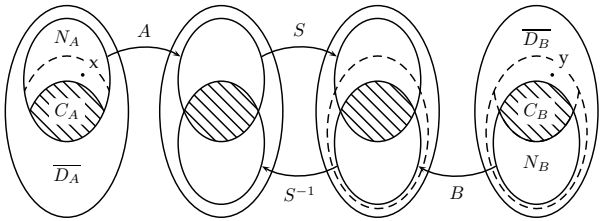


Figure 5.3: The relations between the different sets for the LR algorithm

while $N_A \neq \varnothing$ **do**
  pick $x = \min(\overline{C_A}) = \min(N_A)$ and $y = \min(\overline{D_B}) = |D_B|$
  assign $B[y] \leftarrow S \circ A(x)$, such that $R'_S(x) = y$
  $D_B \leftarrow D_B \oplus y$
  update C_A, C_B, N_A, and N_B according to their definitions
  while $N_A = \varnothing$ and $N_B \neq \varnothing$ **do**
    pick $x = \min(\overline{C_A}) = \min(\overline{D_A}) = |D_A|$ and $y = \min(\overline{C_B}) = \min(N_B)$
    assign $A[x] \leftarrow S^{-1} \circ B(y)$, such that $R'_S(x) = y$
    $D_A \leftarrow D_A \oplus x$
    update C_A, C_B, N_A, and N_B according to their definitions
  end while
end while

When this algorithm finishes, N_A and N_B are both empty. If at this point C_A covers the complete set $\{0,1\}^n$, then R'_S is completely defined. In the opposite case, we need to guess A for the smallest point of $\overline{C_A}$ (which by construction will be $x = |C_A| = |D_A|$). This will add new elements to D_A and thus to N_A, such that we can apply the algorithm once again. In order to be sure to find the smallest representative, we must repeat this for each possible guess.

In most cases, we will only need to guess A for a single point, which means that about 2^n possibilities have to be checked. Completely defining R'_S for a particular guess takes about 2^n steps. However, most guesses will already be rejected after having determined only slightly more than n values, because at that point R'_S will usually already turn out to be larger than the current smallest candidate. Due to this, the total complexity of finding the representative is expected to be $\mathcal{O}(2^n)$.

In order to make the process described above a bit more concrete, we provide a 4×4-bit example in Fig. 5.4.

5.3.3 An Alternative Approach using the Birthday Paradox

The efficiency gain obtained in the previous subsections arises from the fact that the computation is split into two parts, each of which depends on a single

x'	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x')$	1	B	9	C	D	6	F	3	E	8	7	4	A	2	5	0

$$x \xrightarrow{A} x' \xrightarrow{S} y' \xleftarrow{B} y$$

$$0 \rightarrow 0 \xrightarrow{S} 1 \leftrightarrow 1$$
$$1 \leftrightarrow \mathtt{F} \xleftarrow{S^{-1}} 0 \leftarrow 0$$
$$\text{guess} \rightarrow 2 \rightarrow \underline{1} \xrightarrow{S} \mathtt{B} \leftrightarrow 2$$
$$3 \rightarrow \mathtt{E} \xrightarrow{S} 5 \leftrightarrow 4$$
$$4 \leftrightarrow \mathtt{C} \xleftarrow{S^{-1}} \mathtt{A} \leftarrow 3$$
$$5 \rightarrow 3 \xrightarrow{S} \mathtt{C} \leftrightarrow 8$$
$$6 \rightarrow \mathtt{D} \xrightarrow{S} 2 \rightarrow \mathtt{E}$$

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$R'_S(x)$	1	0	2	4	3	8	E									

Figure 5.4: Finding the linear representative

S-box S_1 or S_2 only. In this section, we apply the same idea in a different way and present a second algorithm which is directly based on the birthday method from [90].

First, let us assume that S_1 and S_2 are indeed equivalent, i.e., $S_1 = B_a^{-1} \circ S_2 \circ A_a$, with A_a and B_a two (unknown) affine mappings. Suppose now that we are given two triples $\mathbf{x} = (x_1, x_2, x_3)$ and $\mathbf{x}' = (x'_1, x'_2, x'_3)$, and are asked to determine whether $x'_i = A_a(x_i)$ for $1 \leq i \leq 3$. Note that if this is the case, then the triples $\mathbf{y} = (y_1, y_2, y_3)$ and $\mathbf{y}' = (y'_1, y'_2, y'_3)$, with $y_i = S_1(x_i)$ and $y'_i = S_2(x'_i)$, must also satisfy $y'_i = B_a(y_i)$. In order to verify this, we first expand the tuples $\mathbf{x}$ and $\mathbf{y}$ by adding points x_i and $y_i = S_1(x_i)$ where either x_i or y_i is an odd linear combination of previous elements of $\mathbf{x}$ or $\mathbf{y}$. An example is given in Fig. 5.5. If we now perform the exact same operations on $\mathbf{x}'$ and $\mathbf{y}'$, but using S_2 instead of S_1, then all new points will still be related by $x'_i = A_a(x_i)$ and $y'_i = B_a(y_i)$. As soon as the expanded triples $\mathbf{x}$ and $\mathbf{x}'$ contain more than $n+1$ linearly independent points, we can directly compute A_a, and verify that this mapping holds for all other points. If it does, the same procedure can be applied to recover and verify B_a from $\mathbf{y}$ and $\mathbf{y}'$.

Based on the procedure above, we can now construct an efficient probabilistic algorithm for finding affine equivalences, given that they exist. We start by generating two sets of about $2^{3n/2}$ random triples $\mathbf{x}_j$ and $\mathbf{x}'_j$. If A_a exists, it is likely, because of the birthday paradox, that these sets will contain at least one pair of triples which are related by A_a (we call this a collision). In principle, we could find these collisions by checking each individual pair. This would however require to apply the procedure described earlier about 2^{3n} times. Fortunately, most of these checks can be avoided based on the observation that any linear relation between an even number of elements of one expanded triple must necessarily also hold for the other expanded triple, if both are related by A_a. Hence, it suffices to expand all triples, find all even linear relations for each of them, and only check pairs with identical relations. Examples of such relations are marked with arrows in Fig. 5.5.

How far the triples should be expanded is a trade-off between the work needed for the expansion, and the work saved by reducing the number of pairs to check, but eventually, the algorithm is not expected to require more than $\mathcal{O}(2^{3 \cdot n/2})$ computations. Note that this algorithm is probabilistic (it will fail if no collisions occur), though its success probability can easily be increased by considering a larger number of random sets. Still, it cannot be used to determine with certainty that two S-boxes are not equivalent, and this is an important difference with the previous deterministic algorithms.

5.4 Classification of S-Boxes

Using the tools described in the previous section, it is now relatively straightforward to construct an algorithm which achieves our original goal: classifying S-boxes into equivalence classes. But first, let us examine how many

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

$$
\begin{array}{rcl}
x & \xrightarrow{S} & y \\
x_0 = 0 & \xrightarrow{S} & y_0 = 1 \\
x_1 = 1 & \xrightarrow{S} & y_1 = B \\
x_2 = 2 & \xrightarrow{S} & y_2 = 9 \\
x_3 = 7 & \xleftarrow{S^{-1}} & y_3 = y_0 + y_1 + y_2 = 3 \\
x_4 = x_0 + x_1 + x_2 = 3 & \xrightarrow{S} & y_4 = C \\
x_5 = x_0 + x_1 + x_3 = 6 & \xrightarrow{S} & y_5 = F(= x_0 + x_1 + x_2 + x_4) \\
x_6 = x_0 + x_2 + x_3 = 5 & \xrightarrow{S} & y_6 = 6(= x_0 + x_1 + x_4) \leftarrow \\
x_7 = x_1 + x_2 + x_3 = 4 & \xrightarrow{S} & y_7 = D(= x_1 + x_2 + x_5) \leftarrow \\
x_8 = E & \xleftarrow{S^{-1}} & y_8 = y_0 + y_1 + y_5 = 5 \\
\rightarrow (x_2 + x_3 + x_8 =) x_9 = B & \xleftarrow{S^{-1}} & y_9 = y_0 + y_2 + y_4 = 4
\end{array}
$$

Figure 5.5: Expanding a triple (x_1, x_2, x_3)

Table 5.1: Number of linear and affine equivalence classes of permutations

Dimension	1	2	3	4
Permutations	2	24	40 320	20 922 789 888 000
Lin. Eq. Classes	2	2	10	52 246
Aff. Eq. Classes	1	1	4	302

Dimension	5
Permutations	263 130 836 933 693 530 167 218 012 160 000 000
Lin. Eq. Classes	2 631 645 209 645 100 680 144
Aff. Eq. Classes	2 569 966 041 123 963 092

different equivalence classes we expect for a given dimension n.

5.4.1 The Number of Equivalence Classes

Counting linear and affine equivalence classes is an interesting problem in itself. It was solved in the 1960s by Lorens [71] and Harrison [48] using Polya theory. The solution was based on the computation of the cycle index polynomial of the linear and affine groups, and results were given for $n \leq 5$. During the research that led to this thesis, we independently implemented a similar algorithm, counted the number of equivalence classes for larger n, and verified that this number is very well approximated by $2^n!/|G|^2$, where $|G|$ is the size of the linear or affine group.

A quick look at the figures[1] in Table 5.1 shows that the number of equivalence classes grows very quickly with the dimension n. The number of classes is too small to be really interesting for $n = 3$, and many orders of magnitude too large to be enumerated for $n = 5$. We will therefore concentrate on the case $n = 4$. Fortunately, this case is also relevant in practice, since 4×4-bit S-boxes are frequently used in concrete designs.

5.4.2 Classification of 4×4-bit S-boxes

We now present a very simple algorithm to exhaustively enumerate all affine equivalence classes. In principle, it applies to any dimension n, but as we just saw, it is only practical for $n \leq 4$. The algorithm is based on two simple lemmas.

Lemma 5.1. *All invertible S-boxes can be written as a composition of transpositions.*

Lemma 5.2. *If S_1 and S_2 only differ by a transposition $(u\,v)$, i.e., $S_2 = S_1 \circ (u\,v)$, then, for any S_1' in the equivalence class of S_1, there exists an S_2' in the equivalence class of S_2 such that S_1' and S_2' only differ by a single transposition.*

[1] Note that the number of linear equivalence classes for $n = 5$ mentioned in [71] differs by 2 from what we computed in Table 5.1. We suspect this to be an error.

The basic idea of the algorithm is to construct a list of equivalence classes by iteratively adding the equivalence classes of all S-boxes that can be reached within one transposition from the S-boxes contained in the equivalence classes already included in the list. We start from the identity, and repeat until no new equivalence classes appear. Since every S-box can be reached from the identity by applying successive transpositions (Lemma 5.1), we are sure that all equivalence classes will eventually be covered. On the other hand, Lemma 5.2 tells us that, for every equivalence class, it suffices to check the S-boxes that can be reached from a single element from this class. The complete algorithm is summarized below:

add the identity to the queue Q
while the queue Q is not empty **do**
  take the first S-box S from the queue Q
  for all b **do**
    if the linear representative of $S(x) \oplus b$ is included in table T **then**
      break and continue with the next element in the queue
    end if
  end for
  add S as a representative to the list of equivalence classes
  for all a **do**
    insert the linear representative of $S(\cdot \oplus a)$ in table T
  end for
  for all transpositions $(u\,v)$ **do**
    insert $S \circ (u\,v)$ into the queue Q
  end for
end while

The result of running this algorithm for $n = 4$ is a list of 302 equivalence classes which is graphically represented in Fig. 5.6. The node at the top corresponds to the equivalence class containing the identity (and all affine permutations). The connections between the nodes represent single transpositions. The graph shows that every S-box can be turned into an affine permutation by applying at most 8 transpositions. Two interesting nodes which stand out are the ones on the sixth level at the extreme right, and on the eighth level at the extreme left of the graph. The first node (class 3 in Table 5.2) turns out to be equivalent to the inverse over $GF(2^4)$, the second (class 293 in Table 5.7) is equivalent to both the exponentiation and the logarithm over $GF(2^4)$.

The 302 equivalent classes are listed in Tables 5.2–5.8. Each row contains the hexadecimal lookup table of a representative of the class, and the distribution of the values in the linear approximation and difference distribution tables (which is invariant over all S-boxes in the class). The classes are sorted according to the largest non-trivial imbalance or differential probability in these tables.

Table 5.2: Affine equivalence classes 1–50

	Representative	$\lvert c\rvert = 1/4$	1/2	3/4	1	$p = 1/8$	1/4	3/8	1/2	5/8	3/4	7/8	1
1	401F2B6739A5CDE8	120	30	0	1	90	15	0	0	0	0	0	1
2	801C256943AB7DEF	120	30	0	1	90	15	0	0	0	0	0	1
3	801CF56743AB9DE2	120	30	0	1	90	15	0	0	0	0	0	1
4	20183D6749A5CBEF	120	30	0	1	90	15	0	0	0	0	0	1
5	20183F67495BCDEA	120	30	0	1	90	15	0	0	0	0	0	1
6	20183B6749AFCDE5	120	30	0	1	90	15	0	0	0	0	0	1
7	48123B6709AECD5F	120	30	0	1	90	15	0	0	0	0	0	1
8	801925D746ABC3EF	120	30	0	1	90	15	0	0	0	0	0	1
9	8E1235674CAB9D0F	112	32	0	1	84	18	0	0	0	0	0	1
10	8E12356749FBCD0A	112	32	0	1	84	18	0	0	0	0	0	1
11	8F1235C749AB6DE0	112	32	0	1	84	18	0	0	0	0	0	1
12	8F12356D49ABC7E0	112	32	0	1	84	18	0	0	0	0	0	1
13	C019354762AB8DEF	96	36	0	1	72	24	0	0	0	0	0	1
14	CB12354769A08DEF	96	36	0	1	72	24	0	0	0	0	0	1
15	C912354760AB8DEF	96	36	0	1	72	24	0	0	0	0	0	1
16	8E12354769A0CDBF	96	36	0	1	72	24	0	0	0	0	0	1
17	8A123B674905CDEF	120	30	0	1	93	12	1	0	0	0	0	1
18	40182C3769AB5DEF	112	32	0	1	87	15	1	0	0	0	0	1
19	4018253B69A7CDEF	112	32	0	1	87	15	1	0	0	0	0	1
20	4018253E69ABCD7F	112	32	0	1	87	15	1	0	0	0	0	1
21	6812354B09A7CDEF	112	32	0	1	87	15	1	0	0	0	0	1
22	4018293765ABCDEF	112	32	0	1	87	15	1	0	0	0	0	1
23	8F123567490BCDEA	120	30	0	1	96	9	2	0	0	0	0	1
24	401825F769ABCDE3	120	30	0	1	96	9	2	0	0	0	0	1
25	D01235E769ABC84F	96	36	0	1	78	18	2	0	0	0	0	1
26	CA123547690B8DEF	96	36	0	1	78	18	2	0	0	0	0	1
27	C01235F769AB8DE4	96	36	0	1	78	18	2	0	0	0	0	1
28	8A123047695BCDEF	96	36	0	1	78	18	2	0	0	0	0	1
29	8D12304769ABC5EF	96	36	0	1	78	18	2	0	0	0	0	1
30	681E354709ABCD2F	96	36	0	1	81	15	3	0	0	0	0	1
31	401825A7693BCDEF	96	36	0	1	80	18	0	1	0	0	0	1
32	C0A23547691B8DEF	64	44	0	1	64	24	0	2	0	0	0	1
33	D0A23547691BC8EF	64	44	0	1	64	24	0	2	0	0	0	1
34	8F123C6749AB5DE0	119	28	1	1	78	21	0	0	0	0	0	1
35	801C25B749A63DEF	119	28	1	1	78	21	0	0	0	0	0	1
36	801C25D749AB36EF	119	28	1	1	78	21	0	0	0	0	0	1
37	C0123F6749AB8DE5	111	30	1	1	72	24	0	0	0	0	0	1
38	C012356D49AB87EF	111	30	1	1	72	24	0	0	0	0	0	1
39	8A123569470BCDEF	119	28	1	1	81	18	1	0	0	0	0	1
40	801C2E6749AB3D5F	119	28	1	1	81	18	1	0	0	0	0	1
41	40182D3769ABC5EF	119	28	1	1	81	18	1	0	0	0	0	1
42	40182A37695BCDEF	119	28	1	1	81	18	1	0	0	0	0	1
43	6812354C09AB7DEF	111	30	1	1	75	21	1	0	0	0	0	1
44	6812354D09ABC7EF	111	30	1	1	75	21	1	0	0	0	0	1
45	6812354907ABCDEF	111	30	1	1	75	21	1	0	0	0	0	1
46	DA123567490BC8EF	119	28	1	1	84	15	2	0	0	0	0	1
47	E01A3567492BCD8F	119	28	1	1	84	15	2	0	0	0	0	1
48	8A123597460BCDEF	119	28	1	1	84	15	2	0	0	0	0	1
49	801D2567493BCAEF	119	28	1	1	84	15	2	0	0	0	0	1
50	801625D749ABC3EF	119	28	1	1	84	15	2	0	0	0	0	1

Table 5.3: Affine equivalence classes 51–100

	Representative	$\lvert c\rvert = 1/4$	1/2	3/4	1	$p = 1/8$	1/4	3/8	1/2	5/8	3/4	7/8	1
51	8C12354769AB0DEF	111	30	1	1	78	18	2	0	0	0	0	1
52	68D2354709ABC1EF	111	30	1	1	78	18	2	0	0	0	0	1
53	68E2354709ABCD1F	111	30	1	1	78	18	2	0	0	0	0	1
54	401F253769ABCDE8	111	30	1	1	78	18	2	0	0	0	0	1
55	401A2867395BCDEF	111	30	1	1	78	18	2	0	0	0	0	1
56	401A2568397BCDEF	111	30	1	1	78	18	2	0	0	0	0	1
57	608F354719ABCDE2	111	30	1	1	78	18	2	0	0	0	0	1
58	608C354719AB2DEF	111	30	1	1	78	18	2	0	0	0	0	1
59	681C354709AB2DEF	111	30	1	1	78	18	2	0	0	0	0	1
60	681D354709ABC2EF	111	30	1	1	78	18	2	0	0	0	0	1
61	801C256743AB9DEF	111	30	1	1	78	18	2	0	0	0	0	1
62	801A2563497BCDEF	111	30	1	1	78	18	2	0	0	0	0	1
63	801D253749ABC6EF	111	30	1	1	78	18	2	0	0	0	0	1
64	801A256749DBC3EF	111	30	1	1	78	18	2	0	0	0	0	1
65	801E354769ABCD2F	111	30	1	1	78	18	2	0	0	0	0	1
66	6018D54729ABC3EF	111	30	1	1	78	18	2	0	0	0	0	1
67	6812F54709ABCDE3	111	30	1	1	78	18	2	0	0	0	0	1
68	68123A47095BCDEF	111	30	1	1	78	18	2	0	0	0	0	1
69	80132A6749FBCDE5	111	30	1	1	78	18	2	0	0	0	0	1
70	401825B769A3CDEF	111	30	1	1	78	18	2	0	0	0	0	1
71	801625C749AB3DEF	111	30	1	1	78	18	2	0	0	0	0	1
72	8017256C49AB3DEF	111	30	1	1	78	18	2	0	0	0	0	1
73	4018259763ABCDEF	111	30	1	1	78	18	2	0	0	0	0	1
74	6812394705ABCDEF	111	30	1	1	78	18	2	0	0	0	0	1
75	6819354702ABCDEF	111	30	1	1	78	18	2	0	0	0	0	1
76	8019253746ABCDEF	111	30	1	1	78	18	2	0	0	0	0	1
77	801C253749AB6DEF	111	30	1	1	81	15	3	0	0	0	0	1
78	80132F67495BCDEA	111	30	1	1	81	15	3	0	0	0	0	1
79	6D12354709ABC8EF	95	34	1	1	72	18	4	0	0	0	0	1
80	8B12354769A0CDEF	95	34	1	1	72	18	4	0	0	0	0	1
81	8D12354769ABC0EF	95	34	1	1	72	18	4	0	0	0	0	1
82	68F2354709ABCDE1	95	34	1	1	72	18	4	0	0	0	0	1
83	68C2354709AB1DEF	95	34	1	1	72	18	4	0	0	0	0	1
84	8512304769ABCDEF	63	42	1	1	78	0	14	0	0	0	0	1
85	8A123547690BCDEF	111	30	1	1	80	18	0	1	0	0	0	1
86	681235A7094BCDEF	111	30	1	1	80	18	0	1	0	0	0	1
87	8A1C3567490B2DEF	118	26	2	1	72	21	2	0	0	0	0	1
88	801C256349AB7DEF	118	26	2	1	72	21	2	0	0	0	0	1
89	20183B6749A5CDEF	118	26	2	1	72	21	2	0	0	0	0	1
90	48123B6709A5CDEF	118	26	2	1	72	21	2	0	0	0	0	1
91	4812356907ABCDEF	118	26	2	1	72	21	2	0	0	0	0	1
92	8612950743ABCDEF	118	26	2	1	72	21	2	0	0	0	0	1
93	68123B4709A5CDEF	110	28	2	1	66	24	2	0	0	0	0	1
94	68123E4709ABCD5F	110	28	2	1	66	24	2	0	0	0	0	1
95	68123F4709ABCDE5	110	28	2	1	66	24	2	0	0	0	0	1
96	F0132C6749AB5DE8	118	26	2	1	75	18	3	0	0	0	0	1
97	8D12350749ABC6EF	118	26	2	1	75	18	3	0	0	0	0	1
98	401E2567398BCDAF	118	26	2	1	75	18	3	0	0	0	0	1
99	801F253749ABCDE6	118	26	2	1	75	18	3	0	0	0	0	1
100	2013FA67495BCDE8	118	26	2	1	75	18	3	0	0	0	0	1

Table 5.4: Affine equivalence classes 101–150

	Representative	$\lvert c\rvert = 1/4$	1/2	3/4	1	$p = 1/8$	1/4	3/8	1/2	5/8	3/4	7/8	1
101	20183A67495BCDEF	118	26	2	1	75	18	3	0	0	0	0	1
102	20183D6749ABC5EF	118	26	2	1	75	18	3	0	0	0	0	1
103	40182B6739A5CDEF	118	26	2	1	75	18	3	0	0	0	0	1
104	40182C6739AB5DEF	118	26	2	1	75	18	3	0	0	0	0	1
105	201835674CAB9DEF	118	26	2	1	75	18	3	0	0	0	0	1
106	8F12354769ABCDE0	110	28	2	1	69	21	3	0	0	0	0	1
107	801D354769ABC2EF	110	28	2	1	69	21	3	0	0	0	0	1
108	6812C54709AB3DEF	110	28	2	1	69	21	3	0	0	0	0	1
109	48123E6709ABCD5F	110	28	2	1	69	21	3	0	0	0	0	1
110	80123B6749A5CDEF	110	28	2	1	69	21	3	0	0	0	0	1
111	681235E709ABCD4F	110	28	2	1	69	21	3	0	0	0	0	1
112	4812356D09ABC7EF	110	28	2	1	69	21	3	0	0	0	0	1
113	481235670FABCDE9	110	28	2	1	69	21	3	0	0	0	0	1
114	4812356709AECDBF	110	28	2	1	69	21	3	0	0	0	0	1
115	8012356947ABCDEF	110	28	2	1	69	21	3	0	0	0	0	1
116	F01A2567493BCDE8	118	26	2	1	78	15	4	0	0	0	0	1
117	20183F6749ABCDE5	118	26	2	1	78	15	4	0	0	0	0	1
118	401825F739ABCDE6	118	26	2	1	78	15	4	0	0	0	0	1
119	681B354709A2CDEF	110	28	2	1	72	18	4	0	0	0	0	1
120	48123A67095BCDEF	110	28	2	1	72	18	4	0	0	0	0	1
121	80123F6749ABCDE5	110	28	2	1	72	18	4	0	0	0	0	1
122	481235B709A6CDEF	110	28	2	1	72	18	4	0	0	0	0	1
123	481235D709ABC6EF	110	28	2	1	72	18	4	0	0	0	0	1
124	4812356A097BCDEF	110	28	2	1	72	18	4	0	0	0	0	1
125	8012356D49ABC7EF	110	28	2	1	72	18	4	0	0	0	0	1
126	801235674CAB9DEF	110	28	2	1	72	18	4	0	0	0	0	1
127	4812356709FBCDEA	110	28	2	1	72	18	4	0	0	0	0	1
128	4812359706ABCDEF	110	28	2	1	72	18	4	0	0	0	0	1
129	8012356749AECDBF	110	28	2	1	72	18	4	0	0	0	0	1
130	401A2537698BCDEF	118	26	2	1	74	21	0	1	0	0	0	1
131	481235A7096BCDEF	118	26	2	1	77	18	1	1	0	0	0	1
132	80B2354769A1CDEF	110	28	2	1	71	21	1	1	0	0	0	1
133	801235B769A4CDEF	110	28	2	1	71	21	1	1	0	0	0	1
134	80C2354769AB1DEF	110	28	2	1	74	18	2	1	0	0	0	1
135	80D2354769ABC1EF	110	28	2	1	74	18	2	1	0	0	0	1
136	8012E54769ABCD3F	110	28	2	1	74	18	2	1	0	0	0	1
137	8012F54769ABCDE3	110	28	2	1	74	18	2	1	0	0	0	1
138	608235B719A4CDEF	110	28	2	1	74	18	2	1	0	0	0	1
139	801235C749AB6DEF	110	28	2	1	74	18	2	1	0	0	0	1
140	4018296735ABCDEF	110	28	2	1	74	18	2	1	0	0	0	1
141	8012356749FBCDEA	110	28	2	1	74	18	2	1	0	0	0	1
142	C012354769AB8DEF	94	32	2	1	62	24	2	1	0	0	0	1
143	D012354769ABC8EF	94	32	2	1	62	24	2	1	0	0	0	1
144	8E12354769ABCD0F	94	32	2	1	62	24	2	1	0	0	0	1
145	801F2567493BCDEA	94	32	2	1	62	24	2	1	0	0	0	1
146	801B354769A2CDEF	94	32	2	1	62	24	2	1	0	0	0	1
147	801C354769AB2DEF	94	32	2	1	62	24	2	1	0	0	0	1
148	6812B54709A3CDEF	94	32	2	1	62	24	2	1	0	0	0	1
149	6812D54709ABC3EF	94	32	2	1	62	24	2	1	0	0	0	1
150	801235F769ABCDE4	94	32	2	1	62	24	2	1	0	0	0	1

Table 5.5: Affine equivalence classes 151–200

	Representative	$\lvert c\rvert = 1/4$	1/2	3/4	1	$p = 1/8$	1/4	3/8	1/2	5/8	3/4	7/8	1
151	801625F749ABCDE3	94	32	2	1	62	24	2	1	0	0	0	1
152	4082356B19A7CDEF	94	32	2	1	62	24	2	1	0	0	0	1
153	4812356709ABEDCF	94	32	2	1	62	24	2	1	0	0	0	1
154	4812356709ABCFED	94	32	2	1	62	24	2	1	0	0	0	1
155	80F2354769ABCDE1	94	32	2	1	64	24	0	2	0	0	0	1
156	801925674FABCDE3	117	24	3	1	66	21	4	0	0	0	0	1
157	4018256739DBCAEF	117	24	3	1	66	21	4	0	0	0	0	1
158	B012356749ADC8EF	117	24	3	1	69	18	5	0	0	0	0	1
159	20183C6749AB5DEF	117	24	3	1	69	18	5	0	0	0	0	1
160	40182A67395BCDEF	117	24	3	1	69	18	5	0	0	0	0	1
161	201835674DABC9EF	117	24	3	1	69	18	5	0	0	0	0	1
162	4018259736ABCDEF	117	24	3	1	69	18	5	0	0	0	0	1
163	40823B6719A5CDEF	109	26	3	1	63	21	5	0	0	0	0	1
164	80123E6749ABCD5F	109	26	3	1	63	21	5	0	0	0	0	1
165	2013846E59ABCD7F	109	26	3	1	63	21	5	0	0	0	0	1
166	8012356C49AB7DEF	109	26	3	1	63	21	5	0	0	0	0	1
167	401825673CAB9DEF	109	26	3	1	63	21	5	0	0	0	0	1
168	801235674DABC9EF	109	26	3	1	63	21	5	0	0	0	0	1
169	4018256739AECDBF	109	26	3	1	63	21	5	0	0	0	0	1
170	8012356749AFCDEB	109	26	3	1	63	21	5	0	0	0	0	1
171	401F256739ABCDE8	117	24	3	1	72	15	6	0	0	0	0	1
172	8E12356749ABCD0F	109	26	3	1	66	18	6	0	0	0	0	1
173	8F12356749ABCDE0	109	26	3	1	66	18	6	0	0	0	0	1
174	401D256739ABC8EF	109	26	3	1	66	18	6	0	0	0	0	1
175	801D356749ABC2EF	109	26	3	1	66	18	6	0	0	0	0	1
176	80123C6749AB5DEF	109	26	3	1	66	18	6	0	0	0	0	1
177	8012356E49ABCD7F	109	26	3	1	66	18	6	0	0	0	0	1
178	48E2356709ABCD1F	109	26	3	1	69	15	7	0	0	0	0	1
179	80123A67495BCDEF	117	24	3	1	65	24	1	1	0	0	0	1
180	408235B719A6CDEF	117	24	3	1	65	24	1	1	0	0	0	1
181	8012359746ABCDEF	117	24	3	1	65	24	1	1	0	0	0	1
182	801235B749A6CDEF	117	24	3	1	68	21	2	1	0	0	0	1
183	861D350749ABC2EF	109	26	3	1	62	24	2	1	0	0	0	1
184	201384D759ABC6EF	109	26	3	1	62	24	2	1	0	0	0	1
185	401825A7396BCDEF	109	26	3	1	62	24	2	1	0	0	0	1
186	801235D749ABC6EF	109	26	3	1	62	24	2	1	0	0	0	1
187	4018256739ABEDCF	109	26	3	1	62	24	2	1	0	0	0	1
188	8012356749EBCDAF	109	26	3	1	62	24	2	1	0	0	0	1
189	8A123567490BCDEF	117	24	3	1	80	9	6	1	0	0	0	1
190	401A2567398BCDEF	109	26	3	1	74	12	6	1	0	0	0	1
191	4018253769ABCDEF	109	26	3	1	74	12	6	1	0	0	0	1
192	6018354729ABCDEF	93	30	3	1	65	15	7	1	0	0	0	1
193	6812354709ABCDEF	93	30	3	1	65	15	7	1	0	0	0	1
194	2013FB6749A5CDE8	116	22	4	1	60	21	6	0	0	0	0	1
195	104825F739ABCDE6	116	22	4	1	60	21	6	0	0	0	0	1
196	401825673EABCD9F	116	22	4	1	60	21	6	0	0	0	0	1
197	80132A67495BCDEF	116	22	4	1	63	18	7	0	0	0	0	1
198	801C256749AB3DEF	116	22	4	1	66	15	8	0	0	0	0	1
199	801D256749ABC3EF	116	22	4	1	66	15	8	0	0	0	0	1
200	10482A67395BCDEF	116	22	4	1	59	24	3	1	0	0	0	1

Table 5.6: Affine equivalence classes 201–250

	Representative	$\lvert c\rvert = 1/4$	1/2	3/4	1	$p = 1/8$	1/4	3/8	1/2	5/8	3/4	7/8	1
201	30128C6749AB5DEF	108	24	4	1	56	24	4	1	0	0	0	1
202	201384E759ABCD6F	108	24	4	1	56	24	4	1	0	0	0	1
203	201384F759ABCDE6	108	24	4	1	56	24	4	1	0	0	0	1
204	4018256739ABCFED	108	24	4	1	56	24	4	1	0	0	0	1
205	D012356749ABC8EF	108	24	4	1	62	18	6	1	0	0	0	1
206	E012356749ABCD8F	108	24	4	1	62	18	6	1	0	0	0	1
207	8D12356749ABC0EF	108	24	4	1	62	18	6	1	0	0	0	1
208	80C2356749AB1DEF	108	24	4	1	62	18	6	1	0	0	0	1
209	401C256739AB8DEF	108	24	4	1	62	18	6	1	0	0	0	1
210	401E256739ABCD8F	108	24	4	1	62	18	6	1	0	0	0	1
211	80132F6749ABCDE5	108	24	4	1	62	18	6	1	0	0	0	1
212	8612350749ABCDEF	116	22	4	1	73	12	5	2	0	0	0	1
213	801A2567493BCDEF	108	24	4	1	67	15	5	2	0	0	0	1
214	8016253749ABCDEF	108	24	4	1	67	15	5	2	0	0	0	1
215	8012C54769AB3DEF	92	28	4	1	48	30	0	3	0	0	0	1
216	8012D54769ABC3EF	92	28	4	1	48	30	0	3	0	0	0	1
217	20138E6749ABCD5F	115	20	5	1	56	21	6	1	0	0	0	1
218	C012356749AB8DEF	107	22	5	1	52	24	4	2	0	0	0	1
219	80D2356749ABC1EF	107	22	5	1	52	24	4	2	0	0	0	1
220	2014386759ABCDEF	115	20	5	1	64	15	6	2	0	0	0	1
221	4012356879ABCDEF	107	22	5	1	58	18	6	2	0	0	0	1
222	4012386759ABCDEF	107	22	5	1	58	18	6	2	0	0	0	1
223	F013256749ABCDE8	114	18	6	1	63	6	15	0	0	0	0	1
224	2013F56749ABCDE8	114	18	6	1	63	6	15	0	0	0	0	1
225	20138C6749AB5DEF	114	18	6	1	54	21	4	3	0	0	0	1
226	7012356849ABCDEF	114	18	6	1	54	21	4	3	0	0	0	1
227	801F256749ABCDE3	106	20	6	1	54	18	6	3	0	0	0	1
228	2018356749ABCDEF	114	18	6	1	69	6	9	3	0	0	0	1
229	4812356709ABCDEF	106	20	6	1	63	9	9	3	0	0	0	1
230	4018256739ABCDEF	113	16	7	1	62	9	8	4	0	0	0	1
231	8013256749ABCDEF	110	10	10	1	65	0	5	10	0	0	0	1
232	408235A7196BCDEF	94	32	2	1	66	23	1	0	1	0	0	1
233	4082356719ABEDCF	94	32	2	1	66	23	1	0	1	0	0	1
234	608235A7194BCDEF	109	26	3	1	66	23	1	0	1	0	0	1
235	6082354719ABCDEF	92	28	4	1	60	17	7	0	1	0	0	1
236	8012354769ABCDEF	92	28	4	1	60	17	7	0	1	0	0	1
237	201384C759AB6DEF	107	22	5	1	48	29	3	0	1	0	0	1
238	301285C749AB6DEF	107	22	5	1	48	29	3	0	1	0	0	1
239	3012C56749AB8DEF	107	22	5	1	54	23	5	0	1	0	0	1
240	4012358769ABCDEF	107	22	5	1	60	17	7	0	1	0	0	1
241	4082356719ABCDEF	105	18	7	1	58	11	9	2	1	0	0	1
242	8012356749ABCDEF	105	18	7	1	58	11	9	2	1	0	0	1
243	80A23547691BCDEF	62	40	2	1	48	33	0	0	0	1	0	1
244	8062351749ABCDEF	90	24	6	1	42	27	6	0	0	1	0	1
245	40132567E8A9CDBF	112	28	0	2	57	21	7	0	0	0	0	1
246	40132567D98BCAEF	96	32	0	2	51	21	9	0	0	0	0	1
247	40132567AC8B9DEF	112	28	0	2	50	30	2	1	0	0	0	1
248	30128B6749A5CDEF	96	32	0	2	44	30	4	1	0	0	0	1
249	104825A7396BCDEF	96	32	0	2	44	30	4	1	0	0	0	1
250	40123567E98BCDAF	96	32	0	2	44	30	4	1	0	0	0	1

Table 5.7: Affine equivalence classes 251–300

	Representative	$\lvert c\rvert = 1/4$	1/2	3/4	1	$p = 1/8$	1/4	3/8	1/2	5/8	3/4	7/8	1
251	40123567F98BCDEA	96	32	0	2	44	30	4	1	0	0	0	1
252	40132567AF9BCDE8	96	32	0	2	44	30	4	1	0	0	0	1
253	40123567C98BADEF	64	40	0	2	32	36	0	4	0	0	0	1
254	40123567D98BCAEF	64	40	0	2	32	36	0	4	0	0	0	1
255	40132567AD9BC8EF	64	40	0	2	32	36	0	4	0	0	0	1
256	60123547C9AB8DEF	64	40	0	2	32	36	0	4	0	0	0	1
257	60123547D9ABC8EF	64	40	0	2	32	36	0	4	0	0	0	1
258	6512304789ABCDEF	0	56	0	2	64	0	0	14	0	0	0	1
259	20138A67495BCDEF	110	24	2	2	45	21	11	0	0	0	0	1
260	20143567C9AB8DEF	110	24	2	2	50	18	10	1	0	0	0	1
261	40123567D9ABC8EF	94	28	2	2	40	24	8	2	0	0	0	1
262	40123567E9ABCD8F	94	28	2	2	40	24	8	2	0	0	0	1
263	40132567E8ABCD9F	94	28	2	2	40	24	8	2	0	0	0	1
264	40132567B8A9CDEF	110	24	2	2	48	24	4	3	0	0	0	1
265	40132567F9ABCDE8	108	20	4	2	44	12	16	1	0	0	0	1
266	20135467E9ABCD8F	92	24	4	2	28	30	4	5	0	0	0	1
267	40123567C9AB8DEF	92	24	4	2	28	30	4	5	0	0	0	1
268	40123567F9ABCDE8	92	24	4	2	28	30	4	5	0	0	0	1
269	10462537A98BCDEF	96	32	0	2	48	29	3	0	1	0	0	1
270	40132567D89BCAEF	96	32	0	2	48	29	3	0	1	0	0	1
271	30128A67495BCDEF	110	24	2	2	36	35	3	0	1	0	0	1
272	40132567A98BCDEF	92	24	4	2	40	17	11	2	1	0	0	1
273	2013846759ABCDEF	92	24	4	2	40	17	11	2	1	0	0	1
274	1048256739ABCDEF	106	16	6	2	32	23	7	4	1	0	0	1
275	2013856749ABCDEF	104	12	8	2	42	11	5	9	1	0	0	1
276	4612350789ABCDEF	108	20	4	2	58	16	0	5	2	0	0	1
277	4016253789ABCDEF	92	24	4	2	46	22	0	5	2	0	0	1
278	4013256798ABCDEF	90	20	6	2	42	9	15	0	3	0	0	1
279	1046253789ABCDEF	104	12	8	2	24	32	0	3	4	0	0	1
280	2014356789ABCDEF	104	12	8	2	48	8	8	3	4	0	0	1
281	4013256789ABCDEF	100	4	12	2	54	0	0	9	6	0	0	1
282	40123567A98BCDEF	62	36	2	2	36	21	12	0	0	1	0	1
283	40132567A89BCDEF	62	36	2	2	36	21	12	0	0	1	0	1
284	6012354789ABCDEF	60	32	4	2	32	27	0	7	0	1	0	1
285	3012856749ABCDEF	90	20	6	2	24	27	8	3	0	1	0	1
286	4012356789ABCDEF	88	16	8	2	38	13	8	4	2	1	0	1
287	10432567F9ABCDE8	96	24	0	4	24	6	24	3	0	0	0	1
288	20135467A98BCDEF	64	32	0	4	0	36	0	12	0	0	0	1
289	2013746589ABCDEF	96	24	0	4	12	38	0	3	4	0	0	1
290	1043256789ABCDEF	88	8	8	4	12	20	0	9	4	2	0	1
291	2013546789ABCDEF	60	24	4	4	16	9	16	5	0	3	0	1
292	2013456789ABCDEF	84	0	12	4	36	3	0	0	12	3	0	1
293	80A23517496BCDEF	56	28	8	1	0	56	0	0	0	0	0	2
294	40132567C89BADEF	0	56	0	2	0	56	0	0	0	0	0	2
295	40623517A98BCDEF	0	56	0	2	0	56	0	0	0	0	0	2
296	4062351789ABCDEF	56	24	8	2	0	44	0	6	0	0	0	2
297	2013645789ABCDEF	0	48	0	4	0	32	0	12	0	0	0	2
298	3012654789ABCDEF	0	48	0	4	0	32	0	12	0	0	0	2
299	3012456789ABCDEF	56	16	8	4	0	26	0	12	0	2	0	2
300	1023456789ABCDEF	56	0	8	8	0	14	0	0	0	14	0	2

Table 5.8: Affine equivalence classes 301–302

	Representative	$\|c\| = 1/4$	1/2	3/4	1	$p = 1/8$	1/4	3/8	1/2	5/8	3/4	7/8	1
301	1032456789ABCDEF	0	32	0	8	0	0	0	24	0	0	0	4
302	0123456789ABCDEF	0	0	0	16	0	0	0	0	0	0	0	16

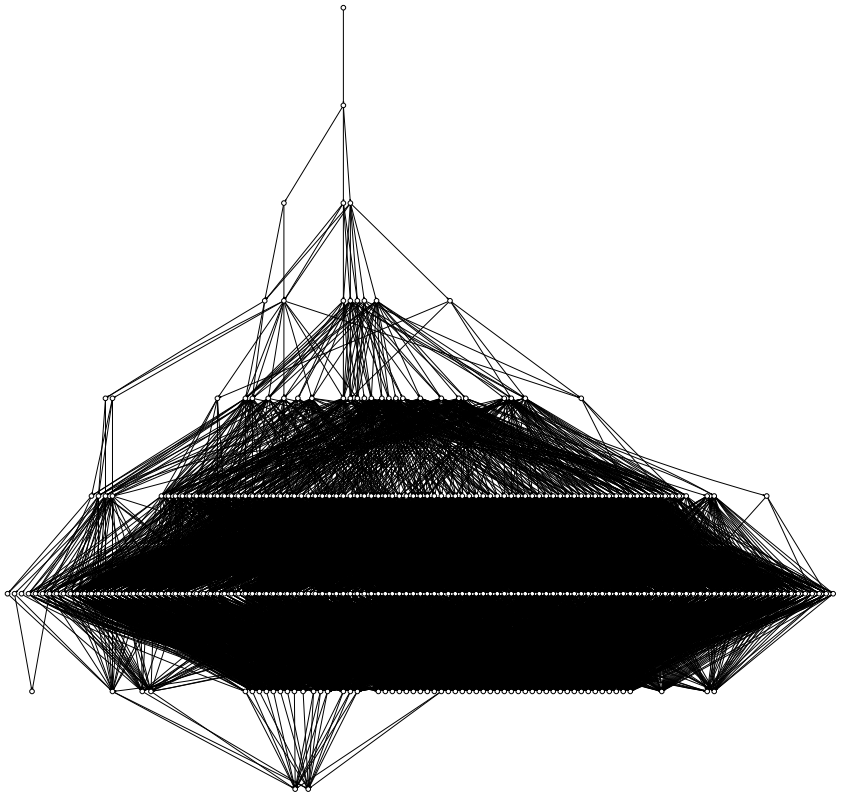


Figure 5.6: Affine equivalence classes for $n = 4$, connected by transpositions

5.5 Extensions

In this section, we briefly discuss a number of possible extensions of the linear and affine equivalence algorithms presented in the previous sections.

5.5.1 Self-Equivalent S-boxes

The original intention of the affine equivalence algorithm was to discover equivalence relations between different S-boxes, but nothing prevents us from running the algorithm for a single S-box S. In this case, the algorithm will return affine mappings A_a and B_a such that $B_a^{-1} \circ S \circ A_a = S$. The number of different solutions for this equation (denoted by $s \geq 1$) can be seen as a measure for the symmetry of the S-box. We call S-boxes that have at least one non-trivial solution ($s > 1$) self-equivalent S-boxes. We will see some examples in Sect. 5.6.

5.5.2 Equivalence of Non-invertible S-boxes

So far, we only considered equivalences between invertible $n \times n$-bit S-boxes, but similar equivalence relations exist for (non-invertible) n to m-bit S-boxes with $m < n$. This leads to a natural extension of our equivalence problem: find an $n \times n$-bit affine mapping A_a and an $m \times m$-bit affine mapping B_a such that $S_1 = B_a^{-1} \circ S_2 \circ A_a$ for two given $n \times m$-bit S-boxes S_1 and S_2.

The main problem when trying to apply the algorithms described above in this new situation, is that the procedure '`check-`N_B' explicitly relies on the fact that the S-boxes are invertible. In cases where the difference $n - m$ is not too large, slightly adapted versions of the algorithms still appear to be very useful, however.

The difference between the extended and the original algorithm resides in the way information about A_a is gathered. In the original algorithm, each iteration yields a number of additional distinct points which can directly be used to complete the affine mapping A_a. This time, the S-boxes are not uniquely invertible and the information obtained after each iteration will consist of two unordered sets of about 2^{n-m} values which are known to be mapped onto each other. In order to continue, the algorithm first needs to determine which are the corresponding values in both sets. This can be done exhaustively if 2^{n-m} is not too large, say less than 8. Once the order has been guessed, 2^{n-m} points are obtained. Since slightly more than n points should suffice to reject a candidate for the representative, one would expect that the total complexity is in the order:

$$2^n \cdot \left(2^{n-m}!\right)^{n/2^{n-m}} . \tag{5.1}$$

In order to test the extended algorithm, we can apply it to the eight 6×4-bit S-boxes of DES. The algorithm shows that no affine equivalences exist between any pair of S-boxes, with the single exception of S_4 with itself. The

equivalence relation is found to be $B^{-1}S_4(A \cdot x \oplus a) \oplus b = S_4(x)$ with $A = I$ and B a simple bit permutation $[4, 3, 2, 1]$, $a = 101111_2$ and $b = 0110_2$. Note that this specific property of S_4 was already discovered by Hellman et al. [52] by looking at patterns in the lookup table.

5.5.3 Almost Affine Equivalent S-boxes

Another interesting problem related to equivalence is the problem of detecting whether two S-boxes are *almost* equivalent. The S-boxes S_1 and S_2 are called almost equivalent if there exist two affine mappings A_a and B_a such that S_1 and $B_a^{-1} \circ S_2 \circ A_a$ are equal, except in a few points (e.g., two values in the lookup table are swapped, or some fixed fraction of the entries are misplaced).

A solution to this problem in the case of linear equivalence can be found by observing that the linear equivalence algorithm of Sect. 5.2 requires only about $\mathcal{O}(n)$ S-box queries to uniquely determine the mappings A and B that correspond with a particular guess. Once the mappings have been discovered, all that is left to do is to check the consistency for all other points. In order to detect "almost" equivalence, we may tolerate inconsistencies for a given fraction f of the remaining points. The algorithm should make sure, however, that no inconsistent points are used during the construction of the mappings. If the fraction of inconsistent points is small, it is sufficient to run the algorithm about $\mathcal{O}((1-f)^{-n})$ times, each time picking different values $x \notin C_A$, and select the mappings with the minimal number of inconsistencies. For example for $n = 8$, and a fraction $f = 0.2$ of inconsistencies, one will need to iterate the algorithm about 10 times. Note that the birthday algorithm of Sect. 5.3.3 can be adapted in a similar way.

5.6 Equivalent Descriptions of Various Ciphers

In the last section of this chapter, we apply our equivalence tools to various block ciphers, and show how this can be used to find equivalent descriptions.

5.6.1 Rijndael

When we feed the 8×8-bit S-box S used in RIJNDAEL [28] to the affine equivalence tool, it will reveal as many as 2040 different self-equivalence relations (see Table 5.9). Although this number might seem surprisingly high at first, we will show that it can easily be explained from the special algebraic structure of the S-box of RIJNDAEL.

In order to avoid the confusion of working in $GF(2^8)$ and $GF(2)^8$ simultaneously, we first introduce the notation $[a]$, which denotes the 8×8-bit matrix corresponding to a multiplication by a in $GF(2^8)$. Similarly, we denote by Q the 8×8-bit matrix which performs the squaring[2] operation in $GF(2^8)$. Considering the fact that the RIJNDAEL S-box is defined as $S(x) = A(x^{-1})$ with A a fixed affine mapping (not to be confused with A_a), we can now derive a general expression for all pairs of affine mappings A_a and B_a that satisfy $B_a^{-1} \circ S \circ A_a = S$:

$$A_a(x) = [a] \cdot Q^i \cdot x \,,$$
$$B_a^{-1}(x) = A\left(Q^{-i} \cdot [a] \cdot A^{-1}(x)\right) \,,$$

with $0 \leq i < 8$ and $a \in GF(2^8) \setminus \{0\}$. Since i takes on 8 different values[3] and there are 255 different choices for a, we obtain exactly 2040 different solutions, which confirms the output of the AE algorithm.

The existence of these affine self-equivalences in RIJNDAEL implies that we can insert an additional affine layer before and after the S-boxes without affecting the cipher. Moreover, since the mixing layer of RIJNDAEL only consists of additions and multiplications with constants in $GF(2^8)$, and since $[a] \cdot Q^i \cdot [c] = [c^{2^i}] \cdot [a] \cdot Q^i$, we can easily push the input mapping A_a through the mixing layer. This allows us to combine A_a with the output mapping of a previous layer of S-boxes, with the plaintext, the round constants or with the key. The resulting ciphers are generalizations[4] of the eight "squares" of RIJNDAEL, obtained in a somewhat different way by Barkan and Biham [6]. By modifying the field polynomial used in these 2040 ciphers, one should be able to expand the set of 240 dual ciphers in *The Book of Rijndaels* [7] to a set of 61 200 ciphers.

Note that these ideas also apply to a large extent to other ciphers that use S-boxes based on power functions. These include CAMELLIA, MISTY and KASUMI (see Table 5.9), whose S-boxes S_7 and S_9 are both designed to be affine equivalent to a power function over $GF(2^7)$ and $GF(2^9)$ respectively.

Table 5.9: RIJNDAEL, CAMELLIA, MISTY, and Kasumi

Cipher	Members	s
RIJNDAEL/CAMELLIA	S, S^{-1}	$2040 = 8 \times 255$
MISTY/KASUMI	S_7	$889 = 7 \times 127$
	S_9	$4599 = 9 \times 511$

5.6.2 Other SPN Ciphers

All affine equivalences in the RIJNDAEL S-box are directly related to its simple algebraic structure, but using our general AE tool, we can also build equivalent representations for S-boxes that are harder to analyze algebraically. Two examples are SERPENT [2] and KHAZAD [8].

[2] This is possible since squaring in $GF(2^8)$ is a linear operation in $GF(2)^8$ (see also [6]).

[3] One can easily check that $Q^8 = I$ and thus $Q^{-i} = Q^{8-i}$.

[4] For $a = 1$ we obtain the 8 square ciphers constructed in [6].

Table 5.10: SERPENT and KHAZAD

Cipher	Members	s	Class
SERPENT	S_0, S_1^{-1}	4	13
	S_0^{-1}, S_1	4	16
	$S_2, S_2^{-1}, S_6, S_6^{-1}$	4	14
	$S_3, S_3^{-1}, S_7, S_7^{-1}$	1	9
	S_4, S_5	1	10
	S_4^{-1}, S_5^{-1}	1	11
KHAZAD	P, P^{-1}, Q, Q^{-1}	4	4

An interesting property that is revealed by the AE algorithm is that the set of eight S-boxes used in SERPENT (see Table 5.10) contains three pairs of equivalent S-boxes ($\{S_2, S_6\}$, $\{S_3, S_7\}$, $\{S_4, S_5\}$) and one pair of inversely equivalent S-boxes ($\{S_0, S_1^{-1}\}$). Moreover, four of the S-boxes are self-equivalent. This allows to apply specific modifications to the mixing layer and to change the order in which the S-boxes are used, and this without affecting the output of the cipher. Notice also that the two inversely equivalent S-boxes (S_0 and S_1) are used in consecutive rounds. The mixing layer probably prevents this property from being exploited, however.

In the case of KHAZAD, both 4×4-bit S-boxes P and Q are found to be self- and mutually equivalent. This implies that the complete cipher can be described using affine mappings and a single non-linear 4×4-bit lookup table. Note that this is not necessarily as bad as it sounds: each cipher can be described with affine mappings and a single non-linear 2×1-bit AND.

5.7 Conclusions

In this chapter, we have developed efficient algorithms for detecting the linear and affine equivalence of bijective S-boxes, and have used them to completely classify the set of all 4×4-bit S-boxes. We have studied extensions of these algorithms for the case of non-bijective S-boxes with small input/output deficiency, and for detecting almost equivalence between S-boxes. Finally, we have described equivalences found in the S-boxes of a number of popular ciphers: RIJNDAEL, DES, CAMELLIA, MISTY, KASUMI, KHAZAD, and SERPENT.

Chapter 6

Stream Cipher Design

In the previous chapter, we have discussed tools to study the non-linear components of encryption schemes. In this chapter we will highlight the importance of linear components for the diffusion of this non-linearity. To illustrate this, we will propose a stream cipher with very few non-linear components, whose security completely relies on the efficient diffusion achieved by its linear structure.

6.1 Background

In the last few years, widely used stream ciphers have started to be systematically replaced by block ciphers. An example is the A5/1 stream cipher used in the GSM standard. Its successor, A5/3, is a block cipher. A similar shift took place with wireless network standards. The security mechanism specified in the original IEEE 802.11 standard (called 'wired equivalent privacy' or WEP) was based on the stream cipher RC4; the newest standard, IEEE 802.11i, makes use of the block cipher AES.

The declining popularity of stream ciphers can be explained by different factors. The first is the fact that the security of block ciphers seems to be better understood. Over the last decades, cryptographers have developed a rather clear vision of what the internal structure of a secure block cipher should look like. This is much less the case for stream ciphers. Stream ciphers proposed in the past have been based on very different principles, and many of them have shown weaknesses. A second explanation is that efficiency, which has been the traditional motivation for choosing a stream cipher over a block cipher, has ceased to be a decisive factor in many applications: not only is the cost of computing power rapidly decreasing, today's block ciphers are also significantly more efficient than their predecessors.

Still, as pointed out by the eSTREAM Stream Cipher Project, it seems that stream ciphers could continue to play an important role in those applications

where high througput remains critical and/or where resources are very restricted. This poses two challenges for the cryptographic community: first, restoring the confidence in stream ciphers, e.g., by developing simple and reliable design criteria; secondly, increasing the efficiency advantage of stream ciphers compared to block ciphers.

In this chapter, we try to explore both problems. The first part of this chapter reviews some concepts which lie at the base of today's block ciphers (Sect. 6.3), and studies how these could be mapped to stream ciphers (Sects. 6.4–6.5). The design criteria derived this way are then used as a guideline to construct a simple and flexible hardware-oriented stream cipher in the second part (Sect. 6.6).

6.2 Security and Efficiency Considerations

Before devising a design strategy for a stream cipher, it is useful to first clearly specify what we expect from it. Our aim in this chapter is to design hardware-oriented binary additive stream ciphers which are both efficient and secure. The following sections briefly discuss what this implies.

6.2.1 Security

The additive stream cipher which we intend to construct takes as input a k-bit secret key K and a v-bit IV. The cipher is then requested to generate up to 2^d bits of key stream $z_t = S_K(IV, t)$, $0 \leq t < 2^d$, and a bitwise exclusive OR of this key stream with the plaintext produces the ciphertext. The security of this additive stream cipher is determined by the extent to which it mimics a one-time pad, i.e., it should be hard for an adversary, who does not know the key, to distinguish the key stream generated by the cipher from a truly random sequence. In fact, as discussed in Sect. 2.2.4, we would like this to be as hard as we can possibly ask from a cipher with given parameters k, v, and d. This leads to a criterion called K-security [26], which can be formulated as follows:

Definition 6.1. An additive stream cipher is called K-secure if any attack against this scheme would not have been significantly more difficult if the cipher had been replaced by a set of 2^k functions $S_K : \{0,1\}^v \times \{0,\ldots,2^d-1\} \rightarrow \{0,1\}$, uniformly selected from the set of all possible functions.

The definition assumes that the adversary has access to arbitrary amounts of key stream, that he knows or can choose the a priory distribution of the secret key, that he can impose relations between different secret keys, etc.

Attacks against stream ciphers can be classified into two categories, depending on what they intend to achieve:

- *Key recovery attacks*, which try to deduce information about the secret key by observing the key stream.

- *Distinguishing attacks*, the goal of which is merely to detect that the key stream bits are not completely unpredictable.

Owing to their weaker objective, distinguishing attacks are often much easier to apply, and consequently harder to protect against. Features of the key stream that can be exploited by such attacks include periodicity, dependencies between bits at different positions, non-uniformity of distributions of bits or words, etc. In this chapter we will focus in particular on linear correlations, as it appeared to be the weakest aspect in a number of recent stream cipher proposals such as SOBER-tw [49] and SNOW 1.0 [38]. Our first design objective will be to keep the largest correlations below safe bounds. Other important properties, such as a sufficiently long period, are only considered afterwards. Note that this approach differs from the way LFSR or T-function based schemes are constructed. The latter are typically designed by maximizing the period first, and only then imposing additional requirements.

6.2.2 Efficiency

In order for a stream cipher to be an attractive alternative to block ciphers, it must be efficient. In this chapter, we will be targeting hardware applications, and a good measure for the efficiency of a stream cipher in this environment is the number of key stream bits generated per cycle per gate.

There are two ways to obtain an efficient scheme according to this measure. The first approach is illustrated by A5/1, and consists in minimizing the number of gates. A5/1 is extremely compact in hardware, but it cannot generate more than one bit per cycle. The other approach, which was chosen by the designers of PANAMA [27], is to dramatically increase the number of bits per cycle. This allows to reduce the clock frequency (and potentially also the power consumption) at the cost of an increased gate count. As a result, PANAMA is not suited for environments with very tight area constraints. Similarly, designs such as A5/1 will not perform very well in systems which require fast encryption at a low clock frequency. One of the objectives of this chapter is to design a flexible scheme which performs reasonably well in both situations.

6.3 How Block Ciphers are Designed

As explained above, the first requirement we impose on the construction is that it generates key streams without exploitable linear correlations. This problem is very similar to the one faced by block cipher designers. Hence, it is natural to attempt to borrow some of the techniques used in the block cipher world. The ideas relevant to stream ciphers are briefly recapitulated in the following sections.

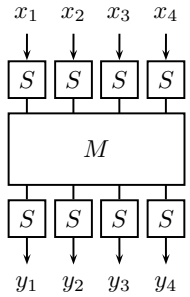


Figure 6.1: Three layers of a block cipher

6.3.1 Block Ciphers and Linear Characteristics

An important problem in the case of block ciphers is that of restricting linear correlations between input and output bits in order to thwart linear cryptanalysis (see Chaps. 3 and 4). More precisely, let P be any plaintext block and C the corresponding ciphertext under a fixed secret key, then any linear combination of bits

$$\Gamma_P^\mathsf{T} \cdot P + \Gamma_C^\mathsf{T} \cdot C \,,$$

where the column vectors Γ_P and Γ_C are called linear masks, should be as balanced as possible. That is, the correlation (or imbalance)

$$c = 2 \cdot \frac{|\{P \mid \Gamma_P^\mathsf{T} \cdot P = \Gamma_C^\mathsf{T} \cdot C\}|}{|\{P\}|} - 1$$

has to be close to 0 for any Γ_P and Γ_C. As explained earlier in this thesis, the well-established way to achieve this consists in alternating two operations. The first splits blocks into smaller words which are independently fed into nonlinear substitution boxes (S-boxes); the second step recombines the outputs of the S-boxes in a linear way in order to 'diffuse' the nonlinearity. The result, called a substitution-permutation network in Sect. 2.1.4, is depicted again in Fig. 6.1.

In order to estimate the strength of a block cipher against linear cryptanalysis, one will typically compute bounds on the correlation of linear characteristics. A linear characteristic describes a possible path over which a correlation might propagate through the block cipher. It is a chain of linear masks, starting with a plaintext mask and ending with a ciphertext mask, such that every two successive masks correspond to a nonzero correlation between consecutive intermediate values in the cipher. The total correlation of the characteristic is then estimated by multiplying the correlations of all separate steps (as dictated by the Piling-up Lemma of Sect. 3.3.3).

6.3.2 Branch Number

Linear diffusion layers, which can be represented by a matrix multiplication $Y = M \cdot X$, do not by themselves contribute in reducing the correlation of a characteristic. Clearly, it suffices to choose $\Gamma_X = M^\mathsf{T} \cdot \Gamma_Y$, where M^T denotes the transpose of M, in order to obtain perfectly correlating linear combinations of X and Y:

$$\Gamma_Y^\mathsf{T} \cdot Y = \Gamma_Y^\mathsf{T} \cdot MX = (M^\mathsf{T}\Gamma_Y)^\mathsf{T} \cdot X = \Gamma_X^\mathsf{T} \cdot X \,.$$

However, diffusion layers play an important indirect role by forcing characteristics to take into account a large number of nonlinear S-boxes in the neighboring layers (called active S-boxes). A useful metric in this context is the *branch number* of M.

Definition 6.2. The branch number of a linear transformation M is defined as

$$\mathcal{B} = \min_{\Gamma_Y \neq 0} [\mathrm{w_h}(\Gamma_Y) + \mathrm{w_h}(M^\mathsf{T}\Gamma_Y)] \,,$$

where $\mathrm{w_h}(\Gamma)$ represents the number of nonzero words in the linear mask Γ.

The definition above implies that any linear characteristic traversing the structure shown in Fig. 6.1 activates at least $\mathcal{B}$ S-boxes. The total number of active S-boxes throughout the cipher multiplied by the maximal correlation over a single S-box gives an upper bound for the correlation of the characteristic.

The straightforward way to minimize this upper bound is to maximize the branch number $\mathcal{B}$. It is easy to see that $\mathcal{B}$ cannot exceed $m + 1$, with m the number of words per block. Matrices M that satisfy this bound (known as the Singleton bound) can be derived from the generator matrices of maximum distance separable (MDS) block codes.

Large MDS matrices are expensive to implement, though. Therefore, it is often more efficient to use smaller matrices, with a relatively low branch number, and to connect them in such a way that linear patterns with a small number of active S-boxes cannot be chained together to cover the complete cipher. This was the approach taken by the designers of RIJNDAEL [28].

6.4 From Blocks to Streams

In this section, we try to adapt the concepts described above to a system where the data is not processed in blocks, but rather as a stream.

Since the data stream enters the system one word at a time, each layer of S-boxes in Fig. 6.1 can be replaced by a single S-box which substitutes individual words as they arrive. A general mth-order linear filter can take over the task of the diffusion matrix. The new system is represented in Fig. 6.2, where D denotes the delay operator (usually written as z^{-1} in signal processing literature), and f and g are linear functions.

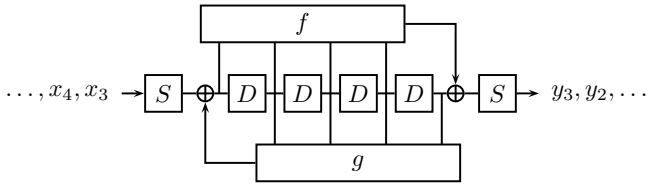


Figure 6.2: Stream equivalent of Fig. 6.1

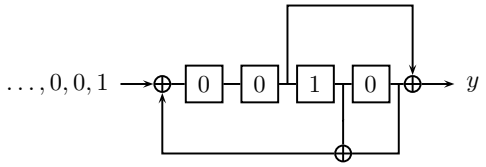


Figure 6.3: A 4th-order linear filter

6.4.1 Polynomial Notation

Before analyzing the properties of this construction, we introduce some notations. First, we adopt the common convention to represent streams of words $x_0, x_1, x_2, \ldots$ as polynomials with coefficients in the finite field:

$$x(D) = x_0 + x_1 D + x_2 D^2 + \ldots .$$

The rationale for this representation is that it simplifies the expression for the input/output relation of the linear filter, as shown in the following equation:

$$y(D) = \frac{f(D)}{g(D)} \cdot \left[x(D) + x^0(D)\right] + y^0(D) . \tag{6.1}$$

The polynomials f and g describe the feedforward and feedback connections of the filter. They can be written as

$$f(D) = D^m \cdot \left(f_m D^{-m} + \cdots + f_1 D^{-1} + 1\right) ,$$
$$g(D) = 1 + g_1 D + g_2 D^2 + \cdots + g_m D^m .$$

The Laurent polynomials x^0 and y^0 represent the influence of the initial state s^0, and are given by $x^0 = D^{-m} \cdot \left(s^0 \cdot g \bmod D^m\right)$ and $y^0 = D^{-m} \cdot \left(s^0 \cdot f \bmod D^m\right)$.

Example 6.1. The 4th-order linear filter depicted in Fig. 6.3 is specified by the polynomials $f(D) = D^4 \cdot (D^{-2} + 1)$ and $g(D) = 1 + D^3 + D^4$. Suppose that the delay elements are initialized as shown in the figure, i.e., $s^0(D) = D$. Knowing

s^0, we can compute $x^0(D) = D^{-3}$ and $y^0(D) = D^{-1}$. Finally, using (6.1), we find the output stream corresponding to an input consisting, for example, of a single 1 followed by 0's (i.e., $x(D) = 1$):

$$\begin{aligned} y(D) &= \frac{D^{-1} + D + D^2 + D^4}{1 + D^3 + D^4} + D^{-1} \\ &= D + D^3 + D^5 + D^6 + D^7 + D^8 + D^{12} + D^{15} + D^{16} + D^{18} + \ldots \end{aligned}$$

6.4.2 Linear Correlations

In order to study correlations in a stream-oriented system we need a suitable way to manipulate linear combinations of bits in a stream. It will prove convenient to represent them as follows:

$$\mathrm{Tr}\left[[\gamma_x(D^{-1}) \cdot x(D)]_0\right] .$$

The operator $[\cdot]_0$ returns the constant term of a polynomial, and $\mathrm{Tr}(\cdot)$ denotes the trace to GF(2).[1] The coefficients of γ_x, called *selection polynomial*, specify which words of x are involved in the linear combination. In order to simplify expressions later on we also introduce the notation $\gamma^*(D) = \gamma(D^{-1})$. The polynomial γ^* is called the *reciprocal* polynomial of γ.

As before, the correlation between x and y for a given pair of selection polynomials is defined as

$$c = 2 \cdot \frac{|\{(x, s^0) \mid \mathrm{Tr}[[\gamma_x^* \cdot x]_0] = \mathrm{Tr}[[\gamma_y^* \cdot y]_0]\}|}{|\{(x, s^0)\}|} - 1 ,$$

where $\deg x \le \max(\deg \gamma_x, \deg \gamma_y)$.

6.4.3 Propagation of Selection Polynomials

Let us now analyze how correlations propagate through the linear filter. For each selection polynomial γ_x at the input, we would like to determine a polynomial γ_y at the output (if it exists) such that the corresponding linear combinations are perfectly correlated, i.e.,

$$\mathrm{Tr}[[\gamma_x^* \cdot x]_0] = \mathrm{Tr}[[\gamma_y^* \cdot y]_0], \quad \forall x, s^0 .$$

If this equation is satisfied, then this will still be the case after replacing x by $x' = x + x^0$ and y by $y' = y + y^0$, since x^0 and y^0 only consist of negative powers, none of which can be selected by γ_x or γ_y. Substituting (6.1), we find

$$\mathrm{Tr}[[\gamma_x^* \cdot x']_0] = \mathrm{Tr}[[\gamma_y^* \cdot f/g \cdot x']_0], \quad \forall x, s^0 ,$$

[1] The trace from $GF(2^n)$ to $GF(2)$ is defined as $\mathrm{Tr}(a) = a + a^2 + a^4 + \cdots + a^{2^{n-1}}$.

which implies that $\gamma_x^* = \gamma_y^* \cdot f/g$. In order to get rid of negative powers, we define $f^\star = D^m \cdot f^*$ and $g^\star = D^m \cdot g^*$ (note the subtle difference between both stars), and obtain the equivalent relation

$$\gamma_y = g^\star / f^\star \cdot \gamma_x \,. \tag{6.2}$$

Note that neither of the selection polynomials γ_x and γ_y can have an infinite number of nonzero coefficients (if it were the case, the linear combinations would be undefined). Hence, they have to be of the form

$$\gamma_x = q \cdot f^\star / \gcd(f^\star, g^\star) \quad \text{and} \quad \gamma_y = q \cdot g^\star / \gcd(f^\star, g^\star) \,, \tag{6.3}$$

with $q(D)$ an arbitrary polynomial.

Example 6.2. For the linear filter in Fig. 6.3, we have that $f^\star(D) = 1 + D^2$ and $g^\star(D) = D^4 \cdot (D^{-4} + D^{-3} + 1)$. In this case, $f^\star$ and $g^\star$ are coprime, i.e., $\gcd(f^\star, g^\star) = 1$. If we arbitrarily choose $q(D) = 1 + D$, we obtain a pair of selection polynomials

$$\gamma_x(D) = 1 + D + D^2 + D^3 \quad \text{and} \quad \gamma_y(D) = 1 + D^2 + D^4 + D^5 \,.$$

By construction, the corresponding linear combinations of input and output bits satisfy the relation

$$\mathrm{Tr}(x_0 + x_1 + x_2 + x_3) = \mathrm{Tr}(y_0 + y_2 + y_4 + y_5), \quad \forall x, s^0 \,.$$

6.4.4 Branch Number

The purpose of the linear filter, just as the diffusion layer of a block cipher, will be to force linear characteristics to pass through as many active S-boxes as possible. Hence, it makes sense to define a branch number here as well.

Definition 6.3. The branch number of a linear filter specified by the polynomials f and g is defined as

$$\begin{aligned} \mathcal{B} &= \min_{\gamma_x \neq 0} [\mathrm{w_h}(\gamma_x) + \mathrm{w_h}(g^\star / f^\star \cdot \gamma_x)] \\ &= \min_{q \neq 0} [\mathrm{w_h}(q \cdot f^\star / \gcd(f^\star, g^\star)) + \mathrm{w_h}(q \cdot g^\star / \gcd(f^\star, g^\star))] \,, \end{aligned}$$

where $\mathrm{w_h}(\gamma)$ represents the number of nonzero coefficients in the selection polynomial γ.

From this definition we immediately obtain the following upper bound on the branch number

$$\mathcal{B} \leq \mathrm{w_h}(f^\star) + \mathrm{w_h}(g^\star) \leq 2 \cdot (m+1) \,. \tag{6.4}$$

Filters for which this bound is attained can be derived from MDS convolutional $(2, 1, m)$-codes [98]. For example, one can verify that the 4th-order linear filter over GF(2^8) with

$$\begin{aligned} f(D) &= D^4 \cdot \left(\mathtt{02}_x D^{-4} + D^{-3} + D^{-2} + \mathtt{02}_x D^{-1} + 1 \right) \,, \\ g(D) &= 1 + \mathtt{03}_x D + \mathtt{03}_x D^2 + D^3 + D^4 \,, \end{aligned}$$

has a branch number of 10. The example uses the same field polynomial as RIJNDAEL, i.e., $x^8 + x^4 + x^3 + x + 1$. Note that in the next sections, we will not try to maximize the branch number, but use much sparser linear filters instead.

6.5 Constructing a Key Stream Generator

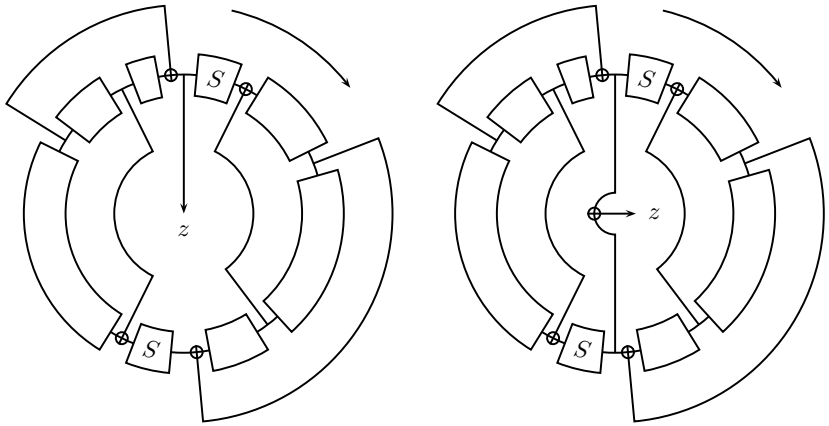
In the previous section, we introduced S-boxes and linear filters as building blocks, and presented some tools to analyze how they interact. Our next task is to determine how these components can be combined into a key stream generator. Again, block ciphers will serve as a source of inspiration.

6.5.1 Basic Construction

A well-known way to construct a key stream generator from a block cipher is to use the cipher in output feedback (OFB) mode. As explained in Sect. 2.1.5, this mode of operation takes as input an initial data block (called initial value or IV), passes it through the block cipher, and feeds the result back to the input. This process is iterated and the consecutive values of the data block are used as key stream. We recall that the block cipher itself typically consists of a sequence of rounds, each comprising a layer of S-boxes and a linear diffusion transformation.

By taking the very same approach, but this time using the stream cipher components presented in Sect. 6.4, we obtain a construction which, in its simplest form, might look like Fig. 6.4(a). The figure represents a key stream generator consisting of two 'rounds', where each round consists of an S-box followed by a very simple linear filter. Data words traverse the structure in clockwise direction, and the output of the second round, which also serves as key stream, is fed back to the input of the first round.

While the scheme proposed above has some interesting structural similarities with a block cipher in OFB mode, there are important differences as well. The most fundamental difference comes from the fact that linear filters, as opposed to diffusion matrices, have an internal state. Hence if the algorithm manages to keep this state (or at least parts of it) secret, then this eliminates the need for a separate key addition layer (another important block cipher component, which we have tacitly ignored so far).



(a) (b)

Figure 6.4: Two-round key stream generators

6.5.2 Analysis of Linear Characteristics

As stated before, the primary goal in this chapter is to construct a scheme which generates a stream of seemingly uncorrelated bits. More specifically, we would like the adversary to be unable to detect any correlation between linear combinations of bits at different positions in the key stream. In the following sections, we will see that the study of linear characteristics provides some guidance on how to design the components of our scheme in order to reduce the magnitude of these correlations.

Applying the tools from Sect. 6.4 to the construction in Fig. 6.4(a), we can easily derive some results on the existence of low-weight linear characteristics. The term 'low-weight' in this context refers to a small number of active S-boxes. Since we are interested in correlations which can be detected by an adversary, we need both ends of the characteristic to be accessible from the key stream. In order to construct such characteristics, we start with a selection polynomial γ_u at the input of the first round, and analyze how it might propagate through the cipher.

First, the characteristic needs to cross an S-box. The S-box preserves the positions of the non-zero coefficients of γ_u, but might modify their values. For now, however, let us only consider characteristics for which the values are preserved as well. Under this assumption and using (6.2), we can compute the selection polynomials γ_v and γ_w at the input and the output of the second round:

$$\gamma_v = g_1^\star / f_1^\star \cdot \gamma_u \quad \text{and} \quad \gamma_w = g_2^\star / f_2^\star \cdot \gamma_v \,.$$

Since all three polynomials γ_u, γ_v, and γ_w need to be finite, we have that

$$\gamma_u = q \cdot f_1^\star f_2^\star / d \,, \quad \gamma_v = q \cdot g_1^\star f_2^\star / d \,, \quad \text{and} \quad \gamma_w = q \cdot g_1^\star g_2^\star / d \,,$$

with $d = \gcd(f_1^\star f_2^\star, g_1^\star f_2^\star, g_1^\star g_2^\star)$ and q an arbitrary polynomial. Note that since both γ_u and γ_w select bits from the key stream z, they can be combined into a single polynomial $\gamma_z = \gamma_u + \gamma_w$.

The number of S-boxes activated by a characteristic of this form is given by $\mathcal{W} = \mathrm{w_h}(\gamma_u) + \mathrm{w_h}(\gamma_v)$. The minimum number of active S-boxes over this set of characteristics can be computed with the formula

$$\mathcal{W}_{\min} = \min_{q \neq 0} [\mathrm{w_h}(q \cdot f_1^\star f_2^\star / d) + \mathrm{w_h}(q \cdot g_1^\star f_2^\star / d)] \,,$$

from which we derive that

$$\mathcal{W}_{\min} \leq \mathrm{w_h}(f_1^\star f_2^\star) + \mathrm{w_h}(g_1^\star f_2^\star) \leq \mathrm{w_h}(f_1^\star) \cdot \mathrm{w_h}(f_2^\star) + \mathrm{w_h}(g_1^\star) \cdot \mathrm{w_h}(f_2^\star) \,.$$

Applying this bound to the specific example of Fig. 6.4(a), where $\mathrm{w_h}(f_i^\star) = \mathrm{w_h}(g_i^\star) = 2$, we conclude that there will always exist characteristics with at most 8 active S-boxes, no matter where the taps of the linear filters are positioned.

6.5.3 An Improvement

We will now show that this bound can potentially be doubled by making the small modification shown in Fig. 6.4(b). This time, each non-zero coefficient in the selection polynomial at the output of the key stream generator needs to propagate to both the upper and the lower part of the scheme. By constructing linear characteristics in the same way as before, we obtain the following selection polynomials:

$$\gamma_u = q \cdot \frac{f_1^\star f_2^\star + f_1^\star g_2^\star}{d} \,, \quad \gamma_v = q \cdot \frac{f_1^\star f_2^\star + g_1^\star f_2^\star}{d} \,, \quad \text{and} \quad \gamma_z = q \cdot \frac{f_1^\star f_2^\star + g_1^\star g_2^\star}{d} \,,$$

with $d = \gcd(f_1^\star f_2^\star + f_1^\star g_2^\star, f_1^\star f_2^\star + g_1^\star f_2^\star, f_1^\star f_2^\star + g_1^\star g_2^\star)$. The new upper bounds on the minimum number of active S-boxes are given by

$$\begin{aligned} \mathcal{W}_{\min} &\leq \mathrm{w_h}(f_1^\star f_2^\star + f_1^\star g_2^\star) + \mathrm{w_h}(f_1^\star f_2^\star + g_1^\star f_2^\star) \\ &\leq 2 \cdot \mathrm{w_h}(f_1^\star) \cdot \mathrm{w_h}(f_2^\star) + \mathrm{w_h}(f_1^\star) \cdot \mathrm{w_h}(g_2^\star) + \mathrm{w_h}(g_1^\star) \cdot \mathrm{w_h}(f_2^\star) \,, \end{aligned}$$

or, in the case of Fig. 6.4(b), $\mathcal{W}_{\min} \leq 16$. In general, if we consider extensions of this scheme with r rounds and $\mathrm{w_h}(f_i^\star) = \mathrm{w_h}(g_i^\star) = w$, then the bound takes the form:

$$\mathcal{W}_{\min} \leq r^2 \cdot w^r \,. \tag{6.5}$$

This result suggests that it might not be necessary to use a large number of rounds, or complicated linear filters, to ensure that the number of active S-boxes in all characteristics is sufficiently large. For example, if we take $w = 2$ as before, but add one more round, the bound jumps to 72.

Of course, since the bound we just derived is an upper bound, the minimal number of active S-boxes might as well be much smaller. First, some of the product terms in $f_1^\star f_2^\star + f_1^\star g_2^\star$ or $f_1^\star f_2^\star + g_1^\star f_2^\star$ might cancel out, or there might exist a $q \neq d$ for which $\mathrm{w_h}(\gamma_u) + \mathrm{w_h}(\gamma_v)$ suddenly drops. These cases are rather easy to detect, though, and can be avoided during the design. A more important problem is that, by fixing the behavior of S-boxes, we have limited ourselves to a special set of characteristics, which might not necessarily include the one with the minimal number of active S-boxes. However, if the feedback and feedforward functions are sparse, and the linear filters sufficiently large, then the bound is increasingly likely to be tight. On the other hand, if the state of the generator is sufficiently small, then we can perform an efficient search for the lowest-weight characteristic without making any additional assumption.

This last approach allows to show, for example, that the smallest instance of the scheme in Fig. 6.4(b) for which the bound of 16 is actually attained, consists of two 11th-order linear filters with

$$f_1^\star(D) = 1 + D^{10}\,, \qquad g_1^\star(D) = D^{11} \cdot (D^{-3} + 1)\,,$$
$$f_2^\star(D) = 1 + D^{9}\,, \qquad g_2^\star(D) = D^{11} \cdot (D^{-8} + 1)\,.$$

6.5.4 Linear Characteristics and Correlations

In the sections above, we have tried to increase the number of active S-boxes of linear characteristics. We now briefly discuss how this number affects the correlation of key stream bits. This problem is treated in several papers in the context of block ciphers (see, e.g., [28]).

We start with the observation that the minimum number of active S-boxes $\mathcal{W}_{\min}$ imposes a bound on the correlation c_c of a linear characteristic:

$$c_c^2 \leq (c_s^2)^{\mathcal{W}_{\min}}\,,$$

where c_s is the largest correlation (in absolute value) between the input and the output values of the S-box. The squares c_c^2 and c_s^2 are often referred to as *linear probability*, or also *correlation potential*. The inverse of this quantity is a good measure for the amount of data that the attacker needs to observe in order to detect a correlation.

What makes the analysis more complicated, however, is that many linear characteristics can contribute to the correlation of the same combination of key stream bits. This occurs in particular when the scheme operates on words, in which case there are typically many possible choices for the coefficients of the intermediate selection polynomials describing the characteristic (this effect is called *clustering*). The different contributions add up or cancel out, depending on the signs of c_c. If we now assume that these signs are randomly distributed, then we can use the approach of [28, Appendix B] to derive a bound on the expected correlation potential of the key stream bits:

$$E(c^2) \leq (c_s^2)^{\mathcal{W}_{\min}-n}\,. \tag{6.6}$$

The parameter n in this inequality represents the number of degrees of freedom for choosing the coefficients of the intermediate selection polynomials.

For the characteristics propagating through the construction presented in Sect. 6.5.3, one will find, in non-degenerate cases, that the values of $n = r \cdot (r-1) \cdot w^{r-1}$ non-zero coefficients can be chosen independently. Hence, for example, if we construct a scheme with $w = 2$ and $r = 3$, and if we assume that it attains the bound given in (6.5), then we expect the largest correlation potential to be at most $c_s^{2 \cdot 48}$. Note that this bound is orders of magnitude higher than the contribution of a single characteristic, which has a correlation potential of at most $c_s^{2 \cdot 72}$.

Remark 6.1. In order to derive (6.6), we replaced the signs of the contributing linear characteristics by random variables. This is a natural approach in the case of block ciphers, where the signs depend on the value of the secret key. In our case, however, the signs are fixed for a particular scheme, and hence they might, for some special designs, take on very peculiar values. This happens for example when $r = 2$, w is even, and all non-zero coefficients of f_i and g_i equal 1 (as in the example at the end of the previous section). In this case, all signs will be positive, and we obtain a significantly worse bound:

$$c^2 \leq (c_s^2)^{\mathcal{W}_{\min}-2 \cdot n}\,.$$

6.6 Trivium's Design

We now present an experimental 80-bit key stream cipher based on the approach outlined above. In this section, we concentrate on the basic design ideas behind the scheme. The complete specifications of the cipher, which was submitted to the eSTREAM Stream Cipher Project under the name TRIVIUM, can be found in Sect. 6.7.

6.6.1 A Bit-Oriented Design

The main idea of TRIVIUM's design is to turn the general scheme of Sect. 6.5.3 into a bit-oriented stream cipher. The first motivation is that bit-oriented schemes are typically more compact in hardware. A second reason is that, by reducing the word-size to a single bit, we may hope to get rid of the clustering phenomenon which, as seen in the previous section, has a significant effect on the correlation.

Of course, if we simply apply the previous scheme to bits instead of words, we run into the problem that the only two existing 1×1-bit S-boxes are both linear. In order to solve this problem, we replace the S-boxes by a component

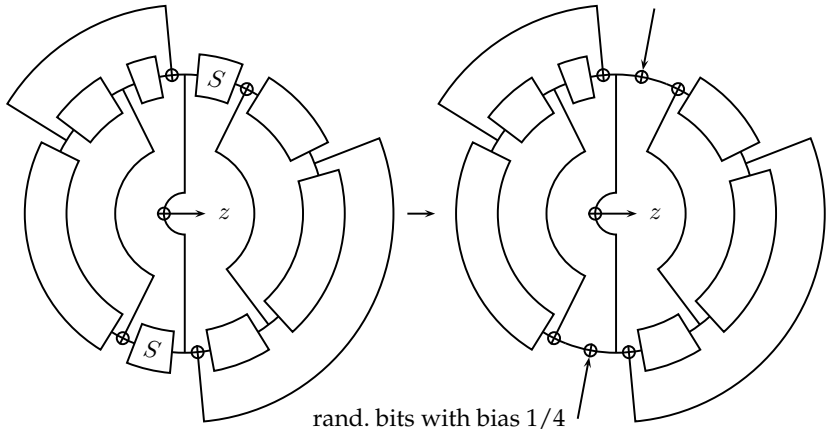


Figure 6.5: How to design 1-bit S-boxes?

which, from the point of view of our correlation analysis, behaves in the same way: an exclusive OR with an external stream of unrelated but biased random bits (see Fig. 6.5). Assuming that these random bits equal 0 with probability $(1 + c_s)/2$, we will find as before that the output of this component correlates with the input with correlation coefficient c_s.

The introduction of this artificial 1×1-bit S-box greatly simplifies the correlation analysis, mainly because of the fact that the selection polynomial at the output of an S-box is now uniquely determined by the input. As a consequence, we neither need to make special assumptions about the values of the non-zero coefficients, nor to consider the effect of clustering: the maximum correlation in the key stream is simply given by the relation

$$c_{\max} = c_s^{\mathcal{W}_{\min}} . \tag{6.7}$$

The obvious drawback, however, is that the construction now relies on external streams of random bits, which have to be generated somehow. TRIVIUM attempts to solve this problem by interleaving three identical key stream generators, where each generator obtains streams of biased bits (with $c_s = 1/2$) by ANDing together state bits of the two other generators.

6.6.2 Specifying the Parameters

Let us now specify suitable parameters for each of those three identical 'sub-generators'. Our goal is to keep all parameters as small and simple as possible, given a number of requirements.

1. The first requirement we impose is that the correlations in the key stream do not exceed 2^{-40}. Since each sub-generator will be fed with streams of bits having correlation coefficient $c_s = 1/2$, we can derive from (6.7) that a minimum weight $\mathcal{W}_{\min}$ of at least 40 is needed. The smallest values of w and r for which this requirement could be satisfied (with a fairly large margin, in fact) are $w = 2$ and $r = 3$.

2. Now that w and r are fixed, we raise our requirements and impose that the minimum weight actually reaches the upper bound of (6.5). In this case, this translates to the condition $\mathcal{W}_{\min} = 72$, which is fulfilled if $\mathrm{w_h}(\gamma_u) + \mathrm{w_h}(\gamma_v) + \mathrm{w_h}(\gamma_w) \geq 72$ for all $q \neq 0$, where

$$\gamma_u = q \cdot \frac{f_1^\star f_2^\star f_3^\star + f_1^\star f_2^\star g_3^\star + f_1^\star g_2^\star g_3^\star}{d} , \quad \gamma_v = \ldots , \quad \text{etc.}$$

3. Although the preceding sections have almost exclusively focused on linear correlations, other security properties such as periodicity remain important. Controlling the period of the scheme is difficult because of the non-linear interaction between the sub-generators, but we can try to decrease the probability of short cycles by maximizing the periods of the individual sub-generators after turning off the streams feeding their 1×1-bit S-boxes. The connection polynomial of these (completely linear) generators is given by $f_1^\star f_2^\star f_3^\star + g_1^\star g_2^\star g_3^\star$, and ideally, we would like this polynomial to be primitive. Our choice of w prevents this, though: for $w = 2$, the polynomial above is always divisible by $(D + 1)^3$. Therefore, we just require that the remaining factor is primitive, and rely on the initialization of the state bits to avoid the few short cycles corresponding to the factor $(D + 1)^3$ (see Sect. 6.8.2).

4. Finally, we also impose some efficiency requirements. The first is that state bits of the sub-generators should not be used for at least 64/3 iterations, once they have been modified. This will provide the final scheme with the flexibility to generate up to 64 bits in parallel. Secondly, the length of the sub-generators should be as short as possible and a multiple of 32.

We can now exhaustively run over all possible polynomials $f_1^\star, \ldots, g_3^\star$ in order to find combinations for which all previous requirements are fulfilled simultaneously. Surprisingly enough, it turns out that the solution is unique:

$$\begin{aligned} f_1^\star(D) &= 1 + D^9 , & g_1^\star(D) &= D^{31} \cdot (D^{-23} + 1) , \\ f_2^\star(D) &= 1 + D^5 , & g_2^\star(D) &= D^{28} \cdot (D^{-26} + 1) , \\ f_3^\star(D) &= 1 + D^{15} , & g_3^\star(D) &= D^{37} \cdot (D^{-29} + 1) . \end{aligned}$$

In order to construct the final cipher, we interleave three of these sub-generators and interconnect them through AND-gates. Since the reasoning above does

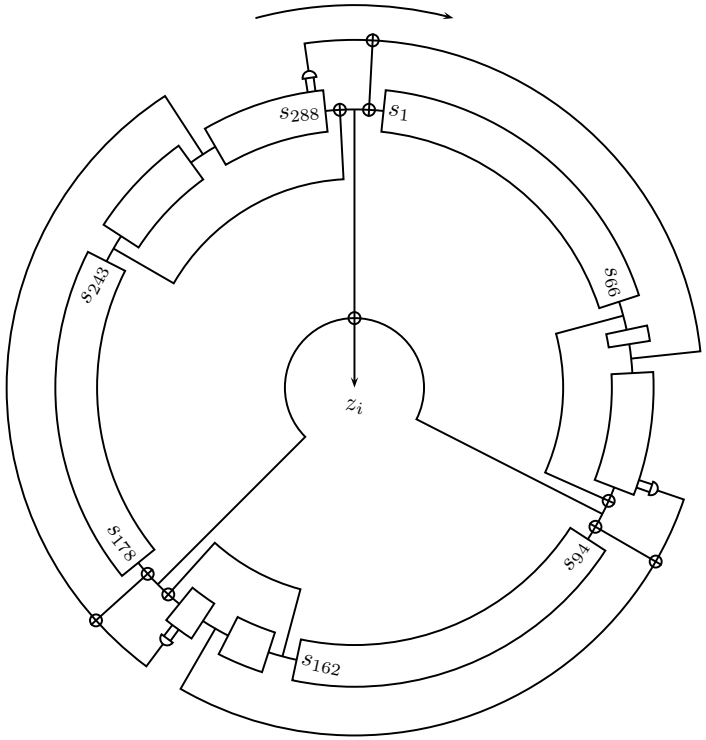


Figure 6.6: TRIVIUM

not suggest which state bits to use as inputs of the AND-gates, we simply choose to minimize the length of the wires. The resulting scheme is shown in Fig. 6.6. The 96 state bits $s_1, s_4, s_7, \ldots, s_{286}$ belong to the first sub-generator, $s_2, s_5, s_8, \ldots, s_{287}$ to the second one, etc.

6.7 Specifications of Trivium

In this section, we give the complete specifications of TRIVIUM. The synchronous stream cipher is designed to generate up to 2^{64} bits of key stream from an 80-bit secret key and an 80-bit initial value (IV). As for most stream ciphers, this process consists of two phases: first the internal state of the cipher is initialized using the key and the IV, then the state is repeatedly updated and used to generate key stream bits. We first describe this second phase.

Table 6.1: Parameters of TRIVIUM

Parameters		
Key size:	80	bit
IV size:	80	bit
Internal state:	288	bit

6.7.1 Key stream generation

The proposed design contains a 288-bit internal state denoted by $(s_1, \ldots, s_{288})$. The key stream generation consists of an iterative process which extracts the values of 15 specific state bits and uses them both to update 3 bits of the state and to compute 1 bit of key stream z_i. The state bits are then rotated and the process repeats itself until the requested $N \leq 2^{64}$ bits of key stream have been generated. A complete description is given by the following simple pseudo-code:

for $i = 1$ to N **do**
$t_1 \leftarrow s_{66} + s_{93}$
$t_2 \leftarrow s_{162} + s_{177}$
$t_3 \leftarrow s_{243} + s_{288}$
$z_i \leftarrow t_1 + t_2 + t_3$
$t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$
$t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
$t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
$(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
end for

We remind the reader that here, as in the rest of this chapter, the '$+$' and '$\cdot$' operations stand for addition and multiplication over GF(2) (i.e., XOR and AND), respectively. A graphical representation of the key stream generation process is given in Fig. 6.6.

6.7.2 Key and IV setup

The algorithm is initialized by loading an 80-bit key and an 80-bit IV into the 288-bit initial state, and setting all remaining bits to 0, except for s_{286}, s_{287}, and s_{288}. Then, the state is rotated over 4 full cycles, in the same way as explained above, but without generating key stream bits. This is summarized in the pseudo-code below:

$(s_1, s_2, \ldots, s_{93}) \leftarrow (K_{80}, \ldots, K_1, 0, \ldots, 0)$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_{80}, \ldots, IV_1, 0, \ldots, 0)$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)$
for $i = 1$ to $4 \cdot 288$ **do**
  $t_1 \leftarrow s_{66} + s_{91} \cdot s_{92} + s_{93} + s_{171}$
  $t_2 \leftarrow s_{162} + s_{175} \cdot s_{176} + s_{177} + s_{264}$
  $t_3 \leftarrow s_{243} + s_{286} \cdot s_{287} + s_{288} + s_{69}$
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
  $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
end for

6.7.3 Alternative Description

Alternatively, TRIVIUM's key stream generation algorithm can also be written in the following recursive way, proposed by Bernstein [11]:

for $i = 1$ to N **do**
  $a_i = c_{i-66} + c_{i-111} + c_{i-110} \cdot c_{i-109} + a_{n-69}$
  $b_i = a_{i-66} + a_{i-93} + a_{i-92} \cdot a_{i-91} + b_{n-78}$
  $c_i = b_{i-69} + b_{i-84} + b_{i-83} \cdot b_{i-82} + c_{n-87}$
  $z_i = c_{i-66} + c_{i-111} + a_{i-66} + a_{i-93} + b_{i-69} + b_{i-84}$
end for

This notation is often more convenient when describing attacks against the stream cipher.

6.8 Security

In this section we briefly discuss some of the cryptographic properties of TRIVIUM. The security requirement we would like to meet is that any type of cryptographic attack should not be significantly easier to apply to TRIVIUM than to any other imaginable stream cipher with the same external parameters (i.e., any cipher capable of generating up to 2^{64} bits of key stream from an 80-bit secret key and an 80-bit IV). Unfortunately, this requirement is not easy to verify, and the best we can do is to provide arguments why we believe that certain common types of attacks are not likely to affect the security of the cipher. A summary of the results discussed in the next sections is given in Table 6.2.

6.8.1 Correlations

When analyzing the security of a synchronous stream cipher, a cryptanalyst will typically consider two different types of correlations The first type are

correlations between linear combinations of key stream bits and internal state bits, which can potentially lead to a complete recovery of the state. The second type, exploited by distinguishing attacks, are correlations between the key stream bits themselves.

Obviously, linear correlations between key stream bits and internal state bits are easy to find, since z_i is simply defined to be equal to $s_{66} + s_{93} + s_{162} + s_{177} + s_{243} + s_{288}$. However, as opposed to LFSR based ciphers, TRIVIUM's state evolves in a nonlinear way, and it is not clear how the attacker should combine these equations in order to efficiently recover the state.

An easy way to find correlations of the second type is to follow linear characteristics through the cipher and to approximate the outputs of all encountered AND gates by 0. However, as explained in the previous section, the positions of the taps in TRIVIUM have been chosen in such a way that any characteristic of this specific type is forced to approximate at least 72 AND gate outputs. An example of a correlated linear combination of key stream bits obtained this way is

$$z_1 + z_{16} + z_{28} + z_{43} + z_{46} + z_{55} + z_{61} + z_{73} + z_{88} + z_{124} + z_{133} + z_{142} + z_{202} + z_{211} + z_{220} + z_{289}\,.$$

If we assume that the correlation of this linear combination is completely explained by the specific characteristic we considered (i.e., the contributions of other characteristics to the correlation of this linear combination can be neglected), then it would have a correlation coefficient of 2^{-72}. Detecting such a correlation would require at least 2^{144} bits of key stream, which is well above the security requirement.

Other more complicated types of linear characteristics with larger correlations might exist in principle, but given the size of the state and the sparseness of the feedback and feedforward functions, the linear combination given above has a good chance to be optimal, and hence, it seems unlikely that the correlations of other characteristics will exceed 2^{-40}. The preliminary results given by Maximov and Biryukov [80] seem to confirm this.

Table 6.2: Cryptanalytical results

Attack	Time	Data	Reference
Linear distinguisher	2^{144}	2^{144}	Sect. 6.8.1
Guess-and-determine attack	2^{195}	288	Sect. 6.8.3
Guess-and-determine attack	2^{135}	288	[61]
Guess-and-determine attack	2^{90}	2^{61}	[80]
Solving system of equations	2^{164}	288	[92]
Exhaustive key search	2^{80}	80	

6.8.2 Period

Because of the fact that the internal state of TRIVIUM evolves in a nonlinear way, its period is hard to determine. Still, a number of observations can be made. First, if the AND gates are omitted (resulting in a completely linear scheme), one can show that any key/IV pair would generate a stream with a period of at least $2^{96-3} - 1$. This has no immediate implications for TRIVIUM itself, but it might be seen as an indication that the taps have been chosen properly.

Secondly, TRIVIUM's state is updated in a reversible way, and the initialization of $(s_{178}, \ldots, s_{288})$ prevents the state from cycling in less than 111 iterations. If we believe that TRIVIUM behaves as a random permutation after a sufficient number of iterations, then all cycle lengths up to 2^{288} would be equiprobable, and hence the probability for a given key/IV pair to cause a cycle smaller than 2^{80} would be 2^{-208}.

6.8.3 Guess and Determine attacks

In each iteration of TRIVIUM, only a few bits of the state are used, despite the general rule-of-thumb that sparse update functions should be avoided. As a result, guess and determine attacks are certainly a concern. A straightforward attack would guess $(s_{25}, \ldots, s_{93})$, $(s_{97}, \ldots, s_{177})$, and $(s_{244}, \ldots, s_{288})$, 195 bits in total, after which the rest of the bits can immediately be determined from the key stream.

More sophisticated attacks can significantly reduce this number, though. A first idea, proposed by Khazaei [61], is to guess a_{i-109}, b_{i-91}, and c_{i-82} for $i = 0, 2, \ldots, 88$ (we use here the alternative description of Sect. 6.7.3). Once these 135 bits are fixed, it can easily be verified that each key stream bit t_i with $0 \leq i \leq 90 + 66$ is reduced to a linear function in $288 - 135$ unknowns. By solving this linear system for all 2^{135} guesses, the attacker will eventually recover the complete internal state.

A considerably improved guess-and-determine attack is presented by Maximov and Biryukov [80]. Instead of guessing one out of two bits of a, b, and c over a certain interval, the authors propose to guess every third bit. In order to get a solvable linear system, they additionally assume that all three AND gates produce zero bits at every third step over a number of consecutive cycles. This assumption is only fulfilled with a small probability, and the attack will therefore have to be repeated for different positions in the stream. With some additional tricks, and given about 2^{61} bits of known key stream, the attack complexity can be reduced to an estimated 2^{90} key setups.

6.8.4 Algebraic attacks

TRIVIUM seems to be a particularly attractive target for algebraic attacks. The complete scheme can easily be described with extremely sparse equations of low degree. However, its state does not evolve in a linear way, and hence the efficient linearization techniques [25] used to solve the systems of equations generated by LFSR based schemes will be hard to apply. Other techniques might be applicable, though, and their efficiency in solving this particular system of equations needs to be investigated.

Recently, some interesting research has been conducted on this topic by several cryptanalysts. In [92], Raddum presents a new technique to solve systems of equations associated with TRIVIUM. His attack has a very high complexity of $\mathcal{O}(2^{164})$ when applied to the full cipher, but breaks BIVIUM-A, a key stream generator similar to the one shown in Fig. 6.4(a), in a day. This same variant is also analyzed by McDonald et al. [81], who show that its state can be recovered in seconds using off-the-shelve satisfiability solvers. While these experiments are useful to test new techniques, it is important to note that the final remark of Sect. 6.5.2, combined with the use of 1-bit S-boxes, indeed implies a fundamental weakness of two-round ciphers such as BIVIUM-A.

Finally, Fischer and Meier [41] analyze TRIVIUM in the context of algebraic attacks based on augmented functions. They show that TRIVIUM's augmented function can easily be analyzed, and conclude that TRIVIUM seems to be resistant against this particular type of algebraic attacks.

6.8.5 Resynchronization attacks

A last type of attacks are resynchronization attacks, in which the adversary is allowed to manipulate the value of the IV, and tries to extract information about the key by examining the corresponding key stream. TRIVIUM tries to preclude this type of attacks by cycling the state a sufficient number of times before producing any output. It can be shown that each state bit depends on each key and IV bit in a nonlinear way after two full cycles (i.e., $2 \cdot 288$ iterations). We expect that two more cycles will suffice to protect the cipher against resynchronization attacks. This seems to be confirmed by the analysis of Turan and Kara [102].

6.9 Implementation Aspects

We conclude this chapter with a discussion of some implementation aspects of TRIVIUM.

6.9.1 Hardware

As stated in Sect. 6.2.2, our aim was to design a cipher which is compact in environments with restrictions on the gate count, power-efficient on platforms with limited power resources, and fast in applications that require high-speed encryption. In TRIVIUM, this flexibility is achieved by ensuring that state bits are not used for at least 64 iterations after they have been modified. This

Table 6.3: Gate counts of 1-bit to 64-bit hardware implementations

Components	1-bit	8-bit	16-bit	32-bit	64-bit
Flip-flops:	288	288	288	288	288
AND gates:	3	24	48	96	192
XOR gates:	11	88	176	352	704
Estimated NAND gates:	3488	3712	3968	4480	5504
NAND gates, 0.13 µm CMOS [44]	2599	2801	3185	3787	4921

way, up to 64 iterations can be computed at once, provided that the 3 AND gates and 11 XOR gates in the original scheme are duplicated a corresponding number of times. This allows the clock frequency to be divided by a factor 64 without affecting the throughput.

Based on the figures stated in [70] (i.e., 12 NAND gates per Flip-flop, 2.5 gates per XOR, and 1.5 gates per AND), we can compute a first estimation of the gate count for different degrees of parallelization. The actual results found by Good and Benaissa [44] for 0.13 µm Standard Cell CMOS show that these estimations are rather pessimistic, however. Both figures are compared in Table. 6.3.

The hardware efficiency of TRIVIUM has been independently evaluated by several other research teams. Gürkaynak et al. [47] report a 64-bit implementation in 0.25 µm 5-metal CMOS technology with a throughput per area ratio of 129 Gbit/s · mm^2, three times higher than for any other eSTREAM candidate. Gaj et al. [42] come to similar conclusions, and also note that TRIVIUM is perceived to be the easiest eSTREAM candidate to implement amongst students following an introductory course on VHDL at the George Mason University. FPGA implementations of TRIVIUM are independently studied by Bulens et al. [22], Good et al. [45], and Rogawski [97]. The general conclusion, here as well, is that TRIVIUM offers a very good trade-off between throughput and area. Finally, Feldhofer [39] analyzes implementations of TRIVIUM for RFID tags, and shows that the power consumption is reduced to one fourth compared to a low-power AES implementation.

6.9.2 Software

Despite the fact that TRIVIUM does not target software applications, the cipher is still reasonably efficient on a standard PC. The measured performance of the reference C-code on a 1 700 MHz Pentium M processor can be found in Table 6.4.

Table 6.4: Measured performance on an Intel® Pentium™ M CPU 1 700 MHz

Operation		
Stream generation:	5.3	cycles/byte
Key setup:	51	cycles
IV setup:	774	cycles

6.10 Conclusion

In this chapter we have presented a simple synchronous stream cipher called TRIVIUM, which seems to be particularly well suited for application requiring a flexible hardware implementation. The design is based on the study of the propagation of linear characteristics, and shows that the effect of a few small non-linear components can be amplified considerably by a carefully designed linear structure. TRIVIUM is currently being evaluated in the framework of the eSTREAM Stream Cipher Project. It is still in an experimental stage, and further research will reveal whether it meets its security requirements or not.

Chapter 7

Conclusions and Open Problems

To conclude this thesis, we summarize its main contributions and suggest a number of directions for further research.

7.1 Contributions of this Thesis

In this thesis, we have investigated different aspects of symmetric encryption algorithms. After a brief discussion of the main principles of symmetric encryption and an explanation of the rationale behind different types of encryption algorithms, we have concentrated on a number of topics which are all connected by one central theme: the importance of (non)linearity in symmetric cryptography. The specific contributions made in these different topics are discussed below.

- Most attacks against block ciphers are based on the ability to build a distinguisher for the inner rounds. In order to illustrate this principle, we have presented two dedicated attacks, both of which exploit specific weaknesses in the linear structure of the ciphers SAFER++ and ARIA. Apart from being illustrative, these attacks have also introduced some new techniques: an alternative way to mount multiset attacks, and a new type of linear attack operating on word level instead of on bit level.

- Linear cryptanalysis is one of the most important general analysis techniques in symmetric cryptography. In this thesis, we have introduced a rigorous statistical framework based on the concept of Maximum Likelihood, which allows to derive optimal attack strategies capable of exploiting multiple linear approximations. We have derived simple expressions to predict the efficiency improvements achieved by these at-

tacks, and have demonstrated their accuracy by running experiments on round-reduced variants of DES.

- S-boxes, which are typically the only source of non-linearity in a block cipher, play an important role in the protection against linear and differential cryptanalysis. In order to reduce S-boxes to their essence (i.e., their non-linearity), we have presented algorithms which allow to make abstraction of affine mappings at the input and the output of the S-boxes. We have introduced the notion of a linear representative, and have used this concept to partition the set of all $2 \cdot 10^{13}$ 4×4-bit S-boxes into 302 equivalence classes.
- In the last part of this thesis, we have presented a new design strategy for stream ciphers based on the same techniques that are used to protect block ciphers against linear cryptanalysis. We have shown that a carefully designed linear structure can be surprisingly efficient in diffusing the nonlinearity of very few small nonlinear components. Based on this strategy, we have developed a compact and efficient hardware oriented stream cipher called TRIVIUM, which has been submitted to the eSTREAM Stream Cipher Project. At the time of writing, it has successfully passed two selection rounds, and is amongst the 8 hardware candidates retained for the third evaluation phase. Because of its simple structure and efficiency, TRIVIUM has attracted the interest of both cryptanalysts [4, 41, 61, 62, 80, 81, 92, 102] and hardware implementers [22, 39, 42, 44, 46, 47, 63, 97].
- In the appendix, we have presented techniques to generate characteristics for SHA-1. This problem, which could previously only be solved on a case-by-case basis after a long and tedious manual analysis, has been a major obstacle in the development of new collision attacks.

7.2 Future Research

Finally, we suggest some directions for further research.

- Generalizing the concept of byte-wise linear cryptanalysis introduced in the attack on ARIA, and finding links with multiple (bit-wise) linear cryptanalysis.
- Effectively implementing a 16-round DES attack based on multiple linear approximations.
- Completely classifying larger S-boxes is infeasible. Instead, try to extrapolate properties observed for 4×4-bit S-boxes to larger S-boxes.
- Classifying diffusion layers up to a reordering of the input/output words and an affine mapping on each individual input/output word.

- Analyzing the security of TRIVIUM, and trying to formalize the resistance against guess-and-determine attacks.
- Designing a class of light-weight stream ciphers which allow to trade area against security.
- Translating the ideas used in TRIVIUM to hash functions, using the duality between linear and differential cryptanalysis.
- Improving the attacks on hash functions and finding collisions for 80-round SHA-1

Appendix A

Differential Characteristics in SHA-1

In this appendix, we present the results of research that was carried out as part of this doctoral work, but is not directly related to symmetric encryption. The problem considered in the following sections should be put into the context of the recent attacks [105, 106, 107, 108] against several widely known hash functions including SHA-1. A critical step in these attacks is the construction of good differential characteristics. Since algorithms similar to the one presented in Sect. 4.6 turn out to be ineffective in this case, new techniques are required.

A.1 Cryptographic Hash Functions

A cryptographic hash function h is an algorithm which maps a message string m of arbitary length to a string $y = h(m)$ of fixed length n, called hash value. Hash functions are used (and sometimes misused) as an elementary building block in a large variety of cryptographic systems, but their primary use is to assign n-bit 'fingerprints' to messages of arbitrary length, with the intent that no two messages will ever be assigned the same fingerprint. An important class of applications where this property is useful are digital signature schemes: if $h(m)$ can be considered to be a unique representation of m, then, instead of the complete message, it suffices to sign this compact hash value.

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

A.1.1 Security Requirements

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

Collision resistance: it is infeasible in practice to find two messages m and $m^* \neq m$ such that $h(m) = h(m^*)$.

Second preimage resistance: for a given message m, it is infeasible in practice to find a second message $m^* \neq m$ such that $h(m) = h(m^*)$.

Preimage resistance: it is infeasible in practice to find, for a given hash value y, a message m such that $h(m) = y$.

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

The resistance of a hash function to collision and (second) preimage attacks depends in the first place on the length n of the hash value. Regardless of how a hash function is designed, an adversary will always be able to find preimages or second preimages after trying out about 2^n different messages. Finding collisions requires a much smaller number of trials: about $2^{n/2}$, as we will see in Sect. A.2.1. As a result, hash functions producing less than 160 bits of output are currently considered inherently insecure. Moreover, if the internal structure of a particular hash function allows collisions or preimages to be found more efficiently than what could be expected based on its hash length, then the function is considered to be broken.

A.1.2 SHA-1

In the remainder of this appendix, we will focus on one particular hash function: SHA-1 [84]. This algorithm, designed by the US National Security Agency (NSA) in 1995, is widely used, and is representative for a large class of hash functions which started with MD4 and includes most algorithms in use today.

SHA-1 is an iterated hash function which processes messages of up to 2^{64} bits in blocks of 512 bits, and produces a 160-bit hash value. It consists of the iterative application of a compression function (denoted by f in Fig. A.1), which transforms a 160-bit chaining variable H_{j-1} into H_j, based on a message block m_j. A fixed IV, specified in the standard [84], is used to initialize the first chaining variable H_0, and the last chaining variable determines the final hash value.

At the core of the compression function lies an invertible transformation on 160-bit blocks, which takes a 512-bit key as a parameter, and can be seen as a block cipher. SHA-1's compression function is constructed by applying this block cipher to the chaining variable H_{j-1}, using the current 512-bit message

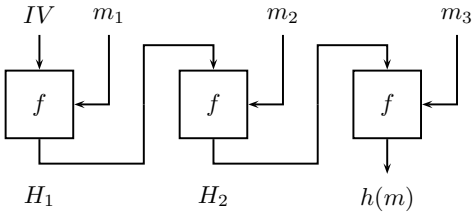


Figure A.1: An iterated hash function

block m_j as a key, and then adding the same H_{j-1} again to the output. This feed-forward construction is very common, and is also known as the Davies-Meyer mode of operation (see [82]).

The block cipher itself consists of two parts: the message expansion and the state update transformation. In the following paragraphs we describe both parts in more detail.

Message Expansion

The purpose of the message expansion is to expand a single 512-bit input message block into eighty 32-bit words $W_0, \ldots, W_{79}$. This is done by splitting the message block into sixteen 32-bit words $M_0, \ldots, M_{15}$, which are then expanded linearly according to the following recursive rule, where the operations $\lll n$ and $\ggg n$ denote rotations over n bit positions to the left and to the right, respectively:

$$W_i = \begin{cases} M_i & \text{for } 0 \leq i < 16, \\ (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \lll 1 & \text{for } 16 \leq i < 80. \end{cases}$$

State Update Transformation

The state update transformation takes as input a 160-bit chaining variable H_{j-1} which is used to initialize five 32-bit registers $A, B, \ldots, E$. These registers, referred to as state variables, are then iteratively updated in 80 steps, indexed by $i = 0, \ldots, 79$. Each step takes as parameters a step constant K_i and one word W_i of the expanded message. A single step consists of the following operations (see also Fig. A.2):

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

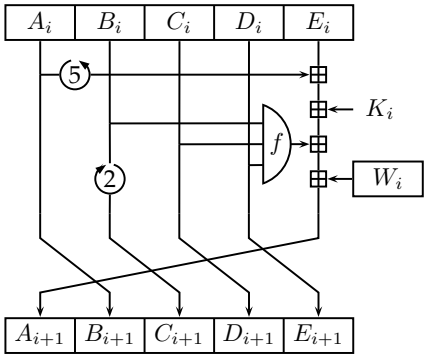


Figure A.2: The state update transformation

The operation f is a boolean function which changes every 20 steps (called a round): the first 20 steps use the function f_{IF}, steps 40 to 59 use the function f_{MAJ}, and the 40 remaining steps use the function f_{XOR}. The definitions of these three functions are given below.

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

The step constants K_i can be found in [84]. Since they do not play any role in this chapter, we do not list them here.

Note that the state update transformation can also be described by a recursive rule in variable A_i only: if we introduce the variables $A_{-1} = B_0$, $A_{-2} = C_0 \lll 2$, $A_{-3} = D_0 \lll 2$, and $A_{-4} = E_0 \lll 2$, then the following recursion holds:

$$A_{i+1} = (A_{i-4} \ggg 2) + (A_i \lll 5) + W_i f(A_{i-1}, A_{i-2} \ggg 2, A_{i-3} \lll 2) + K_i\,.$$

Because of this property, we will only consider the state variable A_i in the remainder of this paper.

A.2 Collision Attacks Revisited

Our objective in this appendix is to develop a method to find SHA-1 characteristics which are suitable for collision attacks. However, in order to solve this problem, we first have to determine exactly what 'suitable' means in this context. In this section, we will therefore consider collision attacks and characteristics in a general setting, and analyze how the choice of the characteristic affects the work factor of the attack.

A.2.1 How Dedicated Collision Attacks Work

If we are given an n-bit hash function whose output values are uniformly distributed and use it to hash an arbitrary pair of messages, then we expect the hash values to collide with a probability of 2^{-n}. Hence, without knowing anything about the internals of the hash function, we should be able to find a collision after trying out 2^n pairs. Since any set of 2^n pairs will do, this approach can be turned into a birthday attack requiring only $2^{n/2}$ hash evaluations.

Instead of testing arbitrary pairs, dedicated collision attacks try to use the internal structure of the hash function to locate a special subset of message pairs which (1) are considerably more likely to collide than random pairs, and (2) can efficiently be enumerated. A particularly effective way to construct such subsets is to restrict the search space to message pairs with a fixed difference. The goal is to pick these differences in such a way that they are likely to propagate through the hash function following a predefined differential characteristic which eventually ends in a zero difference (a collision).

As was observed in [23], the probability for this to happen can be increased by restricting the subset even further and imposing conditions not only on the differences but also on the *values* of specific (expanded) message bits. Moreover, since the internal variables of a hash function only depend on the message (and not on a secret key as for example in block ciphers), we can also restrict the set of message pairs by imposing conditions on the state variables. Depending on their position, however, these conditions might have a considerable impact on the efficiency to enumerate the messages fulfilling them. This important point is analyzed in detail in Sect. A.2.3.

A.2.2 Generalized Characteristics

In order to reflect the fact that both the differences and the actual values of bits play a role in their attack, Wang et al. [107] already extended the notion of differential characteristics by adding a sign to each non-zero bit difference (1 or -1). In this appendix we generalize this concept even further by allowing characteristics to impose arbitrary conditions on the values of pairs of bits.

The conditions imposed by such a generalized characteristic on a particular pair of words (X, X^*) will be denoted by ∇X. In order to make the notation more compact, the pair itself will be written as $X^2 = (X, X^*)$. It will turn out to be convenient to represent ∇X as a set, containing the values for which the conditions are satisfied, for example

$$\nabla X = \{X^2 \mid x_7 \cdot x_7^* = 0,\, x_i = x_i^* \text{ for } 2 \le i < 6,\, x_1 \neq x_1^*, \text{ and } x_0 = x_0^* = 0\}\,.$$

In general, 16 different conditions can be imposed at each bit position, and in order to save space in the rest of this paper, we introduce a symbol for each of them in Table A.1 (including the symbol '#' for the impossible condition). Using this convention, we can rewrite the example above as

$$\nabla X = [\texttt{7?----x0}]\,.$$

Table A.1: Possible conditions on a pair of bits

(x_i, x_i^*)	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
?	✓	✓	✓	✓
-	✓	-	-	✓
x	-	✓	✓	-
0	✓	-	-	-
u	-	✓	-	-
n	-	-	✓	-
1	-	-	-	✓
#	-	-	-	-
3	✓	✓	-	-
5	✓	-	✓	-
7	✓	✓	✓	-
A	-	✓	-	✓
B	✓	✓	-	✓
C	-	-	✓	✓
D	✓	-	✓	✓
E	-	✓	✓	✓

A generalized characteristic for SHA-1 is then simply a pair of sequences $\nabla A_{-4}, \ldots, \nabla A_N$ and $\nabla W_0, \ldots, \nabla W_{N-1}$, which we will represent as a table (see for instance Table A.7).

A.2.3 Work Factor and Probabilities

Let us now assume that we are given a complete characteristic for SHA-1, specified by $\nabla A_{-4}, \ldots, \nabla A_N$ and $\nabla W_0, \ldots, \nabla W_{N-1}$. Our goal is to estimate how much effort it would take to find a pair of messages which follows this characteristic. The algorithm we plan to use is a very simple depth-first search: we first pick a value for W_0^2 such that $W_0^2 \in \nabla W_0$, we compute A_1^2, and verify that $A_1^2 \in \nabla A_1$. Next, we pick $W_1^2 \in \nabla W_1$ such that $A_2^2 \in \nabla A_2$, then $W_2^2 \in \nabla W_2$ such that $A_3^2 \in \nabla A_3$, etc. If, at a certain step i, we cannot find any value W_i^2 which satisfies these conditions, we backtrack.

In order to estimate the work factor of this algorithm, we will compute the expected number of visited nodes in the search tree. But first we introduce some definitions.

Definition A.1. The *message freedom* $F_W(i)$ of a characteristic at step i is the number of ways to choose W_i^2 without violating any (linear) condition imposed on the expanded message, given fixed values W_j^2 for $0 \leq j < i$.

We note that since the expanded message in SHA-1 is completely determined by the first 16 words, we always have $F_W(i) = 1$ for $i \geq 16$.

Definition A.2. The *uncontrolled probability* $P_u(i)$ of a characteristic at step i is the probability that the output A_{i+1}^2 of step i follows the characteristic, given that all input pairs do as well, i.e.,

$$P_u(i) = P\left(A_{i+1}^2 \in \nabla A_{i+1} \mid A_{i-j}^2 \in \nabla A_{i-j} \text{ for } 0 \leq j < 5, \text{and } W_i^2 \in \nabla W_i\right) .$$

Definition A.3. The *controlled probability* $P_c(i)$ of a characteristic at step i is the probability that there exists at least one pair of message words W_i^2 following the characteristic, such that the output A_{i+1}^2 of step i follows the characteristic, given that all other input pairs do as well, i.e.,

$$P_c(i) = P\left(\exists W_i^2 \in \nabla W_i : A_{i+1}^2 \in \nabla A_{i+1} \mid A_{i-j}^2 \in \nabla A_{i-j} \text{ for } 0 \leq j < 5\right) .$$

With the definitions above, we can now easily express the number of nodes $N_s(i)$ visited at each step of the compression function during the collision search. Taking into account that the average number of children of a node at step i is $F_W(i) \cdot P_u(i)$, that only a fraction $P_c(i)$ of the nodes at step i have any children at all, and that the search stops as soon as step N is reached, we can derive the following recursive relation:

$$N_s(i) = \begin{cases} 1 & \text{if } i = N\,, \\ \max\left\{N_s(i+1) \cdot F_W(i)^{-1} \cdot P_u^{-1}(i),\, P_c^{-1}(i)\right\} & \text{if } i < N\,. \end{cases}$$

The total work factor is then given by

$$N_w = \sum_{i=1}^{N} N_s(i)\,.$$

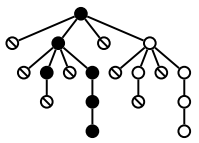
This is the quantity that we would like to minimize when constructing characteristics in Sect. A.3.

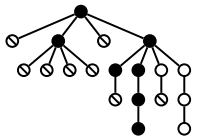
A.2.4 Examples

In order to understand what the different quantities defined above represent, it might be helpful to walk through a small example. Table A.2 shows two hypothetical search trees with corresponding values of F_W, P_u, and P_c. The nodes which are visited by the search algorithm, and hence contribute to the complexity of the collision search, are filled. Note that the values of $P_c(i)$ do not always influence the complexity of the attack. The trees in Table A.2, however, are examples where they do.

Let us now illustrate the previous concepts with two examples on 64-step SHA-1. In the first example, shown in Table A.3, we consider a generalized characteristic which does not impose any conditions, except for a fixed IV value at the input of the compression function and a collision at the output $(A_{64-4}, \ldots, A_{64})$. The values of $N_s(i)$ in the table tell us that the search algorithm is expected to traverse nearly the complete compression function 2^{160}

Table A.2: How P_c affects the search tree

i	tree[a]	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
0:		4	1/2	1	1
1:		4	1/2	1	1
2:		1	1/2	1/2	2
3:		1	1	1	1
4:					1

i	tree	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
0:		4	1/2	1	1
1:		4	1/2	1/2	2
2:		1	1/2	1/2	2
3:		1	1	1	1
4:					1

[a]Both ○ and ● represent values of W_{i-1}^2 which lead to a consistent A_i^2; the nodes visited by the search algorithm are filled. Inconsistent values are denoted by ⊘.

Table A.3: Example 1, collision without additional conditions

i	∇A_i	∇W_i	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
-4:	00001111010010111000011111000011					
-3:	01000000110010010101000111011000					
-2:	01100010111010110111001111111010					
-1:	11101111110011011010101110001001					
0:	01100111010001010010001100000001	????????????????????????????????	64	0.00	0.00	0.00
1:	????????????????????????????????	????????????????????????????????	64	0.00	0.00	0.00
	...	...				
12:	????????????????????????????????	????????????????????????????????	64	0.00	0.00	0.00
13:	????????????????????????????????	????????????????????????????????	64	0.00	0.00	0.00
14:	????????????????????????????????	????????????????????????????????	64	0.00	0.00	32.00
15:	????????????????????????????????	????????????????????????????????	64	0.00	0.00	96.00
16:	????????????????????????????????	????????????????????????????????	0	0.00	0.00	160.00
17:	????????????????????????????????	????????????????????????????????	0	0.00	0.00	160.00
	...	...				
59:	????????????????????????????????	????????????????????????????????	0	-32.00	0.00	160.00
60:	--------------------------------	????????????????????????????????	0	-32.00	0.00	128.00
61:	--------------------------------	????????????????????????????????	0	-32.00	0.00	96.00
62:	--------------------------------	????????????????????????????????	0	-32.00	0.00	64.00
63:	--------------------------------	????????????????????????????????	0	-32.00	0.00	32.00
64:	--------------------------------					

times before finding a colliding pair (note that from here on all values listed in tables will be base 2 logarithms).

In the example of Table A.4, we force the state variables and the expanded message words to follow a given differential characteristic starting from the output of the 16th step (i.e., $A_{16}, \ldots, E_{16}$). How such sparse differential characteristics can be found will be briefly explained in Sect. A.3. The most significant effect is that the five consecutive uncontrolled probabilities of 2^{-32} in the previous example move up to steps 11–15, where their effect on the number of nodes is completely neutralized by the degrees of freedom in the expanded message, resulting in a considerable reduction of the total work factor.

The examples above clearly show that small probabilities have a much larger impact on the work factor when they occur after step 16 (where $F_W(i) = 1$). Therefore, when constructing characteristics, we will in the first place try to optimize the probabilities in the second part of the compression function (steps 16 to $N-1$), even if this comes at the cost of a significant decrease of probabilities in the first part.

A.3 Constructing Characteristics

Having the necessary tools to estimate the work factor corresponding to any given generalized characteristic, we now turn to the problem of finding characteristics which minimize this work factor.

The search method presented in this section constructs characteristics by iteratively adding more conditions as long as it improves the work factor. During this process, two important tasks need to be performed: (1) determining when and where to add which condition, and (2) letting conditions propagate and avoiding inconsistent conditions. We first discuss the second problem.

A.3.1 Consistency and Propagation of Conditions

When analyzing the interaction of bit conditions imposed at the inputs and the outputs of a single step of the state update transformation, three situations can occur: (1) the conditions are inconsistent, (2) the conditions are consistent as such, and (3) the conditions are consistent, provided that a number of additional bit conditions are fulfilled as well (the conditions are said to propagate). This third case is illustrated in Table A.5, where the conditions imposed on the expanded message words in the previous example propagate to the state variables. It should be noted that such consistency checks can be implemented in a very efficient way using trellises, thanks to the fact that bits at different bit positions only interact through the carries of the integer additions.

Table A.4: Example 2, imposing a sparse diff. characteristic from step 16 on

i	∇A_i	∇W_i	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
	...					
0:	01100111010001010010001100000001	-xx----------	32	0.00	0.00	0.00
1:	????????????????????????????????	xxx----------x-x-x-	32	0.00	0.00	0.00
	...	...				
7:	????????????????????????????????	-xx----------xx--x-	32	0.00	0.00	0.00
8:	????????????????????????????????	-xx----------x----xx	32	0.00	0.00	5.00
9:	????????????????????????????????	--x----------x------	32	0.00	0.00	37.00
10:	????????????????????????????????	xxx----------x----x-	32	0.00	0.00	69.00
11:	????????????????????????????????	-xx----------x-	32	-32.00	-29.00	101.00
12:	x----------	x----------x	32	-32.00	-31.00	101.00
13:	x----------	----------x-----	32	-32.00	-31.00	101.00
14:	----------	----------xx	32	-32.00	-31.19	101.00
15:	x----------xx	-x----------x-x--x-	32	-32.00	-27.83	101.00
16:	----------x-	-x----------x------	0	-7.00	-4.00	101.00
17:	x----------x-	xxx----------x-x--x-	0	-7.00	-2.00	94.00
18:	----------	x-x----------	0	-5.00	-3.00	87.00
19:	----------x-	x----------x------	0	-4.00	-3.00	82.00
	...	...				
49:	----------x-	----------x------	0	-2.00	-1.00	7.00
50:	----------	x----------x-	0	-3.00	-2.00	5.00
51:	----------	----------	0	-1.00	-1.00	2.00
52:	----------	x----------	0	-1.00	-1.00	1.00
53:	----------	x----------	0	0.00	0.00	0.00
54:	----------	----------	0	0.00	0.00	0.00
	...	...				
60:	----------	----------	0	0.00	0.00	0.00
61:	----------	----------	0	0.00	0.00	0.00
62:	----------	----------	0	0.00	0.00	0.00
63:	----------	----------	0	0.00	0.00	0.00
64:	----------					

Table A.5: Propagation of conditions in Example 2

i	∇A_i	∇W_i	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
	...					
0:	01100111010001010010001100000001	-xx----------	32	0.00	0.00	0.00
1:	??x----------	xxx----------x-x-x-	32	0.00	0.00	0.00
2:	??????????????????????????????x-	--x----------x----xx	32	0.00	0.00	0.00
3:	????????????????????????????????	x-xx----------x------	32	0.00	0.00	0.00
	...	...				

A.3.2 Determining Which Conditions to Add

In Sect. A.2.4 we noted that conditions in a characteristic affect the work factor in very different ways depending on the step where they are enforced. This is also reflected in the procedure which we are about to propose: in order to determine where to add which conditions, we will proceed in a number of distinct stages, each of which tries to optimize a specific part of the generalized characteristic.

Note that the second stage in this process has long been an important obstacle, which could only be overcome by a long and tedious manual analysis of differential characteristics. This explains why many researchers who tried to improve the attacks on SHA-1 were forced to use the exact same characteristics as those originally found by Wang et al. [107].

Stage 1

As observed in Sect. A.2.4, the work factor of the collision search algorithm is mainly determined by the shape of the characteristic after step 16. Hence, our first goal is to find a high probability differential characteristic, which can start with any difference in the state variables after step 16, but ends in a zero difference in the last step (later on, when we consider multi-block collisions, this constraint will be removed as well).

In general, the sparser a differential characteristic, the higher its probability, and in the case of the SHA family, it has been shown before that sparse characteristics can easily be found by linearizing all components of the state update transformation, representing the resulting compression function as a linear code, and searching for low-weight vectors (see [58, 91, 93, 107]).

Once a suitable differential characteristic is found for the linearized variant (called an L-characteristic), we will use it to fix the differences in all state variables and expanded message words from step 16 on. Because of the linearity of the message expansion, this will automatically fix the difference in the first 16 message words as well.

Stage 2

At this point, the largest part of the work factor is most likely concentrated in step 16 (see, e.g., Table A.4), where the state variables $A_{16-4}, \ldots, A_{16}$, which are not constrained in any way in the previous steps, are suddenly forced to follow a fixed difference. In order to eliminate this bottleneck, we want to guide the state variables to the target difference by imposing conditions to the first steps as well.

Although the probability of this part of the characteristic is not as critical as before, we still want the differences to be reasonably sparse. Unfortunately, because of the high number of constraints (the message difference and both the differences at the input of the first step and at the output of step 16 are fixed already), suitable L-characteristics are extremely unlikely to exist in this

case. In order to solve this problem, we introduce a probabilistic algorithm which bears some resemblance to the algorithms used to find low-weight code words, but instead of feeding it with a linear code, we directly apply it to the unmodified (non-linear) compression function.

The basic idea of the algorithm is to randomly pick a bit position which is not restricted yet (i.e., a '?'-bit), impose a zero-difference at this position (a '-'-bit), and calculate how the condition propagates. This is repeated until all unrestricted bits have been eliminated, or until we run into an inconsistency, in which case we start again. The algorithm can be optimized in several ways, for example by also picking 'x'-bits once they start to appear, guessing the sign of their differences ('u' or 'n'), and backtracking if this does not lead to a solution. It turns out that inconsistencies are discovered considerably earlier this way.

An interesting property of the proposed procedure is that the sparser a characteristic, the higher the probability that it will be discovered. The number of trials before a consistent characteristic is found, is very hard to predict, though. Experiments show that this number can range from a few hundreds to several hundreds of thousands.

Stage 3

In the final stage, we try to further improve the work factor corresponding to the characteristic by performing local optimizations. To this end, we run through all bit positions of every state variable and every expanded message word, check which conditions can be added to improve the total work factor, and finally pick the position and corresponding condition which yields the largest gain. By repeating this many times, we can gradually improve the work factor. The example in Table A.6 shows how our previous characteristic looks like after applying this greedy approach for a number of iterations. A possible further improvement of this approach, which has been investigated recently, is to also consider the joint effect on the work factor of a number of special combinations of conditions.

An interesting issue here, is when to stop adding new conditions. In order to answer this question, we first notice that every additional condition reduces the size of the search tree, but at the same time lowers the expected number of surviving leaves at step N. In general, the work factor will improve as long as the search tree is reduced by a larger factor than the number of surviving leaves. At some point, however, the expected number of leaves will drop below one, meaning that message pairs which actually follow the characteristic are only expected to exist with a certain probability. This is not necessarily a problem if we are prepared to repeat the search for a number of different characteristics, and in fact, that is exactly how we constructed the second block of the 64-step collision presented in the next section. In this case, three different characteristics were used, the third of which is shown in Table A.8 (notice that the expected number of characteristics needed to find one surviving leave

Table A.6: Example 3, after adding conditions to minimize workfactor

i	∇A_i	∇W_i	P_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
	...					
0:	01100111010001010010001100000001	0uu01010110011010000111101110101	0	0.00	0.00	0.00
1:	n0n01010100000001101010000010100 0	unn00001000110100010110111u1u0n0	0	0.00	0.00	0.00
2:	00u1unnnnnnnnnnnnnnnnnnnnnnn01u0	00n11101001100111111111011n1011uu	0	0.00	0.00	0.00
3:	10001010011001001001111u11100u111	n0un01100001101011001101 0u111100	0	0.00	0.00	0.00
4:	u000u01n11uu010u11u10100101010u0	un0n01101001000010001 0110n1u01uu	0	0.00	0.00	0.00
5:	n01001000n100011n1n000101uu0n010	uu1n10101111100111011 10110n000u0	0	0.00	0.00	0.00
6:	010100110n0101u00100001000001100	10n10000111111000000000000010011	0	0.00	0.00	0.00
7:	1011111unnnnnnnnnn100000nu101n10	1nu01000000101110 01----001nu01u1	4	-1.00	0.00	0.00
8:	n11001101110000000101---00110nu00	0nu1101110111-----------u0011nu	12	-8.00	0.00	0.00
9:	n01010010000111101110----n10111n	11u1100001111-----------0u100111	11	-0.13	0.00	0.00
10:	n011010010111----------000000n0	nnn111101---------------n1010u0	16	-4.00	-0.68	0.68
11:	u0110101011------------n1100100	1un1001-0----------------0011u1	18	-6.00	-1.68	5.36
12:	u0010100101------------0-110001	u10110-0-0---------------11000u	18	-11.00	-2.96	17.36
13:	u11100101110010----------0100000	0010010100000------------u00101	13	-4.00	-2.42	24.36
14:	01110011011111------------11000	1001000111111------------1001uu	11	-3.00	-2.00	33.36
15:	u1010110101-1------------1001uu	0n110--0----------------n0n00n0	19	-10.14	-0.14	41.36
16:	11000110000000----------110n0	1u0100101000-----------u100100	0	0.00	0.00	50.22
17:	u000111011-----------------11u1	unn11101000000----------n0n10n1	0	-0.22	-0.21	50.22
18:	11101----------------------1001	n1u0--1----------------01100101	0	-1.00	-0.48	50.00
19:	--0----------------------1u1	u00110-0----------------n101011	0	-1.00	-0.54	49.00
20:	----0----------------------1--	10u00-1-1----------------011100n	0	0.00	0.00	48.00
21:	---------------------------u	00n--0----------------nu01010	0	0.00	0.00	48.00
22:	----------------------------	n1000-0----------------010010u	0	-1.00	-1.00	48.00
	...	...				
60:	----------------------------	------------------------0-------	0	0.00	0.00	0.00
61:	----------------------------	-----------------------1-0------	0	0.00	0.00	0.00
62:	----------------------------	-----------------------1-1-----	0	0.00	0.00	0.00
63:	----------------------------	------------------------0--------	0	0.00	0.00	0.00
64:	----------------------------					

can directly be read from $N_s(0)$, in this example $2^{1.24} \approx 3$). Coming back to our original question, we can conclude that we should in principle continue adding conditions as long as the gain in work factor justifies the cost of generating additional characteristics.

A.4 A Collision for 64-Step SHA-1

To conclude this chapter, we show an example of a 64-step SHA-1 collision found using the techniques described in this chapter. The colliding messages consist of two blocks, where the difference at the output of the first iteration of the compression function cancels the difference at the output of the second iteration through the feed-forward. The generalized characteristics for both blocks were constructed in three stages as explained earlier, and are shown in Tables A.7 and A.8. A simple depth-first search over about 2^{40} nodes, which corresponds to a computational effort equivalent to 2^{35} evaluations of the compression function, was used to construct a message pair which actually follows this characteristic. The colliding messages are given in Table A.9.

Note that, by applying the improvement in Stage 3 briefly mentioned earlier, this result can been extended to compute a 70-step SHA-1 collision within an effort of 2^{44} compression function evaluations. More details will be published in [35].

Table A.7: Characteristic used for the first block of the 64-step collision

i	∇A_i	∇W_i	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
-4:	00001111010010111000011111000011					
-3:	01000000110010010101000011011000					
-2:	01100010111010110111001111111010					
-1:	11101111110011011010101110001001					
0:	01100111010001010010001100000001	01100011110110101111011110111nu	0	0.00	0.00	1.07
1:	00000011100011111000100010010000n	0n11000010100000110l0-010u1n01u1	1	0.00	0.00	1.07
2:	0n0010010100001010110-00011u0un0	0u01001011101101----11011n100100	4	-3.00	0.00	2.07
3:	1u10100001110010100-1un110nuu110	unn100000000000-1001----10u0u11u1	5	-4.00	0.00	3.07
4:	1un0010110011110un1100-0n1n11nu1	n0n01101101101001-01111-10110101	2	-2.00	0.00	4.07
5:	n1u10110101un00010nu10u111000010	u1100101101000111111----1n101011	4	-4.00	0.00	4.07
6:	100u100u01111nu00u1110nu111u1un1	10u01110011001101-1------101011n	7	-5.00	0.00	4.07
7:	nn1100101n1101011-1111-11u1001u0	00n100101010-101------100nu11111	7	-5.00	0.00	6.07
8:	01110111001100u00010--0n11110u11	u1010000001100---00---11-000010u	7	-6.00	0.00	8.07
9:	1n1u000101uuuu0uu1110-1010n110n0	1n00010100000101-100--10-u1111n0	4	-3.00	0.00	9.07
10:	101100010n11111n111u-01n00un100	nu1101010110001--011----1u0110un	6	-5.00	0.00	10.07
11:	nnnnnnnnnnnnnnnnnnnnnnnn-nnnnn0n1	1u01111111111111---------0u110n1	9	-9.00	0.00	11.07
12:	00110100000011110110000110011000	0101100101101101101101---1-0101nu	4	-3.00	0.00	11.07
13:	01000000000010000001111100-011000	0n001000010101----------n1010n1	11	-4.00	0.00	12.07
14:	100110000100011000-0------0110101	nu00001010011----------1n1100uu	11	-2.00	0.00	19.07
15:	11011010111111--1---------00010n	uu10110101O-1-1--------1-1n011n1	11	-0.07	0.00	28.07
16:	11111100----------------0-0111	1101001010100----------1010101u	0	-1.00	-1.00	39.00
17:	0000---------------------1-1111	1u0011100111----------111011u0	0	-1.00	-0.99	38.00
18:	----0----------------------01u-	un00111011-0-0---------0n0011nu	0	0.00	0.00	37.00
19:	----------------------------n	1u1100011111----------1un011n0	0	0.00	0.00	37.00
20:	------------------------------	n1101001100-------------011000n	0	-1.00	-1.00	37.00
21:	----------------------------n-	1u1000110-1-0----------0u1000n0	0	-2.00	-2.00	36.00
22:	----------------------------n-	1n0111010011------------0u0110n1	0	-2.00	-2.00	34.00
23:	------------------------------	0n10011011-------------011111n0	0	-1.00	-1.00	32.00
24:	------------------------------	00101001-0-0-----------001010n1	0	-1.00	-1.00	31.00
25:	----------------------------n-	0001110111-------------1u100100	0	0.00	0.00	30.00
26:	------------------------------	n00010000--------------0-11111n1	0	-1.00	-1.00	30.00
27:	------------------------------	n001111-1-1------------11101010	0	0.00	0.00	29.00
28:	------------------------------	u10111110---------------11001n0	0	-1.00	-1.00	29.00
29:	----------------------------n-	n0011100---------------1u110010	0	0.00	0.00	28.00
30:	------------------------------	001010-1-1------------101010110	0	-2.00	-2.00	28.00
31:	----------------------------n-	u0110101---------------0u110111	0	0.00	0.00	26.00
32:	------------------------------	u101001----------------011111010	0	-2.00	-2.00	26.00
33:	----------------------------u-	00010-1-0-------------110n100000	0	0.00	0.00	24.00
34:	------------------------------	u011010----------------001101110	0	-2.00	-2.00	24.00
35:	----------------------------n-	101111----------------010u111001	0	0.00	0.00	22.00
36:	------------------------------	n111-1-1---------------1010110u0	0	-1.00	-1.00	22.00
37:	------------------------------	110110-----------------100000000	0	0.00	0.00	21.00
38:	------------------------------	n1001------------------010111110	0	0.00	0.00	21.00
39:	------------------------------	u11-0-1----------------101101011	0	0.00	0.00	21.00
40:	------------------------------	01010------------------01011100	0	0.00	0.00	21.00
41:	------------------------------	1011-------------------100100000	0	0.00	0.00	21.00
42:	------------------------------	00-0-0-----------------100111001	0	0.00	0.00	21.00
43:	------------------------------	1101-------------------001111011	0	0.00	0.00	21.00
44:	------------------------------	011--------------------10010000	0	0.00	0.00	21.00
45:	------------------------------	1-1-0------------------101111000	0	0.00	0.00	21.00
46:	------------------------------	110------------------1011010010	0	0.00	0.00	21.00
47:	------------------------------	01---------------------101011000	0	0.00	0.00	21.00
48:	------------------------------	-0-0-------------------101100001	0	0.00	0.00	21.00
49:	------------------------------	10---------------------1101010111	0	0.00	0.00	21.00
50:	------------------------------	0----------------------1010101n11	0	-1.00	-1.00	21.00
51:	----------------------------n--	0-1------------------10u100011-	0	0.00	0.00	20.00
52:	------------------------------	1----------------------001000u11	0	-1.00	-1.00	20.00
53:	------------------------------	---------------------110111n00u	0	-2.00	-2.00	19.00
54:	----------------------------n---	-1---------------------u111011-u	0	-1.00	-1.00	17.00
55:	------------------------------	----------------------101010u00u	0	-1.00	-1.00	16.00
56:	------------------------------	----------------------0111n10u-	0	-2.00	-1.91	15.00
57:	----------------------------n----	0----------------------u111000-u-	0	-1.00	-1.00	13.00
58:	------------------------------	----------------------0-1010un1u-	0	-2.00	-1.83	12.00
59:	----------------------------u---	----------------------1n01n11u--	0	-2.00	-1.87	10.00
60:	------------------------n-----	----------------------u-11000xu-0	0	-2.00	-1.00	8.00
61:	------------------------------	----------------------0000n01ux-	0	-2.00	-1.00	6.00
62:	------------------------------	----------------------1000n00n-x-	0	-3.00	-1.89	4.00
63:	------------------------n------	----------------------u-10010-n-n-	0	-1.00	-1.00	1.00
64:	------------------------------					

Table A.8: Third characteristic used for the 2nd block of the 64-step collision

i	∇A_i	∇W_i	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
-4:	11110011111100010000010000n10011					
-3:	01101110111000001010001110011101					
-2:	11001011101100100011110111000100					
-1:	10010110111101001001110011n110101					
0:	10100000000011110111001010110100O	00111011001010101011O1-0111000nu	1	0.00	0.00	1.24
1:	11110010011100101110010-10000n1nu	1n1010101101000---0100101u1n11u1	3	-3.00	0.00	2.24
2:	uu10001001100001000001nu01un01u0	0u1001010011100010011--11n101110	2	-2.00	0.00	2.24
3:	0u1001011011110000000nnnn01011u1nn	nun11101111010100100011010n0u01n0	0	0.00	0.00	2.24
4:	0nu1110110110n0010uuuu0uunuu1u10	n0n11101101111100010001000001110	0	0.00	0.00	2.24
5:	1000111nu111u0001n11111100100001	u01010110100011010-00101-u100000	2	-1.00	0.00	2.24
6:	u11110un101n0u0111-1011n010u1010	10n1011011100-10110----10011011u	5	-2.00	0.00	3.24
7:	u001u11nn0101011100n---0u011n111	11u10011111001----00-0-10uu00011	6	-4.00	0.00	6.24
8:	1n0101010010u01n10000-0-11000u011	u011110010110----0----1-001110n	9	-7.00	0.00	8.24
9:	01001u1n10100110100101-1-uu10100	1n11110101100----------u0001n0	12	-10.00	0.00	10.24
10:	uuuuuuuuuuuuuuuuuuuuuu--1100u011	nu01000011000100--------n1001nu	9	-8.00	0.00	12.24
11:	01001110111111000111111un-0111100	1n00101101111001001------1n001u0	6	-6.00	0.00	13.24
12:	11000000101111111111111111u110	10101001101100100100---001010nn	3	-2.00	-1.00	13.24
13:	01100001011111111111111--0110110n	1n0000010111011111-------n1110u0	8	-2.24	0.00	14.24
14:	0101111110011010110--------010u0	uu01000110110----------1u0-11nn	12	-4.00	0.00	20.00
15:	01010010010000010----------00nu	un110110000-0-0--------1-0n000n1	11	-1.00	0.00	28.00
16:	001001001011---------------10010	1100010100000----------1101001n	0	0.00	0.00	38.00
17:	100000----------------------1000	0n1101111101-----------11-001u1	0	-1.00	-0.99	38.00
18:	----0----------------------0u1	nn11101111-0-1---------0n0010nu	0	0.00	0.00	37.00
19:	----------------------------n	0u1100011010-----------1un000n1	0	-1.00	-1.00	37.00
20:	-0------------------------------	n0101010011------------11-10110n	0	-1.00	-1.00	36.00
21:	----------------------------n-	1u0001000-0-0----------0u1000n1	0	-1.00	-1.00	35.00
22:	----------------------------n-	0n010001010------------0u-011n1	0	-2.00	-2.00	34.00
23:	------------------------------	1n10010111-------------00101n1	0	-1.00	-1.00	32.00
24:	------------------------------	11011111-0-1-----------000101n1	0	-1.00	-1.00	31.00
25:	----------------------------n-	0010000100-------------0u010000	0	0.00	0.00	30.00
26:	------------------------------	u10011101--------------001000u0	0	-1.00	-1.00	30.00
27:	------------------------------	n100100-0-0------------01010001	0	0.00	0.00	29.00
28:	------------------------------	u11001101--------------0-0-100n0	0	-1.00	-1.00	29.00
29:	----------------------------n-	n1111011---------------1u110000	0	0.00	0.00	28.00
30:	------------------------------	100110-1-1------------00--00100	0	-2.00	-2.00	28.00
31:	----------------------------u-	u0000101---------------1n000111	0	0.00	0.00	26.00
32:	------------------------------	u011010----------------0001111100	0	-2.00	-2.00	26.00
33:	----------------------------n-	11111-0-0--------------0-u100101	0	0.00	0.00	24.00
34:	------------------------------	u011010----------------0-0000000	0	-2.00	-2.00	24.00
35:	----------------------------u-	100100----------------010n011010	0	0.00	0.00	22.00
36:	------------------------------	n100-0-1-------------0-1-11010n0	0	-1.00	-1.00	22.00
37:	------------------------------	010111-----------------100001001	0	0.00	0.00	21.00
38:	------------------------------	u0001------------------0-0001101	0	0.00	0.00	21.00
39:	------------------------------	u00-0-0----------------101010100	0	0.00	0.00	21.00
40:	------------------------------	11110------------------010000101	0	0.00	0.00	21.00
41:	------------------------------	0011-------------------011010010	0	0.00	0.00	21.00
42:	------------------------------	00-1-0-----------------01-001100	0	0.00	0.00	21.00
43:	------------------------------	1010-------------------001111100	0	0.00	0.00	21.00
44:	------------------------------	000------------------11-0011100	0	0.00	0.00	21.00
45:	------------------------------	0-1-0------------------1-1100101	0	0.00	0.00	21.00
46:	------------------------------	000------------------0---010010	0	0.00	0.00	21.00
47:	------------------------------	11---------------------001010101	0	0.00	0.00	21.00
48:	------------------------------	-0-0-------------------1100111001	0	0.00	0.00	21.00
49:	------------------------------	10---------------------0-1111110	0	0.00	0.00	21.00
50:	------------------------------	0----------------------11-1100n10	0	-1.00	-1.00	21.00
51:	----------------------------n--	1-0------------------10u010110-	0	0.00	0.00	20.00
52:	------------------------------	1----------------------0000001u10	0	-1.00	-1.00	20.00
53:	------------------------------	---------------------011011n10u	0	-2.00	-2.00	19.00
54:	----------------------------n---	-1---------------------u-11011-u	0	-1.00	-1.00	17.00
55:	------------------------------	----------------------111011u01u	0	-1.00	-1.00	16.00
56:	------------------------------	----------------------01-00n10u-	0	-2.00	-1.91	15.00
57:	----------------------------n----	1----------------------u101111-u-	0	-1.00	-1.00	13.00
58:	------------------------------	----------------------10-00un0u-	0	-2.00	-1.83	12.00
59:	----------------------------u---	----------------------0n01u11u--	0	-2.00	-1.87	10.00
60:	------------------------u-----	----------------------n-0-111xu-0	0	-2.00	-1.00	8.00
61:	------------------------------	----------------------0100u01ux-	0	-2.00	-1.00	6.00
62:	------------------------------	----------------------0-u11n-x-	0	-3.00	-1.89	4.00
63:	------------------------u------	----------------------n-10110-u-n-	0	-1.00	-1.00	1.00
64:	------------------------------					

Table A.9: A 64-step two-block collision

m_1	m_1^*	$m_1 \oplus m_1^*$	m_2	m_2^*	$m_2 \oplus m_2^*$
63DAEFDD	63DAEFDE	00000003	3B2AB4E1	3B2AB4E2	00000003
30A0D167	70A0D135	40000052	AAD112EF	EAD112BD	40000052
52EDCDA4	12EDCDE4	40000040	669C9BAE	269C9BEE	40000040
90012F5F	70012F0D	E0000052	5DEA4D14	BDEA4D46	E0000052
0DB4DFB5	ADB4DFB5	A0000000	1DBE220E	BDBE220E	A0000000
E5A3F9AB	65A3F9EB	80000040	AB46A5E0	2B46A5A0	80000040
AE66EE56	8E66EE57	20000001	96E2D937	B6E2D936	20000001
12A5663F	32A5665F	20000060	F3E58B63	D3E58B03	20000060
D0320F85	50320F84	80000001	BE594F1C	3E594F1D	80000001
8505C67C	C505C63E	40000042	BD63F044	FD63F006	40000042
756336DA	B5633699	C0000043	50C42AA5	90C42AE6	C0000043
DFFF4DB9	9FFF4D9B	40000022	8B793546	CB793564	40000022
596D6A95	596D6A96	00000003	A9B24128	A9B2412B	00000003
0855F129	4855F16B	40000042	816FD53A	C16FD578	40000042
429A41B3	829A41F0	C0000043	D1B663DC	11B6639F	C0000043
ED5AE1CD	2D5AE1EF	C0000022	B615DD01	7615DD23	C0000022
hash:	A750337B	55FFFDBB	C08DB36C	0C6CFD97	A12EFFE0

A.5 Conclusions

In this appendix, we have studied the problem of finding characteristics in SHA-1, which is an important step in the development of new collision attacks. We have introduced the new concept of a generalized characteristic, and have shown how to compute the effect of such characteristics on the total computational effort of the attack. Finally, we have proposed a method to automate the construction of generalized characteristics which reduce this computational effort.

Bibliography

[1] R. Anderson and M. Roe. A5. `http://jya.com/crack-a5.htm`, 1994. 17

[2] R. J. Anderson, E. Biham, and L. R. Knudsen. Serpent: A new block cipher proposal. In S. Vaudenay, editor, *Fast Software Encryption, FSE'98*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer-Verlag, 1998. 97

[3] ARIA Development Team. Block encryption algorithm ARIA. In *Workshop on Information Security and Cryptography (WISC 2003)*, Sept. 2003. In Korean. 43

[4] S. Babbage. Some thoughts on Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/007, Jan. 2007. `http://www.ecrypt.eu.org/stream`. 124

[5] S. Babbage, C. De Cannière, J. Lano, B. Preneel, and J. Vandewalle. Cryptanalysis of SOBER-t32. In T. Johansson, editor, *Fast Software Encryption, FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 111–128. Springer-Verlag, 2003. 4

[6] E. Barkan and E. Biham. In how many ways can you write Rijndael? In Y. Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 160–175. Springer-Verlag, 2002. 96, 97

[7] E. Barkan and E. Biham. The book of rijndaels. Cryptology ePrint Archive, Report 2002/158, 2002. `http://eprint.iacr.org/`. 97

[8] P. S. L. M. Barreto and V. Rijmen. The KHAZAD legacy-level block cipher. Primitive submitted to NESSIE, Sept. 2000. 97

[9] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, page 394. IEEE Computer Society, 1997. 21

The numbers following the references refer to the pages on which they are cited.

[10] D. J. Bernstein. Understanding brute force. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/036, Apr. 2005. `http://www.ecrypt.eu.org/stream`. 25

[11] D. J. Bernstein. Re: A reformulation of TRIVIUM. Posted on the eSTREAM Forum, Feb. 2006. `http://www.ecrypt.eu.org/stream/phorum/read.php?1,448`. 116

[12] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. In A. Menezes and S. A. Vanstone, editors, *Advances in Cryptology – CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer-Verlag, 1991. 28

[13] E. Biham and A. Shamir. Differential cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 156–171. Springer-Verlag, 1992. 28

[14] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993. 28

[15] E. Biham, A. Biryukov, and A. Shamir. Miss in the middle attacks on IDEA and Khufu. In L. R. Knudsen, editor, *Fast Software Encryption, FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 124–138. Springer-Verlag, 1999. 31

[16] A. Biryukov and C. De Cannière. Block ciphers and systems of quadratic equations. In T. Johansson, editor, *Fast Software Encryption, FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 274–289. Springer-Verlag, 2003. 4

[17] A. Biryukov and A. Shamir. Structural cryptanalysis of SASAS. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 394–405. Springer-Verlag, 2001. 35

[18] A. Biryukov, C. De Cannière, A. Braeken, and B. Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 33–50. Springer-Verlag, 2003. 3, 4

[19] A. Biryukov, C. De Cannière, and G. Dellkrantz. Cryptanalysis of SAFER++. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 195–211. Springer-Verlag, 2003. 3, 4, 38

[20] A. Biryukov, C. De Cannière, J. Lano, S. B. Örs, and B. Preneel. Security and performance analysis of ARIA. Final report, K.U.Leuven ESAT/SCD-COSIC, 2004. 3, 4

[21] A. Biryukov, C. De Cannière, and M. Quisquater. On multiple linear approximations. In M. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004. 3, 4, 35

[22] P. Bulens, K. Kalach, F.-X. Standaert, and J.-J. Quisquater. FPGA implementations of eSTREAM Phase-2 focus candidates with hardware profile. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/024, Jan. 2007. `http://www.ecrypt.eu.org/stream`. 120, 124

[23] F. Chabaud and A. Joux. Differential collisions in SHA-0. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer-Verlag, 1998. 131

[24] J. Choi, D. Hong, S. Hong, and S. Lee. Linear attack using multiple linear approximations. *IEICE Transactions*, 88-A(1):2–8, 2005. 49

[25] N. T. Courtois and W. Meier. Algebraic attacks on stream ciphers with linear feedback. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer-Verlag, 2003. 119

[26] J. Daemen. *Cipher and hash function design. Strategies based on linear and differential cryptanalysis*. PhD thesis, Katholieke Universiteit Leuven, Mar. 1995. 7, 100

[27] J. Daemen and C. S. K. Clapp. Fast hashing and stream encryption with PANAMA. In S. Vaudenay, editor, *Fast Software Encryption, FSE'98*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74. Springer-Verlag, 1998. 101

[28] J. Daemen and V. Rijmen. *The Design of Rijndael: AES — The Advanced Encryption Standard*. Springer-Verlag, 2002. 37, 96, 103, 110

[29] J. Daemen, L. R. Knudsen, and V. Rijmen. The block cipher Square. In E. Biham, editor, *Fast Software Encryption, FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer-Verlag, 1997. 21, 35

[30] C. De Cannière. TRIVIUM: A stream cipher construction inspired by block cipher design principles. In S. K. Katsikas, J. Lopez, M. Backes, S. Gritzalis, and B. Preneel, editors, *Information Security, ISC 2006*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer-Verlag, 2006. 4

[31] C. De Cannière and B. Preneel. TRIVIUM — Specifications. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/030, Apr. 2005. http://www.ecrypt.eu.org/stream. 4

[32] C. De Cannière and C. Rechberger. Finding SHA-1 characteristics: General results and applications. In X. Lai and K. Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 2006. 4

[33] C. De Cannière, J. Lano, and B. Preneel. Cryptanalysis of the two-dimensional circulation encryption algorithm (TDCEA). *EURASIP Journal on Applied Signal Processing*, 2005(12):1923–1927, Dec. 2005. 4

[34] C. De Cannière, A. Biryukov, and B. Preneel. An introduction to block cipher cryptanalysis. *Proceedings of the IEEE*, 94(2):346–356, Feb. 2006. 3, 4

[35] C. De Cannière, F. Mendel, and C. Rechberger. On the full cost of collision search for SHA-1. In *Proceedings of the ECRYPT Workshop on Hash Functions 2007*, May 2007. 4, 139

[36] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976. 6

[37] ECRYPT. Yearly report on algorithms and keysizes (2006). Delivery D.SPA.21 Revision 1.1, Jan. 2007. http://www.ecrypt.eu.org/documents/D.SPA.10-1.1.pdf. xiii, 22, 24

[38] P. Ekdahl and T. Johansson. SNOW – A new stream cipher. In *Proceedings of the First NESSIE Workshop*. NESSIE, Nov. 2000. 101

[39] M. Feldhofer. Comparison of low-power implementations of Trivium and Grain. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/027, Jan. 2007. http://www.ecrypt.eu.org/stream. 120, 124

[40] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting. Improved cryptanalysis of Rijndael. In B. Schneier, editor, *Fast Software Encryption, FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer-Verlag, 2001. 37

[41] S. Fischer and W. Meier. Algebraic immunity of S-boxes and augmented functions. In A. Biryukov, editor, *Fast Software Encryption, FSE 2007*, to appear in *Lecture Notes in Computer Science*. Springer-Verlag, 2007. 119, 124

[42] K. Gaj, G. Southern, and R. Bachimanchi. Comparison of hardware performance of selected Phase II eSTREAM candidates. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/027, Jan. 2007. http://www.ecrypt.eu.org/stream. 120, 124

[43] H. Gilbert and M. Minier. A collision attack on seven rounds of Rijndael. In *Proceedings of the Third AES Candidate Conference*, pages 230–241. National Institute of Standards and Technology, Apr. 2000. 37

[44] T. Good and M. Benaissa. Hardware results for selected stream cipher candidates. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/023, Jan. 2007. http://www.ecrypt.eu.org/stream. 120, 124

[45] T. Good, W. Chelton, and M. Benaissa. Review of stream cipher candidates from a low resource hardware perspective. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/016, Mar. 2006. http://www.ecrypt.eu.org/stream. 120

[46] T. Good, W. Chelton, and M. Benaissa. Review of stream cipher candidates from a low resource hardware perspective. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/016, Jan. 2006. http://www.ecrypt.eu.org/stream. 124

[47] F. K. Gürkaynak, P. Luethi, N. Bernold, R. Blattmann, V. Goode, M. Marghitola, H. Kaeslin, N. Felber, and W. Fichtner. Hardware evaluation of eSTREAM candidates: Achterbahn, Grain, MICKEY, MOSQUITO, SFINKS, TRIVIUM, VEST, ZK-Crypt. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/015, Jan. 2006. http://www.ecrypt.eu.org/stream. 120, 124

[48] M. A. Harrison. On the classification of boolean functions by the general linear and affine groups. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):285–299, June 1964. 86

[49] P. Hawkes and G. G. Rose. Primitive specification and supporting documentation for SOBER-tw submission to NESSIE. In *Proceedings of the First NESSIE Workshop*. NESSIE, Nov. 2000. 101

[50] P. Hawkes, C. McDonald, M. Paddon, G. G. Rose, and M. W. de Vries. Design and primitive specification for Shannon. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/020, Jan. 2007. http://www.qualcomm.com.au/Shannon.html. 19

[51] M. E. Hellman. A cryptanalytic time-memory tradeoff. *IEEE Transactions on Information Theory*, 26:401–406, 1980. 24

[52] M. E. Hellman, R. Merkle, R. Schroppel, L. Washington, W. Diffie, S. Pohlig, and P. Schweitzer. Results of an initial attempt to cryptanalyze the NBS Data Encryption Standard. Technical report, Stanford University, 1976. 96

[53] J. Hong and P. Sarkar. Rediscovery of time memory tradeoffs. Cryptology ePrint Archive, Report 2005/090, 2005. http://eprint.iacr.org/. 25

[54] T. Jakobsen and L. R. Knudsen. The interpolation attack on block ciphers. In E. Biham, editor, *Fast Software Encryption, FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 28–40. Springer-Verlag, 1997. 21

[55] P. Junod. On the optimality of linear, differential, and sequential distinguishers. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 17–32. Springer-Verlag, 2003. 50

[56] P. Junod and S. Vaudenay. Optimal key ranking procedures in a statistical cryptanalysis. In T. Johansson, editor, *Fast Software Encryption, FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2003. 50

[57] C. S. Jutla. Encryption modes with almost free message integrity. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 529–544. Springer-Verlag, 2001. 12

[58] C. S. Jutla and A. C. Patthak. Provably good codes for hash function design. In R. Cramer, editor, *Selected Areas in Cryptography, SAC 2006*, to appear in *Lecture Notes in Computer Science*. Springer-Verlag, 2006. 137

[59] B. S. Kaliski and M. J. Robshaw. Linear cryptanalysis using multiple approximations. In Y. Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 26–39. Springer-Verlag, 1994. 35, 49, 63, 67

[60] B. S. Kaliski and M. J. Robshaw. Linear cryptanalysis using multiple approximations and FEAL. In B. Preneel, editor, *Fast Software Encryption, FSE'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 249–264. Springer-Verlag, 1995. 49

[61] S. Khazaei. Re: A reformulation of TRIVIUM. Posted on the eSTREAM Forum, Feb. 2006. `http://www.ecrypt.eu.org/stream/phorum/read.php?1,448`. 117, 118, 124

[62] S. Khazaei, M. M. Hasanzadeh, and M. S. Kiaei. Linear sequential circuit approximation of Grain and Trivium stream ciphers. Cryptology ePrint Archive, Report 2006/141, 2006. `http://eprint.iacr.org/`. 124

[63] P. Kitsos. Hardware implementations for the ISO/IEC 18033-4:2005 standard for stream ciphers. *International Journal of Signal Processing*, 3(1):1304–4478, 2006. 124

[64] L. R. Knudsen. Practically secure Feistel cyphers. In R. J. Anderson, editor, *Fast Software Encryption, FSE'93*, volume 809 of *Lecture Notes in Computer Science*, pages 211–221. Springer-Verlag, 1994. 31

[65] L. R. Knudsen and J. E. Mathiassen. A chosen-plaintext linear attack on DES. In B. Schneier, editor, *Fast Software Encryption, FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 262–272. Springer-Verlag, 2001. 35, 69

[66] L. R. Knudsen and M. J. B. Robshaw. Non-linear approximations in linear cryptanalysis. In U. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 224–236. Springer-Verlag, 1996. 35

[67] L. R. Knudsen and D. Wagner. Integral cryptanalysis (extended abstract). In J. Daemen and V. Rijmen, editors, *Fast Software Encryption, FSE 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer-Verlag, 2002. 35

[68] D. Kwon, J. Kim, S. Park, S. H. Sung, Y. Sohn, J. H. Song, Y. Yeom, E.-J. Yoon, S. Lee, J. Lee, S. Chee, D. Han, and J. Hong. New block cipher: ARIA. In ong In Lim and D. H. Lee, editors, *International Conference on Information Security and Cryptology, ICISC 2003*, volume 2971 of *Lecture Notes in Computer Science*, pages 432–445. Springer-Verlag, 2004. 43

[69] X. Lai. Higher order derivatives and differential cryptanalysis. In *Proceedings of "Symposium on Communication, Coding and Cryptography", in honor of James L. Massey on the occasion of his 60th birthday*, 1994. 31

[70] J. Lano, N. Mentens, B. Preneel, and I. Verbauwhede. Power analysis of synchronous stream ciphers with resynchronization mechanism. In *ECRYPT Workshop, SASC – The State of the Art of Stream Ciphers*, pages 327–333, 2004. 120

[71] C. S. Lorens. Invertible boolean functions. *IEEE Transactions on Electronic Computers*, EC-13(5):529–541, Oct. 1964. 86

[72] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. In H. C. Williams, editor, *Advances in Cryptology – CRYPTO'85*, volume 218 of *Lecture Notes in Computer Science*, page 447. Springer-Verlag, 1986. Full version in [73]. 151

[73] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, Apr. 1988. An abstract appeared in [72]. 21, 151

[74] S. Lucks. Attacking seven rounds of Rijndael under 192-bit and 256-bit keys. In *Proceedings of the Third AES Candidate Conference*, pages 215–229. National Institute of Standards and Technology, Apr. 2000. 35

[75] J. L. Massey, G. H. Khachatrian, and M. K. Kuregian. Nomination of SAFER++ as candidate algorithm for the New European Schemes for

Signatures, Integrity, and Encryption (NESSIE). Primitive submitted to NESSIE by Cylink Corp., Sept. 2000. 38

[76] M. Matsui. A new method for known plaintext attack of FEAL cipher. In R. A. Rueppel, editor, *Advances in Cryptology – EUROCRYPT'92*, volume 658 of *Lecture Notes in Computer Science*, pages 81–91. Springer-Verlag, 1993. 31

[77] M. Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In Y. Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1994. 64, 66

[78] M. Matsui. On correlation between the order of S-boxes and the strength of DES. In A. D. Santis, editor, *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 366–375. Springer-Verlag, 1995. 67, 69

[79] M. Matsui and A. Yamagishi. A new method for known plaintext attack of FEAL cipher. In R. A. Rueppel, editor, *Advances in Cryptology – EUROCRYPT'92*, volume 658 of *Lecture Notes in Computer Science*, pages 81–91. Springer-Verlag, 1993. 31

[80] A. Maximov and A. Biryukov. Two trivial attacks on Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/003, Jan. 2007. `http://www.ecrypt.eu.org/stream`. 117, 118, 124

[81] C. McDonald, C. Charnes, and J. Pieprzyk. Attacking Bivium with MiniSat. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/040, Apr. 2007. `http://www.ecrypt.eu.org/stream`. 119, 124

[82] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997. Available online at `http://www.cacr.math.uwaterloo.ca/hac/`. 2, 129

[83] S. Murphy, F. Piper, M. Walker, and P. Wild. Likelihood estimation for block cipher keys. Technical report, Information Security Group, Royal Holloway, University of London, 1995. 50

[84] FIPS-180-2. *Secure Hash Standard*. National Institute of Standards and Technology, 2002. FIPS-180-2. 128, 130

[85] FIPS-197. *Advanced Encryption Standard*. National Institute of Standards and Technology, Nov. 2001. URL `http://csrc.nist.gov/encryption/`. 8

[86] FIPS-46. *Data Encryption Standard (DES)*. National Institute of Standards and Technology, 1979. URL `http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf`. Revised as FIPS 46-1:1988, FIPS 46-2:1993, FIPS 46-3:1999. 8

[87] SP-800-38A. *Recommendation for Block Cipher Modes of Operation*. National Institute of Standards and Technology, Dec. 2001. URL `http://csrc.nist.gov/encryption/`. 10

[88] National Security Research Institute. Specification of ARIA, Aug. 2003. Version 0.8. 43

[89] National Security Research Institute. Specification of ARIA, Jan. 2005. Version 1.0, `http://www.nsri.re.kr/ARIA/`. 43

[90] J. Patarin, L. Goubin, and N. T. Courtois. Improved algorithms for isomorphisms of polynomials. In K. Nyberg, editor, *Advances in Cryptology – EUROCRYPT 1998*, volume 2403 of *Lecture Notes in Computer Science*, pages 184–200. Springer-Verlag, 1998. 76, 84

[91] N. Pramstaller, C. Rechberger, and V. Rijmen. Exploiting coding theory for collision attacks on SHA-1. In N. P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, volume 3796 of *Lecture Notes in Computer Science*, pages 78–95. Springer-Verlag, 2005. 137

[92] H. Raddum. Cryptanalytic results on TRIVIUM. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/039, Mar. 2006. `http://www.ecrypt.eu.org/stream`. 117, 119, 124

[93] V. Rijmen and E. Oswald. Update on SHA-1. In A. Menezes, editor, *Topics in Cryptology – CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 58–71. Springer-Verlag, 2005. 137

[94] R. L. Rivest. The RC4 encryption algorithm. RSA Security Inc., Mar. 1992. unpublished. 17

[95] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, Feb. 1978. 6

[96] P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 196–205. ACM Press, 2001. 12

[97] M. Rogawski. Hardware evaluation of eSTREAM candidates: Grain, Lex, Mickey128, Salsa20 and Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/025, Jan. 2007. `http://www.ecrypt.eu.org/stream`. 120, 124

[98] J. Rosenthal and R. Smarandache. Maximum distance separable convolutional codes. *Applicable Algebra in Engineering, Communication and Computing*, 10(1):15–32, Aug. 1999. 107

[99] A. A. Selçuk. On probability of success in differential and linear cryptanalysis. Technical report, Network Systems Lab, Department of Computer Science, Purdue University, 2002. previously published at SCN 2002. 50

[100] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28, 1949. 10, 20

[101] T. Shimoyama and T. Kaneko. Quadratic relation of s-box and its application to the linear attack of full round des. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 200–211. Springer-Verlag, 1998. 35

[102] M. S. Turan and O. Kara. Linear approximations for 2-round Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/008, Jan. 2007. `http://www.ecrypt.eu.org/stream`. 119, 124

[103] S. Vaudenay. An experiment on DES statistical cryptanalysis. In *3rd ACM Conference on Computer and Communications Security, CCS*, pages 139–147. ACM Press, 1996. 50

[104] D. Wagner. The boomerang attack. In L. R. Knudsen, editor, *Fast Software Encryption, FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer-Verlag, 1999. 21, 31

[105] X. Wang and H. Yu. How to break MD5 and other hash functions. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer-Verlag, 2005. 127

[106] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the hash functions MD4 and RIPEMD. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2005. 127

[107] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer-Verlag, 2005. 127, 131, 137

[108] X. Wang, H. Yu, and Y. L. Yin. Efficient collision search attacks on SHA-0. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2005. 127

[109] D. Watanabe, A. Biryukov, and C. De Cannière. A distinguishing attack of SNOW 2.0 with linear masking method. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography, SAC 2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 222–233. Springer-Verlag, 2004. 4

[110] D. Whiting, B. Schneier, S. Lucks, and F. Muller. Phelix – fast encryption and authentication in a single cryptographic primitive. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/020, Apr. 2005. `http://www.ecrypt.eu.org/stream`. 19

[111] M. J. Wiener. The full cost of cryptanalytic attacks. *Journal of Cryptology*, 17(2):105–124, 2004. 24

[112] H. Wu and B. Preneel. Differential-linear attacks against the stream cipher Phelix. In A. Biryukov, editor, *Fast Software Encryption, FSE 2007*, to appear in *Lecture Notes in Computer Science*. Springer-Verlag, 2007. 19

[113] B.-Y. Yang, O. C.-H. Chen, D. J. Bernstein, and J.-M. Chen. Analysis of QUAD. In A. Biryukov, editor, *Fast Software Encryption, FSE 2007*, to appear in *Lecture Notes in Computer Science*. Springer-Verlag, 2007. 22

[114] H. Yoshida, A. Biryukov, C. De Cannière, J. Lano, and B. Preneel. Non-randomness of the full 4 and 5-pass HAVAL. In C. Blundo and S. Cimato, editors, *Security in Communication Networks, SCN 2004*, volume 3352 of *Lecture Notes in Computer Science*, pages 324–336. Springer-Verlag, 2005. 4

Nederlandse Samenvatting

Analyse en Ontwerp van Symmetrische Encryptie-Algoritmen

Hoofdstuk 1: Inleiding

Gedurende de laatste decennia heeft informatietechnologie zich een plaats veroverd in steeds meer domeinen van onze samenleving. Twee opmerkelijke evoluties waren de spectaculaire groei van het Internet en de razendsnelle opgang van draadloze digitale telefoonnetwerken zoals GSM. Het succes van deze nieuwe technologieën kan toegeschreven worden aan een aantal intrinsieke voordelen van digitale systemen. Ten eerste is digitale informatie nagenoeg ongevoelig voor ruis en kan ze naar believen gekopieerd of gewijzigd worden zonder het minste kwaliteitsverlies. Bovendien is de band tussen de informatie en haar drager verdwenen, zodat dezelfde informatie naadloos op erg verschillende toestellen kan verwerkt worden.

Dezelfde eigenschappen die digitale informatiesystemen zo aantrekkelijk maken, bieden echter ook tal van nieuwe mogelijkheden tot misbruik. In tegenstelling tot bijvoorbeeld een brief in een gesloten (en verzegelde) omslag, kan digitale informatie op haar weg van zender naar ontvanger op verschillende manier gemanipuleerd worden, zonder dat dit enig spoor zou nalaten. Al snel is duidelijk geworden, dat de enige manier om dit te verhelpen zonder aan de voordelen van digitale systemen te raken, erin bestaat de informatie zodanig te transformeren dat ze zichzelf beschermt, onafhankelijk van de drager. De wetenschap die zich hiermee bezighoudt wordt de *cryptologie* genoemd.

De bescherming van digitale informatie kan verschillende aspecten omvatten, waarvan de belangrijkste zijn: *geheimhouding* (het voorkomen dat in-

formatie onthuld wordt aan derden) en *authentisering* (het garanderen dat de ontvangen boodschap wel degelijk afkomstig is van de zender, en onderweg niet werd gewijzigd). In deze thesis behandelen we enkel het eerste probleem.

Hoofdstuk 2: Symmetrische Encryptie

Om de geheimhouding van een boodschap P te waarborgen wanneer deze over een onveilig kanaal wordt doorgezonden, moet de data versleuteld worden. Hiervoor wordt beroep gedaan op een *encryptie-algoritme*, dat in het algemeen geval bestaat uit twee wiskundige functies: een encryptiefunctie E die de klaartekst P omzet in een cijfertekst $C = E(P)$, en een decryptiefunctie $D = E^{-1}$ die precies het omgekeerde doet. Vanzelfsprekend is de geheimhouding van D (of van de sleutel die in deze functie als parameter gebruikt wordt), een essentiële voorwaarde voor de veiligheid van dit systeem. Geldt dit ook voor de encryptiefunctie E, dan spreken we van *symmetrische encryptie*; indien het algoritme toelaat dat E publiek gemaakt wordt, dan hebben we te maken met *publieke-sleutel encryptie*. Zoals de titel laat vermoeden, zullen we ons in deze thesis beperken tot symmetrische encryptie.

In Hoofdstuk 2 van deze thesis bespreken we de voornaamste soorten symmetrische encryptie-algoritmen. We maken een onderscheid tussen *stroomcijfers* en *blokcijfers*, en vestigen de aandacht op het verschil in aanpak tussen blok-encryptie, waarbij de cijfertekst op een (voor de aanvaller) onvoorspelbare wijze afhangt van de *waarde* van de klaartekst, en stroom-encryptie, waarbij de encryptie op een onvoorspelbare manier afhangt van de *positie* van de symbolen in de klaartekst. We bespreken de typische iteratieve structuur van blokcijfers en de verschillende modes waarin ze gebruikt kunnen worden, en bestuderen vervolgens een aantal stroomcijfer-constructies.

Tenslotte verduidelijken we wat een aanval op een symmetrisch encryptie-algoritme precies inhoudt, hoe de veiligheid van een algoritme in principe kan beoordeeld worden, en wat verstaan moet worden onder een veilig algoritme.

Hoofdstuk 3: Cryptanalyse van Blokcijfers

In dit hoofdstuk verdiepen we ons in de studie van technieken die de beveiliging van blokcijfers trachten te doorbreken (de *cryptanalyse* van blokcijfers). Typisch bestaan deze aanvalstechnieken uit twee fasen. In een eerste fase wordt getracht een sleutel-onafhankelijke eigenschap te vinden waarmee het blokcijfer, ontdaan van zijn laatste (of eerste) ronde(n), zich onderscheidt van een willekeurige permutatie. In de tweede fase worden dan alle mogelijke waarden van de relevante sleutelwoorden in deze laatste (of eerste) ronde(n) doorlopen, totdat de verwachte eigenschap waargenomen wordt, wat erop wijst dat de waarden correct zouden kunnen zijn. In dit hoofdstuk bespreken we drie belangrijke technieken, die alle drie steunen op dit principe: lineaire cryptanalyse, differentiële cryptanalyse, en multiset cryptanalyse.

In het tweede deel van dit hoofdstuk geven we twee voorbeelden van concrete aanvallen, die nog steeds uitgaan van hetzelfde principe, maar die gebruik maken van een aantal nieuwe ideeën. De eerste aanval is gericht op een gereduceerde versie van het blokcijfer SAFER++, en stelt een nieuw soort multiset aanval voor dat gebruik maakt van botsingen. De tweede aanval wordt geïllustreerd aan de hand van het blokcijfer ARIA, en kan gezien worden als een variant van lineaire cryptanalyse die steunt op lineaire combinaties van woorden in plaats van bits.

Hoofdstuk 4: Lineaire Cryptanalyse Herbekeken

Lineaire cryptanalyse is een van de meest krachtige en algemeen toepasbare aanvalstechieken. In dit hoofdstuk bestuderen we deze aanval vanuit een "Most-Likelihood" perspectief. We ontwerpen een algemeen statistisch kader dat ons toelaat om optimale aanvalsstrategieën af te leiden voor lineaire aanvallen die simultaan gebruik maken van meerder lineaire benaderingen, en tonen aan dat Matsui's oorspronkelijke aanvallen, Algoritme 1 en Algoritme 2, in dit kader op vrijwel identieke manier beschreven kunnen worden. Door het invoeren van het concept van *capaciteit* van een verzameling lineaire benaderingen, kunnen we eenvoudige benaderende formules afleiden om de efficiëntie van de nieuwe aanvallen te voorspellen.

Vervolgens passen we deze aanvallen toe op gereduceerde versies van DES, en verifiëren we dat hun efficiëntie inderdaad nauwkeurig voorspeld kan worden. Op basis van deze resultaten bespreken we mogelijke verbeteringen van de beste aanvallen op het volledige DES algoritme.

We sluiten dit hoofdstuk af met de constructie van een efficiënt algoritme om in een blokcijfer de m beste lineaire benaderingen te vinden.

Hoofdstuk 5: Classificatie van S-Boxen

Een belangrijk bouwblok in het ontwerp van blokcijfers die bestand zijn tegen lineaire en differentiële cryptanalyse zijn substitutie-boxen (S-boxen). In dit hoofdstuk bestuderen we deze componenten door ze te reduceren tot hun essentie: hun niet-lineariteit. Hiervoor ontwikkelen we algoritmen die toelaten om abstractie te maken van affiene afbeeldingen aan de in- en uitgang van de S-boxen, en te detecteren of twee S-boxen equivalent zijn tot op een affiene afbeelding. We tonen aan hoe de zogenaamde *lineaire representant* van een S-box efficiënt kan berekend worden, en gebruiken dit om alle $2 \cdot 10^{13}$ 4×4-bit S-boxen onder te verdelen in 302 equivalentieklassen.

In het laatste deel van dit hoofdstuk breiden we deze algoritmen uit voor het detecteren van bijna-equivalente S-boxen, en passen we ze ook aan om

niet-bijectieve S-boxen te bestuderen. Tenslotte passen we de algoritmen toe op de S-boxen van een aantal populaire blokcijfers en tonen het bestaan aan van een groot aantal equivalentie-relaties.

Hoofdstuk 6: Ontwerp van Stroomcijfers

In het vorige hoofdstuk hebben we algoritmen voorgesteld om de niet-lineaire componenten van encryptie-algoritmen te bestuderen. In dit laatste hoofdstuk richten we onze aandacht op de lineaire componenten en op hun belangrijke rol in het verspreiden van deze niet-lineariteit. We ontwikkelen een nieuwe ontwerpstrategie voor stroomcijfers geïnspireerd door de technieken die gebruikt worden om de lineaire en differentiële cryptanalyse van blokcijfers te bemoeilijken. De methode maakt gebruik van lineaire filters, en leidt uiteindelijke tot het ontwerp van een nieuw stroomcijfer, TRIVIUM, dat slechts bestaat uit een erg klein aantal niet-lineaire componenten (3 EN-poorten), en waarvan de veiligheid volledig berust op een efficiënte spreiding gerealiseerd door zorgvuldig ontworpen lineaire filters.

Het stroomcijfer TRIVIUM is een van de acht hardware kandidaten dat de eerst twee selectieronden van het eSTREAM stroomcijfer project overleefde. In het laatste gedeelte van dit hoofdstuk geven we een kort overzicht van de huidige status van de publieke evaluatie.

Appendix: Differentiële Karakteristieken in SHA-1

In deze appendix worden de resultaten samengevat van onderzoek dat uitgevoerd werd in het kader van dit doctoraat, maar dat geen rechtstreeks verband houdt met symmetrische encryptie. Het probleem dat we bestuderen is het zoeken van karakteristieken in de hashfunctie SHA-1. Dit probleem, dat een cruciale rol speelt in de ontwikkeling van botsingsaanvallen, is lang een belangrijke hindernis geweest die bij iedere nieuwe aanval opnieuw overwonnen moest worden via een ingewikkelde en moeizame manuele analyse. In deze appendix voeren we het concept *veralgemeende karakteristiek* in, en tonen we aan hoe de invloed van deze karakteristieken op de efficiëntie van de aanval berekend kan worden. Vervolgens stellen we een geautomatiseerde methode voor om zulke karakteristieken te construeren.

List of Publications

International Journals

1. C. De Cannière, A. Biryukov, and B. Preneel. An introduction to block cipher cryptanalysis. *Proceedings of the IEEE*, 94(2):346–356, Feb. 2006.

2. C. De Cannière, J. Lano, and B. Preneel. Cryptanalysis of the two-dimensional circulation encryption algorithm (TDCEA). *EURASIP Journal on Applied Signal Processing*, 2005(12):1923–1927, Dec. 2005.

Lecture Notes in Computer Science

1. C. De Cannière and C. Rechberger. Finding SHA-1 characteristics: General results and applications. In X. Lai and K. Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 2006. (Best Paper Award)

2. C. De Cannière. TRIVIUM: A stream cipher construction inspired by block cipher design principles. In S. K. Katsikas, J. Lopez, M. Backes, S. Gritzalis, and B. Preneel, editors, *Information Security, ISC 2006*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer-Verlag, 2006.

3. H. Yoshida, A. Biryukov, C. De Cannière, J. Lano, and B. Preneel. Non-randomness of the full 4 and 5-pass HAVAL. In C. Blundo and S. Cimato, editors, *Security in Communication Networks, SCN 2004*, volume 3352 of *Lecture Notes in Computer Science*, pages 324–336. Springer-Verlag, 2005.

4. A. Biryukov, C. De Cannière, and M. Quisquater. On multiple linear approximations. In M. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

5. D. Watanabe, A. Biryukov, and C. De Cannière. A distinguishing attack of SNOW 2.0 with linear masking method. In M. Matsui and R. Zuc-

cherato, editors, *Selected Areas in Cryptography, SAC 2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 222–233. Springer-Verlag, 2004.

6. A. Biryukov, C. De Cannière, and G. Dellkrantz. Cryptanalysis of SAFER++. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 195–211. Springer-Verlag, 2003.

7. A. Biryukov, C. De Cannière, A. Braeken, and B. Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 33–50. Springer-Verlag, 2003.

8. A. Biryukov and C. De Cannière. Block ciphers and systems of quadratic equations. In T. Johansson, editor, *Fast Software Encryption, FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 274–289. Springer-Verlag, 2003.

9. S. Babbage, C. De Cannière, J. Lano, B. Preneel, and J. Vandewalle. Cryptanalysis of SOBER-t32. In T. Johansson, editor, *Fast Software Encryption, FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 111–128. Springer-Verlag, 2003.

International Workshops

1. C. De Cannière, F. Mendel, and C. Rechberger. On the full cost of collision search for SHA-1. In *Proceedings of the ECRYPT Workshop on Hash Functions 2007*, May 2007.

2. C. De Cannière and C. Rechberger. Finding SHA-1 characteristics. In *Second Cryptographic Hash Workshop*. NIST, Aug. 2006.

3. C. De Cannière and B. Preneel. A stream cipher construction inspired by block cipher design principles. In *SASC 2006 – Stream Ciphers Revisited*, 2006.

4. C. De Cannière, J. Lano, B. Preneel, and J. Vandewalle. Distinguishing attacks on SOBER-t32. In *Proceedings of the Third NESSIE Workshop*. NESSIE, Nov. 2002.

5. A. Biryukov and C. De Cannière. Block ciphers and systems of quadratic equations. In *Proceedings of the Third NESSIE Workshop*. NESSIE, Nov. 2002.

Miscellaneous

1. C. De Cannière and B. Preneel. TRIVIUM — Specifications. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/030, Apr. 2005. `http://www.ecrypt.eu.org/stream`.

2. C. De Cannière. "Blowfish," "Camellia," "CAST," and 10 other entries. In H. van Tilborg, editor, *Encyclopedia of Cryptography and Security*. Springer, 2005.

3. A. Biryukov, C. De Cannière, J. Lano, S. B. Örs, and B. Preneel. Security and performance analysis of ARIA. Final report, K.U.Leuven ESAT/SCD-COSIC, 2004.

4. C. De Cannière and B. Preneel. A short introduction to cryptology. *Revue HF Tijdschrift*, 2004(4):4–14, 2004.

5. C. De Cannière. Hoe onoverwinnelijk is Harald Blåtand? *Het Ingenieursblad*, 72(6/7):56–62, 2003.

Christophe De Cannière was born on October 7, 1978 in Leuven, Belgium. He received the degree of Electrical Engineering (Telecommunications) from the Katholieke Universiteit Leuven, Belgium in June 2001. His Master's thesis dealt with the cryptanalysis of the Bluetooth stream cipher, and was carried out at Lunds Tekniska Högskola, Sweden. In October 2001 he started working as a PhD researcher in the research group COSIC (Computer Security and Industrial Cryptography) at the Department of Electrical Engineering (ESAT) of the K.U.Leuven. His research was sponsored by the National Fund for Scientific Research Flanders (FWO). From October 2005 to September 2006, he visited the IAIK research group of the Technische Universität Graz, Austria. Together with his co-author Christian Rechberger, he received the Best Paper Award at the ASIACRYPT 2006 conference for the paper "Finding SHA-1 Characteristics."