Cipher and Hash Function Design Strategies based on linear and differential cryptanalysis

Joan Daemen

March 1995

Note: This version has been compiled in January 2004. With respect to the original version, the formatting of some formulas was modified for layout purposes and some sentences were rephrased to allow easier line breaks.

Voorwoord

In deze inleiding maak ik van de gelegenheid gebruik om al diegenen te bedanken die er voor hebben gezorgd dat dit doctoraat tot stand is gekomen in een informele en vriendschappelijke werksfeer met veel ruimte voor samenwerking, creativiteit en persoonlijk initiatief.

In de eerste plaats denk ik hierbij aan mijn promotoren Prof. René Govaerts en Prof. Joos Vandewalle, die mij op weg hebben geholpen met concrete voorstellen voor onderzoek. Door mij steeds te wijzen op het belang van de praktische toepasbaarheid van onderzoeksresultaten hebben zij voor een belangrijk deel de toon gezet voor deze verhandeling.

Voorts wil ik graag de overige leden van het leescomité bedanken, Prof. Bart De Decker van het Dept. Computerwetenschappen en Prof. Jean-Jacques Quisquater van de UCL. Mijn oprechte dank gaat ook uit naar mijn collega en jurylid Bart Preneel, die het manuscript op vrijwillige basis heeft nagelezen en met wie ik vele leerrijke gesprekken gevoerd heb.

Een speciaal woord van dank gaat naar jurylid Prof. Luc Claesen van IMEC voor zijn inzet en enthousiasme die hebben geleid tot de vlotte implementatie en demonstratie van de SUBTERRANEAN chip.

I am also grateful to Prof. Peter Landrock of the university of Århus for serving in my jury despite his busy agenda. Als laatste jurylid bedank ik voorzitter Prof. Willy Dutré.

Verder bedank ik graag mijn collega's met wie ik binnen ESAT heb samengewerkt en op wie ik nooit tevergeefs een beroep heb gedaan. Ik denk hierbij speciaal aan Jan Verschuren die meer dan wie ook heeft bijgedragen tot de positieve sfeer. Verder vermeld ik graag Luc Van Linden, Antoon Bosselaers, Cristian Radu, Vincent Rijmen, Ria Vanden Eynde, Rita Dewolf, Steven Vanleemput, Patrick Wuytens, Mark Vandenwauver, André Barbé, Elvira Wouters, Marc Genoe, Gert Peeters en Zohair Sahraoui.

I would also like to thank all people from the international scientific community who have supported or helped me during my research. Among them are Tsutomu Matsumoto, Markus Dichtl, William Wolfowicz, Jorge Davila, Bert Den Boer, Douglas Stinson, Ronald Rivest and Jan Egil Øye.

Ik wil graag besluiten met die mensen te bedanken die mij het nauwst aan het hart liggen. Mijn ouders, voor hun onvoorwaardelijke steun, hun aanmoedigingen en hun oprechte interesse. Mijn verloofde Ilse, voor haar voortdurende toewijding en liefde waardoor het mij nooit aan motivatie en energie ontbroken heeft of zal ontbreken.

Abstract

This thesis contains a new approach to design block ciphers, synchronous and self-synchronizing stream ciphers and cryptographic hash functions. The goal of this approach is the specification of cryptographic schemes that are secure, simple to describe and that can be implemented efficiently on a wide variety of platforms. Key words are simplicity, symmetry and parallelism. An overview of the different types of ciphers, encryption schemes and hash functions is given, the nature of cryptographic security is discussed and some new security-related definitions are presented. The design is mainly guided by the resistance against differential and linear cryptanalysis. The basic mechanisms of these two attacks are investigated and their structure is clarified by adopting a new formalism for their description and analysis. The resistance against differential and linear cryptanalysis is obtained by applying the new *wide trail strategy* that emphasizes the mechanism of diffusion. The application of this strategy for the different types of ciphers and hash functions leads to a number of new structures and specific designs. A new self-reciprocal block cipher structure is introduced together with a new type of cryptographic component: the stream/hash module. The design of single-bit self-synchronizing stream ciphers is treated and the potential weaknesses of ciphers that make use of arithmetic operations are analyzed. The design approach is supported by a number of new cryptanalytic results.

Contents

List of Notations

List of Abbreviations

ASCII	: American Standard Code for Information Interchange
BB	: Bitwise Boolean
BSS	: Binary Symmetric Source
CA	: Cellular Automaton
CBC	: Cipher Block Chaining
CFB	: Cipher Feedback
CL	: Conserved Landscape
CMOS	: Complementary Metal Oxide Semiconductor
CO	: Ciphertext-Only
CS	: Cyclic Shift
DC	: Differential Cryptanalysis
DES	: Data Encryption Standard
ECB	: Electronic Code Book
EXOR	: Exclusive Or
FCS	: Filtered Counter Scheme
FFT	: Fast Fourier Transform
IDEA	: International Data Encryption Algorithm
ISO	: International Organization for Standardization
IV	: Initial Value
KP	: Known-Plaintext
LC	: Linear Cryptanalysis
LFSR	: Linear Feedback Shift Register
LSB	: Least Significant Bit
MD4	: Message Digest 4
MD5	: Message Digest 5
MMB	: Modular Multiplication based Block Cipher
MUX	: Multiplexer
MSB	: Most Significant Bit
NIST	: National Institute for Standards and Technology (USA)
NAND	: Not And
NOR	: Not Or
NP	: Nondeterministic Polynomially
NS	: Non-Secret

OFB	: Output Feedback
P	: Polynomially
RSA	: Rivest-Shamir-Adleman
SHA	: Secure Hash Algorithm

List of Ciphers and Hash Functions

DES	: 64-bit block cipher
FFT-hash	: hash function
IDEA	: 64-bit block cipher
MD4	: hash function
MD5	: hash function
RSA	: public key cipher
SHA	: hash function
Snefru	: hash function
BaseKing	: 192-bit block cipher
Boognish	: hash function
Cellhash	: hash function
Jam	: synchronous stream cipher
MMB	: 128-bit block cipher
StepRightUp	: stream/hash module
Subterranean	: stream/hash module
Subhash	: Subterranean hash function
Substream	: Subterranean synchronous stream cipher
3-Way	: 96-bit block cipher
$\Upsilon\Gamma$	: single-bit self-synchronizing stream cipher

List of mathematical symbols

$\mathbb{Z}_2$	: set of residues modulo 2, i.e., $\{0,1\}$
$\mathbb{Z}_2^n$	: set of all binary vectors with components in $\mathbb{Z}_2$
a	: a binary vector or state
a_i	: the component of a with index i
$\bar{a}$	: bitwise complement of binary vector a
$a+b$	: bitwise sum of binary vectors a and b
ν	: a neighborhood, i.e., a set of indices
$a\|_\nu$	: a confined to its components with indices in ν
$\bar{0}$	: a state or vector with all components equal to 1
$\#\alpha$	: cardinality of set α
$\lfloor x \rfloor$	: the greatest integer less than or equal to x

$\lceil x \rceil$	: the least integer greater than or equal to x
$\delta(w)$	: Kronecker delta function
δ_i	: vector with a single nonzero component at index i
$a \| b$	: concatenation of strings a and b
M	: message
C	: cryptogram
K	: secret key
Q	: public parameter
κ	: cipher key
m^t	: message symbol or block at time step t
c^t	: cryptogram symbol or block at time step t
w^t	: (stream) encrypting symbol at time step t
n_{m}	: finite memory (in symbols) of finite state machine
n_{s}	: symbol length in bits
n_κ	: length of κ
n_{a}	: length of the internal state of a finite state machine
n_{b}	: block length
n_{h}	: length of a hash result
r	: relative redundancy
v	: length of a message
b_{s}	: number of stages in a self-synchronizing stream cipher
m	: number of rounds in an iterated block cipher
r	: number of bits in the buffer
$\mathrm{F}[x](y)$	: Boolean mapping parameterized by x and with argument y
$\mathrm{C}(f, g)$	: correlation coefficient of Boolean functions f and g
$\mathcal{W}$	: Walsh-Hadamard transform
$\mathcal{V}_f$	: support space of Boolean function f
C^h	: correlation matrix of Boolean mapping h
C^h_{uw}	: element of C^h in row u and column w
ϕ	: a binary shift-invariant transformation
τ_r	: translation over r positions to the right
$\mathrm{R_p}$	: prop ratio
$\mathrm{C_p}$	: correlation contribution coefficient
$\mathrm{w_r}$	: restriction weight
$\mathrm{w_c}$	: correlation weight
$\mathrm{w_h}$	: Hamming weight
$\mathrm{w_t}$	: triplet weight
$\mathrm{w_m}$	: minimum ternary weight of a number
$\mathcal{B}$	: branch number of a linear transformation
$\mathcal{C}$	: maximum input-output correlation of a transformation
$\mathcal{D}$	: diffusion factor of a transformation
$\mathcal{R}$	: maximum prop ratio of a transformation

Chapter 1

Introduction

1.1 Communication and computer security

In our modern society more and more activities come to rely on telecommunication networks and computers. It is expected, and can already be observed today, that this technological revolution will affect many aspects of human interaction. The new technology has given rise to important new concepts, such as (portable) *software* and *CPU time sharing*.

At present, an important role is played by all kinds of paper documents such as value documents, administration documents, identification documents, contracts and treaties. Efficient application of communication and computer technology asks for the handling of documents that are no longer inseparable from their physical carrier, but are equivalent to abstract strings of symbols. Moreover, technological advancements are making it easier and easier to physically forge classical paper documents. These new technological developments bring with them an acute need for security mechanisms.

Contracts, identification documents and value documents are examples of **commitment** carriers. Some organization, individual or group of individuals commits herself to certain consequences if unable to keep certain promises. The **authentication** of these classical **documents** is still realized by physical means such as signatures or high-tech printing techniques that are considered hard to counterfeit. Obviously, the authentication of abstract computer documents can only be realized effectively at the logical level.

In face-to-face communication it is relatively easy to create circumstances in which eavesdropping is infeasible. In telecommunication applications (communication) messages travel over easily accessible channels and their **privacy** can no longer be taken for granted. Face-to-face communication has the inherent aspect of **authenticity**. It can be verified by sensory perception that the communication partner is the person he or she claims to be and that the utterances attributed to him or her are actually his or hers. In telecommunication applications the authenticity of received messages is no longer obvious. Messages can be modified during transmission or an adversary can create messages that are wrongly attributed to the legitimate communication partner.

Modern computer and communication technology has given rise to a broad range of new services. Widespread communication networks, some even world-wide (i.e., Internet), connect consumers to millions of computers, large numbers of databases or the providers of video/audio entertainment. It is widely expected that in the near future the available services will be integrated in multifunctional networks. In commercial and political campaigns this is made clear with buzzwords such as *information superhighway*, *multimedia* and *B-ISDN* (Broadband Integrated Services Digital Network). Other well-known services include financial transaction systems with automatic teller machines (ATM) and electronic funds transfer at the point-of-sale (EFTPoS).

Clearly, the management of these networks is only possible if there are effective mechanisms of **authorization**. It should be infeasible for a user to gain access to services that he is not entitled to. Additionally, these systems are preferably designed in such a way that possible abuses by the network administration, such as eavesdropping or blackmail, are prevented as much as possible. The systems should be robust, i.e., the risk of breakdown due to sabotage should be minimized. These requirements also hold for local multi-user computer networks. An example of a desirable feature in the latter systems is the ability of a user to protect the confidentiality and the integrity of his files with respect to any other user and even the system administration.

1.2 Cryptography

Cryptographic techniques have been used for many centuries to protect the secrecy and authenticity of diplomatic and political correspondence and military communications. A brilliant account of the history of cryptography is given by David Kahn in [58]. Most historical cryptographic schemes are in fact *encryption schemes*. These convert a message into a cryptogram by an invertible operation that depends on a secret key. Decryption is relatively easy for someone in possession of the key and is considered infeasible for someone not in possession of the key. However, often intercepted messages have actually been reconstructed from the cryptogram by adversaries without a priori knowledge of the secret key. This is called *cryptanalysis*. The science that studies both cryptography and cryptanalysis is called *cryptology*.

Until well after World War II, any cryptologic research and practice of significance was confined to some isolated sites at army headquarters, secret services and manufacturing companies. Publications were scarce and cryptologic knowledge was carefully guarded. The publication of the Data Encryption Standard and the invention of public-key cryptography at the end of the seventies [31] marked the transition of cryptology from a rather occult practice to a scientific research area [32]. Under the impulse of the revolution in communication and computer technology, the field of cryptology has known a spectacular growth. Today there is a community of cryptologic researchers who organize and attend specialized conferences and workshops, publish in scientific journals and have their own international association (IACR). This new openness in cryptologic research has been a fertile soil for the emergence of completely new concepts [68, 81]. The main object of cryptologic research, the encryption scheme, has been complemented with a wide variety of cryptographic schemes and corresponding cryptanalytic attacks.

Nowadays, the term cryptography is generally used to refer to the set of mathematical and logical tools for providing communication and computer security. With the exception of so-called *quantum cryptography* [3], cryptographic techniques aim at realizing the security at the logical level, independent of the physical communication channel or data carrier. Cryptanalysis corresponds to the detection of weaknesses in cryptographic schemes. In the modern scientific context, cryptanalysis is no longer a goal in itself, but has become the main set of tools available for cryptographic design. For a treatment of the most important *families* of cryptographic schemes and cryptanalytic attacks we refer to [68, 81, 8].

1.3 Research goals

Our research has concentrated on the design of bulk-processing cryptographic schemes. By this we mean cryptographic schemes with a work factor proportional to the length of the messages to be treated. The data throughput attainable by these cryptographic schemes determines for a large part the performance of communication and computer security systems. Clearly, encryption schemes belong to this class.

Encryption schemes are designed to protect the secrecy of the message contents. Still, it has often been tacitly assumed that encryption also provides authentication in that the secret key is needed to construct a cryptogram that decrypts to a meaningful message. Nowadays it is widely realized that the amount of authentication protection provided by encryption is very low for some types of encryption schemes and depends on a characteristic of the message denoted by the term *relative redundancy*.

Effective authentication can be realized by a cryptographic checksum mechanism. The basic component of such a checksum mechanism is a *cryptographic hash function*. This is a function that maps a message of arbitrary length to a *hash result* with fixed length. This mapping can be parameterized by a secret key. In a checksum mechanism the cryptographic hashing is the only operation with a work factor proportional to the message length. All other operations act on the short hash result. Other applications of hash functions include digital signatures and certain identification protocols. For an extensive overview of the applications of cryptographic hash functions we refer to the doctoral dissertation of Bart Preneel [84].

In most applications it is sufficient that a hash function provides a unique imprint of a message. This may mean that it should be infeasible to find pairs of *colliding* messages, i.e., that hash to the same result. Sometimes it may be sufficient that for any given hash result it is infeasible to find a corresponding message or that, given a message, finding another message that hashes to the same result is infeasible. Cryptographic hashing is also proposed to destroy potentially exploitable algebraic properties of a scheme, as in signature schemes based on the RSA public key scheme

[91, 80]. Collision-resistance is an important basic requirement that guides our design strategy of cryptographic hash functions. Still, in our notion of security with respect to cryptographic hash functions, collision-resistance is more of a *symptom* than a security-defining property.

To our knowledge all bulk-processing in cryptography can be reduced to encryption or cryptographic hashing.

Preferably security in computer and communication systems is realized without giving rise to noticeable delay or data overhead. Since the use of modern technology like optical fiber will allow communication in the Gbit/s range in a few years, there is a need for encryption and cryptographic hashing at these rates. Still, in the majority of applications a simple software implementation on a general purpose processor will probably be sufficient.

At this moment, there are already a large number of specific proposals for the different types of encryption schemes and cryptographic hash functions. Hence, why did we choose to study the design of these schemes, resulting in even more proposals?

In fact, our motivation is given in Chapters 3 to 5 and supported by results in Chapter 10. In short, it is a critique of the design practices that are currently dominant. There are several problems with these practices that severely limit the resulting designs. The most important problems belong to one of the following three classes:

- design goals and criteria that are irrelevant in practice,
- design frameworks that are too narrow,
- disregard for criteria that are essential in practice such as software efficiency and/or hardware suitability.

Our goal is to contribute to the design of encryption schemes and cryptographic hash functions that are

1. **Portable:** they can be implemented cheaply and efficiently on a wide variety of platforms.
2. **Secure:** they are as secure as their dimensions (key length, block length, . . .) suggest, independent of the encrypted data or accessibility of the encryptor and decryptor.

1.4 About this thesis

This thesis contains the motivation, foundation, formulation, elaboration and illustration of a new approach for the design of single-key encryption schemes, ciphers and cryptographic hash functions.

In Chapter 2 we introduce the cryptographic schemes to be designed. We describe the external properties of the different ciphers and encryption schemes, and give a general model for sequential hashing.

Chapter 3 is devoted to the essential property of cryptographic schemes, namely security. We discuss the problems of provable security and provide some new security-related concepts.

In Chapter 4 we elaborate on our design goals and introduce our design strategies for the different types of ciphers and cryptographic hash functions.

In Chapter 5 we give a new formalism for the description and analysis of difference propagation and correlation in cryptographic mappings. These are the most important analytical tools in our design approach.

Chapter 6 is devoted to a class of transformations to which belong the main building blocks in all our designs. These are the binary shift-invariant transformations. We discuss their invertibility, difference propagation and correlation properties.

In Chapter 7 we elaborate on our design strategy for block ciphers, resulting in two specific designs.

In Chapter 8 we introduce dedicated modules that can be used both as a synchronous stream cipher and as a cryptographic hash function. This includes the description of our hardware-oriented design SUBTERRANEAN and a more software-oriented design.

Chapter 9 is devoted to the design of hardware-oriented single-bit self-synchronizing stream ciphers. This is illustrated by a design example.

Chapter 10 contains three cryptanalytic results that are particularly illustrative in the context of this thesis.

Chapter 11 is completely devoted to cryptographic schemes that have modular arithmetic as their basic operations. After discussing some weaknesses in two particular schemes, we present a new block cipher design.

Finally, in Chapter 12 we give our conclusions and some directions for further research.

In Chapters 2 to 6 we clearly indicate what our own contributions are. In Chapters 7 to 12 this has been omitted since these contain only ideas of our own.

Chapter 2

Encryption, Ciphers and Hash Functions

2.1 Introduction

In this chapter we introduce the different types of single-key ciphers, encryption schemes and cryptographic hash functions. The design of these cryptographic schemes is the subject of our research.

Encryption schemes can be seen as modes of use of underlying cryptographic building blocks that are denoted by the term ciphers. The new and seemingly unnecessary distinction between ciphers and encryption schemes forced itself on us in trying to categorize the different primitives with respect to external characteristics. The resulting categorization is compact and elegant.

The different ciphers and encryption schemes are derived from a generic encryption scheme, thereby clearly indicating their mutual similarities, differences and relations. With respect to cryptographic hash functions, a generic model for sequential hashing is given.

All resulting primitives can be characterized by a small set of dimensions. For all cases we demonstrate that the choice of these dimensions is a compromise between the concern for repeated occurrences on the one hand and for error propagation on the other hand.

2.2 The generic encryption scheme

Informally, the purpose of a single-key encryption scheme is the transformation of a *message* M into a *cryptogram* C such that

- it is easy to extract M from C with the key,
- it is infeasible to derive M from C without the key.

We present a simple generic model to describe the external behavior of discrete single-key encryption schemes. In this model, messages are treated as sequences

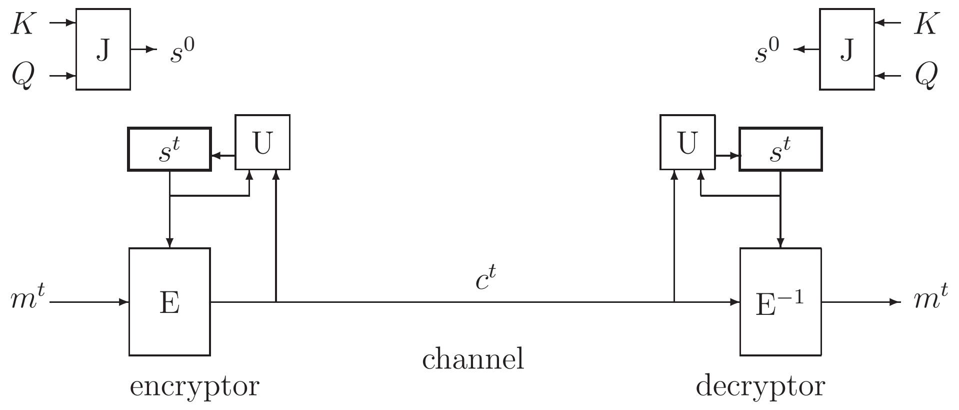


Figure 2.1: Generic model for encryption schemes.

of symbols: $m^1 m^2 \ldots$ These symbols belong to some finite set, called the *plaintext alphabet*. Encryption is performed by transforming the message symbols sequentially into cryptogram symbols. The cryptogram symbols belong to the (finite) *ciphertext alphabet*. A block diagram of the generic model is given in Fig. 2.1. It is described by three equations:

$$s^0 = \mathrm{J}(K, Q) \ , \tag{2.1}$$
$$s^{t+1} = \mathrm{U}[c^t](s^t) \ , \tag{2.2}$$
$$c^t = \mathrm{E}[s^t](m^t) \ . \tag{2.3}$$

The *encryption transformation* E converts a message symbol m into a cryptogram symbol c in an invertible way, parameterized by an *internal state* s. The *state updating transformation* U updates the internal state s^t to s^{t+1}, parameterized by the last cryptogram symbol c^t. In general, the internal state depends on all past cryptogram (and corresponding message) symbols. The state updating transformation U can be designed in such a way that the internal state depends only on the last n_{m} cryptogram symbols for some finite value of n_{m}. The resulting finite state machine is said to have *finite input memory* n_{m} [51]. The *initialization mapping* J converts a secret key K and a public parameter Q into the initial internal state s^0.

The channel models all possible deformations of the cryptogram between encryptor and decryptor. Typically, the channel represents a communication link or the process of storage on and retrieval from an information carrier.

2.2.1 Encryption through transposition

In our generic model the message symbols are sequentially *substituted* in a one-to-one fashion by cryptogram symbols. For the large class of discrete encryption schemes that are based on transposition rather than substitution, the given model does not adequately describe the essential mechanisms involved. In transposition-based schemes encryption is performed by changing the order of the message symbols in a key-dependent way. Many historical examples of this type of encryption schemes can be found in [58]. The application of transposition as encryption mechanism has some inherent limitations:

- **Concealment:** transposition fails to conceal some essentials characteristics of the message. This includes the frequency distribution of symbols in a message.
- **Flexibility:** for messages consisting of only a few symbols, the number of possible transpositions is too small to result in effective concealment.
- **Storage:** encryption and decryption are not simple sequential operations that can be done on-the-fly. In general, the transposition of message symbols needs a buffer with a size comparable to that of the complete message.

Because of these disadvantages transposition is nowadays only applied in or proposed for encryption in systems where substitution is technically infeasible and only a moderate level of protection is needed. A typical example is the scrambling of analog video images for pay-TV.

In [70] it is proposed to replace the row-by-row scanning of the analog frames of a video sequence by a key-dependent scanning pattern based on space-filling curves. The keys that determine the scanning patterns are renewed for every frame and are produced by an underlying sequence generator. Cryptanalysis of this proposal in [4] showed that large parts of the original image could easily be restored from the cryptogram by exploiting the correlation between subsequent frames.

In a pay-TV system described in [33] every line of a frame is cyclically shifted with one of 256 possible offsets, generated by a cryptographic number generator. As can be read in Chapter 10, the correlation between corresponding lines in subsequent frames can be exploited to gain sufficient information about the output of the specific number generator to allow for on-line descrambling without the key.

Although in these examples the data to be encrypted are continuous in time and magnitude, the encryption schemes are discrete since the space of possible transpositions in these schemes is finite. It can be observed that in both examples the transposition scheme is only part of a larger compound system that also involves the generation of key material to parameterize the transpositions. In both cases the application of a fixed transposition would be very weak. The security of the compound systems relies on the security of the key generation mechanisms.

Because of their small application domain and the limitations of the security that can be attained, the design of transposition-based encryption schemes is not treated in this thesis.

2.2.2 The real world

Our goal is to design discrete encryption schemes that are competitive in as many applications as possible in the real world. Therefore it is important to take into account the biases inherent in modern technology and their impact on certain design decisions.

To enable storage and transmission, data must be “written” or “coded” in some alphabet. Before the advent of computer technology and data automation, data was stored and transmitted mainly as characters printed on paper. The alphabet coincided with the "natural" alphabet of the corresponding language the document was written in, supplemented with punctuation marks, the digits 0 to 9 and possibly some special characters. Superficially, this did not change with the coming of computer technology. Instead of being written or printed on paper, the alphabet symbols were now stored on punched cards, magnetic tapes or disks. However, the symbols are not stored "as they are" but are first coded as sequences of binary values (called bits). These bits are represented on magnetic disk or tape as differently oriented magnetizations. In electronic circuits these bits are represented by low/high currents, low/high charges or low/high voltages. In light-based communication, bits can be represented by the presence or absence of a light pulse or as two orthogonal polarizations. Clearly, there is a very strong bias towards binary representation in modern communication and computer technology. Discrete data to be processed by modern automated systems, whether it represents a readable text, digitized sound or video or any other type of data, is invariably coded as a stream of bits.

Efficiency dictates that encryption schemes reflect the way messages are coded. In the rotor-based encryption devices [58] that were extensively used during World War II encryption was performed by sequentially substituting the message characters (belonging to the 26-letter natural alphabet) by cryptogram characters. This substitution was governed by the positions of rotors inside the device that were rotated according to a fixed schedule to make the substitution varying in time. Many other examples of encryption schemes working on natural alphabets can also be found in [58].

Efficient present-day encryption schemes process binary coded data. In the most simple case the plaintext alphabet consists of the two binary symbols that are denoted by 0 and 1. For the sake of encryption the bits may be grouped in *blocks*, *vectors* or *symbols*. Although it is not inconceivable that schemes could be designed where the length of these blocks varies during encryption, in practical schemes it is invariably a constant. If this fixed *symbol length* is denoted by n_{s}, the size of the plaintext alphabet is $2^{n_{\mathrm{s}}}$ and the plaintext symbols can be represented by n_{s}-bit vectors.

The invertibility requirement inherent in encryption implies that the size of the ciphertext alphabet is not smaller than the size of the plaintext alphabet. If the size of the ciphertext alphabet is larger than the size of the plaintext alphabet, the number of bits needed to code a ciphertext symbol is on the average larger than the symbol length. The resulting expansion is highly undesirable in storage and transmission. Therefore we will only consider the design of encryption schemes where the plaintext and the ciphertext symbols can both be denoted by n_{s}-bit symbols, with n_{s} a constant.

2.2.3 Some definitions

A binary variable or bit is a variable that can have only two values, denoted by 0 and 1. The set $\{0, 1\}$ is also denoted by $\mathbb{Z}_2$, i.e., the residues modulo 2. A binary *vector* consists of an array of binary-valued components, that are indexed starting from

0. A binary vector a with *dimension* (or *length*) n has components $a_0, a_1, \ldots, a_{n-1}$. The space of all binary vectors with dimension n is denoted by $\mathbb{Z}_2^n$.

A *Boolean function* $f(a)$ is a two-valued function with domain $\mathbb{Z}_2^n$ for some n. A *Boolean mapping* $h(a)$ maps $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^m$ for some n, m. A Boolean function is a special case of a Boolean mapping with $m = 1$. A Boolean mapping can be seen as the parallel application of m Boolean functions: $(h_1(a), h_2(a), \ldots, h_{m-1}(a))$. If $m = n$, the Boolean mapping is called a *transformation* of $\mathbb{Z}_2^m$. This transformation is called *invertible* if it is a bijection.

The addition modulo 2 of two binary variables α and β is denoted by $\alpha + \beta$. Hence, $\alpha+\beta$ equals 0 if $\alpha = \beta$ and 1 if $\alpha \neq \beta$. A binary variable can be *complemented* by adding 1 modulo 2, i.e., $\bar{\alpha} = \alpha + 1$. The bitwise addition, sum or difference of two binary vectors a and b is denoted by $a + b$ and consists of a binary vector c with components $c_i = a_i + b_i$. Sometimes the plus sign is used to denote arithmetic addition. The meaning of + will however always be clear from the context.

A Boolean mapping h is *linear* (with respect to bitwise addition) if

$$h(a + b) = h(a) + h(b) \ , \tag{2.4}$$

for all $a, b \in \mathbb{Z}_2^n$.

The *Kronecker delta function* $\delta(x)$ is equal to 1 if its argument is 0 and equal to 0 if its argument differs from 0. A binary vector with a single nonzero component at index i is denoted by δ_i.

2.2.4 Error sensitivity

In many modern applications the channel between encryptor and decryptor cannot be considered error-free. Transmission errors will occur due to a wide range of phenomena such as noise and fading in digital carrier modulation schemes. Therefore it is essential to investigate the operation of encryption schemes in the presence of transmission errors.

For many cases the transmission error behavior can be approximated by a symmetric binary memoryless channel. In this model the probability of a bit error is independent of its absolute value and independent of time. For typical error rates ($<$ 1%) the errors are isolated and appear to be randomly distributed over the stream.

The symmetric binary memoryless channel model describes only *sign* errors: every bit received has been sent, but its value may be altered in the channel. Imperfect synchronization due to bad recovery of timing can make it appear that bits are "lost" or that new bits are "created" during transmission. This type of error is called a *bit slip*.

In practice there are many transmission channels that cannot be modeled as memoryless. Due to man-made noise, fading and temporary loss of contact (especially in mobile communication), errors occur in long bursts interleaved by periods of relatively few isolated errors. Apart from the loss of data, these bursts also hamper the synchronization between sender and receiver.

In the generic model sufficient conditions for correct decryption of a message symbol m^t are:

- **Error-free reception:** of the cryptogram symbol c^t.
- **History:** internal state of the decryptor is equal to that of the encryptor. This implies
  - **Synchronism:** internal "clocks" of encryptor and decryptor are synchronized;
  - **Infinite input memory:** all cryptogram symbols that have been sent from the initialization onwards have been correctly received.

In our generic model a single transmission error can corrupt all following plaintext symbols. Because of this lack of robustness, almost all practical encryption schemes are described by restricted versions of the generic model. By imposing different robustness conditions a number of particular models can be deduced from the generic model. Every model corresponds in a natural way to a specific class of encryption schemes.

In the generic model the infinite input memory makes correct decryption of a cryptogram progressively less likely as the length of the original message grows. In many applications this accumulation of error probability cannot be tolerated. Therefore, we restrict ourselves to the design of encryption schemes that have a finite input memory. The internal state s^t of an encryption scheme with memory n_{m} is independent of all cryptogram symbols encrypted before time step $t - n_{\mathrm{m}}$. If $n_{\mathrm{m}} = 0$, the internal state is independent of the cryptogram symbols. The finite state machine consisting of the internal state and the state updating transformation is autonomous.

With the finite input memory requirement imposed, we identify the factors that determine the robustness behavior of an encryption scheme:

- **Symbol length**
  Because the cryptosystem decrypts the cryptogram one symbol at a time, a single sign error results in the faulty reception of a complete symbol. Since the cryptogram symbols consist of n_{s}-bit vectors, the error propagation due to this effect is proportional to the symbol length. If $n_{\mathrm{s}} \neq 1$, a bit slip can result in a loss of symbol alignment between sender and receiver.
- **Need for synchronism**
  If there is no need for synchronism, bit slips only have a local effect, i.e., only a limited number of message symbols are corrupted. In synchronous encryption schemes a bit slip or a burst resulting in loss of synchronization corrupts all following message symbols.
- **Input memory length**
  If the memory length is 0, a sign error results in the corruption of only a single message symbol. If the memory length is equal to a nonzero constant n_{m}, a sign error results in the corruption of up to $n_{\mathrm{m}} + 1$ message symbols.

Maximum robustness would be achieved by an encryption scheme with a minimum symbol length (a single bit), zero input memory and no need for synchronism. Such a scheme can however not offer any security since for a given key it invariably maps the symbol 1 to 1 or 0 and 0 to the complementary bit. For this system the *repeated occurrence* of a cryptogram symbol leaks information about the message. In general, the expected probability of repeated occurrences imposes a lower bound on the most important dimensions of the scheme. In practical encryption schemes the actual choice of these dimensions is a compromise between robustness and repeated occurrence concerns. In the following sections we describe the different types of encryption schemes that emerge when certain design decisions are made.

2.2.5 Stream versus block encryption

In most stream encryption schemes the message bits m^t are encrypted by adding an *encrypting* bit z^t modulo 2:

$$c^t = m^t + z^t \ . \tag{2.5}$$

Decryption corresponds to adding the encrypting bit again. This is called *single-bit* stream encryption. Stream ciphers are not restricted to an encryption rate of a single bit per time step. In more efficient designs every time step n_{s} encrypting bits are generated allowing the encryption of n_{s} message bits by simple bitwise addition. In this scheme every time step a *block* of n_{s} message bits are encrypted. Still, the encryption mechanism is clearly stream encryption since the encrypting transformation operates on the individual bits of the symbol block.

This encryption mechanism can be generalized by allowing more general group operations over the set of n_{s}-bit vectors. If $\odot$ denotes a group operation, we have

$$c^t = m^t \odot z^t \text{ and } m^t = c^t \odot (z^t)^{-1} \ . \tag{2.6}$$

z^t is called the *encrypting symbol* and the sequence of encrypting symbols is called the *encrypting sequence*. Although this operation does not encrypt the message bit by bit, it is not considered a block encryption scheme. In general, we consider an encryption transformation to be a block cipher if it is hard to reconstruct the encryption transformation from pairs m, c. In the case of a group operation the encryption transformation is completely determined by $z = c \odot m^{-1}$. The distinction between block and stream encryption can be summarized as follows.

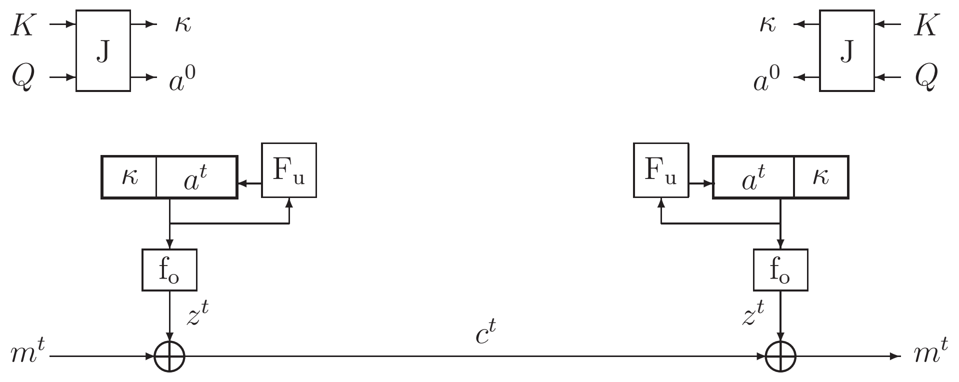


Figure 2.2: Synchronous stream encryption.

Given some plaintext-ciphertext symbol pairs,

- **Block encryption:** reconstruction of the encryption transformation is hard.
- **Stream encryption:** reconstruction of the encryption transformation is easy. The security relies on changing this transformation for every symbol in a way that is hard to predict for an adversary.

In the following it is is assumed that bitwise addition is used, since there is no additional security in choosing an encrypting transformation different from this. The encrypting symbol consists of n_s bits and is denoted by $z^t = z^t_0, z^t_1, \ldots, z^t_{n_\mathrm{s}-1}$. Finally, like block encryption, stream encryption with $n_\mathrm{s} \neq 1$ requires symbol alignment for correct decryption.

2.3 Synchronous stream ciphers

The heart of a synchronous stream encryptor (or decryptor) consists of an autonomous finite state machine, referred to as a *synchronous stream cipher* or equivalently a *cryptographic sequence generator*. The internal state of this finite state machine consists of the actual internal state a that is updated and the *cipher key* κ that is fixed during encryption. The updating transformation is parameterized by κ. The initial internal state and the cipher key are determined by the initialization mapping J. The encrypting symbol depends on the internal state by the output function $\mathrm{f_o}$. A block diagram of a synchronous encryption scheme is given in Fig. 2.2. The generation of encrypting symbols is governed by three equations:

$$\begin{aligned}(\kappa, a^0) &= \mathrm{J}(K, Q)\ , &(2.7)\\ a^{t+1} &= \mathrm{F_u}[\kappa](a^t)\ , &(2.8)\\ z^{t+1} &= \mathrm{f_o}[\kappa](a^t)\ . &(2.9)\end{aligned}$$

The length of the internal state a is denoted by n_a and the length of κ by n_κ.

If the encryption transformation is implemented by bitwise addition, sign errors in the cryptogram only affect the corresponding bits in the message. The disadvantage of synchronous encryption is the requirement of perfect synchronization between encryptor and decryptor. If synchronization is lost, the output of the decryptor is unintelligible for the receiver and synchronization has to be regained.

If the plaintext has sufficient redundancy to verify correctness of decryption, the receiver can recover synchronization without the intervention of the sender. This involves trying different offsets between the sender's and the receiver's clock. In applications where high-speed online decryption is needed, automatic synchronization recovery can be technically infeasible. The risk of synchronization loss can be reduced somewhat at the cost of inserting synchronization patterns in the ciphertext.

In most applications it is essential to limit the consequences of synchronization loss. This can be achieved by initializing at fixed time steps. For instance, if initialization occurs every μ symbols, synchronization loss will on the average result in the loss of $\mu/2$ symbols. The appropriate frequency of initialization depends on the probability of synchronization loss and the nature of the application. If there is a channel from the receiver to the sender, the receiver can request initialization to the sender upon detection of synchronization loss.

The aim of initialization is to ensure that encryptor and decryptor have the same internal state at a certain time. The presence of the public parameter Q in the initialization mapping J allows resynchronization without the introduction of new key material. Initialization consists of simultaneously loading encryptor and decryptor with a couple (a^0, κ) computed from the secret key K and an agreed upon value for the public parameter Q. For initialization on fixed time steps, Q stands for its serial number. For initialization upon request, Q can be the time given by a commonly available clock or a string that is sent along with the request.

Although in applications the possibility of initialization without the need for new key material is crucial for efficiency and robustness, most synchronous stream encryption proposals in literature do not mention an initialization mapping. When these schemes are used in practical systems, there is almost always an initialization mapping. It is however typically described at a higher architectural level, masking the interaction with the actual synchronous stream cipher. Chapter 10 shows that the specific choice of the initialization mapping can have an important impact on the cryptographic security of the resulting synchronous stream encryption scheme.

Since the encrypting stream is independent of the data to be enciphered, it can be generated in advance and stored for later use. This makes synchronous stream encryption the most flexible encryption mechanism, ideal for encryption of data packets in computer networks. Encryption of the communication between two nodes sharing a secret key can be performed by bitwise addition of an encrypting stream that

- corresponds to the key K and a parameter Q specific for the package, i.e., its (not to be encrypted) identifier,
- corresponds to a substring of a precomputed encrypting stream with the offset sent along with the package.

2.3.1 Repeated occurrences

If the same value of Q (and K) is used for two different initializations, the corresponding encrypting sequences are equal. This allows an adversary to calculate the bitwise difference of the two corresponding message sequences from the cryptogram sequences and should therefore be avoided. This imposes a lower bound on the number of possible values for Q.

In practical implementations the number of possible internal states a is finite and hence the iteration of the cryptographic sequence generator must end up in a finite cycle. To avoid repetitions in the encrypting sequence, these cycles must be sufficiently long. It is therefore useful to study the cyclic behavior of the cryptographic sequence generators. In general, it is hard to predict the length of a cycle starting from a specific initial state. However, for updating transformations that can be considered as representative members of certain classes of transformations, fairly reliable statistical predictions can be made. In our analysis we make a distinction between three classes: the set of transformations of $\mathbb{Z}_2^{n_\mathrm{a}}$, the set of invertible transformations of $\mathbb{Z}_2^{n_\mathrm{a}}$ and transformations with additional algebraic structure.

Let the state updating transformation be uniformly chosen from the set of transformations of $\mathbb{Z}_2^{n_\mathrm{a}}$. After loading an initial state a^0, the state updating transformation is iterated until the internal state a^{τ_1} is equal to a state a^{τ_0} with $0 \leq \tau_0 < \tau_1$. Clearly a^{τ_1} is on a cycle with length $\tau_1 - \tau_0$. The sequence of internal states from the a^0 to a^{τ_0} is called the *tail*. In [41] it is shown that with a state length of n_a, the expected tail and cycle lengths are equal and given by $(\sqrt{\pi/8})2^{n_\mathrm{a}/2}$. The expected total number of states on cycles is only $(\sqrt{\pi/2})2^{n_\mathrm{a}/2}$.

Let the state updating transformation be uniformly chosen from the set of invertible transformations of $\mathbb{Z}_2^{n_\mathrm{a}}$. Since every state has a single predecessor, all states are on cycles and there are no tails. According to [41], the cycle length distribution is flat, i.e., every cycle length between 1 and 2^{n_a} is equiprobable. The probability that a cycle starting from a given state is shorter than some value λ is equal to $\lambda/2^{n_\mathrm{a}}$ and the expected cycle length is $2^{n_\mathrm{a}-1}$. The risk of short cycles cannot be completely eliminated but it can be made arbitrarily small by augmenting the state length n_a. Hence, invertible updating transformations are generally more efficient in the use of the state space than updating transformations that are not invertible.

If the updating transformation has some additional algebraic structure these statistical predictions are no longer valid. The most important class of finite state machines with this type of updating transformation are linear feedback shift registers [95]. These can be designed in such a way that all internal states (except the all-0 state) lie on a single cycle. In this case the risk of short cycles can be completely eliminated by choosing the length of the linear feedback shift register sufficiently large.

2.3.2 Filtered counter stream ciphers

The decryption of small parts of a long cryptogram obtained by synchronous encryption can become very inefficient. Decryption of a single cryptogram symbol m^i

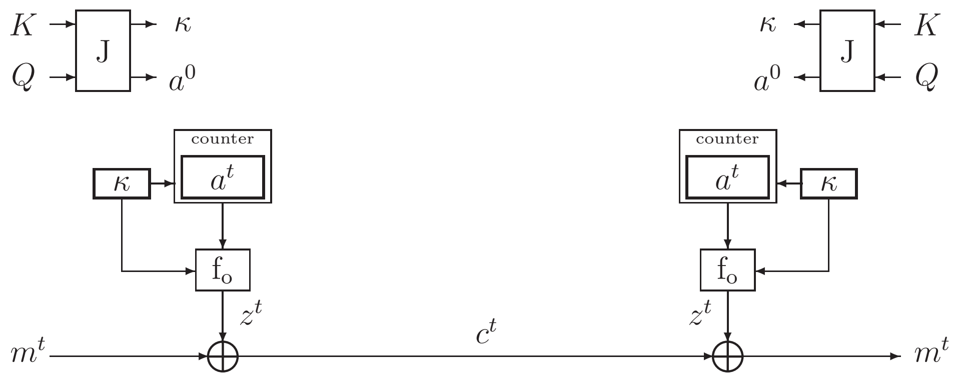


Figure 2.3: Filtered counter stream encryption.

requires the knowledge of the corresponding internal state a^i. This internal state can be obtained by loading the state register with the initialization values a^0 and κ corresponding to the known key K and parameter Q and subsequently iterating the stream cipher i times. The effort is the same as for decryption of all cryptogram symbols before m^i. In general, individual message symbols are not easily accessible from the cryptogram if encryption is synchronous.

The efficiency problems with decryption of a single cryptogram symbol are due to the iterations of the cipher necessary to transform a^0 to a^i. For certain updating transformations, a^i can be calculated efficiently from a^0 and i. For example, if the updating transformation consists of a simple incremental counter, we have $a^i = a^0 + i$. Other examples of updating transformations that allow such a shortcut are linear updating transformations such as in linear feedback shift registers or congruential generators with an updating transformation given by $a^{i+1} = (\mathrm{u}a^i + \mathrm{v}) \bmod \mathrm{m}$.

Stream ciphers with the property that ${\mathrm{F_u}}^i$ can be efficiently calculated from $\mathrm{F_u}$ are called *filtered counter stream ciphers* or, equivalently, *cryptographic filtered counters*. The algebraic structure inherent in the updating transformation determines the cyclic behavior of these finite state machines. By choosing the "counter" parameters in an intelligent way, all states can be arranged in long cycles. Hence, the two most important motivations for using filtered counter stream ciphers are the relative ease of access to individual message symbols and the lack of short cycles. A disadvantage of most filtered counter stream ciphers is that the updating transformation is very simple and calls for a "strong" output function $\mathrm{f_o}$. The operation of a filtered counter stream cipher is illustrated in Fig. 2.3.

Repeated Occurrences

For filtered counter stream ciphers the design of the initialization mapping is critical. Suppose for instance that the counter is a simple arithmetic incrementing counter and the initialization mapping simply maps K to κ and Q to the initial state. If the adversary chooses two values of Q that differ only by a small number, one of the corresponding encrypting sequences is simply a delayed version of the other. In

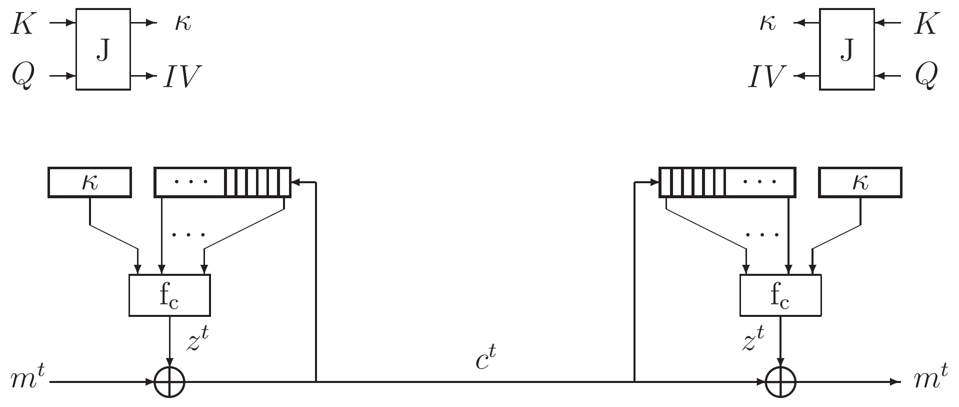


Figure 2.4: Self-synchronizing stream encryption.

some cases it may also be possible to choose the difference in Q in such a way that the input to the output function for some time step after initialization is equal for both sequences.

2.4 Self-synchronizing stream ciphers

In self-synchronizing stream encryption the encrypting bits only depend on the last n_{m} (called the memory) cryptogram symbols and a *cipher key* κ. This is modeled by a self-synchronizing stream cipher consisting of a shift register that contains the last n_{m} cryptogram symbols and a *cipher function* $\mathrm{f_c}$. This is in fact only a conceptual model to illustrate the external dependencies. In a specific design the finite input memory of the self-synchronizing stream cipher can be realized in numerous ways. A block diagram of self-synchronizing stream encryption is given in Fig. 2.4. The generation of encrypting symbols is governed by three equations:

$$(\kappa, IV) = \mathrm{J}(K, Q) \ , \tag{2.10}$$
$$c^{-n_{\mathrm{m}}+1} \dots c^0 = IV \ , \tag{2.11}$$
$$z^t = \mathrm{f_c}[\kappa](c^{t-n_{\mathrm{m}}} \dots c^{t-1}) \ . \tag{2.12}$$

Unlike for synchronous stream encryption, the initialization mapping is not an essential component of the self-synchronizing encryption scheme. Loading the key K and the initial value IV without the intervention of an initialization mapping would in no way restrict the application domain. However, the presence of the initialization mapping is important in the description of certain cryptographic attacks that exploit relations between keys, such as the so-called *chosen-key* attacks described by Eli Biham [7].

Decryption is correct if the last n_{m} cryptogram symbols have been correctly received. An isolated error on the channel between encryptor and decryptor gives rise to an extra burst of n_{m} potentially incorrectly decrypted symbols, i.e., $n_{\mathrm{m}}n_{\mathrm{s}}$ incorrectly decrypted bits at the receiver. If the transmission error behavior can be modeled by a binary memoryless channel, self-synchronizing stream encryption is only practical if the bit error rate is significantly smaller than $(n_\mathrm{m} n_\mathrm{s})^{-1}$. In this case the presence of the decryptor multiplies the error rate by a factor of $n_\mathrm{m} n_\mathrm{s}$.

For channels that suffer from error bursts, self-synchronizing stream encryption is the natural solution. In this setting the error propagation effect of encryption is only an extension of every burst with $n_\mathrm{m} n_\mathrm{s}$ bits. If n_s differs from 1, there can however be an alignment problem after a burst.

The specific advantage of self-synchronizing stream encryption (especially binary, i.e., with $n_\mathrm{s} = 1$) is that they can be built on top of any digital communication system with minimum cost. This is what makes self-synchronizing stream encryption a very popular encryption mechanism.

2.4.1 Repeated occurrences

For a given κ the cipher function maps all vectors consisting of n_m symbols to the ciphertext alphabet. A cryptanalyst with temporary access to the input and output of a decryptor or encryptor can reconstruct part of this mapping. Observed cryptograms can then be scanned for occurrences of n_m-symbol strings with known cipher function output, giving a single symbol, i.e., n_s bits of plaintext for every occurrence. The expected ratio of ciphertext that can be decrypted in this way is equal to the number of observed outputs divided by $2^{n_\mathrm{m} n_\mathrm{s}}$. For example, if the mapping is reconstructed for 2^ℓ inputs, scanning a cryptogram of 2^ω symbols yields on the average $2^{\ell+\omega-n_\mathrm{m} n_\mathrm{s}}$ message symbols. This only becomes significant if $\ell + \omega$ is larger than $n_\mathrm{m} n_\mathrm{s}$.

A cryptanalyst with access to the cryptogram only can scan it for multiple occurrences of n_m-symbol strings. For every repeated occurrence, the bitwise addition of the two message symbols immediately following these strings is equal to the bitwise addition of two corresponding cryptogram symbols. This yields n_s bits of message information. Since every cryptogram symbol depends on all previous message symbols in a complicated way, the distribution of symbol sequences occurring in the cryptogram can be considered unbiased. Measurable biases in the distribution of subsequences in the cryptogram indicate cryptographic weaknesses in the self-synchronous encryption scheme. The probability of having at least one repeated occurrence, leaking n_s bits of plaintext information, only becomes significant if the length of the ciphertext is of the order $2^{n_\mathrm{m} n_\mathrm{s}/2}$. This is in fact a manifestation of the well-known *birthday paradox* [84, p. 273].

2.4.2 Output feedback (OFB) mode

Feeding the output of the cipher function of a self-synchronizing stream cipher back into the shift register yields an autonomous finite state machine. This machine can be used as the cryptographic sequence generator in a synchronous stream encryption scheme. This is illustrated in the left-hand side of Fig. 2.5.

This straightforward construction has the important drawback that the resulting updating transformation is not invertible. If the symbol length is 1, it is expected that half of the states have a single predecessor, one in four has no predecessors and

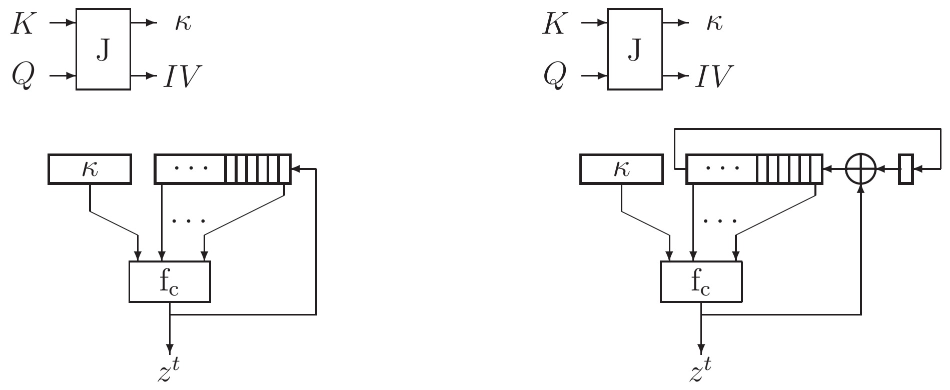


Figure 2.5: OFB mode of a self-synchronizing stream cipher. At the left: non-invertible updating transformation, at the right: invertible updating transformation.

one in four has two predecessors. For larger symbol length $n_{\mathrm{s}} > 4$ the number of predecessors of a state is expected to have a Poisson distribution with $\lambda = 1$, i.e., $\Pr(k) = \mathrm{e}^{-1}/k!$. This yields a state-transition diagram with tails leading to cycles with a typical length smaller than the square root of the total number of states.

This undesirable cyclic behavior can be avoided by modifying the feedback in such a way that the updating transformation is invertible. In the right-hand side of Fig. 2.5 it is shown how this can be realized by introducing an extra memory cell and a bitwise addition. The predecessor b of a state a can be obtained by shifting the contents of the shift register a single cell to the right, where the contents of the leftmost cell is obtained from the additional memory cell. The predecessor value of the additional cell can be obtained by bitwise addition of the rightmost shift register cell of a and $\mathrm{f_c}(\kappa, b)$. For this construction the cycle length is expected to have a flat distribution.

2.5 Block ciphers

A block cipher is an invertible transformation parameterized by a cipher key κ. An essential property of a block cipher is that it is hard to determine the value of κ given matching pairs of message and cryptogram blocks. A block cipher is a component that can be used in many different modes [38]. In this section we present the five most important modes. Two of these modes constitute block encryption schemes and three stream encryption schemes. These modes have been standardized by ISO [52].

In all these schemes there is an explicit initialization mapping. As in the case of self-synchronizing stream ciphers, the initialization mapping is not an essential component for all the modes. Using K for κ and (for some of the modes) Q for IV does not restrict their application domain. The presence of the initialization mapping is needed to model the resistance against chosen-key attacks [7] and because it is sometimes actually used in practice.

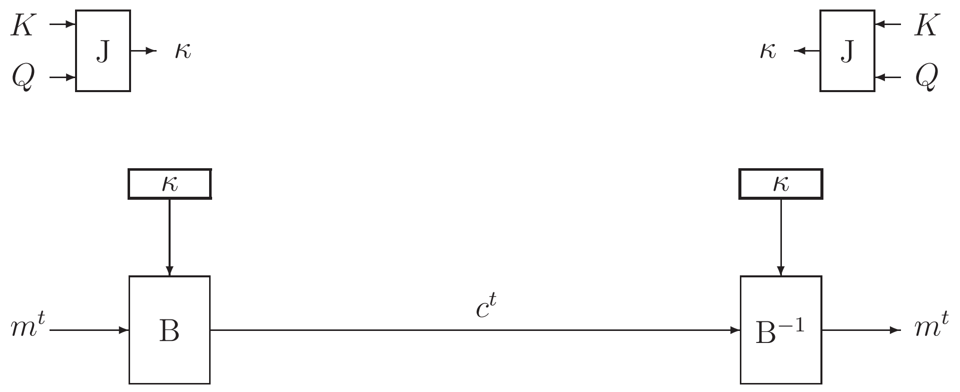


Figure 2.6: Block cipher in ECB mode.

2.5.1 Electronic code book (ECB) mode

Simple block encryption or the ECB mode of a block cipher is conceptually the simplest of encryption methods. Encryption and decryption are described by

$$\begin{aligned} \kappa &= \mathrm{J}(K, Q) \ , & (2.13) \\ c &= \mathrm{B}[\kappa](m) & (2.14) \\ m &= \mathrm{B}[\kappa]^{-1}(c) \ . & (2.15) \end{aligned}$$

The corresponding block diagram is given in Fig. 2.6. The block length is denoted by n_{b}. Isolated sign errors propagate only over a single block while bit slips may result in the loss of block alignment between sender and receiver.

Ciphertext stealing

If a message consists of a number of bits that is not a multiple of the block length n_{b}, the last incomplete block can be padded in some reversible way and encrypted. An example of reversible padding is appending a single 1 followed by zeroes until the length is a multiple of the block length. This padding operation results in a cryptogram that is longer than the message. This can give storage or bandwidth problems if many relatively short messages have to be encrypted. In [79, p. 78] a method is given to encrypt all messages not smaller than the block length in a non-expanding way. The message is simply encrypted block by block except the last complete block and the incomplete block. This method is called *ciphertext stealing* and is illustrated in Fig. 2.7.

Repeated Occurrences

If the statistical distribution of the plaintext blocks can be considered uniform, the analysis of repeated occurrences is analogous to that for self-synchronizing encryption schemes, with the cipher function mapping replaced by the block cipher transformation and $n_{\mathrm{m}} n_{\mathrm{s}}$ replaced by the block length n_{b}. However, in most practical cases the statistical distribution of the plaintext symbols may not be considered

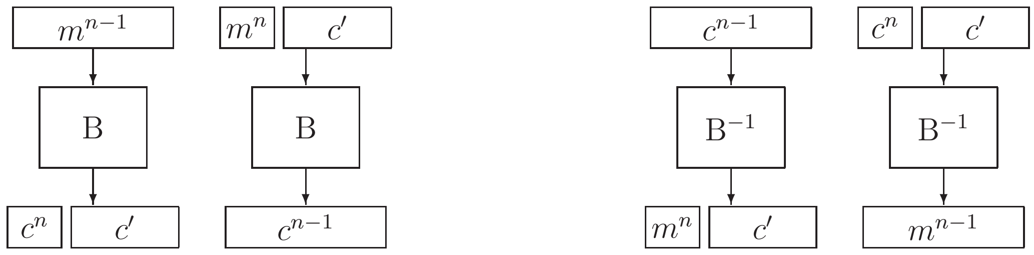


Figure 2.7: Ciphertext stealing for ECB block encryption (left) and corresponding decryption (right). The block c' is just an intermediate result and is not a part of the cryptogram.

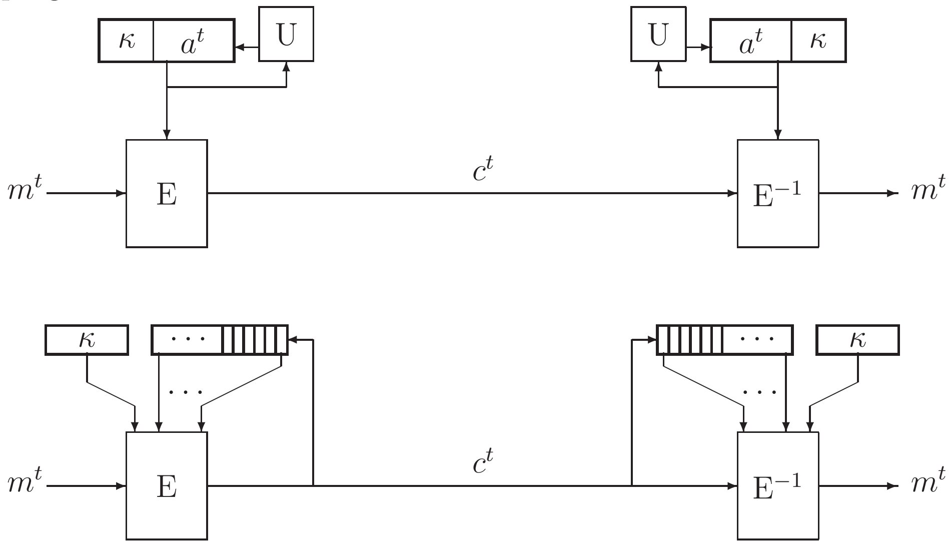


Figure 2.8: Synchronous (above) and self-synchronizing (below) block encryption.

uniform. Messages often are highly redundant, resulting in repeated blocks, even for relatively short messages. Since the encryption is context independent, the equality of message blocks can be detected by checking for repeated cryptogram blocks. Clearly, the repeated block occurrences in the message could be eliminated by proper compression before encryption.

Still, ECB block encryption has the important drawback that the quality of the concealment it offers depends on the structure of the message itself. This has led to the design of a block cipher mode that makes the block encryption context dependent without augmenting the system complexity and affecting the robustness too much: the cipher block chaining mode.

2.5.2 Cipher block chaining (CBC) mode

There are essentially two robust mechanisms to make block encryption context dependent. These are illustrated in Fig. 2.8. In one of them the encryption transformation is made to depend on time. The encryption transformation is parameterized by the internal state of an autonomous finite state machine, resulting in a *synchronous block encryption scheme*. In fact, most of the rotor-based encryption machines in use before, during and after World War II [58] perform synchronous block encryption, with the blocks consisting of characters. This excludes the famous Hagelin M-209 cipher machine [58], which performs synchronous stream encryption with the symbols consisting of characters. A more recent example is the proposal of Cees Jansen and Dick Boekee in [56]. In this system the cipher key of the block cipher consists of the internal state of an autonomous finite state machine.

The alternative mechanism is to make the encryption transformation depend on a finite number of past cryptogram symbols, resulting in a *self-synchronizing block encryption scheme*.

In many practical block cipher implementations, changing the cipher key κ is a time-consuming operation, typically a few times slower than performing a single encryption. Therefore, most schemes avoid the need for frequent changes in κ during encryption. This restricts the application of the context dependent parameters in the encryption scheme to bitwise addition (or any efficient invertible operation) before or after encryption.

In the case of synchronous block encryption, the additional system complexity can be reduced to a minimum by taking a very simple finite state machine such as an incrementing counter or a linear feedback shift register. However, due to the predictability of these finite state machines, addition after encryption will in general not prevent the detection of repeated blocks in the message. For example, a sequence of several equal blocks cannot be disguised by the addition of the highly regular sequence. Addition before encryption, on the other hand, has the problem that there can be interference between the message and the output of the simple finite state machine, giving rise to repeated inputs to the block cipher. The concealment is still not message independent. In both cases the concealment can be made message independent if the finite state machine is realized by a synchronous stream cipher. However, if a synchronous stream cipher is needed anyway, one might as well remove the block cipher from the scheme and just do synchronous stream encryption.

A better alternative is self-synchronizing block encryption. The statistical distribution of the input to the block cipher can be made uniform by adding a value to the message block that depends on previous cryptogram blocks. The simplest and most robust solution is to take for this purpose the previous cryptogram block itself. We have

$$(\kappa, IV) = \mathrm{J}(K, Q) \;, \tag{2.16}$$

$$c^0 = IV \;, \tag{2.17}$$

$$c^t = \mathrm{B}[\kappa](m^t + c^{t-1}) \text{ and } m^t = c^{t-1} + \mathrm{B}[\kappa]^{-1}(c^t) \;. \tag{2.18}$$

This is called the cipher block chaining (CBC) mode of a block cipher and is illustrated in Fig. 2.9. In this scheme every ciphertext depends in a complicated way on all past message blocks and can be considered uniformly distributed. Error propagation is limited since a message block only depends on two cryptogram blocks.

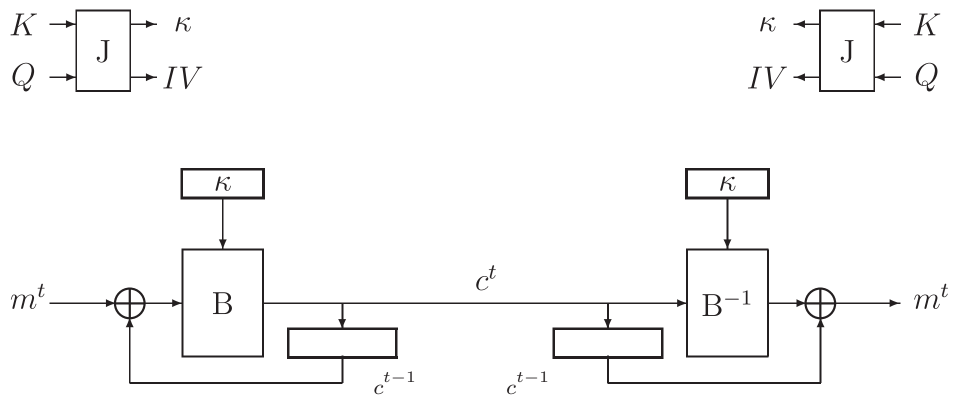


Figure 2.9: CBC mode of a block cipher.

Repeated occurrences

As in the case of the ECB mode, a cryptanalyst with temporary access to the input and output of a decryptor or encryptor can reconstruct part of the block cipher mapping. Observed ciphertext can then be scanned for occurrences of cryptogram blocks of which the corresponding plaintext is known, giving a message block for every occurrence. The expected ratio of ciphertext that can be decrypted in this way is equal to the number of obtained plaintext-ciphertext block pairs divided by 2^{n_b}.

A cryptanalyst who only has access to the cryptogram can scan it for multiple occurrences of blocks. Say we find $c^i = c^j$. We have $B(m^i + c^{i-1}, \kappa) = B(m^j + c^{j-1}, \kappa)$, hence $m^j = m^i + c^{i-1} + c^{j-1}$. Since the cryptogram symbols are freely available to the cryptanalyst, this gives n_b bits of information about the message. The probability of having at least one repeated occurrence, leaking n_b bits of plaintext information, only becomes significant if the length of the ciphertext is of the order $2^{n_\mathrm{b}/2}$.

Ciphertext stealing

In [79, p. 81] it is shown that ciphertext stealing is also possible in CBC block encryption. The mechanism is illustrated in Fig. 2.10.

2.5.3 Output feedback (OFB) mode

The updating transformation of a cryptographic sequence generator can be realized by a block cipher, resulting in a synchronous stream encryption scheme. The most obvious way to do this is given in Fig. 2.11. For this construction we have

$$(\kappa, s^0) = \mathrm{J}(K, Q) \ , \tag{2.19}$$

$$a^{t+1} = \mathrm{B}[\kappa](a^t) \ , \tag{2.20}$$

$$z^{t+1} = \mathrm{sel}(a^t) \ . \tag{2.21}$$

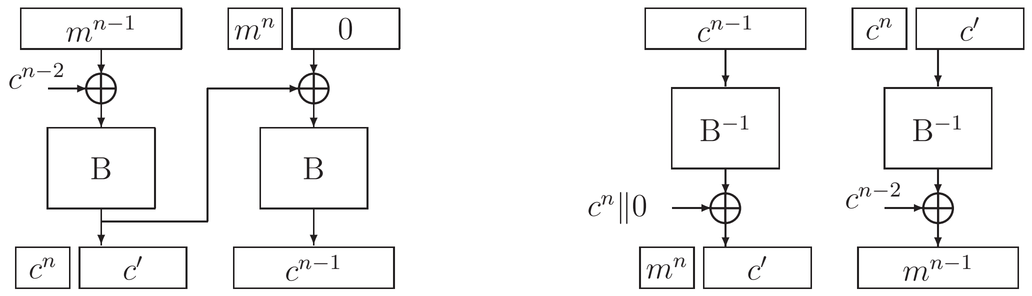


Figure 2.10: Ciphertext stealing in CBC block encryption (left) and corresponding decryption (right). The block c' is just an intermediate result and is no part of the cryptogram.

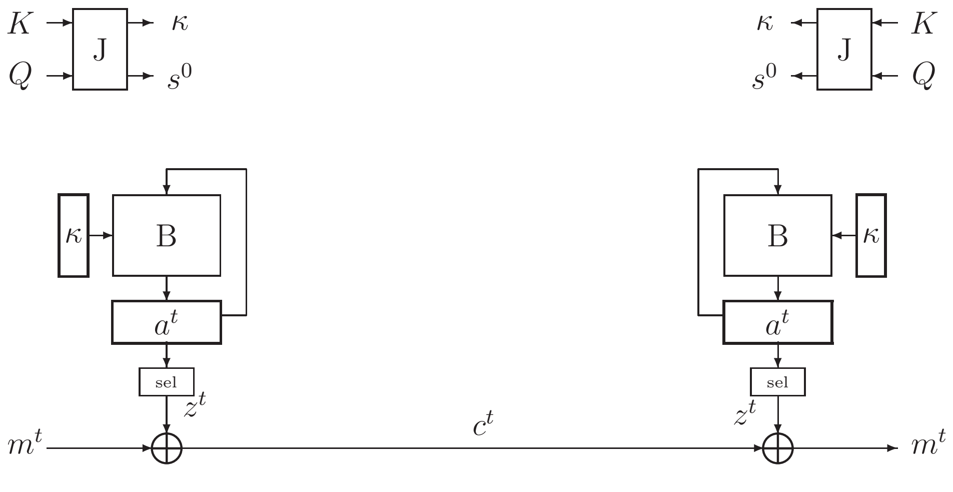


Figure 2.11: OFB mode of a block cipher.

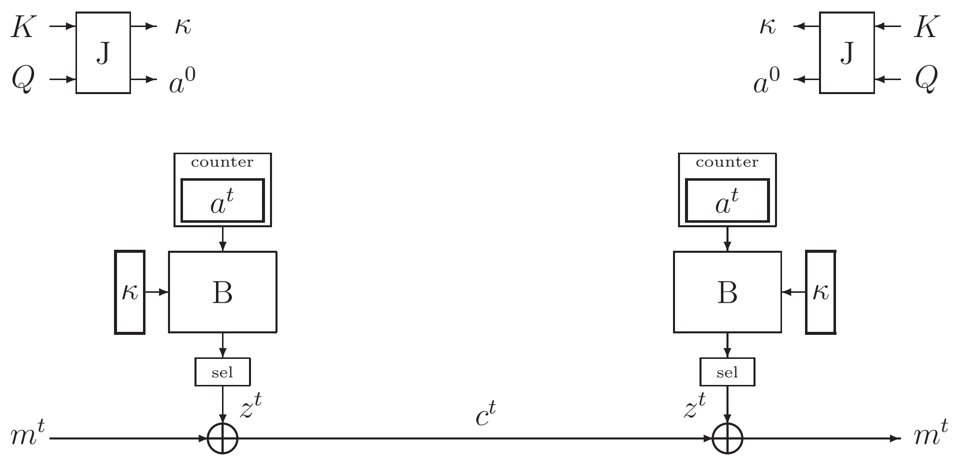


Figure 2.12: FCS mode of a block cipher.

The functional block denoted by "sel" selects n_{s} bits of a. It can be seen that the updating transformation is invertible. The cycle length starting from a certain initial state has a flat distribution. The symbol length n_{s} can range between 1 and the block length n_{b}. The efficiency of this mode relative to the ECB mode is $n_{\mathrm{s}}/n_{\mathrm{b}}$.

2.5.4 Filtered counter scheme (FCS) mode

The output function of a filtered counter stream cipher can be realized by a block cipher, resulting in a synchronous stream encryption scheme. This is illustrated in Fig. 2.12 and described by

$$(\kappa, a^0) = \mathrm{J}(K, Q) , \tag{2.22}$$
$$a^{t+1} = a^t + 1 \bmod 2^{n_{\mathrm{b}}} , \tag{2.23}$$
$$z^{t+1} = \mathrm{sel}(\mathrm{B}[\kappa](a^t)) . \tag{2.24}$$

Here (2.23) is just an illustrative example of a counter-like updating transformation. This can also be a linear feedback shift register or a congruential generator. In the case of (2.23) all internal states are on a single cycle.

2.5.5 Cipher feedback (CFB) mode

The cipher function in a self-synchronizing stream cipher can be realized by a block cipher, called the CFB mode of a block cipher. This is illustrated in Fig. 2.13 and described by

$$(\kappa, IV) = \mathrm{J}(K, Q) , \tag{2.25}$$
$$c_{-n_{\mathrm{m}}+1} \dots c_0 = IV , \tag{2.26}$$
$$z^t = \mathrm{sel}(\mathrm{B}[\kappa](c^{t-n_{\mathrm{m}}} \dots c^{t-1})) . \tag{2.27}$$

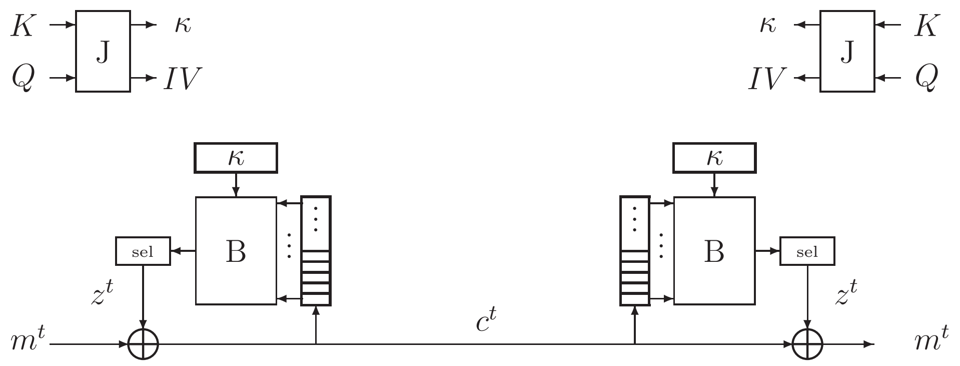


Figure 2.13: CFB mode of a block cipher.

The symbol length n_s can range from 1 to the block length n_b. The input memory of the self-synchronizing stream cipher is $\lceil n_\mathrm{b}/n_\mathrm{s} \rceil$. The efficiency of the block cipher in this mode relative to the ECB mode is $n_\mathrm{s}/n_\mathrm{b}$.

2.6 Ciphers and encryption schemes

In the preceding sections we identified a number of different types of encryption schemes and a number of different types of ciphers that can be used in the realization of these encryption schemes. Table 2.1 gives an overview of the relations between ciphers and encryption schemes.

type of cipher:	sync. stream	s.-s. stream	block
sync. stream encryption	straight	OFB	OFB/FCS
s.-s. stream encryption	-	straight	CFB
simple block encryption	-	-	ECB
CBC block encryption	-	-	CBC

Table 2.1: Modes of use of ciphers to implement different types of encryption schemes.

2.7 Cryptographic hash functions

A hash function maps messages of arbitrary length to a hash result with a fixed length. Cryptographic hash functions are designed to make the mapping "resistant against analysis." What we mean by this will be treated in Chapter 3.

Next to ordinary cryptographic hash functions, there are also cryptographic hash functions that are parameterized by a secret key. The external behavior of a cryptographic hash function is simply described by

$$h = \mathrm{H}(M) \ , \tag{2.28}$$

and of a keyed cryptographic hash function by

$$h = \mathrm{H}[K](M) \ . \tag{2.29}$$

The hash result h consists of a bit string of a specified length denoted by n_h.

In the hashing of long messages, it may be infeasible to store the complete message during the computation of the hash result. However, this is no problem for practical cryptographic hash function proposals. They all operate in a sequential manner, so that messages can be hashed on-the-fly without the need for external storage. Almost all existing cryptographic hash function proposals can be described by the succession of the following operations:

- **Segmentation:** the input is divided into a number (denoted by s) of blocks m^i of equal length and the last, generally incomplete, block m^s is padded in a unique and reversible way;
- **Initialization:** the initial *chaining state* d^0 is set equal to a value IV fixed by the specification;
- **Iteration:** the chaining state is updated sequentially by a *chaining transformation* G for all message blocks, starting from m^1 and ending with m^s;
- **Result:** the hash result is calculated from the final chaining state by the *output transformation* $\mathrm{G_o}$.

The chaining of a sequential keyed cryptographic hash function is described by

$$(d^0, \kappa) = \mathrm{J}(K) \ , \tag{2.30}$$
$$d^i = \mathrm{G}[\kappa](d^{i-1}, m^i) \ , \tag{2.31}$$
$$h = \mathrm{G_o}[\kappa](d^s) \ . \tag{2.32}$$

For a sequential non-keyed hash function we have

$$d^0 = \mathrm{IV} \ , \tag{2.33}$$
$$d^i = \mathrm{G}(d^{i-1}, m^i) \ , \tag{2.34}$$
$$h = \mathrm{G_o}(d^s) \ , \tag{2.35}$$

with IV fixed by the specification. A block diagram of the generic model for sequential hashing is given in Fig. 2.14.

In many designs it is proposed to append the length of the input in the padding operation to avoid attacks based on *fixed points*, such as described on p. 143. In the scheme of Fig. 2.14 this can be realized by including a counter in the chaining transformation. If it is detected that the last (incomplete) block arrives, the chaining transformation performs the padding operation before updating the chaining state.

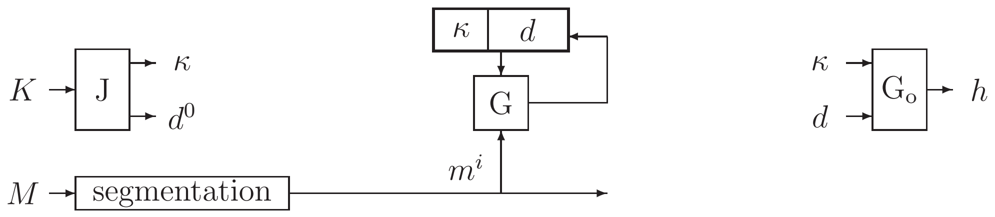


Figure 2.14: Generic model for sequential hash functions.

2.7.1 Hierarchical hashing schemes

In some applications the message to be hashed may be a very large data structure, consisting of a hierarchy of files and (sub)directories. A hash result may be required for every file and directory. This hierarchy can be reflected in the calculation of the hash results by a recursive structure. The hash result of a file is given by the simple application of the hash function on the file itself. The hash result of a directory is given by the application of the hash function on the message consisting of the concatenation of the hash results of its subdirectories and files.

Such a hierarchical hashing scheme can also be used for speeding up the hashing process for a single large file by splitting it up into a number of parts. The parts can be hashed in parallel by a number of processors. The hash result of the file is obtained by applying the hash function to the message that consists of the concatenation of hash results corresponding to the parts. This is a variant of a scheme proposed by Ivan Damgård in [29].

For the hash results to be reproducible it is necessary that the order of the intermediate hash results and the splitting of the file are fully specified.

2.7.2 Repeated occurrences

In most applications of cryptographic hash functions it is essential for security that, given a hash result, it is infeasible to find a message that hashes to this value. This property is denoted by the term *preimage-resistant*. The property that, given a message and its hash result, it is infeasible to generate a second message that hashes to the same result is denoted by the term *2nd preimage-resistant*. Another important property is *collision-resistance*, denoting that it is infeasible to find a pair of colliding messages, i.e., that hash to the same result.

The probability of having a collision in a set of ℓ unrelated messages becomes large if ℓ is larger than $2^{n_\mathrm{h}/2}$. This is again a manifestation of the birthday paradox. Finding a collision in such a set of ℓ messages in a straightforward way requires the generation and storage of the ℓ hash results and finding a matching pair by cumbersome sorting procedures. In practice the memory requirements of such an attack are more restrictive than the work factor. However, in [87, 88] Jean-Jacques Quisquater and Jean-Paul Delescaille presented an elegant algorithm to generate collisions with an expected work factor of approximately $2^{n_\mathrm{h}/2}$ applications of the hash function and with negligible memory requirements. In this attack, the hash function $\mathrm{H}(M)$ is used as a transformation $\vartheta(a)$ in the space of n_{h}-bit vectors. Starting from some initial value a^0, a sequence is generated by the iterated application of ϑ, i.e., $a^{i+1} = \vartheta(a^i)$. For "good" hash function designs, the state-transition diagram generated by $\vartheta(a)$ has a structure that corresponds to that of a transformation chosen uniformly from the space of all possible transformations (see p. 16). This means that it can be expected that after $(\sqrt{\pi/2})2^{n_{\mathrm{a}}/2}$ iterations of ϑ, a state a^{τ_1} is reached equal to some earlier state a^{τ_0}. This state is called the *point of contact* and marks the place where the tail is joined to the cycle. Clearly, the state a^{τ_1} (or equivalently a^{τ_0}) has two preimages: $a^{\tau_1 - 1}$ on the cycle and $a^{\tau_0 - 1}$ on the tail. Hence, these two preimages form a colliding pair of messages: $\mathrm{H}(a^{\tau_0 - 1}) = \mathrm{H}(a^{\tau_1 - 1})$. The point of contact can be located at the cost of a negligible number of additional iterations by storing only past states which satisfy a specific condition. This is called the method of *distinguished points* and was proposed by the same authors in [86]. Clearly, this collision-generating attack imposes a lower bound on n_{h}.

If the hash function is sequential, a collision may also appear in an intermediate chaining state. Two different partial messages leading to equal chaining states can be turned into colliding messages by simply completing both messages with equal blocks. Hence, it makes no sense to have a hash result longer than the chaining state.

It can be argued whether the chaining transformation should be invertible or not. By invertibility is meant that it is easy to calculate the predecessor of a chaining state given the corresponding message block. Invertibility allows the generation of a message with a given hash result by performing a meet-in-the-middle attack. Before the attack a *target* final chaining state is chosen that corresponds to the desired hash result. For a large set of *message heads* an intermediate chaining state is generated by forward iteration from the initial state. For another large set of *message tails* an intermediate chaining state is generated by backward iteration from the target state. Subsequently, the two sets are inspected for matches in the intermediate chaining state. By concatenating a head and tail with matching intermediate chaining states a message is obtained that hashes to the desired result. The probability of success becomes significant if the product of the sizes of the two sets of messages is of the order of magnitude of the size of the state space. The lower bound on the length of the chaining state imposed by this attack has the same order of magnitude to that imposed by collisions in the chaining state.

If the chaining transformation is non-invertible, this attack is not possible. However, in this case there must exist pairs of chaining states mapping to the same chaining state with equal message blocks. It is not inconceivable that for some proposals this property could be exploited in a global attack.

Finally, for a hierarchical hashing scheme it can be seen that a collision in some node implies a collision for the hash function itself.

2.8 Conclusions

The main part of this chapter has been devoted to our taxonomy of encryption schemes and ciphers. We have shown that the different families of encryption schemes and ciphers emerge in a natural way from a generic model by imposing robustness conditions. The choice of the dimensions is a compromise between robustness and concern for repeated occurrences.

Chapter 3

Cryptographic Security

3.1 Introduction

Before attacking the actual problem of designing cryptographic primitives, this chapter focuses on the limits to the specification and realization of their cryptographic properties. We start by introducing the cryptanalytic setting by defining the different ways of access of the cryptanalyst. This is followed by a description of the important and universal attack of exhaustive key search.

Given a clear and practically relevant definition of security, it would be very valuable to have a set of practical cryptographic schemes that can be proved secure with respect to this definition. The difficulties that arise in trying to achieve this are discussed in Sect. 3.3. We argue that the traditional approaches to the design of provably secure schemes are irrelevant to the design of cryptographic hash functions, single-key ciphers and encryption schemes. In Sect. 3.4 we show that terms such as "cryptographic security" actually must be seen as carriers of commitment in the interaction between cryptologic experts and laymen and within the cryptologic community itself.

We introduce the terms *K-secure* and *hermetic* to denote two intuitive security-related notions about cryptographic schemes. Both notions are formulated as the security *relative to* the majority of so called *super* schemes that model *all possible* instances of the given types of cryptographic schemes. Finally, we treat the limitations to manipulation detection realized by the encryption of redundant messages.

3.2 The cryptanalytic setting

One of the basic working assumptions of cryptography was first explicitly formulated over a century ago by the leading cryptographer Auguste Kerckhoffs (1835-1903) [58], and is therefore called:

Kerckhoffs' Principle: a designer should assume that the entire mechanism of encipherment (or hashing), except for the value of the secret key, will be known to the enemy cryptanalyst.

This can be reformulated to give: the part of the scheme that is unknown to the enemy cryptanalyst is called the *key*. If the key of one field application is compromised, it should not give away any information on other field keys. Taken to its ultimate consequence, this implies that every field key must be chosen from the key space according to a uniform probability distribution. Therefore it is a working assumption in most cryptologic literature that the cryptanalyst has no a priori information about the key.

In many real-world applications this is not the case. Non-uniform key selection occurs for instance when keys are derived from user-chosen passwords. Another typical situation is the "partial" use of the key in an encryption scheme. Consider an application for which it has been decided that a key length of 64 bits is sufficient, and the selected encryption scheme has a key length of 256 bits. This incompatibility is typically resolved by letting the 256-bit encryption key consist of the 64-bit key followed by zeros (or any other constant), or the 64-bit key four times repeated.

Non-uniform key selection as compared to uniform key selection diminishes the level of protection and is widely considered to be bad practice. The solution to problems related to it should consist in replacing the key selection mechanism itself. This point of view plainly ignores the reality of systems design. Typically, a working system has to be assembled using prespecified building blocks, among them cryptographic ones. Allowing only uniform key selection strongly limits the application domain of specific cryptographic schemes. Therefore it is *essential* to take the effects of non-uniform key selection into account.

3.2.1 Ways of access

We consider an encryption scheme consisting of an encryptor and a decryptor, both loaded with a certain key K_t and connected by a publicly accessible channel. According to the situation at hand, the access of the cryptanalyst can vary greatly. In analyzing the privacy protection given by encryption schemes, a number of different access modes for the cryptanalyst can be considered:

1. Seeing cryptograms
   (a) while having some statistical knowledge about the messages;
   (b) corresponding to (partially) known messages;
   (c) corresponding to (partially) chosen messages;
2. Seeing messages corresponding to chosen cryptograms.

1(a) is traditionally called *ciphertext-only*, 1(b) *known-plaintext*, 1(c) *chosen-plaintext* and 2 *chosen-ciphertext*. In many cases it is advantageous to base the choices on information obtained from previous steps. This is denoted by the term *adaptive*.

Complementary to all these access modes is the ability of the cryptanalyst to manipulate the initialization mapping:

1. Choose the public parameter Q;

2. choose the initialization mapping itself.

These two additional modes of access are useful if an encryption scheme has to be designed using a specific cipher. An example of a class of attacks that specifically exploit these modes of access are the *related-key* attacks [7].

For a keyed cryptographic hash function the modes of access available to a cryptanalyst are seeing the hash results

1. and having some statistical knowledge about the messages;

2. corresponding to known messages;

3. corresponding to chosen messages.

For a non-keyed cryptographic hash function, there is no difference between the access of legitimate users and enemy cryptanalysts since there is no secret information involved.

3.2.2 Exhaustive key search

In exhaustive key search the goal of the cryptanalyst is to determine the key K_t (target key) that is used in a cryptographic scheme. It consists of exhaustively eliminating all keys not equal to the target key by establishing that they are inconsistent with some available piece of information. For keyed hash functions this information consists of some messages and their corresponding hash results. For encryption schemes this information is a cryptogram obtained by encryption of a message using the target key, together with the message itself or some a priori knowledge about the message. This a priori knowledge can be the fact that the message is ASCII coded text, contains digitized images or sound or contains checksums that must match.

In most cases the a priori information can be characterized by a certain amount of redundancy. A message that has a certain amount of redundancy in a given context can be compressed. The *relative redundancy* r of a message is the difference between the length of the message and the length of the message that would be obtained using an "optimum" coding scheme, divided by the message length. For example, the ASCII file corresponding to Chapter 1 of this thesis contains 13,442 bytes. After compression with the `gnu software` compression facility `gzip` this is reduced to 5,201 bytes. Hence, the relative redundancy in this file is at least 61 %.

If the message is known, a key is eliminated if it does not map the message to the observed cryptogram or hash result. A wrong key will map a v-bit message to the correct cryptogram by chance with probability 2^{-v}. If the key length is n_k, the expected amount of wrong keys that are not eliminated is $2^{n_\mathrm{k}-v}$. This also holds for keyed hash functions, with v denoting the total number of hash result bits available to the cryptanalyst. Hence, it is expected that the target key will be uniquely determined if $v > n_{\mathrm{k}}$.

If the message itself is not known, but only some a priori information, a key can be eliminated if the key maps the cryptogram to a message that is "invalid" with respect to the a priori information. The probability that a wrong key maps the cryptogram to a valid message is equal to 2^{-rv} with v denoting the length of the cryptogram. Hence, the target key can be uniquely determined if $v > n_{\mathrm{k}}/r$. The right-hand side of this inequality is called the *unicity distance*, a term that was introduced by Claude Shannon [99]. Since it has the relative message redundancy in the denominator, the unicity distance can be made arbitrarily large by compressing the message. This may result in a unicity distance larger than the total length of all cryptograms that are encrypted using the target key. In this situation a cryptanalyst with only access to the cryptogram is in principle unable to determine the target key. There are however two problems with this approach:

- No practical compression scheme can squeeze out all redundancy. In fact, redundancy is not an absolute property of an individual message but depends on the cryptanalyst's contextual and a priori knowledge.

- More fundamentally: we want encryption schemes to be secure, independent of the structure of the encrypted data.

In [49] Martin Hellman showed that the expected work factor of exhaustive key search for block ciphers can be decreased at the cost of additional storage and pre-computations. In this attack the cryptanalyst requires the ciphertext corresponding to a *chosen* plaintext block. The work factor for finding a single secret key can be decreased to $2^{2n_{\mathrm{k}}/3}$ encryptions if a table of $2^{2n_{\mathrm{k}}/3}$ blocks is precomputed in a preprocessing phase. This preprocessing phase has a work factor of $2^{n_{\mathrm{k}}}$ encryptions and must be done only once for a given block cipher. The attack also applies to keyed hash functions and can be easily parallelized.

Finally, if the cryptanalyst has some a priori knowledge of the key, the expected effort of exhaustive key search can be minimized by scanning the key space starting with the keys with the highest probabilities.

3.3 Provable security

It would be very desirable for a cryptographic scheme to have some *provable* security properties, guaranteeing absolute protection or some lower bound on the amount of work it takes to successfully cryptanalyze the cryptographic scheme. In the following sections we discuss in an informal way the complications that arise in trying to design provably secure cryptographic schemes. The design effort of provably secure schemes is mainly inspired by two distinct theories: information theory and computational complexity theory.

3.3.1 Information theoretic approach

Information theory was founded by Claude Shannon in [98] and applied to cryptography in [99]. It deals with the *possibility* of attack, without taking the cryptanalytic effort into account. Encryption and cryptographic hashing can be applied for the protection of two aspects of communication: its *privacy* and its *authentication*.

Privacy

The privacy provided by an encryption scheme is unconditional if the cryptograms are of no use in trying to guess the messages. Encryption schemes with this property exist and are called *perfect*. Shannon [99] defined an encryption scheme to be *perfect* if the cryptogram is statistically independent of the message. He showed that perfect privacy can be achieved only when the secret key is at least as long as the plaintext. This was already achieved in the so-called *one-time pad* proposed by Gilbert Vernam two decades earlier [103]. The fact that the key is as long as the message restricts the application domain of perfect encryption schemes to some rare exotic instances such as spy communications and the Washington-Moscou hotline. In virtually all modern applications encryption is applied to reduce the problem of protecting the confidentiality of a large message by that of a short key. For this purpose perfect encryption schemes are useless.

In the remaining part of this section we briefly discuss the concepts of *local randomness* and *randomized ciphers*, both inspired by information theory. For a thorough information-theoretic treatment of these and some other concepts related to provably secure privacy we refer to the doctoral dissertation of Ueli Maurer [73].

The concept of *local randomness* in encrypting sequences of synchronous stream ciphers was introduced by Claus Schnorr [97]. It is motivated by considering provable security against an adversary with limited access. The adversary is assumed to be limited in the number of encrypting bits that he can obtain but is free to choose the positions of these bits. This unrealistic cryptanalytic setting renders this concept quasi useless in the context of provably secure stream encryption. An upper bound for the number of encrypting bits that the adversary is allowed to obtain in this model is given by the number of key bits. If more encrypting bits are given, the key can be identified by exhaustive key search.

In [73] Ueli Maurer presents a *strongly randomized* cipher that is shown to be "perfect with high probability". The basic idea of this encryption scheme is that the adversary obtains no information about the plaintext with probability very close to one unless he accesses a substantial fraction of a large publicly accessible string of bits, called the *randomizer*. It is proved that, if this fraction is below some limit, the enemy's entire observation, consisting of the cryptogram and the examined randomizer bits, is statistically independent of the message. Although this is not an unconditionally secure encryption scheme, we agree with the designer that, of all proposals, this is the closest thing to a provably secure cipher. Unfortunately, the need for the publicly accessible randomizer makes this encryption scheme highly impractical.

Authentication

Authentication is a more complex concept than privacy. The simplest model is that of a single sender, a single receiver and an insecure channel. In this model, authentication is concerned with the fact that every received message was sent by the legitimate sender and has not been modified during transmission. For a good treatment of authentication from an information-theoretic point of view we refer to the third chapter of the doctoral dissertation of Bart Preneel [84]. An extensive overview of authentication is given by Gus Simmons in [100].

The messages are converted into cryptograms, parameterized by some secret key that is shared between sender and receiver. These cryptograms may or may not conceal the message. In the simplest model, a deception attack performed by an adversary consists of the observation and subsequently the modification of that cryptogram (substitution attack) or the sending of a cryptogram without prior observation (impersonation attack). Clearly, unconditional security in the sense that the probability of detection of substitution or impersonation is 1, is impossible. For, it is always possible that an adversary sends a random cryptogram that happens to be valid.

Information theoretically, a lower bound on the probability for a successful deception attack is given by $2^{-n_\mathrm{k}/2}$ if the n_k-bit key is only used once [84, 100]. If the same key is used for more than a single message, this probability drops to $2^{-n_\mathrm{k}/(\ell+1)}$ with ℓ the number of observed cryptograms. This implies that for a given probability of success of a deception attack $\mathrm{P_d}$ the amount of key bits per authenticated message must be larger than $-\log_2 \mathrm{P_d}$. Hence, for a given security level the number of key bits is proportional to the number of messages that have to be authenticated.

One could argue that this key material could be communicated over the same channel, using a high-level cryptographic system that ensures the privacy and the authentication of the communication based on some fixed secret master key. However, the security of the resulting system consisting of the message authentication system and this key communication system is also limited by the information theoretic bounds. Hence, cryptographic systems that have some (information-theoretic) provable level of security with respect to authentication are impractical for the same reasons that encryption schemes providing perfect privacy are impractical.

3.3.2 The complexity theoretic approach

Computational complexity theory is a subdiscipline of computer science that studies the time and memory resources needed in the execution of algorithms. After the introduction of public key cryptography by Whitfield Diffie and Martin Hellman in [31], a growing part of the cryptologic community has concentrated on devising cryptographic schemes and protocols inspired by this theory. Its popularity in cryptography is due to different factors. Some researchers have (had) the idea that computational complexity theory enables one to design cryptographic schemes that are provably secure under reasonable assumptions. Other researchers believe that, despite the fact that practical provable security by applying computational complexity seems to be out of reach, it can serve as a guiding light towards sound design principles.

For a thorough general treatment of computational complexity theory we refer to [42]. In this section we give an informal discussion on the theory as applied in the field of cryptography. Complexity theory considers computational problems that are defined for arbitrary input lengths. The central subject is the asymptotic time and memory resources, denoted by the term complexity, needed by problem-solving algorithms for some reasonable model of computation. By emphasizing on the distinction between polynomial complexity and non-polynomial complexity, the theory is to a large extent independent of the particular computation model. Problems that can be solved with resources (time, memory) that grow only polynomially with the input size are considered easy. Problems for which no solving algorithm has been found with a complexity that grows polynomially with its input size are considered difficult.

This is expressed in the central concept in computational complexity theory, the distinction between two classes of problems: P and NP. The class NP (nondeterministic polynomial) consists basically of all decision problems (that can be answered by yes or no) that, if the answer is yes, it can be verified in polynomial time. A typical example of a problem in NP is the traveling salesman problem, i.e., "Given a set of cities, the distances between them and a bound B, does there exist a tour of all the cities with a total length B or less?". If someone claims that the answer is yes, we can ask him or her to give us the order of the cities that constitutes the tour and simply verify the answer by adding the corresponding distances. The class P (polynomial) is the subset of NP that can be solved in polynomial complexity. It is still an open question whether P equals NP, and many computer scientists consider this to be the single most important open question in computational complexity theory.

The principal technique in finding interrelations and bringing structure in this theory is that of *reducing* one problem to another. This involves a constructive transformation of any instance of the first problem to an equivalent instance of the second. Such a transformation provides the means for converting any algorithm that solves the second problem into a corresponding algorithm for solving the first problem. A transformation is called polynomial if the input length of the second problem is polynomial in the input length of the first problem. The reduction of one problem to another must be realized by a polynomial transformation.

Any problem that can be reduced to a problem in P, belongs to P itself. In this way, another subset of NP is defined as the set of problems to which any problem in NP can be reduced. This set has been shown to be non-empty and its elements are called NP-complete. These can be considered the "hardest" problems in NP. If a single NP-complete problem could be reduced to a problem in P, this would imply P = NP. The fact that nobody has succeeded in doing this, is taken as strong circumstantial evidence that P differs from NP. Related to the NP-complete problems are the NP-hard problems. This is a larger class of not necessarily decision problems, that can be proved to be at least as hard as an NP-complete problem.

From the computational complexity theoretic point of view, the best one can do is to base a cryptographic scheme on some NP-hard problem. Intuitively, this means that some NP-hard problem can be reduced to a problem of which the solution consists of successfully cryptanalyzing the cryptographic scheme. There are some constructions and corresponding definitions of security for which this has been achieved. For example, Russell Impagliazzo and Moni Naor used the knapsack to construct a "Pseudo-random String Generator" and a "Universal One-Way Hash Function" [55]. They proved the equivalence of the security (with their definition) of these primitives to the intractability of the knapsack problem. Still, the resulting "provable security" of schemes this type of must be seen in the framework of computational complexity theory:

- Computational complexity theory only makes a distinction between polynomial and non-polynomial asymptotic complexity, disregarding constant factors and the degrees of the polynomials involved. In every practical application of a cryptographic scheme the dimensions, linked to the input length of the corresponding "hard" problem, must be fixed. Hence, computational complexity does not give any information whatsoever on the security of a specific cryptographic scheme with fixed dimensions.

- The complexity of a class of problems is the minimum complexity of all known algorithms that solve **all** problems in that class. This is a *worst-case* measure. This is illustrated by the fact that every class of NP-complete problems has an indefinite number of subclasses that are in P [42]. Hence, even in the asymptotic framework computational complexity theory fails to provide a lower bound on the complexity for successful cryptanalysis of a cryptographic scheme.

The main results of applying computational complexity theory in cryptography are the constructions of cryptographic schemes and protocols in terms of simpler building blocks. Families of cryptographic schemes make use of a "general" family of functions that has some well-defined asymptotic intractability property. The resulting families of schemes have an intractability property that provably holds if the intractability property of the underlying function family holds. Examples of these underlying function families are families of "one-way functions" [94] or "claw-free functions" [28]. The supposed relevance of these constructions is twofold. First of all, these constructions give a conditional proof of existence for cryptographic schemes with specific intractability properties based on some simple (albeit unprovable) assumption. Second, the constructions can be used to build real cryptographic schemes by specifying the underlying function family. It is argued that these constructions can be useful in the design of cryptographic schemes that have some sort of provable security. Here the term provable refers to the reduction that makes successful cryptanalysis of the cryptographic scheme equivalent to solving the intractable problem associated with the underlying function.

Computational complexity theory has contributed to a narrow-minded reductionistic design approach that we denote by *N-reductionism.* In this approach the emphasis is on the provable reduction of the security of a cryptographic scheme to that of its components. The underlying assumption is that the design of components with the desired properties is easier than the more general design problem of the cryptographic scheme itself. We believe the effect of this approach to range between useless to harmful for several reasons:

- In practice there is no way to verify the underlying assumption. In some cases it can even be shown that the component functions are in fact harder to build. The N-reductionist approach is effectively a compartmentalization of the design problem. The design of the component functions can be seen as local optimization as opposed to global optimization when a less restricted design approach is taken.

- By emphasizing the reduction, the real design issues are often neglected. The reduction is considered to be the "fundamental" and "scientific" part of the construction, while the design of the underlying component functions is by its nature "ad hoc" and therefore less interesting.

- The term "provably secure" that is sometimes used in this context gives a false impression of security if not all assumptions are explicitly stated.

For schemes where the reductions are performed in a computational complexity theoretic framework, some additional complications occur:

- Reductions that are valid in a computational-theoretic framework can be completely useless in constructions for cryptographic schemes where some fixed level of security is expected. For example, a simple block cipher is not considered secure if an attack exists that is very much faster than exhaustive key search. In this way it is possible to build a block cipher family that is provably secure in a computational complexity theoretic framework, but is still hopelessly insecure and inefficient compared to even the weakest of practical proposals with the same dimensions.

- Because of the asymptotic nature of computational complexity theory, computer scientists are inclined to postpone the decision of fixing the dimensions even after specification of the underlying function. Instead of a single function a complete function *family* must be specified, one for every dimension value. This restricts the underlying function to scalable functions that are typically very inefficient.

Our objections to the N-reductionist approach are illustrated by three specific examples. In Chapter 4 we discuss the design principle for hash functions proposed by Ivan Damgård in [29] and by Ralph Merkle in [77]. In Chapter 9 we discuss two design principles for self-synchronizing stream ciphers proposed by Ueli Maurer [74] and in Chapter 10 a block cipher construction proposed by Shimon Even and Yishay Mansour [34].

3.4 Security in practice

From the previous discussions it can be concluded that we have no hope of building practical ciphers, encryption schemes or cryptographic hash functions that are provably secure under some realistic assumptions. Still, the need for ciphers and cryptographic hash functions faces us with the design problem.

The absence of a proof of security for practical cryptographic schemes causes cryptographic security to be linked to the concept of trust. Even if an objective definition of cryptographic security is possible, to a proposition such as "cipher X is cryptographically secure" cannot be attributed a truth value since the validity of the statement cannot be checked (in principle). However, the proposition "Person B trusts cipher X to be cryptographically secure" can indeed be true or false, depending on the willingness of person B to use cipher X in his security implementation. If a person C makes the statement "Cipher X is cryptographically secure" addressed to person D, this must not be seen as a transfer of information from C to D, but as a commitment of person C to cipher X addressed to D. If D can be convinced that cipher X does not comply with the agreed upon definition of cryptographic security C, this must have consequences for person C. It is the importance of these consequences and the reputation of person C in the eyes of person D that make the strength of the commitment.

The widespread use of a cryptographic scheme implies the widespread trust in the cryptographic security of the scheme. Every user must be convinced that the scheme provides the cryptographic security that is needed for his or her application. Since not everyone considers himself to be an expert in cryptography, most users are unable to convince themselves of the security by inspecting and analyzing the cryptographic scheme itself. The trust of the users originates in the commitment of individuals or institutions that are trusted by the user. This can be applied in a transitive manner, e.g., a citizen trusts his government that trusts its cryptographers. Observe however that every chain of trust must originate in a commitment. In this context, the "cryptographic strength" that is attributed to a cryptographic scheme is the strength of the original commitment.

When a cryptographic function is first published, the trust in its security is typically limited to a very small group of people. If the function is an attractive goal for cryptanalysis, it can gain the trust of more people by resisting cryptanalysis, despite public scrutiny by colleague cryptologists. When no weaknesses are found that affect the cryptographic security of the function, the amount of trust in it grows with the joint cryptanalytic effort against it. This gradual build-up of trust is abruptly broken off if a weakness is demonstrated.

In this context it is easy to see the appeal of cryptographic schemes of which the successful cryptanalysis is related to solving some long-standing (typically number theoretical) open problem. These functions can gain a high trust level almost immediately, based on the widespread belief in the hardness of the long-standing open problem.

The differences in the implicit understandings of what constitutes security may give rise to arguments whether certain demonstrated weaknesses are harmful. This can be minimized if the scheme is accompanied by a *claim* of security and a clear definition of this security. Ideally, a definition of security would involve the real-world concerns related to economical cost. This would however exclude a rigorous formal mathematical definition and make the security depend on the economical and technological circumstances. The complexity and indeterminism in these areas would leave room for different interpretations that can give rise to arguments. It seems plausible that a rigorous definition based on some formal model could eliminate all margins of interpretation. However, in that case the appropriateness of the formal model itself can be questioned. Therefore, our definitions of security are stated in an informal manner.

A security claim is not something that one should expect to be proved by the designer. The claim has two different functions. For potential cryptanalysts, it serves as a challenge. Effective cryptanalysis consists of demonstrating weaknesses that refute the claim. If the function has withstood public scrutiny through time by the absence of effective cryptanalysis, the claim serves as the specification of the security of the cipher for system engineers and users who trust it.

3.4.1 The cryptologic activity

The "cryptologic activity" is the joint effort of the design of cryptographic primitives and the analysis of their possibilities and limitations. The group of people that are involved in this activity is generally referred to as the cryptologic community. Important results are communicated in presentations at scientific and technical conferences and in papers in the proceedings of these conferences or in journals.

Central in the cryptologic activity is the iterative interaction between cryptography and cryptanalysis. Cryptanalysis shapes cryptography by the fact that new designs and design strategies must be resistant against known attacks or types of cryptanalysis. New cryptanalytic results in their turn arise from attacks on previously proposed designs.

Cryptologic results range from specific particular designs and attacks to general applicable design strategies and cryptanalysis. Many members of the cryptologic community consider it to be the goal of cryptologic activity to expand our theoretical cryptologic knowledge by doing fundamental scientific research. In this approach the emphasis is on general and fundamental results that hold irrespective of the details of the cryptographic functions to be designed. The design of particular cryptographic functions is considered to be a mere application of this theory. In our opinion this point of view does not correspond to what really goes on and gives rise to a false impression of theoretical knowledge.

On the one hand, some theories have developed in a direction of their own, detached from the limitations and needs of the real world. In some cases they only describe aspects of models that are irrelevant in practical applications, in other cases models have been introduced that can have no application at all. On the other hand, for many so-called fundamental results it can be observed that they are in fact very specific for a certain design approach inspired by existing proposals, that are by nature *historical* instead of fundamental. The actual influence of existing, specific designs is so pervasive that it is even implicit in the choice of the research problems to be addressed.

Even most of the fundamental (in that they are widely applicable) cryptologic principles have emerged in the context of particular designs. For example, both differential and linear cryptanalysis, described in Chapter 5, were developed in the cryptanalysis of particular block ciphers.

In the domain of single-key cryptography and cryptographic hash functions, the emphasis on theory has given rise to the fact that, for the different types of ciphers and hash functions, there is only little variation between the different particular designs. Since most of the widely applicable results originate from particular designs, we consider this to be an undesirable situation.

In our opinion the useful results of cryptologic activity consist of practical cryptographic designs and a better understanding of their possibilities and limitations. In the context of cryptology, supporting theories are nothing more than means to reach that goal. The central activity of cryptology is the iterative interaction of particular designs and cryptanalysis. It can be argued that the widespread adoption of this point of view may lead to a chaotic situation with an ever growing number of competing proposals and attacks. This argument ignores the very important criteria of cost, performance and portability that will mark some designs as more valuable than others.

For a new design to be noticed, it should at least be superior in some way to existing designs with comparable security. Of course, new proposals are not designed from scratch. Typically, many new proposals will be improved (with respect to cost, performance and portability) or adapted (to patch up security holes) versions of existing designs. With the introduction of improved and adapted versions the previous proposals will disappear from the field of interest. In some cases the new versions may in fact turn out to be inferior, and the old proposals may surface again. As time passes, "better" proposals are expected to emerge. The proposals that combine low cost, high performance, high portability and immunity to cryptanalysis will demand most of the attention of the cryptologists. In this way inferior designs would simply disappear into oblivion.

The reputation of a cryptographic design may be affected as much by rumors and speculation as by effective attacks. A good example is the speculation about the block cipher DES [37]. The details of the design process of DES have never been published, and this has given rise to speculation that DES would have a so-called *trapdoor*. A trapdoor is a feature built into a cryptographic primitive that gives the designer the possibility to circumvent in some way the security offered by the primitive without being noticed by the public. Of course, the risk of rumors and speculations cannot be eliminated. However, their credibility can be minimized by completely specifying the design process and fully explaining and motivating every design decision. In this respect it is a good idea to make the design as simple as possible.

Finally, it must be mentioned that in practice personal, commercial, political and strategic interests have an important impact on the course of cryptologic activity. We realize that the situation described in the previous paragraphs is a remote ideal and that even a (miraculous) change of mentality in the cryptographic community will not be sufficient to reach it.

3.5 Definitions of security

In general, the cryptographic security that is required from a cipher depends on the application. Therefore, *cryptographic security of a cipher* can best be defined as security in the worst possible circumstances. Clearly, a cipher that is claimed to be cryptographically secure by this definition, is claimed to be secure in all applications. In this section we introduce two new security-defining terms that capture the high security requirements that we expect from our designs.

We will define a number of types of cryptographic "super" schemes. The set of super schemes of a particular type with given dimensions contain all possible schemes of that type. The essential property of a uniformly chosen super scheme is the absence of redundancy in the mappings it realizes. The properties of the sets of super schemes serve as a reference in our definitions of security.

3.5.1 Super ciphers

We distinguish four different classes of super ciphers: synchronous stream ciphers, filtered counter ciphers, self-synchronizing stream ciphers and block ciphers. For every super cipher it is assumed that the bits, used to specify it, originate from a binary symmetric source (BSS). A BSS is a generator of bits, where the probabilities that particular bits are 1 or 0 are equal and mutually independent.

- **Synchronous stream cipher**
  - dimensions: n_κ, n_s, n_a.
  - for every value of κ an *invertible* state updating transformation is specified by a table with 2^{n_a} entries containing n_a-bit states. This cipher can be described by $n_\mathrm{a}2^{n_\mathrm{a}+n_\kappa}$ bits. The encrypting symbol z^t consists of the state bits with indices 0 to $n_\mathrm{s}-1$.
- **Filtered counter cipher**
  - dimensions and parameter: n_κ, n_s, n_a, specific counter.
  - for every value of κ an output function is specified by a table with 2^{n_a} entries containing n_s bit output symbols. The number of bits needed to specify this output function is $n_\mathrm{s}2^{n_\mathrm{a}+n_\kappa}$.
- **Self-synchronizing stream cipher**
  - dimensions: n_κ, n_s, n_m.
  - for every value of κ a cipher function is specified by a table of $2^{n_\mathrm{s}n_\mathrm{m}}$ entries containing n_s-bit output symbols. The number of bits needed to specify the cipher function is $n_\mathrm{s}2^{n_\mathrm{s}n_\mathrm{m}+n_\kappa}$.

- **Block cipher**
  - dimensions: n_κ, n_{b}.
  - for every value of κ a permutation is specified by $2^{n_{\mathrm{b}}}$ entries containing n_{b}-bit blocks. This cipher can be described by $n_{\mathrm{b}}2^{n_{\mathrm{b}}+n_\kappa}$ bits.

In the case of the updating functions of the super synchronous stream cipher and the block cipher mappings, the invertibility condition prevents the straightforward generation of the tables by simply filling in bits originating from the BSS. Another procedure must be followed to use the BSS bits to select the transformations of $\mathbb{Z}_2^{n_{\mathrm{a}}}$ (or $\mathbb{Z}_2^{n_{\mathrm{b}}}$) from the set of $2^{n_{\mathrm{a}}}!$ (or $2^{n_{\mathrm{b}}}!$) possible invertible transformations in a non-biased way.

It is not impossible that a super cipher is generated with an exploitable weakness. In fact, given the type of cipher and its dimensions, all possible instances have the same probability of being generated. For instance, in the case of block ciphers this includes the block cipher consisting of the identity transformation for every key. For practical values of the dimensions however, these sets of ciphers with exploitable weaknesses form a negligible minority.

3.5.2 Super encryption schemes

The injudicious choice of an initialization mapping can give rise to an insecure encryption scheme, even when a super cipher is used as its main building block. Such an initialization mapping is called *unsound*. A necessary condition for an initialization mapping to be sound is that for a given value of K (or Q), its output determines the value of Q (K) completely. Hence, it is invertible both with respect to the key K and to the public parameter Q. In a filtered counter cipher the initialization mapping must also prevent the exploitation of repeated occurrences.

An encryption scheme is a super encryption scheme if its initialization mapping is sound and the cipher that it contains is a super cipher. A super encryption scheme is described by a type-dependent set of dimensions.

3.5.3 Super cryptographic hash functions

A super cryptographic hash function has only a single dimension: the length of the hash result n_{h}. Since its specification would require an infinite number of bits because there is an infinite number of possible messages, a more “operational” specification will be given.

We will first describe a non-keyed super cryptographic hash function. The hash function can be evaluated by requesting the hash result for a given message at a central site. This site handles the requests in a sequential manner, one message at a time, and keeps a data base containing all previous messages. When a request is handled, it is checked whether the given message is in the data base. If not, a new hash result is generated by a BSS. The data base is updated and the hash result is sent back to the applicant. If the message is in the data base, the corresponding hash result is sent to the applicant. In this way, the same message always gives rise to the same hash result.

A keyed variant has an additional dimension, the key length n_{k}. It can be specified by a variant of the above procedure where every occurrence of “message” is replaced by “(message,key) couple”. This can also be considered as a central site handling $2^{n_{\mathrm{k}}}$ different hash functions, one for each key.

3.5.4 K-secure primitives

Definition 3.1 *An encryption scheme is K-secure if all possible attack strategies for this scheme have the same expected work factor and storage requirements for the majority of corresponding super encryption schemes. This must be the case for all possible modes of access for the adversary and for any a priori key distribution.*

Definition 3.2 *A cipher is K-secure with respect to an initialization mapping if all possible attack strategies for the resulting encryption scheme have the same expected work factor and storage requirements for the majority of corresponding super encryption schemes. This must be the case for all possible modes of access for the adversary and for any a priori key distribution.*

Definition 3.3 *A keyed cryptographic hash function is K-secure if all possible attack strategies for this function have the same expected work factor and storage requirements for the majority of corresponding super keyed hash functions. This must be the case for all possible modes of access for the adversary and for any a priori key distribution.*

One can consider the security of a cipher when used with a variety of initialization mappings and in a variety of modes of use. For example, the simplest way to turn a block cipher into an encryption scheme is the ECB mode, with an initialization mapping that simply maps K to κ without the intervention of a public parameter.

At best, a cipher can be so strong that the specific choice of the (sound) initialization mapping has no influence on the security of the resulting scheme. At worst, no secure scheme can be built around the cipher without choosing an initialization mapping that has certain cryptographic properties on its own. For a given cipher, classes in the space of all possible initialization mappings can be identified with respect to which the cipher is K-secure. The flexibility of the cipher grows with the size of these classes.

K-security is a very strong notion of security. It can easily be seen that all the following weaknesses cannot occur in K-secure encryption schemes or keyed hash functions:

- existence of key-recovering attacks faster than exhaustive search,
- certain symmetry properties in the mapping,
- occurrence of large classes of weak keys,

- related-key and resynchronization attacks,
- (for keyed hashing) *existential forgery attacks* more efficient than exhaustive key search.

Existential forgery [84, p. 41] consists of the determination of the hash result for at least one plaintext (that may be nonsensical).

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

The term *cascade cipher* was defined in [99] as a cipher that consists of the composition of a number of ciphers with *independent cipher keys*. Hence, a cascade cipher converts a plaintext symbol into a ciphertext symbol by first applying cipher mapping 1, followed by cipher mapping 2 and so on. In [75] Ueli Maurer and Jim Massey prove that a cascade cipher with commutative component ciphers (e.g., most synchronous stream ciphers), is as cryptographically secure as its most secure component cipher. The results are presented in a rather informal manner, since it is argued that they hold for every reasonable formalism. Remarkably, they do **not** hold with respect to K-security.

The ciphers treated in [75, 99] are in fact what we call encryption schemes. A cascade encryption scheme is composed of a number of encryption schemes. The global key K is the concatenation of the keys of the component encryption schemes. The global parameter Q is the concatenation of the parameters of the component encryption schemes. Since K-security is a type of security that must hold for any a priori distribution of K, the cascade encryption scheme can only be K-secure if *all* of its component encryption schemes are K-secure. This can be seen as follows. Suppose we have a cascade encryption scheme with a component encryption scheme Ψ that fails to be K-secure. If K is chosen according to a distribution that fixes all component keys except K_Ψ, the access to encryption scheme Ψ is identical to the case that no other encryption schemes are present. This argument only holds if the keys are independent, but so does the proof in [75]. K-security is a concept that discourages the application of N-reductionist design approaches that often trade efficiency for a false feeling of cryptographic security.

In the context of cryptographic hash functions, K-secure can only be used with respect to keyed ones. For non-keyed hash functions the definition of K-secure makes no sense.

3.5.5 Hermetic schemes

It is possible to imagine ciphers that have certain weaknesses and still are K-secure. An example of such a weakness would be a block cipher with $n_\text{b} \geq n_\kappa$ and a single weak key, for which the cipher mapping is linear. The detection of this would take at least a few encryptions, while checking whether this key is used would only take a single encryption. If this cipher would be used in an encryption scheme, this single weak key would pose no problem. However, used as a component in a larger scheme, for instance a hash function, this property could introduce a crucial weakness.

For these reasons we introduce another definition of security, denoted by the term *hermetic*.

Definition 3.4 *A cryptographic scheme is hermetic with respect to some application if it does not have weaknesses that are exploitable in that application* and *are not present for the majority of corresponding super cryptographic schemes.*

Definition 3.5 *A cryptographic scheme is hermetic if it is hermetic with respect to all conceivable applications.*

Informally, a cipher or hash function is hermetic if its internal structure cannot be exploited in any application.

For example, for an hermetic cryptographic hash function, the work factor of finding a collision is approximately $2^{n_\mathrm{k}/2}$ hash function applications with negligible storage requirements. For this is the complexity of the best known method for finding collisions in the majority of super cryptographic hash functions.

3.6 Manipulation detection in decryption

In many applications it is tacitly assumed that encryption guarantees that cryptograms decrypting to meaningful messages originate from the legitimate sender (in possession of the secret key) and that modifications to the cryptogram by an adversary (not in possession of the secret key) are detected. In this section we give some upper bounds to the protection that can be offered by different types of encryption.

For detection to be possible, the messages must have a well-defined redundancy. The best attack strategy and the probability of success depend strongly on the type of encryption that is applied and the nature of the redundancy. We only consider two types of redundancy here: an appended cryptographic hash result and distributed local redundancy. An example of the latter is the knowledge that a message is an ASCII-coded text.

An obvious type of attack is the replay attack, in which the adversary simply sends a previously recorded cryptogram to the receiver. This kind of attack can be thwarted by ensuring that for every message the encryption makes use of a different public parameter Q. This public parameter can be sent along with the cryptogram or can be a sequential message number. A disadvantage of this scheme is that the receiver has to keep a record of the Q parameters and must check for every incoming message whether its Q parameter has already been used in the past. In some applications, Q can be derived from external parameters known to both sender and receiver such as the time or the physical (or logical) address of the encrypted file on a storage medium.

For distributed redundancy we will consider only redundancy at the symbol level. It is assumed that only a subset of all possible plaintext symbols is used. In the case of stream encryption with $n_\mathrm{s} = 1$, the bits may be grouped in larger symbols with respect to this effect. If the relative redundancy is r, only $2^{(1-r)n_s}$ of all 2^{n_s} symbols are allowed in the message. We consider two types of distributed redundancy. In *type A* the allowed symbols form a vector space over $< \mathbb{Z}_2^{n_s}, + >$ while in *type B* they form a subset with no exploitable structure.

We consider two types of access for the adversary: ciphertext-only (CO) and known-plaintext (KP). Although in some circumstances chosen-plaintext attacks may be plausible, we do not consider them here because of the complexity of the setting. In a situation where the adversary is completely free to choose the messages, he has the same access as the legitimate sender and there is *no protection at all*. Hence, in any interesting chosen-plaintext situation, additional restrictions (of time and/or total amount of input) have to be imposed upon the access. It must be kept in mind that in these restricted chosen-plaintext circumstances the amount of protection offered can degrade drastically compared to the ciphertext-only and known-plaintext situations.

In the case that the redundancy originates from a cryptographic hash function, we have also included the case *non secret communication* (NS). By this we denote messages sent in plaintext with an appended hash result that is encrypted with the specified type of encryption.

A first type of attacks are *transposition, omission or insertion* of blocks or symbols in the message. If the redundancy results from an appended hash result, these simple attacks are very unlikely to succeed. In the case of distributed redundancy however, they are a realistic threat. In ECB block encryption, these attacks can be executed without a risk of detection. For CBC block encryption and the n_b-bit CFB mode of a block cipher, these attacks require a known-plaintext situation and the probability of detection is only small if the relative redundancy is very low.

Tables 3.1 and 3.2 give an overview of the success probability of some simple message modification attacks in which some cryptogram symbols are substituted by others to change the message. For self-synchronizing stream encryption in Table 3.1 and CBC block encryption in Table 3.2 we have made a distinction between three cases: attacks that involve the manipulation of the IV by the adversary, substitutions in the bulk of the message and substitutions in the last symbols or last block of the message. It is assumed that the block length is at least as large as the hash result, i.e., that $n_b \leq n_h$.

In these attacks it is assumed that the adversary does not

- perform exhaustive key search,
- exploit the internal structure of the used ciphers and cryptographic hash functions, i.e., the attacks also work for super encryption schemes and hash functions,
- make use of the knowledge gathered from previous cryptograms or message-cryptogram pairs.

Because of these restrictions for the adversary, the given success probabilities must be considered as upper bounds to the amount of protection that can be achieved against substitution attacks .

		stream encryption			
		sync.	self-synchronizing		
			IV manip.	bulk	last symbol
dist. red. A		0	ℓn_{s}	$n_{\mathrm{m}} n_{\mathrm{s}} r$	$(\ell - 1) n_{\mathrm{s}} r$
dist. red. B	KP	0	ℓn_{s}	$n_{\mathrm{m}} n_{\mathrm{s}} r$	$(\ell - 1) n_{\mathrm{s}} r$
	CO	$n_{\mathrm{s}} r$	$(n_{\mathrm{m}} + 1) n_{\mathrm{s}} r$	$(n_{\mathrm{m}} + 1) n_{\mathrm{s}} r$	$\ell n_{\mathrm{s}} r$
hash	NS	0	n_{h}	n_{h}	$n_{\mathrm{h}} - n_{\mathrm{s}}$
	KP	0	n_{h}	n_{h}	n_{h}
	CO	n_{h}	n_{h}	n_{h}	n_{h}

Table 3.1: $-\log_2$ of the probability that modifications in the cryptogram are not detected after decryption for stream encryption.

		block encryption			
		ECB	CBC		
			IV manip.	bulk	last block
dist. red. A		$n_{\mathrm{b}} r$	0	$n_{\mathrm{b}} r$	$n_{\mathrm{b}} r$
dist. red. B	KP	$n_{\mathrm{b}} r$	0	$n_{\mathrm{b}} r$	$n_{\mathrm{b}} r$
	CO	$n_{\mathrm{b}} r$	$n_{\mathrm{s}} r$	$(n_{\mathrm{b}} + n_{\mathrm{s}}) r$	$n_{\mathrm{b}} r$
hash	NS	n_{h}	0	n_{h}	n_{h}
	KP	n_{h}	n_{h}	n_{h}	n_{h}
	CO	n_{h}	n_{h}	n_{h}	n_{h}

Table 3.2: $-\log_2$ of the probability that substitutions in the cryptogram are not detected after decryption for block encryption.

From Tables 3.1 and 3.2 it can be seen that synchronous stream encryption can only offer protection in the most restricted access modes. For the other types of encryption, the probability of success decreases as the relative redundancy increases. For block encryption and self-synchronizing stream encryption, fair protection levels can be attained against the simple substitution attacks. However, as the relative redundancy increases, another type of attack has to be taken into account that limits the usable lifetime of the cipher keys. In ECB block encryption, the number of possible plaintext blocks, and hence also the number of possible ciphertext blocks is $2^{(1-r)n_{\mathrm{b}}}$. In a ciphertext-only situation, the adversary can build a table of valid cryptogram blocks that forms a non-negligible fraction of the total set of allowed cryptogram blocks. In a known-plaintext situation, a substantial part of this restricted encryption table may be reconstructed. This knowledge can be used by the adversary to form valid cryptograms. This is in general not true for CBC block encryption and self-synchronizing stream encryption. However, if manipulation of IV is possible or if IV is chosen from a small subset of all possible values, similar attacks are possible for these modes. It can be seen that in all cases manipulation of IV should be prevented. This can be achieved by not allowing the sender to choose IV himself.

As a general conclusion we can state that if manipulation detection is required, one can best append a cryptographic hash result and encrypt with block or self-synchronizing encryption. The corresponding expansion can be compensated by compressing the message prior to hashing and encryption.

3.7 Conclusions

In this chapter we have given a motivation for our belief that designing practical cryptographic schemes that have a high degree of provable security is infeasible. It is argued that the notion of security is closely linked to that of trust. We have given a description of the cryptologic activity as it is going on, and as we think that it should be.

Our own contributions in this chapter are:

- expansion of the standard ways of access with key manipulations by means of the initialization mechanism,
- the removal of the requirement of uniformity of key selection and its implications,
- the explicit formulation of the role of the cryptographic claim,
- the notions of K-secure and hermetic schemes,
- the systematic treatment of the limitations of manipulation detection by encryption.

Chapter 4

Design Strategies

4.1 Introduction

Our goal is contributing to the design of ciphers, encryption schemes and cryptographic hash functions that are at the same time cheap, portable and secure against known attacks. First we will show how simplicity and structural transparency can contribute to realizing these properties. This is followed by a discussion on the restrictions imposed by the need for portability and suitability for dedicated hardware implementation.

The central part of this chapter is devoted to the formulation of our design approach and its relation to the existing design practice. We discuss the dedicated design of block ciphers, self-synchronizing stream ciphers, synchronous stream ciphers and cryptographic hash functions. In this chapter we only describe the global design strategies. They are further elaborated in Chapter 7 to 9.

Finally, we discuss the portability and hardware suitability of a number of operations that are the most important candidate building blocks for cryptographic schemes.

4.2 General requirements

4.2.1 Simplicity

It is an advantage for a cryptographic scheme to have a simple description for a number of different reasons.

The *risk of* implementation *errors* increases with the complexity of the description. It is therefore advantageous to have a succinct and clear specification that makes an appeal to a pre-understanding of the reader that is as small as possible. The design is preferably specified at the **bit-level**. Additionally, a simple design can be specified in a short software program.

In most practical applications the cryptographic scheme is only a small *building block* in a compound system. The system complexity increases with the complexity of all of its building blocks. Especially important in this respect is the simplicity of the external behavior of the scheme, i.e., the so-called *interface*.

In general, the work factor of the *verification of resistance* against known types of attack grows with the complexity of the design. If the design is too complex, any verification of this sort may become infeasible.

It is infeasible to prove that a given design has no *trapdoor*. The designer has to convince the cryptographic community and the public that there is no trapdoor by motivating all design decisions. Obviously, this is easier for a simple design than for a complex design.

4.2.2 Structural transparency

In Chapter 3 we explained why we believe it to be infeasible to prove that a certain cryptographic scheme is secure against all possible attacks. There are however a number of generally applicable attacks and cryptanalytic principles for each of the different types of encryption schemes and hash functions. It is clearly an advantage if a design can be checked to be secure against these known attacks.

Some keyed schemes have been designed in such a way that the structure of the transformations used depends strongly on the specific value of key bits. General analysis of the design with respect to the known attacks is assumed to be simply impossible since the propagation properties depend strongly on the key. This is indeed the case in the traditional point of view, where an adversary has no a priori information about the key. As we stated in Chapter 3, in many practical situations the adversary actually *does* have a priori information about the key. Hence, it is meaningful to consider cryptanalysis scenarios in which a number of key bits are given specific values and the goal is the determination of the remaining ones. If an attack can be constructed that is more efficient than exhaustive search over the reduced key space, the cryptographic scheme is not K-secure. The fixed key bits can be chosen in such a way that the corresponding transformations have unfavorable propagation properties. This may lead to a vast amount of different attacks for a single scheme.

Typical weaknesses of ciphers composed of strongly key-dependent operations are so called *classes of weak keys*. A class of weak keys is a subset of the key space for which the cipher has an easily detectable weakness. In Chapter 11 we show that there are large classes of weak keys for the block cipher IDEA.

In our opinion, one of the main goals of a sound pragmatic design strategy is to enable verification of resistance against known types of cryptanalysis. To facilitate this we impose the restriction that the propagation properties of the component operations seen at the bit level should be independent of the actual value of the key bits. In Chapter 5 it can be seen that this restricts key application to bitwise addition.

4.3 Specific requirements

4.3.1 Portability

A design is said to be *adapted* to a given processor if it efficiently exploits the instruction set of this processor to realize its desired properties. For a given processor, a well adapted design can be translated in a straightforward way to a sequence of machine instructions. For a cryptographic design that is aimed at widespread application, it is advantageous that it is well adapted to a wide variety of processors. In this respect it is best to use only operations that are **widely available** as **machine instructions**. Clearly, processors such as those in the Intel 80X86 series have more weight in this context than rare processors.

The number of machine instructions that the execution of the cryptographic scheme takes is roughly inversely proportional to the processor word length for processors that it is well adapted to.

4.3.2 Dedicated hardware suitability

In some applications it is appropriate to have dedicated hardware implementations. When encryption and hashing rates larger than 100 Mbit/s are needed (e.g., optical fiber), software implementations may not be fast enough. Some applications (e.g., GSM) require compact (single-chip) integrated systems, consisting of a number of dedicated functional modules, including a cryptographic scheme. In such a modular system, the throughput required from the cryptographic module has to be realized with minimum area. This is also the case for a cryptographic scheme that has to fit on a custom *smart card*.

For these applications it is necessary that a cryptographic design lends itself to dedicated implementations. Ideally, there is a trade-off between area and processing speed for a given design. At one end of the spectrum there is a serial implementation consisting of a small dedicated microprocessor and some memory. At the other end, there is an implementation that is optimized for speed by applying parallelism.

The degree of parallelism that one can achieve depends on the algorithmic structure of the cryptographic scheme. The suitability for parallel implementations can be enforced by first designing a highly parallel finite state machine and later specifying the functions of the cryptographic scheme in terms of it. This leads to cryptographic schemes with dedicated implementations that are both fast and compact.

Such a finite state machine, with a uniform and small gate delay and especially designed for cryptographic implementations, is called a *cryptographic finite state machine*. Figure 4.1 gives a model of a cryptographic finite state machine that can be used in the design of block ciphers, synchronous stream ciphers and cryptographic hash functions. In this model, the cryptographic finite state machine consists of 4 basic components: the internal state a, the buffer b, the round transformation logic R and the control- and load logic. By means of the control pins, one can choose between a number of different operations. The most important function is the round transformation that updates the state a parameterized by (part of) the contents of

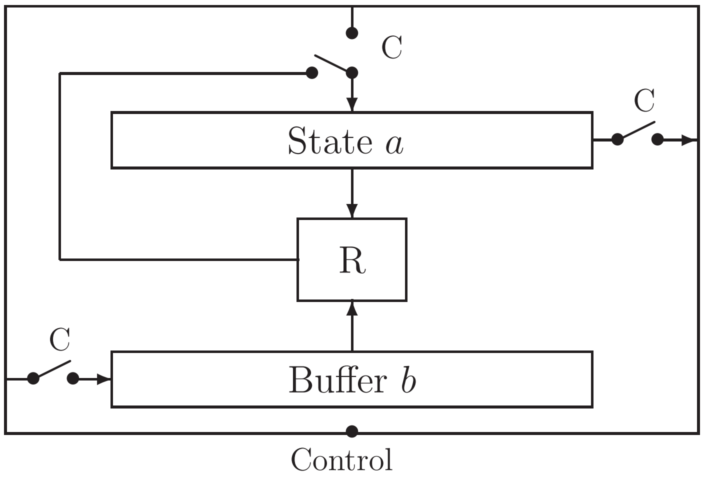


Figure 4.1: simple model of a cryptographic finite state machine.

the buffer. Possible operations include *state update, state or buffer load, state or buffer hold, state or buffer reset and state read*. We have first introduced the concept of the cryptographic finite state machine in [9].

Important implementation aspects of a cryptographic finite state machine are the needed number of logical gates, the complexity of the connection pattern and the number of pins for on/off chip communication.

Since only part of the chips coming from the foundry are without defects, a very important aspect of dedicated hardware design is *testability*. Classically, the testability of an integrated circuit is realized by the inclusion of *scan pads*. This consists of some extra circuitry on the chip, connecting all the memory cells on the chip in a single shift register. In *test mode* the values of all internal memory cells can be read out. With the resulting access to the internal memory cells it is possible to check the correctness of the operation of the chip locally. For a cryptographic module, the existence of such a test mode can be a threat to the overall security of the system. Therefore, it is an advantage for a cryptographic design that its hardware implementations can be tested *without* the need for scan pads.

4.4 Block ciphers

Block ciphers are the most versatile of the three types of cipher. Their different modes enable them to perform ECB block encryption, CBC encryption, self-synchronizing stream encryption and synchronous stream encryption.

Almost all single-key block cipher proposals are of the *iterated* type. In these ciphers the ciphertext is obtained from the plaintext by a specified number m of iterations of an invertible round transformation. These round transformations are parameterized by round subkeys, that are derived from κ as specified in the *key schedule*. The Data Encryption Standard or DES [37], described in Chapter 5, is

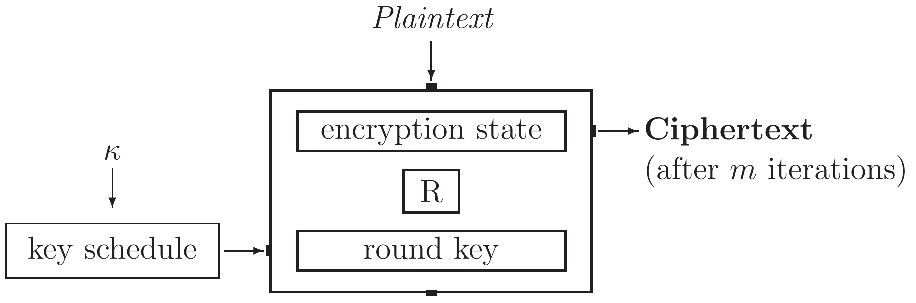


Figure 4.2: Cryptographic finite state machine implementation of an iterated block cipher.

the best known iterated block cipher. Ever since its publication, DES has dominated the cryptologic activity in the area of block ciphers.

As illustrated in Fig. 4.2, an iterated block cipher can be implemented as a cryptographic finite state machine in a straightforward way. The state updating transformation consists of the round transformation. The plaintext is loaded into the state register and converted into the ciphertext by simply performing m iterations. The buffer contains the round key that is updated according to the external key schedule that has as input κ.

For decryption to be practical, the inverse of the round transformation must be easily implementable. Preferably this inverse can be realized by reusing the same cryptographic finite state machine in some manner. Hence, the inverse of the block cipher should have the same structure as the block cipher itself. We denote this by the term *structural self-reciprocity*.

In the case of DES the state is split into a left part L_i and a right part R_i. The round transformation consists of two steps. First the left part is updated according to $L_i = L_i + F(R_i, K_i)$, with F the so-called F-function. In the second step the left and the right part are switched. Both steps are involutions. The structure of this round transformation is named after Horst Feistel, who led within IBM the research activity that gave rise to iterated block ciphers and more specifically LUCIFER, the predecessor of DES [35, 36].

The main body of DES consists of 16 iterations of the round transformation with the left-right switch of the last round removed. This is preceded by a bit permutation IP and followed by the inverse of this bit permutation IP^{-1}. The round keys K_i are derived from the global key according to a key schedule.

DES can also be seen as 16 applications of the F-function alternated by left-right switches. Since both operations are involutions, DES and its inverse have the same structure with exception of the key schedule. The most important feature of the Feistel structure is that the involution property does not restrict the design of the F-function in any way. During the last years, there have been a number of block cipher proposals with the Feistel structure. All of these can be considered as variants of DES: different components in a fixed (or only slightly modified) global computational graph.

4.4.1 Our approach

Our goal is to design iterated block ciphers that have a structural self-reciprocity. We consider the Feistel structure undesirable since only half of the internal state bits enter the F-function. As described in Chapter 5, this has been extensively exploited in recent attacks.

We propose to design the round transformation consisting of a number of different steps. These steps are transformations with a high degree of symmetry and uniformity, and can be realized in hardware with a small gate delay. In this way the block cipher can be implemented in hardware as a cryptographic finite state machine, as illustrated in Fig. 4.2.

One of the round transformation steps is a bitwise addition of the round key, all others are key-independent. These transformations cannot be chosen freely as the F-function in the Feistel structure. Self-reciprocity is realized by imposing certain algebraic conditions on the step transformations and carefully arranging them. The key schedule is kept as simple as possible. The choice of the step transformations and their arrangement is mainly governed by the two important types of cryptanalysis as described in Chapter 5. In Chapter 7 this approach is further elaborated and illustrated by a completely specified design and some variants.

4.5 Self-synchronizing stream ciphers

Single-bit self-synchronizing stream encryption has a specific advantage over all other types of encryption. For, in providing an existing communication system with encryption, single-bit self-synchronizing stream encryption can be applied without the need for additional synchronization or segmentation. In communication systems that transmit symbols consisting of more than a single bit, this is also the case if the encryption symbol length n_{s} divides the symbol length in the communication system. An example of such a communication system is a quadrature phase shift keying (QPSK) modulation scheme [101], that transmits symbols consisting of two bits.

The most widely adopted approach to self-synchronizing stream encryption is the use of a block cipher in CFB mode. For this mode, the attainable encryption speed is a factor $n_{\mathrm{b}}/n_{\mathrm{s}}$ slower than the encryption speed of the underlying block cipher implementation. For the single-bit CFB mode for DES this factor is 64. Hence, the CFB mode with small symbol length is very inefficient compared to the ECB and CBC modes. This poor efficiency seems to be inherent in self-synchronizing stream encryption with small symbol length. Its only appropriate domain of application is in the addition of encryption in existing systems that have no segmentation or synchronization provisions facilitating block or synchronous stream encryption. For this reason, we have not spent any efforts in trying to design dedicated self-synchronizing stream ciphers for portability.

For applications in which single-bit self-synchronizing stream encryption is needed with rates above 10 Mbit/s, even a hardware implemented block cipher in CFB mode may not be fast enough. For these high-speed applications it can be worthwhile to

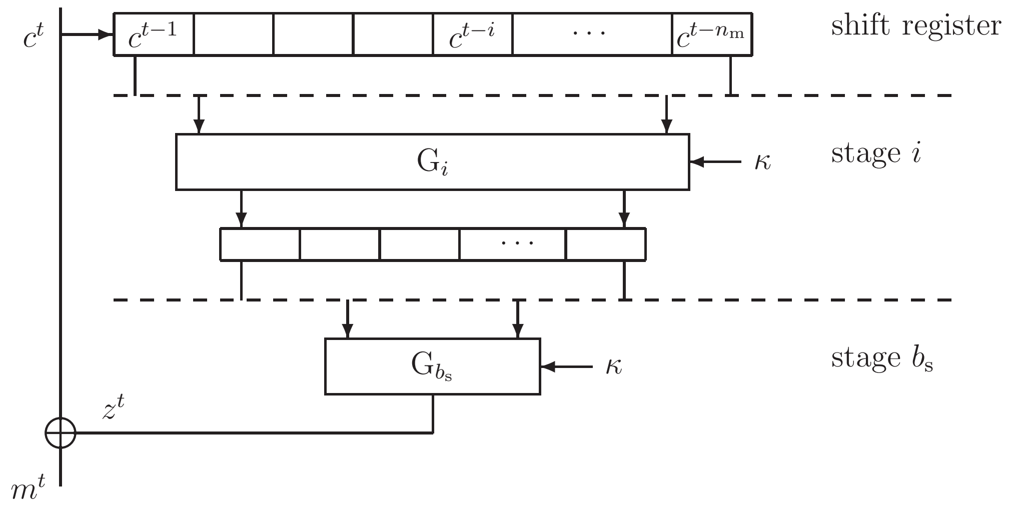


Figure 4.3: Self-synchronizing stream cipher with a cipher function consisting of stages.

design a dedicated hardware-oriented single-bit self-synchronizing stream cipher.

4.5.1 Our approach

We propose to compose the cipher function of a number, denoted by b_s, of *stages* G_i. In hardware, every stage can be implemented by a combinatorial circuit and a register storing the intermediate result. This pipelined approach is illustrated in Fig. 4.3. It can be seen that the encryption speed is limited by the largest occurring gate delay. The implementation of the cipher function in b_s stages causes the output z^t to depend on the contents of the shift register b_s time steps ago. Hence, the encryption equations of Chapter 2 have to be adapted to:

$$\begin{aligned}(\kappa, IV) &= \mathrm{J}(K, Q)\ , && (4.1)\\ c^{1-n_\mathrm{m}-b_\mathrm{s}} \dots c^0 &= IV\ , && (4.2)\\ z^t &= \mathrm{f_c}[\kappa](c^{t-n_\mathrm{m}-b_\mathrm{s}} \dots c^{t-1-b_\mathrm{s}})\ . && (4.3)\end{aligned}$$

In Chapter 9 we elaborate on the design of high-speed single-bit self-synchronizing stream ciphers.

4.6 Synchronous stream ciphers

Most of the cryptologic activity in the area of synchronous stream ciphers is concentrated in two closely linked subdisciplines:

1. The design and analysis of synchronous stream ciphers with an updating transformation realized by a combination of (most often binary) linear feedback shift registers (LFSR).

2. The study and description of “randomness criteria” for sequences, especially those originating from LFSR-type ciphers.

Next to the synchronous stream ciphers of the LFSR type, there have been a number of practical proposals inspired by specific interdisciplinary theories such as chaos theory. Our early proposals based on cellular automata that are briefly described in Chapter 8 are of this type. Finally, we must mention the existence of proposals inspired by computational complexity theory and the information-theoretical concepts of local randomness and randomized ciphers. These proposals are more theoretically oriented and have no real practical relevance. A good overview of the scientific activity in the area of synchronous stream ciphers containing an extensive bibliography is given by Rainer Rueppel in [95].

In the class of the LFSR-based synchronous stream ciphers we can distinguish the regularly clocked ciphers and the clock-controlled ciphers. The former are in fact filtered counter ciphers. In the latter, certain parts of the finite state machine are irregularly clocked. In *forward clock control* one regularly clocked LFSR is used to control the clock of another LFSR. In *feedback clock control* the number of iterations before the next output symbol is determined by the value of the last output symbol. For a more detailed overview of clock-controlled shift registers we refer to Dieter Gollman and William Chambers in [44].

Although indispensable in almost all real-world applications, no initialization mappings are mentioned in the publications that contain the LFSR-based proposals. In most cases, the key is directly mapped to the initial state and there is no public parameter. For all these LFSR-based encryption schemes it can be shown that they are not K-secure. For the filtered counter ciphers there exist very powerful correlation attacks that exploit the correlation of encrypting bits with the input bits to the output function. [46, 95]

Turning the LFSR-based schemes into acceptable synchronous stream encryption schemes requires the design of initialization mappings in such a way that the K-security is not jeopardized. The problem with LFSR-based ciphers is that the simplicity of the updating transformation makes them very weak against attacks that involve internal states with a specified bitwise difference. It is therefore necessary that the initialization mappings have a certain degree of complexity of their own. In Chapter 10 a number of attacks are described that exploit the linearity of the initialization mapping and the updating transformation.

Another very important disadvantage of most LFSR-based ciphers is their limited speed both in software and dedicated hardware implementations due to the fact that only a single encrypting bit is produced per iteration. In hardware this is compensated by the high attainable clock speed resulting in reasonable throughput rates. In software however, the complete updating transformation and the output function have to be simulated for every encrypting bit.

The emphasis on LFSR-based ciphers has had its impact on the research of “randomness of sequences”. In the rhetoric of the corresponding literature, a sequence cannot be classified as “random” if it fails to meet the known criteria. A sequence meets a given criterion if a characteristic that is extracted by a specific test does

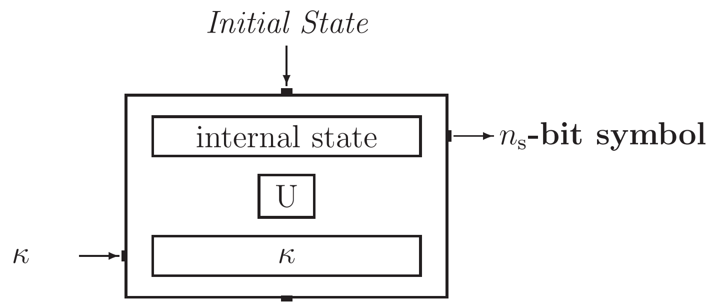


Figure 4.4: Cryptographic finite state machine implementation of a synchronous stream cipher.

not show a significant deviation from what can be expected from a uniformly chosen sequence of equal length.

The most prominent criterion is that of the *linear complexity profile*. This is in part thanks to the existence of an efficient algorithm to calculate the linear complexity profile of a sequence in the form of the Berlekamp-Massey algorithm [69]. The linear complexity profile is especially suited for detecting "non-randomness" in sequences originating from simple LFSR-based ciphers. Alternatively, LFSR-based ciphers can be designed that have a guaranteed minimum linear complexity. Other criteria that are considered important are statistical distribution properties, such as 0/1 balance and n-tuple distribution, and generalizations of linear complexity, such as quadratic span and maximum-order complexity [57].

The only limit to the variety of criteria that can be formulated is ones imagination. Consider the subset of cryptanalytic attacks on synchronous stream ciphers that can be described in terms of "randomness" criteria. These attacks consist of two parts: the choice (or construction) of the criterion and the application of the test. Clearly, the creative and most important part of the cryptanalysis is the construction of the criterion, exploiting structural weaknesses of the cipher itself. Nontrivially breakable ciphers with cryptographic weaknesses that are revealed by one of the general criteria described in literature invariably belong to one of two specific categories. The first category is that of academic examples especially designed to demonstrate the supposed relevance of the proposed criterion. The second category is that of bona-fide proposals designed by cryptologic illiterates.

4.6.1 Our approach

The work factor of the generation of a single encrypting symbol is the execution of the updating transformation and the output function. In specific implementations, practical constraints typically impose an upper limit to this work factor. Hence, we are faced with the problem of dividing the available resources over the updating transformation and the output function.

Turning a synchronous stream cipher into a synchronous encryption scheme requires the specification of an initialization mapping. Preferably, this initialization mapping is as simple as possible, requiring a minimum of additional circuitry in dedicated hardware implementations and a minimum amount of code in software implementations.

There is a strong difference between the effect of the output function and the updating transformation. While an execution of the former only affects a single encrypting symbol, an execution of the latter affects the evolution of the internal state and thus all consecutive encrypting symbols. As illustrated by the resynchronization attacks in Chapter 10, this is especially relevant in attacks that exploit the initialization mapping. It is shown that, in the case of a linear initialization mapping and updating function, the cryptanalyst has a very high degree of access to the input of the output function. This access enables the cryptanalyst to reconstruct the key, even in the presence of a very complex output function.

These considerations have led us to a design approach quite distant from the LFSR-based ciphers. In our approach, the output function is reduced to a mere *selection* of n_{s} bits of the internal state. The resistance against attacks must be realized by the propagation properties of the (iterated) updating transformation. The resulting cipher must allow extremely simple initialization mappings, requiring no additional processing at all. A synchronous stream cipher that has been designed by this approach can be implemented in a straightforward way by means of a cryptographic finite state machine. This is illustrated in Fig. 4.4.

Clearly, a K-secure synchronous stream encryption scheme can be used for other purposes than encryption. This includes the generation of large quantities of key material from a single master key and the simulation of a BSS.

4.7 Cryptographic hash functions

During the last decade, the design and analysis of cryptographic hash functions have developed into an active field with relatively many specific proposals. For a thorough treatment of this field we refer to the doctoral dissertation of Bart Preneel [84].

All cryptographic hash function proposals are of the sequential type, as described in Fig. 2.14. The padded and segmented message is fed block by block to a process governed by a chaining transformation that transforms the chaining state parameterized by the message block. The initial chaining state is specified and the final chaining state is used to determine the hash result. Clearly, the component that is crucial for both performance and resistance against cryptanalysis is the chaining transformation.

Two important properties that distinguish cryptographic hash functions from ordinary ones are collision-resistance and (2nd) preimage-resistance. Most of the cryptologic literature on hash functions has concentrated on these aspects. In [29], Ivan Damgård presented a design principle for collision-resistant hash functions that has determined the structure of most hash function proposals. This principle was also proposed independently by Ralph Merkle in [77]. Essentially, it is a method to construct a collision-resistant hash function by using as the chaining transformation a collision-resistant compression function $\mathrm{f}_{\mathrm{com}}$ with a fixed-length input. This

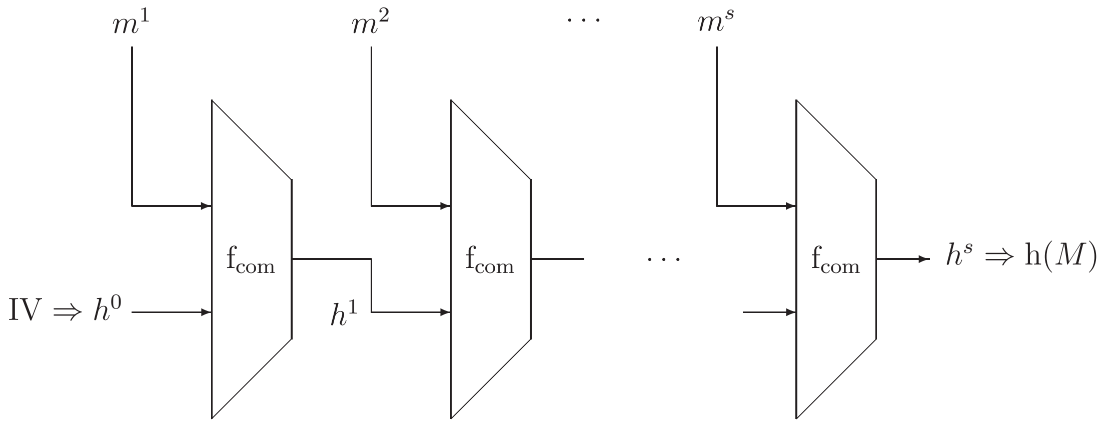

m^1
m^2
$\cdots$
m^s
$\mathrm{f}_{\mathrm{com}}$
$\mathrm{f}_{\mathrm{com}}$
$\cdots$
$\mathrm{f}_{\mathrm{com}}$
$h^s \Rightarrow \mathrm{h}(M)$
$\mathrm{IV} \Rightarrow h^0$
h^1

Figure 4.5: Traditional iterated hash function construction.

"traditional" hash function construction is shown in Fig. 4.5.

Ivan Damgård proved in a computational complexity theoretic framework that generating a collision for such a hash function involves either generating a collision for $\mathrm{f}_{\mathrm{com}}$ or solving a problem with comparable complexity. This reduced the problem of designing a collision-resistant hash function to that of designing a collision-resistant fixed-length input compression function. The weak point of this N-reductionist approach lies in the uncertainty whether this second problem is in fact *easier*. Moreover, seen within our generic model for sequential hash functions, application of the Damgård-Merkle principle imposes important restrictions.

In the traditional construction the chaining transformation of our generic model, $d^i = \mathrm{G}(d^{i-1}, m^i)$ consists of the compression function $\mathrm{f}_{\mathrm{com}}(m^i \| d^{i-1})$ with fixed-length input. The output function G_{o} is omitted or, equivalently, taken to be the identity. Clearly, this restricts the length of the chaining state to that of the hash result. In the Damgård proof it is an essential condition that the compression function $\mathrm{f}_{\mathrm{com}}$ that forms the chaining transformation is collision-resistant. In this context, a collision for the chaining transformation denotes two different couples (d, m) and (d^*, m^*) with $\mathrm{G}(d, m) = \mathrm{G}(d^*, m^*)$. For an invertible chaining transformation collisions can be generated ad libitum. Therefore the Damgård-Merkle design principle can only be applied if the chaining transformation is not invertible.

Despite its restrictions, almost all hash function proposals are designed by the traditional approach. In many proposals the compression function is realized by combinations of a block cipher and some simple operations. An important part of [84] is devoted to the study of these schemes. Other block-cipher based proposals can be found in [64]. Besides these there are many dedicated designs. The best known examples are MD4 [92] and MD5 [93] by Ronald Rivest, Snefru [78] by Ralph Merkle and the standard NIST-SHA [39, 40]. For their description and analysis, together with that of many lesser known proposals, we refer again to [84].

Most dedicated hash function designs are tailored to run very fast on processors with a 32-bit word length. Their sequential nature and the use of arithmetic operations however form an obstacle for feasible hardware realizations that are substantially faster than their corresponding software implementations. Moreover, none of them has a short and elegant description.

4.7.1 Our approach

Our goal is the design of simple and portable unkeyed and keyed cryptographic hash functions that are hermetic.

The basic principle of our approach is derived from the compression function of an early version of MD4. The operation of this compression function can be described as encrypting in an invertible way the 128-bit chaining state d^{i-1} to d^i with m^i as a key, or

$$d^i = \mathrm{f}_{\mathrm{MD4}}(d^{i-1}, m^i) \ . \tag{4.4}$$

In a later version this compression function was modified by adding d^{i-1} to the "encryption result" :

$$d^i = d^{i-1} + \mathrm{f}_{\mathrm{MD4}}(d^{i-1}, m^i) \bmod 2^{128} \ . \tag{4.5}$$

It can be seen that for the compression function (4.4) collisions can be generated by simply inverting the encryption. In (4.5) this invertibility is removed and the Damgård-Merkle principle can be applied. Moreover, in the case of (4.4), a message that hashes to a given result can be generated by a meet-in-the-middle attack with a work factor of approximately 2^{64} executions of the compression function. This attack is not possible in the case of (4.5).

In our design approach the chaining state d is split into two parts: the *hashing state* a and the *buffer contents* b. The application of the chaining transformation affects both parts. The buffer consists typically of a number of parallel shift registers with either no or very simple feedback. Without feedback, every bit of the buffer contents corresponds to some past message bit. With feedback, the bits of the buffer contents are linear functions of message bits. The buffer can be initialized by setting it to 0.

The hashing state is updated by a simple nonlinear invertible updating transformation, called the *round transformation* that is parameterized by part of the buffer contents. We have

$$a^i = \mathrm{G_h}[\mathrm{sel}(b^i)](a^{i-1}) \ , \tag{4.6}$$

with sel(b) denoting a prespecified subset of the buffer bits. The hashing state is initialized by setting it to some prespecified value, denoted by IV. Similar to MD4, the effect of the repeated application of the chaining transformation on the hashing state can be seen as an encryption of IV to the final hashing state with the variable-length message as key. The buffer and its updating function can be seen as the key schedule.

After all message blocks have appeared at the input, a number of extra blank iterations are performed in order to make the final value of the hashing state depend in a complicated way on the last message blocks. The hash result is derived from

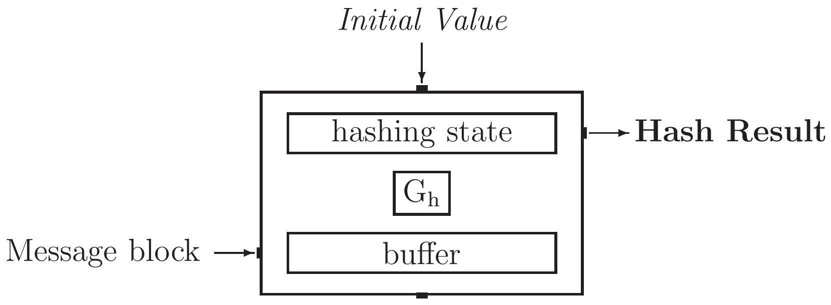


Figure 4.6: Cryptographic finite state machine implementation of a cryptographic hash function.

this final value by simply taking a subset of its bits, possibly interleaved by some additional blank iterations of the chaining transformations.

It can be seen that a hash function designed by this approach can be directly implemented as a cryptographic finite state machine. This is depicted in Fig. 4.6.

To be hermetic, the hashing state must be at least two times as long as the hash result. Otherwise there exists a meet-in-the-middle attack that is not possible in the case of a super hash function. This attack can be applied to find a message that hashes to a given result. This meet-in-the-middle attack is similar to the one described in Sect. 2.7.2.

Because of the simplicity and invertibility of the round transformation, it is essential for the collision-resistance of the hash function that every linear combination of message bits affects sel(b) at least two times, preferably with a large number of iterations in between. This must be realized by carefully tuning the buffer updating function and the selection function sel(b). It also imposes a minimum size on the buffer length.

We have only treated the design of non-keyed cryptographic hash functions thus far. In our approach, the hashing process consists for a large part of the encryption of IV into the final value of the hashing state with the message as key. For this reason it seems natural to include the actual key in the message. We propose to turn a non-keyed hash function into a keyed one by preceding the first message blocks $m^1 m^2 \ldots$ by a number of key blocks and by succeeding the last message blocks $\ldots m^{s-1} m^s$ by the same key blocks. In this way the key pervades the complete hashing process.

4.8 Choice of basic operations

In this section we discuss the simplicity and implementation properties of the most common basic operations that are available to specify cryptographic schemes. From this discussion we motivate our choice of building blocks.

4.8.1 Bit permutations

A bit permutation of a vector modifies the order of its component bits without affecting their value. A straightforward description of a bit permutation acting upon an n-bit vector takes $n \log_2 n$ bits. Hence, for simplicity of description it is desirable to introduce some regularity or symmetry. Typical regular bit permutations are those in which the new bit positions can be calculated from the old ones using a simple expression. For example, in a simple bit rotation over d positions the bit with index i moves to position $i + \mathrm{d}$ for some d.

In a dedicated hardware implementation, a bit permutation can be implemented without the need for additional logical gates. The bit permutation is contained in the interconnection pattern. However, for bit permutations with a non-local character the interconnections can constitute a large part of the area of the resulting chip.

A bit permutation in which bits are moved individually is not suited for software implementations on general-purpose processors. The bit permutation can be done fully serial, moving every individual bit to its desired position. This is however very slow. A faster method using lookup tables can treat k bits at a time. In general this method requires $\lceil \frac{n}{k} \rceil$ tables of 2^k elements of size n. The exponential growth of these tables in the number of input bits limits the feasibility of this method. The subclass of bit permutations that allows a blocked approach is much better suited for software implementation. This includes vector rotations and permutations within and acting on subblocks. These can be implemented by combining the widely available left and right word shift instructions and bitwise Boolean instructions. The unavoidable choice of the length of the subblocks favors one specific word length over the others and therefore limits the portability of these bit permutations.

4.8.2 Bitwise Boolean operations

Bitwise Boolean operations treat all individual components of binary vectors in the same way. The most important are bitwise complementation and the binary operations of bitwise AND, OR and EXOR. They are in general very simple to describe.

In a dedicated hardware implementation, Boolean operations can be implemented in a straightforward way using their hardware counterparts: compact logical gates with small gate delay.

In software, bitwise Boolean operations can be implemented in straightforward way using widespread efficient bitwise Boolean instructions. There are no portability problems since, for a processor word length of k bits, a bitwise Boolean operation acting on vectors of length n can be split into the execution of $\lceil \frac{n}{k} \rceil$ bitwise Boolean instructions.

4.8.3 Substitution boxes

An s-bit to u-bit substitution box (or S-box) returns an u-bit block if given an s-bit block. Such a substitution box can be described by a table of 2^s elements of u bits. Hence, for simplicity of description it is desirable to keep these boxes small or specify their elements by a succinct recipe. Another useful measure in this respect is limiting the number of different substitution boxes used in a single design.

In dedicated hardware implementations, substitution boxes can be implemented as a straightforward ROM lookup table or as a fully optimized custom cell. Since the area of these modules grows exponentially with the the length of the input s, small boxes are most desirable from this point of view.

In software, a substitution box is implemented as an array of constants that is indexed by the s-bit input. Although they can sometimes be efficiently implemented, substitution boxes have in general small portability since there is no benefit in using processors with a word length larger than the substitution box input length. For processors with a word length several times the substitution box input length, a speedup can be obtained by handling two (or more) substitution boxes at the same time. For this purpose two S-boxes are combined in a single (typically very large) table with 2^{2s} elements.

4.8.4 Modular arithmetic operations

In modular arithmetic operations binary vectors are taken to be binary representations of integers. If the implicit mapping between vectors and numbers is established, these operations can be described as compactly as bitwise Boolean operations. The most important arithmetic operations are addition, subtraction, multiplication and modular reduction.

At the Boolean level, arithmetic operations can best be described in a recursive way. Addition consists of an array of simply connected Boolean bit adders, with the interaction between the adders by *carry bits*. Multiplication can be expressed as a number of additions and shift operations. For dedicated hardware implementations of addition and subtraction, there is a trade-off between gate delay and area due to the carry propagation. For the simplest serial implementation this gate delay is proportionate to the length of the numbers to be processed. Still, addition and subtraction can be efficiently implemented in hardware. Multiplication and modular reduction are very complicated operations, and fast dedicated implementations lead to a very large area even for a modest word length.

In software, modular arithmetic operations are implemented using the widespread arithmetic instructions. If the length of the numbers in the cryptographic scheme is adapted to the word length of the processor, this can be done very efficiently. As in the case of substitution boxes there is however no benefit in larger processor word lengths. If the processor word length is smaller than required, the operations have to be implemented by cumbersome multiple precision routines. Hence, the portability of modular arithmetic operations is rather small.

These modular arithmetic operations distinguish themselves from the other ones by their representative character. While for the other operations the appropriate level of description is the bit level, this is not the case for modular arithmetic. The latter is usually described at a higher level carrying with it a rich algebraic structure. Injudicious use of modular arithmetic operations can enable cryptanalysis exploiting these algebraic properties, without the need to resort to the bit level. Still, the inability of exploiting these algebraic properties does not imply that successful cryptanalysis at the bit level is impossible.

Chapter 11 contains two examples of attacks on cryptographic schemes based on arithmetic operations and a new block cipher design with modular multiplication as its main component.

4.8.5 Our choice

From these four classes of basic operations those with the best hardware suitability, portability and simplicity properties are the bitwise Boolean operations and the blockwise bit permutations. However, Bitwise Boolean operations are not directly usable as transformations in cryptographic schemes. Therefore we propose to combine them into invertible transformations with a high degree of symmetry and a simple specification. These are the *shift-invariant transformations on binary arrays* that will be introduced in Chapter 6.

4.9 Conclusions

This chapter has given a description of and a motivation for our own design approach. For each of the different types of cryptographic schemes we have highlighted the differences with the conventional approaches and indicated the origin of the initial concepts.

Chapter 5

Propagation and Correlation

5.1 Introduction

In this chapter we treat difference propagation and input-output correlation in Boolean mappings and iterated Boolean transformations. Difference propagation is specifically exploited in differential cryptanalysis (DC), invented by Eli Biham and Adi Shamir [5]. Input-output correlation is exploited in linear cryptanalysis (LC), invented by Mitsuru Matsui [71]. Both DC and LC were successfully applied on the block cipher DES [37]. DC was the first chosen-plaintext attack, LC the first known-plaintext attack more efficient than exhaustive key search for DES.

We start with a brief description of DES and the original DC and LC attacks using the terminology of their inventors. For a more detailed treatment of the attacks, we refer to the original publications [5, 71]. The only aim of our description is to indicate the aspects of the attacks that determine their expected work factor. For DC the critical aspect is the maximum *probability* for difference propagations, for LC it is the maximum *deviation* from 1/2 of the probability that linear expressions hold.

We introduce a number of algebraic tools that more adequately describe the essential mechanisms of LC and DC. This includes a number of powerful new concepts, such as the *correlation matrix* of a Boolean mapping. Using these new concepts a number of new relations and equalities are derived. These tools are further refined to describe propagation and correlation in iterated Boolean transformations.

Finally, we formulate and motivate our new structural design strategy for the round transformation of block ciphers and, more generally, the updating transformation of cryptographic finite state machines.

5.2 The Data Encryption Standard

The cipher that was the most important object of the attacks to be discussed is the Data Encryption Standard (DES) [37]. Therefore, we start with a brief description of its structure.

DES is a block cipher with a block length of 64 bits and a key length of 56

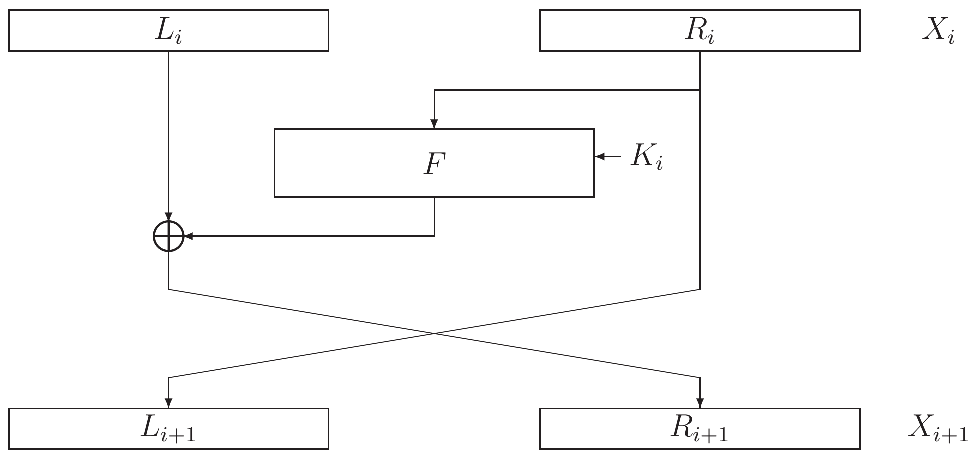


Figure 5.1: Computational graph of the DES round transformation.

bits. Its main body consists of 16 iterations of the *keyed round transformation*. The computational graph of the round transformation is depicted in Fig. 5.1. It can be seen that the intermediate encryption value is split into a 32-bit left part L_i and a 32-bit right part R_i. The latter is the argument of the keyed F-function. The output of the F-function is added bitwise to L_i. Subsequently, left and right part are interchanged.

The computational graph of the F-function is depicted in Fig. 5.2. It can be seen that it consists of the succession of four steps. In the expansion E the 32 input bits are expanded to 48 bits. Subsequently, a 48-bit round key is added bitwise to this 48-bit vector. The resulting 48-bit vector is mapped onto a 32-bit vector by 8 nonlinear S-boxes that each convert 6 input bits into 4 output bits. As an example, Fig. 5.3 gives the specification of the second S-box. Finally, these 32 bits are transposed by the bit permutation P. Observe that the only nonlinear step in the F-function (and also in the round transformation) consists of the S-boxes. The 48-bit round keys are extracted from the 56-bit cipher key by means of a linear key schedule.

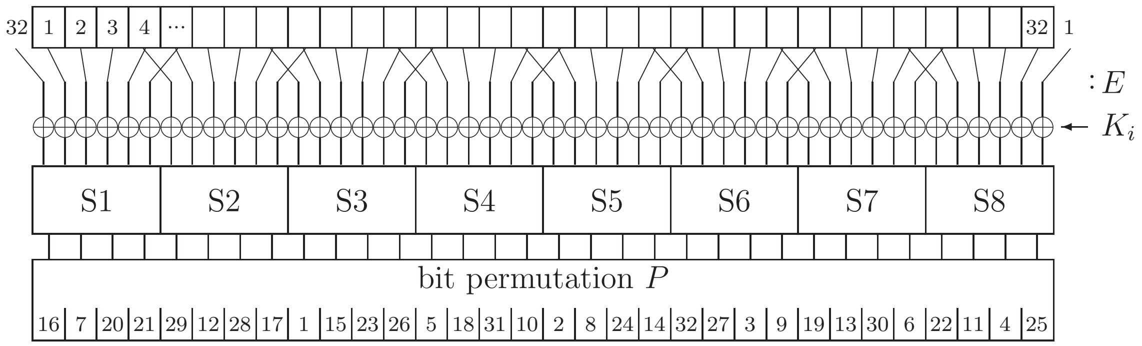


Figure 5.2: Computational graph of the DES F-function.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0 :	f	1	8	e	6	b	3	4	9	7	2	d	c	0	5	a
1 :	3	d	4	7	f	2	8	e	c	0	1	a	6	9	b	5
2 :	0	e	7	b	a	4	d	1	5	8	c	6	9	3	2	f
3 :	d	8	a	1	3	f	4	2	b	6	7	c	0	5	e	9

Figure 5.3: Specification of DES S-box S_2. If the 6-bit input is denoted by $a_1a_2a_3a_4a_5a_6$ the output is given by the entry in row $a_1 + 2a_6$ and column $a_2 + 2a_3 + 4a_4 + 8a_5$. The 4-bit values are given in hexadecimal notation, e.g., d denotes 1101.

5.3 Differential and linear cryptanalysis

In this section we summarize differential cryptanalysis as described in [5] and linear cryptanalysis as presented in [71].

5.3.1 Differential cryptanalysis

Differential cryptanalysis is a chosen-plaintext (difference) attack in which a large amount of plaintext-ciphertext pairs are used to determine the value of key bits. Statistical key information is deduced from ciphertext blocks obtained by encrypting pairs of plaintext blocks with a specific bitwise difference A' under the target key. The work factor of the attack depends critically on the largest probability $\mathrm{P}(B'|A')$ with B' a difference at some fixed intermediate stage of the cryptographic function, e.g., at the input of the last round. In a first approximation, the probabilities $\mathrm{P}(B'|A')$ for DES are assumed to be independent of the specific value of the key.

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

The attack is successful if the right value of the subkey is suggested significantly more often than any other value. Pairs with an intermediate difference not equal to B' are called wrong pairs. Subkey values suggested by these pairs are in general wrong. Right pairs, with an intermediate difference equal to B', do not only suggest the right subkey value but often also a number of wrong subkey values. For DES the wrong suggestions may be considered uniformly distributed among the possible key values if the value $\mathrm{P}(B'|A')$ is significantly larger than $\mathrm{P}(C'|A')$ for any $C' \neq B'$.

Under these conditions it makes sense to calculate the ratio between the number of times the right value is suggested and the average number of suggestions per entry, the so-called *signal-to-noise or S/N ratio*. If the size of the table is 2^ℓ and the average number of suggested subkeys per pair is γ, this ratio is equal to $\mathrm{P}(B'|A')2^{\ell}/\gamma$. The S/N ratio strongly affects the number of right pairs needed to uniquely identify the right subkey value. Experimental results [5] showed that for a ratio of 1-2 about 20-40 right pairs are enough. For larger ratios only a few right pairs are needed and for ratios that are much smaller than 1 the required amount of right pairs can make a practical attack infeasible.

Large probabilities $\mathrm{P}(B'|A')$ are localized by the construction of so-called *characteristics*. An m-round characteristic constitutes an $m+1$-tuple of difference patterns: $(X'_0, X'_1, \ldots, X'_m)$. The probability of this characteristic is the probability that an initial difference pattern X'_0 propagates to difference patterns $X'_1, X'_2, \ldots, X'_m$ respectively after 1, 2, ..., m rounds. In the assumption that the propagation probability from X'_{i-1} to X'_i is independent of the propagation from X'_0 to X'_{i-1}, this probability is given by

$$\prod_i \mathrm{P}(X'_i|X'_{i-1}) \ , \tag{5.1}$$

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

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

Trivial characteristics have been exploited to construct high-probability iterative characteristics, i.e., characteristics with a periodic sequence of differences. The iterative characteristic with highest probability has period 2. Of the two involved single-round characteristics, one is trivial. In the other one there is a nonzero difference pattern at the input of three neighboring S-boxes, that propagates to a zero difference pattern at the output of the S-boxes with probability 1/234. Hence, the resulting iterative characteristics have a probability of 1/234 per 2 rounds.

5.3.2 Linear cryptanalysis

Linear cryptanalysis is a known-plaintext attack in which a large amount of plaintext-ciphertext pairs are used to determine the value of key bits. For the 8-round variant of DES, linear cryptanalysis can also be applied in a ciphertext-only context.

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

$$A[i_1, i_2, \ldots, i_a] = A[i_1] + A[i_2] + \cdots + A[i_a] \ .$$

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

$$P[i_1, i_2, \ldots, i_a] + C[j_1, j_2, \ldots, j_b] = K[k_1, k_2, \ldots, k_c] \ , \tag{5.2}$$

with $i_1, i_2, \ldots, i_a, j_1, j_2, \ldots, j_b$ and $k_1, k_2, \ldots, k_c$ fixed bit locations. The effectiveness of such a linear expression in linear cryptanalysis is given by $|p - 1/2|$ with p the probability that it holds. By checking the value of the left-hand side of (5.2) for a large number of plaintext-ciphertext pairs, the right-hand side can be guessed by taking the value that occurs most often. This gives a single bit of information about the key. In [71] it is shown that the probability of making a wrong guess is very small if the number of plaintext-ciphertext pairs is larger than $|p - 1/2|^{-2}$.

In [71] another algorithm is given that determines more than a single bit of key information using a similar linear expression. Instead of (5.2), an expression is used that contains no plaintext or ciphertext bits, but instead bits of the intermediate encryption values I_1 and I_{15} respectively after a single and all but a single round:

$$I_1[i_1, i_2, \ldots, i_a] + I_{15}[j_1, j_2, \ldots, j_b] = K[\ell_1, \ell_2, \ldots, \ell_c] \ . \tag{5.3}$$

By assuming values for a subset ν_k of the subkey bits of the first and last round, the bits of I_1 and I_{15} that occur in (5.3) can be calculated. These bits are correct if the values assumed for the key bits with indices in ν_k are correct. Given a large number ℓ of plaintext-ciphertext pairs, the correct values of all bits in ν_k and the value of the right-hand side of (5.3) can be determined in the following way. For all values of the key bits with indices in ν_k, the number of plaintext-ciphertext pairs are counted for which (5.3) holds. For the correct assumption the expected value of this sum is $p\ell$ or $(1-p)\ell$. Thanks to the nonlinear behavior of the round transformation this sum is expected to have significantly less bias for all wrongly assumed subkey values. Given a linear expression (5.3) that holds with probability p, the probability that this algorithm leads to a wrong guess is very small if the number of plaintext-ciphertext pairs is significantly (say more than a factor 8) larger than $|p - 1/2|^{-2}$. In a recent improvement of this attack this factor 8 is reduced to 1 [72]. Hence, in both variants the value of $|p - 1/2|$ is critical for the work factor of the attack.

Effective linear expressions (5.2) and (5.3) are constructed by "chaining" single-round linear expressions. An $(m-1)$-round linear expression can be turned into an m-round linear expression by appending a single-round linear expression such that all the intermediate bits cancel:

$$\begin{array}{rcccl}
P[i_1, i_2, \ldots, i_a] & + & I_{m-1}[j_1, j_2, \ldots, j_b] & = & K[k_1, k_2, \ldots, k_c] \\
 & & + & & \\
I_{m-1}[j_1, j_2, \ldots, j_b] & + & I_m[m_1, m_2, \ldots, m_a] & = & K[k_2, k_5, \ldots, k_d] \\
 & & = & & \\
P[i_1, i_2, \ldots, i_a] & + & I_m[m_1, m_2, \ldots, m_a] & = & K[k_1, k_3, \ldots, k_d]
\end{array}$$

In [71] it is shown that the probability that the resulting linear expression holds can be approximated by $1/2 + 2(p_1 - 1/2)(p_2 - 1/2)$, given that the component linear expressions hold with probability p_1 and p_2 respectively.

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

$$I_{\ell-1}[j_1 + 32, j_2 + 32, \ldots, j_a + 32] = I_\ell[j_1, j_2, \ldots, j_a] \ ,$$

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

5.4 Analytical and descriptive tools

In this section we present a formalism and some useful tools for the description and analysis of difference propagation and the chaining of linear expressions. We establish a relation between Boolean mappings and linear mappings over real vector spaces, allowing a much simpler treatment of linear expressions. More importantly, the proposed formalisms forces us to look at the phenomena from a different angle, giving new insights. In the original descriptions of LC and DC, the propagation and chaining are described as *probabilistic* phenomena, with an emphasis on probabilities of events. In our formalism we describe the phenomena in terms of ratios and correlations, reflecting a more deterministic view.

5.4.1 The Walsh-Hadamard transform

Linear cryptanalysis can be seen as the exploitation of *correlations* between linear combinations of bits of different intermediate encryption values (or *states*). The correlation between two Boolean functions with domain $\mathbb{Z}_2^n$ can be expressed by a *correlation coefficient* that ranges between -1 and 1:

Definition 5.1 *The correlation coefficient* $\mathrm{C}(f, g)$ *associated with a pair of Boolean functions* $f(a)$ *and* $g(a)$ *is given by*

$$\mathrm{C}(f, g) = 2 \cdot \Pr(f(a) = g(a)) - 1 \ .$$

From this definition it follows that $\mathrm{C}(f, g) = \mathrm{C}(g, f)$. If the correlation coefficient is different from zero, the functions are said to be *correlated*.

A selection vector w is a binary vector that *selects* all components i of a vector for which $w_i = 1$. Analogous to the inner product of vectors in linear algebra, the linear combination of the components of a vector a selected by w can be expressed as $w^{\mathrm{t}}a$ with the t suffix denoting transposition of the vector w. A linear Boolean function $w^{\mathrm{t}}a$ is completely specified by its corresponding selection vector w.

Let $\hat{f}(a)$ be a real-valued function that is -1 for $f(a) = 1$ and $+1$ for $f(a) = 0$. This can be expressed by $\hat{f}(a) = (-1)^{f(a)}$. In this notation the real-valued function corresponding to a linear Boolean function $w^{\mathrm{t}}a$ becomes $(-1)^{w^{\mathrm{t}}a}$. The bitwise sum of two Boolean functions corresponds to the bitwise product of their real-valued counterparts, i.e.,

$$\widehat{f(a) + g(a)} = \hat{f}(a)\hat{g}(a) \ . \tag{5.4}$$

We define an *inner product* for real-valued functions, not to be confused with the inner product of *vectors*, by

$$< \hat{f}, \hat{g} >= \sum_a \hat{f}(a)\hat{g}(a) \ , \tag{5.5}$$

and the corresponding *norm* by

$$\|\hat{f}\| = \sqrt{< \hat{f}, \hat{f} >} \ . \tag{5.6}$$

For a Boolean function $f(a)$, the norm of $\hat{f}(a)$ is equal to the square root of its domain size, i.e., $2^{n/2}$. From the definition of the correlation coefficient it follows that

$$\mathrm{C}(f, g) = \frac{< \hat{f}, \hat{g} >}{\|\hat{f}\| \cdot \|\hat{g}\|} \ . \tag{5.7}$$

In the space of all Boolean functions, the real-valued functions corresponding to the linear Boolean functions form an orthogonal basis with respect to the defined inner product:

$$< (-1)^{u^{\mathrm{t}}a}, (-1)^{v^{\mathrm{t}}a} >= 2^n \delta(u + v) \ , \tag{5.8}$$

with $\delta(w)$ the Kronecker delta function that is equal to 1 if w is the zero vector and 0 otherwise. The representation of a Boolean function with respect to this basis is called its Walsh-Hadamard transform [45, 84]. If the correlation coefficients $\mathrm{C}(f(a), w^{\mathrm{t}}a)$ are denoted by $\hat{F}(w)$, we have

$$\hat{f}(a) = \sum_w \hat{F}(w)(-1)^{w^{\mathrm{t}}a} \ , \tag{5.9}$$

and dually,

$$\hat{F}(w) = 2^{-n} \sum_a \hat{f}(a)(-1)^{w^{\mathrm{t}}a} \ , \tag{5.10}$$

summarized by

$$\hat{F}(w) = \mathcal{W}(f(a)) \ . \tag{5.11}$$

Hence, a Boolean function is completely specified by the set of correlation coefficients with all linear functions.

Taking the square of the norm of both sides of (5.9) and dividing by 2^n yields the theorem of Parseval [84, p. 223]:

$$1 = \sum_v \sum_w \hat{F}(v)\hat{F}(w) < (-1)^{v^t a}, (-1)^{w^t a} >= \sum_w \hat{F}^2(w) \ , \tag{5.12}$$

expressing a relation between the number of linear functions that are correlated with a given Boolean function and the amplitude of their correlations.

The Walsh-Hadamard transform of the sum of two Boolean functions $f(a)+g(a)$ can be derived using (5.9):

$$\begin{aligned} \hat{f}(a)\hat{g}(a) &= \sum_u \hat{F}(u)(-1)^{u^t a} \sum_v \hat{G}(v)(-1)^{v^t a} \\ &= \sum_u \sum_v \hat{F}(u)\hat{G}(v)(-1)^{(u+v)^t a} \\ &= \sum_w \left(\sum_v \hat{F}(v+w)\hat{G}(v) \right) (-1)^{w^t a} \ . \end{aligned} \tag{5.13}$$

The values of $\hat{H}(w) = \mathcal{W}(f+g)$ are therefore given by

$$\hat{H}(w) = \sum_v \hat{F}(v+w)\hat{G}(v) \ . \tag{5.14}$$

Hence, addition modulo 2 in the Boolean domain corresponds to convolution in the transform domain. If the convolution operation is denoted by $\otimes$ this can be expressed by

$$\mathcal{W}(f+g) = \mathcal{W}(f) \otimes \mathcal{W}(g) \ . \tag{5.15}$$

For multiplication (bitwise AND) of two Boolean functions it can easily be seen that

$$\widehat{f(a)g(a)} = \frac{1}{2}(1 + \hat{f}(a) + \hat{g}(a) - \hat{f}(a)\hat{g}(a)) \ . \tag{5.16}$$

Hence, we have

$$\mathcal{W}(fg) = \frac{1}{2}(\delta(w) + \mathcal{W}(f) + \mathcal{W}(g) - \mathcal{W}(f+g)) \ . \tag{5.17}$$

Given the convolution property it is easy to demonstrate some composition properties that are useful in the study of linear cryptanalysis.

- Complementation of a Boolean function $g(a) = f(a) + 1$ corresponds to multiplication by -1 in the transform domain: $\hat{G}(w) = -\hat{F}(w)$.

- Adding a linear function $g(a) = f(a) + \mathrm{u}^t a$ corresponds to a dyadic shift operation in the transform domain: $\hat{G}(w) = \hat{F}(w + \mathrm{u})$.

The subspace of $\mathbb{Z}_2^n$ generated by the vectors w for which $\hat{F}(w) \neq 0$ is called its *support space* $\mathcal{V}_f$. The support space of the sum of two Boolean functions is a subspace of the (vector) sum of their corresponding support spaces: $\mathcal{V}_{f+g} \subseteq \mathcal{V}_f + \mathcal{V}_g$. This follows directly from the convolution property. Two Boolean functions are called *disjunct* if their support spaces are disjunct, i.e., if the intersection of their support spaces only contains the origin. A vector $v \in \mathcal{V}_{f+g}$ with f and g disjunct has a unique decomposition into a component $u \in \mathcal{V}_f$ and a component $w \in \mathcal{V}_g$. In this case the transform values of $h = f + g$ are given by

$$\hat{H}(v) = \hat{F}(u)\hat{G}(w) \text{ with } v = u + w \text{ and } u \in \mathcal{V}_f, w \in \mathcal{V}_g \ . \tag{5.18}$$

A pair of Boolean functions that depend on non-overlapping sets of input bits is a special case of disjunct functions. Other examples can be found on p. 110.

5.4.2 Correlation matrices

Almost all components in encryption schemes, including S-boxes, state updating transformations and block ciphers are simply mappings from a space of n-dimensional binary vectors to a space of m-dimensional binary vectors. Often $m = n$. These mappings can be represented by their *correlation matrix*.

A mapping $h : \mathbb{Z}_2^n \leftarrow \mathbb{Z}_2^m$ can be decomposed into m *component* Boolean functions: $(h_0, h_1, \ldots, h_{m-1})$. Each of these component functions h_i has a Walsh-Hadamard transform $\hat{H}_i$. The vector function with components $\hat{H}_i$ is denoted by $\hat{H}$ and can be considered the Walsh-Hadamard transform of the mapping h. As in the case of Boolean functions, $\hat{H}$ completely determines the transformation h. The Walsh-Hadamard transform of any linear combination of components of h is specified by a simple extension of (5.15):

$$\mathcal{W}(u^{\mathrm{t}}h) = \bigotimes_{u_i=1} \hat{H}_i \ . \tag{5.19}$$

All correlation coefficients between linear combinations of input bits and those of output bits of the mapping h can be arranged in a $2^m \times 2^n$ *correlation matrix* C^h. The element C^h_{uw} in row u and column w is equal to $\mathrm{C}(u^{\mathrm{t}}h(a), w^{\mathrm{t}}a)$. The rows of this matrix can be interpreted as

$$(-1)^{u^{\mathrm{t}}h(a)} = \sum_w C^h_{uw}(-1)^{w^{\mathrm{t}}a} \ . \tag{5.20}$$

In words, the real-valued function corresponding to a linear combination of output bits can be written as a linear combination of the real-valued functions corresponding to the linear combinations of input bits.

A Boolean function $f(a)$ can be seen as a special case of a mapping and has a correlation matrix with two rows: row 0 and row 1. Row 1 contains the Walsh-Hadamard transform values of $f(a)$ and row 0 the Walsh-Hadamard transform values of the Boolean function that is equal to 0.

A matrix C^h defines a linear mapping with domain $\mathbb{R}^{2^n}$ and range $\mathbb{R}^{2^m}$. Let $\mathcal{L}$ be a mapping from the space of binary vectors to the space of real vectors, depicting a binary vector of dimension n onto a real vector of dimension 2^n. $\mathcal{L}$ is defined by

$$\mathcal{L} : \mathbb{Z}_2^n \rightarrow \mathbb{R}^{2^n} : a \mapsto \mathcal{L}(a) = \alpha \Leftrightarrow \alpha_u = (-1)^{u^{\mathrm{t}} a} \ . \tag{5.21}$$

Since $\mathcal{L}(a+b) = \mathcal{L}(a) \cdot \mathcal{L}(b)$, $\mathcal{L}$ is a group-homomorphism from $< \mathbb{Z}_2^n, + >$ to $< \mathbb{R}^{2^n}, \cdot >$, with "$\cdot$" denoting the componentwise product. From (5.20) it can easily be seen that

$$C^h \mathcal{L}(a) = \mathcal{L}(h(a)) \ . \tag{5.22}$$

Consider the composition of two Boolean mappings $h = h_2 \circ h_1$ or $h(a) = h_2(h_1(a))$, with h_1 mapping n-dimensional vectors to p-dimensional vectors and with h_2 mapping p-dimensional vectors to m-dimensional vectors. The correlation matrix of h is determined by the correlation matrices of the component mappings. We have

$$\begin{aligned}
(-1)^{u^{\mathrm{t}} h(a)} &= \sum_v C^{h_2}_{uv} (-1)^{v^{\mathrm{t}} h_1(a)} \\
&= \sum_v C^{h_2}_{uv} \sum_w C^{h_1}_{vw} (-1)^{w^{\mathrm{t}} a} \\
&= \sum_w (\sum_v C^{h_2}_{uv} C^{h_1}_{vw}) (-1)^{w^{\mathrm{t}} a} \ .
\end{aligned}$$

Hence, we have

$$C^{h_2 \circ h_1} = C^{h_2} \times C^{h_1} \ , \tag{5.23}$$

with $\times$ denoting the matrix product, C^{h_1} a $2^p \times 2^n$ matrix and C^{h_2} a $2^m \times 2^p$ matrix. The input-output correlations of $h = h_2 \circ h_1$ are given by

$$\mathrm{C}(u^{\mathrm{t}} h(a), w^{\mathrm{t}} a) = \sum_v \mathrm{C}(u^{\mathrm{t}} h_1(a), v^{\mathrm{t}} a) \mathrm{C}(v^{\mathrm{t}} h_2(a), w^{\mathrm{t}} a) \ . \tag{5.24}$$

If h is an invertible transformation in $\mathbb{Z}_2^n$, we have

$$\mathrm{C}(u^{\mathrm{t}} h^{-1}(a), w^{\mathrm{t}} a) = \mathrm{C}(u^{\mathrm{t}} b, w^{\mathrm{t}} h(b)) = \mathrm{C}(w^{\mathrm{t}} h(b), u^{\mathrm{t}} b) \ . \tag{5.25}$$

Using this and $C^h \times C^{(h^{-1})} = C^{h \circ h^{-1}} = I = C^h \times (C^h)^{-1}$ we obtain

$$(C^h)^{-1} = C^{(h^{-1})} = (C^h)^{\mathrm{t}} \ , \tag{5.26}$$

hence, C^h is an orthogonal matrix. Conversely, a Boolean mapping with an orthogonal correlation matrix is invertible.

Special mappings

In the following, the suffix h will be omitted. Consider the transformation that consists of the bitwise addition of a constant vector k: $h(a) = a + k$. Since $u^{\mathrm{t}}h(a) = u^{\mathrm{t}}a + u^{\mathrm{t}}k$, the correlation matrix is a diagonal matrix with

$$C_{uu} = (-1)^{u^{\mathrm{t}}k} \ . \tag{5.27}$$

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

Consider a linear mapping $h(a) = Ma$ with M an $m \times n$ binary matrix. Since $u^{\mathrm{t}}h(a) = u^{\mathrm{t}}Ma = (M^{\mathrm{t}}u)^{\mathrm{t}}a$, the elements of the corresponding correlation matrix are given by

$$C_{uw} = \delta(M^{\mathrm{t}}u + w) \ . \tag{5.28}$$

If M is an invertible square matrix, the correlation matrix is a permutation matrix. The single nonzero element in row u is in column $M^{\mathrm{t}}u$. The effect of applying an invertible linear transformation before (or after) a transformation h on the correlation matrix is only a permutation of its columns (or rows).

Consider a mapping from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^m$ that consists of the parallel application of ℓ component mappings (S-boxes) from $\mathbb{Z}_2^{n_i}$ to $\mathbb{Z}_2^{m_i}$ with $\sum_i n_i = n$ and $\sum_i m_i = m$. We will call such a mapping a *juxtaposed* mapping. We have $a = (a_{(0)}, a_{(1)}, \ldots, a_{(\ell-1)})$ and $b = (b_{(0)}, b_{(1)}, \ldots, b_{(\ell-1)})$ with the $a_{(i)}$ vectors of dimension n_i and the $b_{(i)}$ vectors of dimension m_i. The mapping $b = h(a)$ is defined by $b_{(i)} = h_{(i)}(a_{(i)})$ for $0 \leq i < \ell$. With every S-box $h_{(i)}$ corresponds a $2^{n_i} \times 2^{m_i}$ correlation matrix denoted by $C^{(i)}$. Since the $h_{(i)}$ are disjunct, (5.18) can be applied and the elements of the correlation matrix of h are given by

$$C_{uw} = \prod_i C^{(i)}_{u_{(i)}w_{(i)}} \ , \tag{5.29}$$

with $u = (u_{(0)}, u_{(1)}, \ldots, u_{(\ell-1)})$ and $w = (w_{(0)}, w_{(1)}, \ldots, w_{(\ell-1)})$. In words this can be expressed as: the correlation associated with input selection w and output selection u is the product of its corresponding S-box input-output correlations $C^{(i)}_{u_{(i)}w_{(i)}}$.

On p. 90 it is explained how these properties can be used to efficiently calculate the input-output correlations of the DES round transformation.

5.4.3 Derived properties

The concept of the correlation matrix is a valuable tool to demonstrate properties of Boolean transformations, functions and their spectrum. We will illustrate this with some examples.

Lemma 5.1 *The elements of the correlation matrix of a Boolean transformation satisfy*

$$C_{(u+v)x} = \sum_w C_{u(w+x)}C_{vw} \ , \tag{5.30}$$

for all $u, v, x \in \mathbb{Z}_2^n$.

Proof : Using the convolution property, we have

$$\begin{aligned} \mathcal{W}((u+v)^{\mathrm{t}}h(a)) &= \mathcal{W}(u^{\mathrm{t}}h(a)+v^{\mathrm{t}}h(a)) \\ &= \mathcal{W}(u^{\mathrm{t}}h(a)) \otimes \mathcal{W}(v^{\mathrm{t}}h(a)) \ . \end{aligned} \tag{5.31}$$

Since the components of $\mathcal{W}(\xi^{\mathrm{t}}h(a))$ are given by $C_{\xi w}$, the projection of (5.32) onto the component with index x gives rise to (5.30). □

A Boolean function is *balanced* if it is 1 (0) for exactly half of the elements in the domain. Clearly, being balanced is equivalent to being uncorrelated to the Boolean function equal to 0 (or 1).

Using the properties of correlation matrices we can now give an elegant proof of the following well-known theorem.

Theorem 5.1 *A Boolean transformation is invertible if and only if every linear combination of output bits is a balanced Boolean function of its input bits.*

Proof :

$\Rightarrow$ If h is an invertible transformation, its correlation matrix C is orthogonal. Since $C_{00}=1$ and all rows and columns have norm 1, it follows that there are no other elements in row 0 or column 0 different from 0. Hence, $\mathrm{C}(u^{\mathrm{t}}h(a),0)=\delta(u)$ or $u^{\mathrm{t}}h(a)$ is balanced for all $u \neq 0$.

$\Leftarrow$ The condition that all linear combinations of output bits are balanced Boolean functions of input bits corresponds to $C_{u0}=0$ for $u \neq 0$. If this is the case, it can be shown that the correlation matrix is orthogonal. The expression $C^{\mathrm{t}} \times C = \mathrm{I}$ is equivalent to the following set of conditions

$$\sum_w C_{uw}C_{vw} = \delta(u+v) \text{ for all } u,v \in \mathbb{Z}_2^n \ . \tag{5.32}$$

Now, the substitution $x=0$ in (5.30) gives rise to

$$\sum_w C_{uw}C_{vw} = C_{(u+v)0} \ .$$

Since we have $C_{u0}=0$ for all $u \neq 0$ and $C_{00}=1$, (5.32) holds for all possible pairs u,v. It follows that C is an orthogonal matrix, hence h^{-1} exists and is defined by C^{t}.

□

Lemma 5.2 *The elements of the correlation matrix of a mapping with domain* $\mathbb{Z}_2^n$ *and the Walsh-Hadamard transform values of a Boolean function with domain* $\mathbb{Z}_2^n$ *are integer multiples of* 2^{1-n}.

Proof : The sum in the right-hand side of (5.10) is always even since its value is of the form $(k \cdot (1) + (2^n - k) \cdot (-1) = 2k - 2^n$. It follows that the Walsh-Hadamard coordinates must be integer multiples of 2^{1-n}. □

A mapping from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^m$ can be converted into a mapping from $\mathbb{Z}_2^{n-1}$ to $\mathbb{Z}_2^m$ by fixing a single component of the input. More generally, a component of the input can be set equal to a linear combination of other input components, possibly complemented. Such a restriction is of the type

$$v^{\mathrm{t}}a = \epsilon \ , \tag{5.33}$$

with $\epsilon \in \mathbb{Z}_2$. Assume that $v_s \neq 0$. The restriction can be seen as the result of a mapping $a' = h_r(a)$ from $\mathbb{Z}_2^{n-1}$ to $\mathbb{Z}_2^n$ specified by $a'_i = a_i$ for $i \neq s$ and $a'_s = \epsilon + v^{\mathrm{t}}a + a_s$. The nonzero elements of the correlation matrix of h_r are

$$C^{h_r}_{ww} = 1 \text{ and } C^{h_r}_{(v+w)w} = (-1)^{\epsilon} \text{ for all } w \text{ with } w_s = 0 \ . \tag{5.34}$$

It can be seen that columns indexed by w with $w_s = 0$ have exactly two nonzero entries with magnitude 1 and those with $w_s = 1$ are all-zero. Omitting the latter gives a $2^n \times 2^{n-1}$ correlation matrix C^{h_r} with only columns indexed by the vectors with $w_s = 0$.

The transformation restricted to the specified subset of inputs can be seen as the consecutive application of h_r and the transformation itself. Hence, its correlation matrix C' is given by $C \times C^{h_r}$. The elements of this matrix are

$$C'_{uw} = C_{uw} + (-1)^{\epsilon} C_{u(w+v)} \ , \tag{5.35}$$

if $w_s = 0$ and 0 if $w_s = 1$. The elements in C' are the Walsh-Hadamard transform values of Boolean functions of $(n-1)$-dimensional vectors, hence, from Lemma 5.2 they must be integer multiples of 2^{2-n}. This can be easily generalized to multiple linear restrictions on the input.

Applying (5.12) to the rows of the restricted correlation matrices gives additional laws for the Walsh-Hadamard transform values of Boolean functions. For the single restrictions of the type $v^{\mathrm{t}}a = \epsilon$ we have

$$\sum_w (F(w) + F(w+v))^2 = \sum_w (F(w) - F(w+v))^2 = 2 \ . \tag{5.36}$$

Lemma 5.3 *The elements of a correlation matrix corresponding to an invertible transformation of n-bit vectors are integer multiples of 2^{2-n}.*

Proof : Consider an element of the correlation matrix C_{uw}. If the input of the corresponding transformation is restricted by $w^{\mathrm{t}}a = 0$, the correlation of the output function $u^{\mathrm{t}}h(a)$ to 0 becomes $C_{uw} + C_{u0}$. According to Lemma 5.2, this value is an integer multiple of 2^{2-n}. From Theorem 5.1 it follows that $C_{u0} = 0$ and hence that C_{uw} must be an integer multiple of 2^{2-n}. □

With a similar argument it can be shown that either *all* elements of the Walsh-Hadamard transform of a Boolean function are an integer multiple of 2^{2-n} or none of them is.

5.4.4 Difference propagation

Consider a couple of n-dimensional vectors a and a^* with bitwise difference $a + a^* = a'$. Let $b = h(a), b^* = h(a^*)$ and $b' = b + b^*$, hence, the difference a' propagates to the difference b' through h. This is denoted by $(a' \dashv h \vdash b')$ or, if h is clear from the context, simply (a', b'). In general, b' is not determined by a' but depends on the value of a (or a^*).

Definition 5.2 *The prop ratio* $\mathrm{R_p}$ *of a difference propagation* $(a' \dashv h \vdash b')$ *is given by*

$$\mathrm{R_p}(a' \dashv h \vdash b') = 2^{-n} \sum_a \delta(b' + h(a + a') + h(a)) \ . \tag{5.37}$$

If a pair is chosen uniformly from the set of all pairs (a, a^*) with $a + a^* = a'$, the probability that $h(a) + h(a^*) = b'$ is given by $\mathrm{R_p}(a' \dashv h \vdash b')$. In this specific experimental set-up the prop ratio corresponds to a probability. This is however not the case in general and we believe that the widespread use of the term "probability" to denote what we call "prop ratio" has given rise to considerable confusion.

The prop ratio ranges between 0 and 1. Since $h(a+a')+h(a) = h(a)+h(a+a')$, it must be an integer multiple of 2^{1-n}. The difference propagation $(a' \dashv h \vdash b')$ restricts the values of a to a fraction of all possible inputs. This fraction is given by $\mathrm{R_p}(a' \dashv h \vdash b')$. It can easily be seen that

$$\sum_{b'} \mathrm{R_p}(a' \dashv h \vdash b') = 1 \ . \tag{5.38}$$

If $\mathrm{R_p}(a' \dashv h \vdash b') = 0$, the difference propagation $(a' \dashv h \vdash b')$ is called *invalid*. The input difference a' and the output difference b' are said to be *incompatible* through h.

Definition 5.3 *The restriction weight of a valid difference propagation* $(a' \dashv h \vdash b')$ *is the negative of the binary logarithm of the prop ratio, i.e.,*

$$\mathrm{w_r}(a' \dashv h \vdash b') = -\log_2 \mathrm{R_p}(a' \dashv h \vdash b') \ . \tag{5.39}$$

The restriction weight can be seen as the amount of information (in bits) that is given by $(a' \dashv h \vdash b')$ on a, or the loss in *entropy* [98] of a due to the restriction $(a' \dashv h \vdash b')$. The restriction weight ranges between 0 and $n - 1$.

If h is linear, $b' = b + b^* = h(a) + h(a^*) = h(a + a^*) = h(a')$, i.e., a' completely determines b'. From $\mathrm{w_r}(a' \dashv h \vdash b') = 0$ it can be seen that this difference propagation does not restrict or gives away information on a.

Special mappings

An *affine* mapping h from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^m$ is specified by

$$b = Ma + k \ , \tag{5.40}$$

with M a $m \times n$ matrix and k a m-dimensional vector. The difference propagation for this mapping is determined by

$$b' = Ma' \ . \tag{5.41}$$

For a juxtaposed mapping h, it can easily be seen that

$$\mathrm{R_p}(a' \dashv h \vdash b') = \prod_i \mathrm{R_p}(a'_{(i)} \dashv h_{(i)} \vdash b'_{(i)}) \ , \tag{5.42}$$

and

$$\mathrm{w_r}(a' \dashv h \vdash b') = \sum_i \mathrm{w_r}(a'_{(i)} \dashv h_{(i)} \vdash b'_{(i)}) \ , \tag{5.43}$$

with $a' = (a'_{(0)}, a'_{(1)}, \ldots, a'_{(\ell-1)})$ and $b' = (b'_{(0)}, b'_{(1)}, \ldots, b'_{(\ell-1)})$.

A mapping h from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^m$ can be converted into a mapping h_s from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^{m-1}$ by discarding a single output bit a_s. The prop ratios of h_s can easily be expressed in terms of the prop ratios of h:

$$\mathrm{R_p}(a' \dashv h_s \vdash b') = \mathrm{R_p}(a' \dashv h \vdash \omega^0) + \mathrm{R_p}(a' \dashv h \vdash \omega^1) \ , \tag{5.44}$$

with $b'_i = \omega_i^0 = \omega_i^1$ for $i \neq s$ and $\omega_s^1 = 1$ and $\omega_s^0 = 0$. This can be generalized to the situation in which only a number of linear combinations of the output are considered. Let θ be a linear mapping corresponding to an $m \times \ell$ binary matrix M. The prop ratios of $\theta \circ h$ are given by

$$\mathrm{R_p}(a' \dashv \theta \circ h \vdash b') = \sum_{\omega | b' = M\omega} \mathrm{R_p}(a' \dashv h \vdash \omega) \ . \tag{5.45}$$

Prop ratios in terms of correlation coefficients

The prop ratios of the difference propagations of Boolean functions and mappings can be expressed respectively in terms of their Walsh-Hadamard transform values and their correlation matrix elements. With a derivation similar to (5.14) it can be shown that the components of the inverse transform of the componentwise product of two spectra $\hat{c}_{fg} = \mathcal{W}^{-1}(\hat{F}\hat{G})$ are given by

$$\hat{c}_{fg}(b) = 2^{-n} \sum_a \hat{f}(a)\hat{g}(a+b) = 2^{-n} \sum_a (-1)^{f(a)+g(a+b)} \ . \tag{5.46}$$

$\hat{c}_{fg}(b)$ is not a Boolean function. It is generally referred to as the *cross correlation function* of f and g. Hence, the cross correlation function of two Boolean functions is the inverse Walsh-Hadamard transform of the componentwise product of their spectra. If $g = f$ it is called the autocorrelation function of f and denoted by

$\hat{r}_f$. The components of the spectrum of the autocorrelation function consist of the squares of the spectrum of f, i.e.,

$$\hat{F}^2 = \mathcal{W}(\hat{r}_f) \ . \tag{5.47}$$

This is generally referred to as the Wiener-Khintchine theorem [84].

The difference propagation in a Boolean function f can be expressed easily in terms of the autocorrelation function. The prop ratio of the difference propagation $(a' \dashv f \vdash 0)$ is given by

$$\begin{aligned} \mathrm{R_p}(a' \dashv f \vdash 0) &= 2^{-n} \sum_a \delta(f(a) + f(a + a')) \\ &= 2^{-n} \sum_a \frac{1}{2}(1 + \hat{f}(a)\hat{f}(a + a')) \\ &= \frac{1}{2}(1 + \hat{r}_f(a')) \\ &= \frac{1}{2}(1 + \sum_w (-1)^{w^{\mathrm{t}} a'} \hat{F}^2(w)) \ . \end{aligned} \tag{5.48}$$

The component of the autocorrelation function $\hat{r}_f(a')$ corresponds to the amount that $\mathrm{R_p}(a' \dashv f \vdash 0)$ deviates from $1/2$.

For mappings from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^m$, let the autocorrelation function of $u^{\mathrm{t}} h(a)$ be denoted by $\hat{r}_u(a')$, i.e.,

$$\hat{r}_u(a') = 2^{-n} \sum_a (-1)^{u^{\mathrm{t}} h(a) + u^{\mathrm{t}} h(a + a')} \ . \tag{5.49}$$

Now we can easily prove the following remarkable theorem that expresses the duality between the difference propagation and the correlation properties of a Boolean mapping.

Theorem 5.2 *The table of prop ratios and the table containing the squared elements of the correlation matrix of a Boolean mapping are linked by a (scaled) Walsh-Hadamard transform. We have*

$$\mathrm{R_p}(a' \dashv h \vdash b') = 2^{-m} \sum_{u,w} (-1)^{w^{\mathrm{t}} a' + u^{\mathrm{t}} b'} C_{uw}^2 \ , \tag{5.50}$$

and dually

$$C_{uw}^2 = 2^{-n} \sum_{a',b'} (-1)^{w^{\mathrm{t}} a' + u^{\mathrm{t}} b'} \mathrm{R_p}(a' \dashv h \vdash b') \ . \tag{5.51}$$

Proof : we have

$$\begin{aligned}
\mathrm{R_p}(a' \dashv h \vdash b') &= 2^{-n}\sum_a \delta(h(a)+h(a+a')+b') \\
&= 2^{-n}\sum_a \prod_i \frac{1}{2}((-1)^{h_i(a)+h_i(a+a')+b'_i}+1) \\
&= 2^{-n}\sum_a 2^{-m}\sum_u (-1)^{u^{\mathrm{t}}h(a)+u^{\mathrm{t}}h(a+a')+u^{\mathrm{t}}b'} \\
&= 2^{-m}\sum_u (-1)^{u^{\mathrm{t}}b'} 2^{-n}\sum_a (-1)^{u^{\mathrm{t}}h(a)+u^{\mathrm{t}}h(a+a')} \\
&= 2^{-m}\sum_u (-1)^{u^{\mathrm{t}}b'}\hat{r}_u(a') \\
&= 2^{-m}\sum_u (-1)^{u^{\mathrm{t}}b'}\sum_w (-1)^{w^{\mathrm{t}}a'} C_{uw}^2 \\
&= 2^{-m}\sum_{u,w} (-1)^{w^{\mathrm{t}}a'+u^{\mathrm{t}}b'} C_{uw}^2 \ .
\end{aligned}$$

□

5.5 Application to iterated transformations

The described tools and formalisms can be applied to the propagation of differences and the calculation of correlations in iterated transformations. This includes iterated block ciphers such as DES and the repeated application of state updating transformations in synchronous stream ciphers and the round transformations in cryptographic hash functions.

The studied iterated transformations are of the form

$$\beta = \rho_m \circ \ldots \circ \rho_2 \circ \rho_1 \ . \tag{5.52}$$

In a block cipher the ρ_i are selected from a set of round transformations $\{\rho[b] | b \in \mathbb{Z}_2^{n_\mathrm{b}}\}$ by n_b-bit round keys $\kappa^{(i)}$, i.e., $\rho_i = \rho[\kappa^{(i)}]$. These round keys are derived from the cipher key κ by the key schedule. In the iterated application of the state updating transformation of a synchronous stream cipher or a hash function, the ρ_i are selected by (part of) the buffer contents. The correspondence between our formalism and the terminology of the original descriptions of LC and DC is treated in Sect. 5.5.3.

5.5.1 Correlation

Fixed key

In the Walsh-Hadamard transform domain, a fixed succession of round transformations corresponds to a $2^n \times 2^n$ correlation matrix that is the product of the correlation matrices corresponding to the round transformations. We have

$$C^\beta = C^{\rho_m} \times \ldots \times C^{\rho_2} \times C^{\rho_1} \ . \tag{5.53}$$

Linear cryptanalysis exploits the occurrence of large elements in this product matrix.

An m-round *linear trail* Ξ, denoted by

$$\Xi = (\xi_0 \lhd \rho_1 \rhd \xi_1 \lhd \rho_2 \rhd \xi_2 \lhd \ldots \rhd \xi_{m-1} \lhd \rho_m \rhd \xi_m) \;, \tag{5.54}$$

consists of the chaining of m round correlations of the type $\mathrm{C}({\xi_i}^{\mathrm{t}}\rho_i(a), {\xi_{i-1}}^{\mathrm{t}}a)$. To this linear trail corresponds a *correlation contribution coefficient* $\mathrm{C_p}$ ranging between -1 and $+1$. We have

$$\mathrm{C_p}(\Xi) = \prod_i C^{\rho_i}_{\xi_i\xi_{i-1}} \;. \tag{5.55}$$

From this definition and (5.53) we have

$$\mathrm{C}(u^{\mathrm{t}}\beta(a), w^{\mathrm{t}}a) = \sum_{\xi_0 = w, \xi_m = u} \mathrm{C_p}(\Xi) \;. \tag{5.56}$$

Hence, the correlation between $u^{\mathrm{t}}\beta(a)$ and $w^{\mathrm{t}}a$ is the sum of the correlation contribution coefficients of all m-round linear trails Ξ with initial selection vector w and terminal selection vector u.

Variable key

In actual cryptanalysis the succession of round transformations is not known in advance but is governed by an unknown key or some input-dependent value. In general, the elements of the correlation matrix of ρ_i depend on the specific value of the round key $\kappa^{(i)}$.

For some block ciphers the strong round-key dependence of the correlation and propagation properties of the round transformation has been cited as a design criterion. The analysis of correlation or difference propagation would have to be repeated for every specific value of the cipher key, making linear and differential analysis infeasible. A typical problem with this approach is that the *quality* of the round transformation with respect to LC or DC strongly depends on the specific value of the round key. While the resistance against LC and DC may be very good on the average, specific classes of cipher keys can exhibit linear trails with excessive correlation contributions (or differential trails with excessive prop ratios).

These complications can be avoided by designing the round transformation in such a way that the amplitudes of the elements of its correlation matrix are independent of the specific value of the round key. As was shown in Sect. 5.4.2, this is the case if the round transformation consists of a fixed transformation ρ followed (or preceded) by the bitwise addition of the round key $\kappa^{(i)}$ to (part of) the state.

The correlation matrix C^{ρ} is determined by the fixed transformation ρ. The correlation contribution coefficient of the linear trail Ξ becomes

$$\mathrm{C_p}(\Xi) = \prod_i (-1)^{\xi_i^{\mathrm{t}}\kappa^{(i)}} C^{\rho}_{\xi_i\xi_{i-1}} = (-1)^{\epsilon_\Xi + \sum_i \xi_i^{\mathrm{t}}\kappa^{(i)}} |\mathrm{C_p}(\Xi)| \;, \tag{5.57}$$

with $\epsilon_\Xi = 1$ if $\prod_i C^{\rho}_{\xi_i\xi_{i-1}}$ is negative and $\epsilon_\Xi = 0$ otherwise. $|\mathrm{C_p}(\Xi)|$ is independent of the round keys, and hence, only the sign of the correlation contribution coefficient is key-dependent. Analogous to the restriction weight for differential trails, we can define:

Definition 5.4 *The* correlation weight $\mathrm{w_c}$ *of a linear trail* Ξ *is given by*

$$\mathrm{w_c}(\Xi) = -\log_2 |\mathrm{C_p}(\Xi)| \ . \tag{5.58}$$

The correlation weight of a linear trail is the sum of the correlation weights of its linear steps given by $-\log_2 |C^{\rho}_{\xi_i \xi_{i-1}}|$.

The correlation coefficient between $u^{\mathrm{t}}\beta(a)$ and $w^{\mathrm{t}}a$ can be expressed in terms of the correlation contribution coefficients of linear trails:

$$\mathrm{C}(u^{\mathrm{t}}\beta(a), w^{\mathrm{t}}a) = \sum_{\xi_0 = w, \xi_m = u} (-1)^{\epsilon_\Xi + \sum_i \xi_i^{\mathrm{t}} \kappa^{(i)}} |\mathrm{C_p}(\Xi)| \ . \tag{5.59}$$

The amplitude of this correlation coefficient is no longer independent of the round keys since the terms are added or subtracted depending on the value of the round keys.

Correlation analysis

The analysis of a round transformation with respect to its correlation properties consists of the investigation of two aspects.

The first aspect concerns the basic entities in LC, i.e., linear trails. The round transformation can be investigated by identifying the *critical* multiple-round linear trails, i.e., with the highest correlation contribution coefficient. For block ciphers the maximum correlation contribution coefficient for linear trails that span all but a few rounds has to be investigated. An efficient round transformation combines a low work factor with critical correlation contribution coefficients that decrease rapidly when the number of rounds increases. We give a strategy for the design of this type of round transformations at the end of this chapter, called the *wide trail strategy*.

The second aspect concerns the way in which linear trails combine to multiple-round correlations. Constructive interference of many linear trails with small correlation contribution coefficients may result in large correlations. Analysis includes investigating whether the round transformation can give rise to such *clustering*. For a well-designed round transformation multiple-round correlation coefficients larger than $2^{-n/2}$ are dominated by a single linear trail.

5.5.2 Difference propagation

Fixed key

An m-round *differential trail* Ω, denoted by

$$\Omega = (\omega_0 \dashv \rho_1 \vdash \omega_1 \dashv \rho_2 \vdash \omega_2 \dashv \ldots \vdash \omega_{m-1} \dashv \rho_m \vdash \omega_m) \ , \tag{5.60}$$

consists of the chaining of difference propagations of the type $(\omega_{i-1} \dashv \rho_i \vdash \omega_i)$. These are called the (differential) *steps* of the trail. The prop ratio of Ω, denoted by $\mathrm{R_p}(\Omega)$ is the relative portion of values of a_0 that exhibit the specified differential trail.

A differential step $(\omega_{i-1} \dashv \rho_i \vdash \omega_i)$ imposes restrictions on the intermediate state a_{i-1}. If the succession of round transformations is assumed to be fixed, a_{i-1} is completely determined by a_0. Consequently, the restrictions on a_{i-1} can (in principle) be converted into restrictions on a_0. Since the round transformations are invertible, the relative size of the subset of allowed a_0 values is still given by $\mathrm{R_p}(\omega_{i-1} \dashv \rho_i \vdash \omega_i)$. The relative size of the set of values a_0 that satisfy the restrictions imposed by all the differential steps of a differential trail Ω is per definition the prop ratio of Ω.

Definition 5.5 *The restriction weight of a differential trail Ω is the sum of the restriction weights of its differential steps, i.e.,*

$$\mathrm{w_r}(\Omega) = \sum_i \mathrm{w_r}(\omega_{i-1} \dashv \rho_i \vdash \omega_i) \ . \tag{5.61}$$

Now consider a two-round differential trail. The first step imposes restrictions on a_0 and the second on a_1. Typically, these restrictions involve only a subset of the components of each of the vectors. If for every selection vector v_0 of the involved components of a_0 and every selection v_1 of the involved components of a_1 the correlation $C^{\rho_1}_{v_1 v_0} = 0$, the restrictions are said to be uncorrelated. If this is the case, imposing values upon the involved components of a_0 does not restrict the involved components of a_1 and vice versa. Hence, the two restrictions are independent and the prop ratio of the two-round differential trail is equal to the product of the prop ratios of its two differential steps. This can readily be generalized to more than two rounds.

It is generally infeasible to calculate the exact value of $\mathrm{R_p}(\Omega)$, while it is easy for the restriction weight. Under the assumption that the restrictions originating from the different steps are not (or only very weakly) correlated, the prop ratio can be approximated by

$$\mathrm{R_p}(\Omega) \approx 2^{-\mathrm{w_r}(\Omega)} \ . \tag{5.62}$$

In practice, e.g., for DES, the approximation is very good if the restriction weight is significantly below n. If $\mathrm{w_r}(\Omega)$ is of the order n or larger, (5.62) can no longer be a valid approximation. This is due to the inevitable (albeit small) correlations between the restrictions. The prop ratio multiplied by 2^n is the number of inputs a_0 that exhibit the specified differential trail, and it must therefore be an (even) integer. Of the differential trails Ω with a restriction weight $\mathrm{w_r}(\Omega)$ above n, only a fraction $2^{n-\mathrm{w_r}(\Omega)}$ can be expected to actually occur for some a_0.

Variable key

If the round transformation consists of a fixed transformation followed by the bitwise addition of the round key, the distribution of differential steps and their restriction weight is independent of the round key. Since the restriction weight of a differential trail consists of the sum of the restriction weights of its differential steps, it is independent of the cipher key.

The reduction of the restrictions imposed upon a_{i-1} by $(\omega_{i-1} \dashv \rho_i \vdash \omega_i)$ to restrictions on a_0 involves the round keys, and hence, the prop ratio of a differential trail is in principle not independent of the cipher key. Or alternatively, the signs of the correlations between the different restrictions depend on the round keys. Since for the proposed round transformations the approximation given by (5.62) is key independent, differential trails with restriction weight significantly below n have prop ratios that can be considered independent of the round keys. Differential trails Ω with restriction weights $\mathrm{w_r}(\Omega)$ above n will only actually occur for an expected portion $2^{n-\mathrm{w_r}(\Omega)}$ of the cipher keys.

DC exploits difference propagations $(\omega_0 \dashv \beta \vdash \omega_m)$ with large prop ratios. Since for a given input value a_0 exactly one differential trail is followed, the prop ratio of (a', b') is given by the sum of the prop ratios of all m-round differential trails with initial difference a' and terminal difference b', i.e.,

$$\mathrm{R_p}(a' \dashv \beta \vdash b') = \sum_{\omega_0 = a', \omega_m = b'} \mathrm{R_p}(\Omega) \ . \tag{5.63}$$

Propagation analysis

The analysis of a round transformation with respect to its difference propagation properties consists of the investigation of three aspects.

The first aspect concerns the basic entities in DC, i.e., differential trails. The round transformation can be investigated by identifying the *critical* multiple-round differential trails, i.e., with the lowest restriction weights. For block ciphers it is relevant to check the minimum restriction weight for differential trails that span all but a few rounds of the block cipher. An efficient round transformation combines a low work factor with critical restriction weights that grow rapidly as the number of rounds increases. This type of round transformations can also be designed by the *wide trail strategy*, described at the end of this chapter.

The second aspect concerns the approximation of the prop ratio of the differential trails by the product of the prop ratio of its steps. Specifically for the critical differential trails it must be checked whether the restrictions imposed by the differential steps can indeed be considered uncorrelated.

The third aspect concerns the way in which differential trails combine to difference propagations over multiple rounds. Many differential trails with high restriction weight and equal initial and terminal difference may result in difference propagation with a large prop ratio. As in the case of LC, analysis includes the investigation whether the round transformation can give rise to such *clustering*. For a well-designed round transformation multiple-round difference propagation with prop ratios larger than 2^{1-n} are dominated by a single differential trail.

5.5.3 DES cryptanalysis revisited

In this section we match the elements of linear and differential cryptanalysis as described in Sect. 5.3 with those of our framework.

Linear cryptanalysis

The multiple-round linear expressions described in [71] correspond to what we call linear trails. The probability p that such an expression holds is $\frac{1}{2}(1 + \mathrm{C_p}(\Xi))$, with

$\mathrm{C_p}(\Xi)$ the correlation contribution coefficient of the corresponding linear trail. This implies that the considered correlation coefficient is assumed to be dominated by a single linear trail. This assumption is valid because of the large amplitude of the described correlation coefficients on the one hand and the structure of the DES round transformation on the other hand.

The correlation of the linear trail is independent of the key and consists of the product of the correlations of its steps. In general, the elements of the correlation matrix of the DES round transformation are not independent of the round keys. In the described linear trails the actual independence is caused by the fact that the steps of the described linear trail only involve bits of a single S-box.

The input-output correlations of F-function of DES can be calculated by applying the rules given in Sect. 5.4.2. The 32-bit selection vector b at the output of the bit permutation P is converted into a 32-bit selection vector c at the output of the S-boxes by a simple linear transformation. The 32-bit selection vector a at the input of the (linear) expansion E gives rise to a set α of $2^{2\ell}$ 48-bit selection vectors after the expansion, with ℓ the number of neighboring S-box pairs that are addressed by a.

In the assumption that the round key is all-zero, the correlation between c and a can now be calculated by simply adding the correlations corresponding to c and all vectors in α. Since the S-boxes form a juxtaposed mapping, these correlations can be easily calculated from the correlation matrices of the individual S-boxes. For $\ell > 0$ the calculations can be greatly simplified by recursively reusing intermediate results in computing these correlations. The total number of calculations can be reduced to less than 16ℓ multiplications and additions of S-box correlations.

The effect of a nonzero round key is the multiplication of some of these correlations by -1. Hence, if $\ell > 0$, the correlation depends on the value of 2ℓ different linear combinations of round key bits. If $\ell = 0$, α only contains a single vector and the correlation is independent of the key.

Differential cryptanalysis

The characteristics with their characteristic probability described in [5] correspond to what we call differential trails and their (approximated) prop ratio. The prop ratio of a differential trail is taken to be the prop ratio of the difference propagation from its initial difference to its terminal difference. For the DC of DES this is a valid approximation because of the large prop ratios of the considered differential trails and the structure of the DES round transformation.

For the DES round transformation the distribution of the differential steps and their restriction weights are not independent of the round keys. This dependence was already recognized in [5] where in the analysis the restriction weights of the differential steps are approximated by an average value. Lars Knudsen has shown that the two-round iterative differential with approximate prop ratio 1/234 in fact has a prop ratio of either 1/146 or 1/585 depending on the value of a linear combination of round key bits [60].

Recently, Martin Hellman and Susan Langford published an attack on an 8-round variant of DES that combines the mechanisms of differential and linear cryptanalysis [50]. In their attack they apply plaintext pairs with a specific difference that propagates with prop ratio 1 to a certain difference in the intermediate state after 3 rounds confined to a subset of its bits. Then a 3-round linear trail is constructed between the output of round 7 and the input of round 4. The correlation between certain linear combinations of intermediate bits in the pair is exploited to gain information about key bits. It can easily be shown that this correlation is the square of the correlation contribution coefficient of the 3-round linear trail. The number of required plaintext-ciphertext pairs can be approximated by raising this correlation contribution coefficient to the power -4 while in simple linear cryptanalysis the required number of pairs is approximately the critical correlation contribution coefficient to the power -2. This limits the usability of this attack to ciphers with poor resistance against differential and linear cryptanalysis.

5.6 The wide trail strategy

In this section we present our strategy for the design of round transformations without low-weight multiple-round linear and differential trails.

For both types of trails, the weight is given by the sum of the weights of its steps. Let the round transformation consist of three steps: an invertible nonlinear transformation γ, an invertible linear transformation θ and the round key addition.

Suppose γ is a juxtaposed transformation. As explained on p. 79, a correlation coefficient C^{γ}_{uw} is the product of the corresponding input-output correlation coefficients of the S-boxes. With the correlation weight of the input-output correlation of an S-box equal to minus the binary logarithm of its correlation, the correlation weight of a linear step is given by the sum of the correlation weights of the corresponding input-output correlations of the S-boxes. Similarly, the restriction weight of a differential step is the sum of the restriction weights of the corresponding difference propagations of the S-boxes.

An S-box of a specific round is said to be *active* with respect to a linear trail if its output selection vector is nonzero for that linear trail. It is said to be active with respect to a differential trail if its input difference vector is nonzero for that differential trail. Now, both for linear and differential trails it can be seen that the weight of a trail is the sum of the active S-boxes.

This suggests two possible mechanisms of eliminating low-weight trails:

- Choose S-boxes with difference propagations that have high restriction weight and with input-output correlations that have high correlation weight.

- Design the round transformation in such a way that only trails with many S-boxes occur.

The wide trail strategy emphasizes the second mechanism. The round transformation must be designed in such a way that linear (or differential) steps with only few active S-boxes are followed by linear (or differential) steps with many active S-boxes. This is closely linked to the concept of *diffusion*, introduced by Shannon [99] to denote the quantitative spreading of information. The only requirement for the S-boxes themselves is that their input-output correlations have a certain minimum correlation weight and that their difference propagations have a certain minimum restriction weight.

The wide trail strategy does not restrict the nonlinear step to juxtaposed transformations. It can equally well be applied to the shift-invariant transformations that are treated in the following chapter.

5.6.1 Traditional approach

The wide trail strategy contrasts highly with the approach taken by the majority of cryptographic researchers working in cipher design. This traditional approach is dominated by the structure of DES and fully concentrates on the S-boxes. This is illustrated by the small width of the linear and differential trails in DES. Its most effective differential trail contains only 3 S-boxes per 2 rounds, its most effective linear trail only 3 S-boxes per 4 rounds.

Typically, the S-boxes are (tacitly) assumed to be located in the F-function of a Feistel structure or in some academic round transformation model such as *so-called* substitution-permutation (or transposition) networks [1, 82]. These networks consist of the alternation of parallel S-boxes and bit permutations and were proposed in [36, 59]. The S-boxes are considered to be *the* active elements in the cipher and must be designed to eliminate low-weight trails. In practice this requirement is translated to "criteria" for S-boxes, such as maximum input-output correlation, maximum prop ratio and diffusion criteria. These criteria impose conflicting restrictions, and finding S-boxes that have an acceptable score with respect to all them becomes less difficult when their size grows.

This has led many researchers to the conclusion that resistance against DC and LC is best realized by adopting large S-boxes. This one-sided point of view plainly ignores the potential of high diffusion provided by a well-designed round transformation.

5.7 Conclusions

We have given a number of tools to describe and investigate the propagation of differences and the correlations in Boolean mappings and iterated transformations. An explicit design strategy has been formulated and motivated. Our most important contributions in this chapter are:

- the concept of the correlation matrix and its properties,
- the relation between the correlations and the prop ratios of a Boolean mapping,
- the systematic treatment of the silent assumptions made in differential and linear cryptanalysis,
- the wide trail strategy.

We are convinced that the theory of correlation matrices can further be extended to provide additional insights in both correlation and difference propagation behavior. This is an opportunity for doing some further research that is both challenging and relevant.

Chapter 6

Shift-Invariant Transformations

6.1 Introduction

The defining property of shift-invariant transformations is the commutativity with translation. Shift-invariant transformations on binary vectors have a number of properties that make them suitable components for the state updating transformation of cryptographic finite state machines. In hardware, these transformations can be implemented as an interconnected array of identical 1-bit output "processors." The shift-invariance ensures that the computational load is optimally distributed. In software, their regularity allows efficient implementations by employing bitwise logical operations. Moreover, binary shift-invariant transformations can be specified by a single Boolean function.

Shift-invariant transformations are closely linked with so-called *cellular automata* since these are defined as (infinite) automata with a local shift-invariant state updating function. However, while research on cellular automata focuses on evolution of long-term structures and patterns in time, this chapter treats the short-term aspects of invertibility and local propagation and correlation properties.

We make a distinction between two types of invertibility. While *local* invertibility is in fact an inherent property of some shift-invariant transformations, *global* invertibility is context dependent. In this chapter new procedures are given for detection and proof of both types of invertibility.

The transformations naturally fall into the categories linear (with respect to bitwise addition) and nonlinear. In the nonlinear case, a distinction is made between transformations with finite and those with infinite neighborhood. The most important example of the latter corresponds to *cyclic multiplication*, i.e., multiplication by a constant modulo $2^n - 1$.

The last part of this chapter is dedicated to the study of the propagation and correlation properties of binary shift-invariant transformations with finite neighborhood. This includes the introduction of some new useful tools such as the *Hamming weight distribution table* and the *branch number* of linear mappings and our in-depth study of the correlation and propagation properties of a particularly useful nonlinear shift-invariant transformation, denoted by χ.

The propagation and correlation properties of cyclic multiplication require a different approach and are treated in Chapter 11. A substantial part of this chapter has already been published in our papers [23] and [24].

6.2 Shift-invariance and the state space

In spite of the fact that in practical implementations the arrays have to be finite, our theoretical treatment will cover the infinite case. The envisioned finite transformations are equivalent to restrictions of more general infinite transformations.

The studied transformations operate on a one-dimensional infinite array of binary-valued ($\mathbb{Z}_2$) cells with indices $i \in \mathbb{Z}$. The cells with the higher indices are assumed to be at the *right*. The (infinite) set of all possible states is denoted by $\mathcal{A}$. The ith component of a state a is denoted by a_i. More generally, a projection operator $|_\nu$, with ν a subset of $\mathbb{Z}$, extracts from a state a the $\#\nu$-dimensional vector with components indixed by the elements of ν, denoted by $a|_\nu$. If ν is a singleton, say $\{k\}$, we write $a|_k$ to denote $a|_{\{k\}}$.

A *translation* τ_r over r cells is a transformation that shifts a r positions to the right.

$$\tau_r : \mathcal{A} \rightarrow \mathcal{A} : a \mapsto b \Leftrightarrow b_{i+r} = a_i, \forall i \in \mathbb{Z} \ . \tag{6.1}$$

Shift-invariance can be expressed as

Definition 6.1 *A transformation $\phi : \mathcal{A} \rightarrow \mathcal{A}$ is shift-invariant if*

$$\forall a \in \mathcal{A}, \forall r \in \mathbb{Z} : \phi(\tau_r(a)) = \tau_r(\phi(a)) \ .$$

In the following, shift-invariant transformations are denoted by the symbol ϕ.

Two states a and b are *congruent* if there is an integer $r \in \mathbb{Z}$ such that $b = \tau_r(a)$. This is an equivalence relation and thus establishes a partition of the state space $\mathcal{A}$. The quotient space is denoted by $\mathcal{T}$.

A state is an assignment of values for an infinite array. Only a limited part of the infinite array can be realized and boundary conditions have to be specified for the transformation ϕ. This must be done in such a way that the shift-invariance of the transformation is preserved. Therefore periodic states are considered.

Definition 6.2 *The period of a state a is the smallest $p \in \mathbb{N}_0$ such that $a = \tau_p(a)$. If there is no such integer the state is said to be aperiodic.*

The subset of $\mathcal{A}$ of all states with period dividing n is denoted by $\mathcal{A}_n$. We have $a \in \mathcal{A}_n \Longleftrightarrow a = \tau_n(a)$. There are two states with period 1, the all-zero state and the all-one state. These are denoted by 0 and $\bar{0}$ respectively.

A state in $\mathcal{A}_n$ is uniquely defined by its components with indices 0 to $n-1$. The ϕ-image of a state in $\mathcal{A}_n$ must be in $\mathcal{A}_n$ since $\phi(a) = \phi(\tau_n(a)) = \tau_n(\phi(a))$. The transformation ϕ restricted to $\mathcal{A}_n$ can be implemented with the state contained in the finite array of length n. The transformation ϕ is applied to a finite vector with periodic boundary conditions.

The subset of the quotient space $\mathcal{T}$ with period p is denoted by $\mathcal{T}_p$. The congruence quotient space of $\mathcal{A}_n$ is called $\mathcal{Z}_n$ and can be expressed in terms of the $\mathcal{T}_i$ as $\mathcal{Z}_n = \bigcup_{d|n} \mathcal{T}_d$. For the calculation of the cardinalities of $\mathcal{T}_n$ and $\mathcal{Z}_n$ we refer to Appendix A.1.

If two states a and b are congruent, $\phi(a)$ and $\phi(b)$ are congruent. Therefore ϕ defines a transformation ϕ_τ over the quotient space $\mathcal{T}$.

6.3 Local maps

The kth component of the ϕ-image of a, $\phi(a)|_k$, can be expressed as a Boolean function $f(a)$. Any other component of the image can be calculated by $\phi(a)|_i = \tau_{k-i}(\phi(a))|_k = \phi(\tau_{k-i}(a))|_k = f(\tau_{k-i}(a))$. Hence, ϕ can be completely specified by the Boolean function $f_l(a)$ describing $\phi(a)|_0$.

$f_l(a)$ does not necessarily depend on all components of a. The subset of $\mathbb{Z}$ containing the indices of the components that $f_l(a)$ explicitly depends upon is called the *neighborhood* ν. We have $f_l(a) = f_l(a|_\nu)$. If ν is finite, the transformation is generally referred to as a *cellular automaton mapping*. In this case the *span* of ν is defined by $\max(\nu) - \min(\nu)$ and $f_l(a)$ is called the *local map*. Studying transformations with infinite ν is significant with respect to finite implementations.

The composition of two transformations ϕ_1 and ϕ_2 yields a new shift-invariant transformation $\phi = \phi_2 \circ \phi_1$ defined by $\phi(a) = \phi_2(\phi_1(a))$. In general, composition of shift-invariant transformations is not commutative.

The transformations ϕ with $\#\nu = 1$ are called trivial. These contain the identity transformation $1_\mathcal{A} : f_l(a) = a_0$, the translations $\tau_r : f_l(a) = a_{-r}$ and the bitwise complementation $\bar{1}_\mathcal{A} : f_l(a) = \bar{a}_0 = a_0 + 1$.

6.4 Invertibility

Definition 6.3 *A transformation ϕ is invertible if for every state $b \in \mathcal{A}$ there is exactly one state $a \in \mathcal{A}$ such that $b = \phi(a)$.*

The definition implies that for an invertible ϕ there must exist an inverse mapping ϕ^{-1} such that $\phi^{-1} \circ \phi = \phi \circ \phi^{-1} = 1_\mathcal{A}$. Moreover, this inverse is shift-invariant, since from $\tau_r \circ \phi = \phi \circ \tau_r$ follows $\phi^{-1} \circ \tau_r \circ \phi = \phi^{-1} \circ \phi \circ \tau_r$ and consequently $\phi^{-1} \circ \tau_r = \tau_r \circ \phi^{-1}$. For transformations with finite ν, D. Richardson proved the following theorem in [89]:

Theorem 6.1 *If a transformation ϕ with finite ν is invertible, then its inverse ϕ^{-1} is a shift-invariant transformation with finite ν.*

Therefore an invertible ϕ with finite ν will be called *locally* invertible. Observe that this theorem does not give an explicit method of constructing the inverse transformation.

Important classes of transformations are not invertible over $\mathcal{A}$, but they are invertible over subsets of periodic states $\mathcal{A}_n$.

Definition 6.4 *A transformation ϕ is globally invertible over $\mathcal{A}_n$, denoted by $\phi \bowtie n$ if for every state $b \in \mathcal{A}_n$ there is exactly one state $a \in \mathcal{A}_n$ such that $b = \phi(a)$.*

In this case it is possible to define the inverse mapping of ϕ restricted to $\mathcal{A}_n$. The inverse is shift-invariant but it is only defined for states in $\mathcal{A}_n$. Theorem 6.1 does not apply in this case.

Proposition 6.1 *if $\phi \bowtie n$, ϕ preserves the period for all states with a period d that divides n.*

Proof : Let a be a state with period $d|n$. It follows from $a \in \mathcal{A}_d$ that $\phi(a) \in \mathcal{A}_d$ or, equivalently, that the period of $\phi(a)$ must divide d. Suppose $\phi(a)$ has period $p < d$. We have $\phi(\tau_p(a)) = \tau_p(\phi(a)) = \phi(a)$. The consequence that two different states in $\mathcal{A}_n$, $\tau_p(a)$ and a, have the same ϕ-image contradicts the definition of global invertibility. □

If $\phi \bowtie n$, it follows from Prop. 6.1 that $\phi \bowtie d$ for all $d : d|n$. Conversely $\phi \not\bowtie n \Rightarrow \phi \not\bowtie k$ if k is a multiple of n. Therefore, the invertibility properties of a mapping with respect to the period can be expressed in the form of a set ξ of integers. We have

$$\phi \bowtie n \Longleftrightarrow m_i \not| n, \forall m_i \in \xi \ . \tag{6.2}$$

From one invertible shift-invariant transformation, many other invertible shift-invariant transformations can be derived by simply composing the original one with translation, complementation or both. Obviously, these transformations are not "essentially different." In order to reveal other obvious relations in the set of all possible ϕ, some more transformations are introduced.

The *reflection* ρ is the transformation that maps a state to its mirror-image around 0:

$$\rho : \mathcal{A} \to \mathcal{A} : a \mapsto b \Leftrightarrow b_i = a_{-i}, \forall i \in \mathbb{Z} \ . \tag{6.3}$$

It can easily be seen that $\rho \circ \phi \circ \rho$, the *reflection* of ϕ, is a shift-invariant transformation.

A *decimation* Δ with base $e \in \mathbb{N}_0$ is a function that maps one state to a vector of e states:

$$\Delta_e : \mathcal{A} \to \mathcal{A}^e : a \mapsto (b^0, b^1 \ldots, b^{e-1}) \Leftrightarrow b^j_i = a_{ei+j}, \tag{6.4}$$

$\forall i \in \mathbb{Z}, 0 \le j < e$. The inverse of the Δ_e is the *interweaving* of e states:

$$\Delta_e^{-1} : \mathcal{A}^e \to \mathcal{A} : (a^0, a^1 \ldots, a^{e-1}) \mapsto b \Leftrightarrow b_{ei+j} = a^j_i, \ . \tag{6.5}$$

$\forall i \in \mathbb{Z}, 0 \le j < e$. Let ϕ_e be defined by

$$\phi_e : \mathcal{A}^e \to \mathcal{A}^e : a \mapsto b \Leftrightarrow b^j = \phi(a^j) 0 \le j < e \ . \tag{6.6}$$

It can be seen that $\Delta_e^{-1} \circ \phi_e \circ \Delta_e$, the *expansion of ϕ by e*, is a shift-invariant transformation.

With respect to propagation properties, these cannot be considered essentially different from ϕ. The set ξ associated with two transformations that are not essentially different are equal, except for expansions of ϕ. If $\phi' = \Delta_e^{-1} \circ \phi \circ \Delta_e$ it can be derived that

$$\forall m_i \in \xi_\phi \Rightarrow m_i \cdot \gcd(m_i, e) \in \xi_{\phi'} \ . \tag{6.7}$$

6.5 Linear shift-invariant transformations

The local map $f_l(a|_\nu)$ must be a linear Boolean function of $a|_\nu$, i.e., a sum modulo 2 of the components with indices in ν. Observe that ϕ is completely specified by the neighborhood ν.

A linear transformation ϕ can be described by a polynomial multiplication with coefficients in $\mathbb{Z}_2$. A state a can be represented by a double infinite power series

$$\alpha(x) = \sum_{-\infty}^{+\infty} a_i x^i \ .$$

A linear transformation ϕ specified by a neighborhood ν is represented by a characteristic dipolynomial $c(x) = \sum_{i \in \nu} x^{-i}$. The power series representation of $\phi(a)$ is given by $c(x)\alpha(x)$.

A state a with period n is represented by $\alpha(x) = a(x) \sum_{j=-\infty}^{+\infty} x^{j \cdot n}$ with $a(x) = \sum_{i=0}^{n-1} a_i x^i$ completely specifying a. For a state a with period n it can be seen that $(1 + x^n)\alpha(x) = 0$ since this is equivalent to $a + \tau_n(a) = a + a = 0$. We have

$$c(x)\alpha(x) = c(x)a(x) \sum_{j=-\infty}^{+\infty} x^{j \cdot n} = c(x)a(x) \bmod (1 + x^n) \sum_{j=-\infty}^{+\infty} x^{j \cdot n} \ .$$

The linear transformation ϕ restricted to $\mathcal{A}_n$ can therefore be described as a multiplication by a polynomial $c(x)$ in the commutative ring of binary polynomials modulo $1 + x^n$. If $b = \phi(a)$ we have

$$b(x) = c(x)a(x) \bmod (1 + x^n) \ . \tag{6.8}$$

6.5.1 Invertibility

Obviously, the inverse of an invertible linear transformation is linear. From this it can be derived that there are only trivial locally invertible linear shift-invariant transformations. Consider an invertible linear ϕ specified by $c(x)$. Its inverse can be specified by a dipolynomial $d(x)$ since it must be linear and shift-invariant. This polynomial must satisfy $c(x)d(x)\alpha(x) = \alpha(x)$ for all possible $\alpha(x)$ or $c(x)d(x) = 1$. This is only possible if both $c(x)$ and $d(x)$ have only one term, i.e., if ϕ is a translation.

On the other hand, finding linear shift-invariant transformations that are globally invertible over some $\mathcal{A}_n$ is easy. In the following, only dipolynomials without negative powers will be considered since every linear ϕ can be decomposed in a translation and a linear ϕ' that can be represented by a polynomial.

Proposition 6.2 *For ϕ defined by $c(x)$: $\phi \bowtie n \iff \gcd(c(x), 1+x^n) = 1$. The inverse ϕ^{-1} can be described as multiplication modulo $1+x^n$ by a polynomial $d(x)$ with $d(x)c(x) \equiv 1 \pmod{1+x^n}$.*

This proposition follows directly from (6.8) . The inverse polynomial $d(x)$ can easily be computed with the (extended) Euclidean algorithm.

Example 6.1 *Given $c(x) = 1+x+x^3$. This c(x) is invertible over $\mathcal{A}_8$ because $\gcd(c(x), 1+x^8) = 1$. Its inverse is given by $d(x) = 1+x^2+x^5+x^6+x^7$. It is not invertible over $\mathcal{A}_7$ because $c(x)$ divides $1+x^7$.*

Every binary polynomial can be factored in a unique product of powers of distinct irreducible (or equivalently prime) binary polynomials. The invertibility properties of $c(x)$ depend on the algebraic order [65] of its irreducible factors. The algebraic order of a polynomial $c(x)$ is the smallest number $m \in \mathbb{N}_0$ such that $x^m = 1 \bmod c(x)$.

If $c(x)$ is irreducible we have:

Lemma 6.1 *An irreducible $c(x)$ of algebraic order m is invertible modulo $1+x^n$ iff $m \not| n$. We have $\xi = \{m\}$.*

Proof : We have $x^{km} \equiv 1 \pmod{c(x)}$ for any integer k or, equivalently, $\gcd(c(x), 1+x^{km}) = c(x)$. Therefore, $c(x)$ is not invertible modulo $1+x^{km}$. On the other hand, for n no multiple of m we have $x^n \not\equiv 1 \pmod{c(x)}$ or equivalently $\gcd(c(x), 1+x^n) \neq c(x)$. For $c(x)$ irreducible we must have $\gcd(c(x), 1+x^n) = 1$, hence, $c(x)$ is invertible modulo $1+x^n$. □

Example 6.2 $c(x) = 1+x+x^2$ *: this is an irreducible polynomial of order $m = 3$. Hence, $c(x)$ is invertible for all n that are no multiple of 3.*

If $c(x)$ is composite then $c(x)$ is invertible for a given length n if all of its factors are invertible for that length. The transformation with polynomial $c(x)$ can be seen as the composition of the transformations represented by the factors of $c(x)$. It immediately follows :

Theorem 6.2 *Let the factorization of $c(x)$ in irreducible factors be given by*

$$p_1(x)^{\alpha_1} p_2(x)^{\alpha_2} \dots p_g(x)^{\alpha_g} \ ,$$

and the algebraic orders of the factors $p_i(x)$ denoted by m_i. Then

$$\xi = \{m_1, m_2 \dots, m_g\} \ .$$

Example 6.3 $c(x) = 1+x+x^5$ *factors in $(1+x+x^2)(1+x^2+x^3)$. These polynomials have respectively orders $m = 3$ and $m = 7$, hence, $c(x)$ is invertible for all lengths that are no multiple of 3 or 7.*

Corollary 6.1 *If $c(x)$ contains an even number of terms there is no length n for which ϕ is invertible over $\mathcal{A}_n$. If $c(x)$ contains an odd number of terms there are always lengths n for which it is invertible.*

If $c(x)$ contains an even number of terms then $c(1) = 0$, i.e., 1 is a root of $c(x)$ or equivalently $(1 + x)|c(x)$. This means that $\xi = \{1\}$. If $c(x)$ contains an odd number of terms, $1 + x$ cannot be a factor of $c(x)$. An example of a length n for which such a $c(x)$ is invertible is $n = 1 + \prod_{m_i \in \xi} m_i$. It can easily be seen that this number is coprime to all m_i.

6.6 Nonlinear transformations with finite ν

Transformations in this class are always specified in terms of a local map. A particularly useful description of this local map when invertible ϕ's are envisioned is the *complementing landscape* (CL) specification [102]. Here the local map is specified by a set of patterns, called the *complementing landscapes* (CL). The value of a component is complemented if its neighborhood takes on one of these patterns. A landscape is a pattern consisting of symbols 1, 0, and - denoting "don't care," positioned relative to an origin, denoted by $*$. In this context, the all-zero state will be denoted by 0^* and the all-one state by 1^*.

Example 6.4 *Say ϕ is specified by $\{1*\text{-}01\}$ and $b = \phi(a)$. We have:*
$b_i = \bar{a}_i$ *if* $a_{i-1} = 1$ *&* $a_{i+2} = 0$ *&* $a_{i+3} = 1$,
$b_i = a_i$ *otherwise.*
The value of a_{i+1} is irrelevant in this calculation. We say a_i is complemented if component i is in landscape $1*\text{-}01$.

The implementation of a transformation in the form of a two layer NAND circuit followed by an EXOR is given by its complementing landscape specification. In this respect the CL representation gives a good idea of the implementation complexity of the transformation.

Definition 6.5 *Two landscapes ℓ_1 and ℓ_2 are called compatible if a component can be in (the origin of) both landscapes at the same time.*

Example 6.5 $\ell_1 = 1*\text{-}01$, $\ell_2 = *111$, $\ell_3 = *1\text{-}1$. *Here ℓ_1 and ℓ_2 are incompatible, ℓ_3 is compatible with both ℓ_1 and ℓ_2.*

6.6.1 Local invertibility

The only locally invertible binary ϕ's that have been described in the literature, are of the (single) conserved-landscape type [102]. Here ϕ is defined by a landscape ℓ with the property that any component partaking it cannot be in the origin of ℓ. Every cell in the landscape is left unchanged by the transformation and therefore the whole landscape is preserved. This will be illustrated in the following example by the only locally invertible ϕ with span 4.

Example 6.6 *ϕ is defined by ℓ =* `0*10`*. If component* 0 *is in landscape* `0*10`*, components* −1*,*1 *and* 2 *must be in landscapes incompatible with ℓ.*

- *Component* −1 *is in landscape* `*-10`*,*
- *Component* 1 *is in landscape* `0-*0`*,*
- *Component* 2 *is in landscape* `0-1*`*.*

It can be seen that if the neighborhood of component 0 *takes on the landscape, this will still be the case after applying ϕ.*

It follows that the inverse of an invertible ϕ of the conserved-landscape type is ϕ itself, i.e., ϕ is an involution. This can be useful in applications where ϕ and ϕ^{-1} must both be implemented. If a landscape is given, it can easily be checked whether it is a conserved landscape.

We discovered that there are also invertible ϕ's that are defined by a *set* of several landscapes that is preserved by the transformation. A component in a landscape (not) belonging to this set, will (not) be in a landscape belonging to this set after application of ϕ. The simplest example is given by the set {`0*110`, `10*10`}.

It can easily be checked whether a given landscape (or set of landscapes) is conserved. Numerous locally invertible ϕ's can be found by searching in the space of possible landscapes. The search can be optimized by taking into account that conserved landscapes must have certain "overlap" properties. For instance, there are no conserved landscapes where the origin is in the rightmost or leftmost position.

In our search for invertible shift-invariant transformations, we found locally invertible ϕ's of which the invertibility *cannot* be explained by the conserved-landscape principle. From Theorem 6.1 it follows that the inverse of these ϕ's can be expressed as a local map. The invertibility can be proved by giving an algorithm for the calculation of $\phi^{-1}(a)$ for all $a \in \mathcal{A}$. The algorithms to calculate the inverse of invertible ϕ's that are not involutions consist basically of two operations, denoted by *seed* and *leap*:

1. **Seed :** For all $a \in \mathcal{A}$, one component or a certain pattern of components of $\phi^{-1}(a)$ can be found.
2. **Leap :** Starting from a seed, other components of $\phi^{-1}(a)$ can be reconstructed.

This can best be illustrated by an example. We prove the invertibility of a class of invertible ϕ's that are not of the conserved-landscape type.

Example 6.7 *ϕ is specified by* `0`◁*i*▷`*1`◁*i*▷`0` *. Here* ◁*i*▷ *denotes the symbol* `-` *repeated i times.*
Seed: *The two* 0 *symbols of this landscape are preserved. Therefore, if a_i is in landscape* `1`◁*i*▷`*` *or in* `*-`◁*i*▷`1` *it yields $\phi^{-1}(a)|_i = a_i$. The only state without seeds is 0^*, but it can easily be seen that $\phi^{-1}(0^*) = 0^*$.*
Leap: *If $\phi^{-1}(a)|_i$ is known, $\phi^{-1}(a)|_{i-1}$ can be found. This can be expressed in a leap tree. This is a binary tree that summarizes all combinations of relevant*

components of state "a" near the seed. The diagrams in the nodes consist of two lines. The bottom line denotes a substring of state "a" while the top line denotes the corresponding part of $\phi^{-1}(a)$. In the initial node (at the left) a seed is depicted with respect to an origin $*$. The component in the origin will be determined by the leap. For a node there are two possibilities. In one case, the component in the origin can be determined from the knowledge of components in its neighborhood. This is denoted by a symbol ℓ or $\bar{\ell}$ in the origin. In the other case, more knowledge is needed from the neighborhood. If there is a component in the neighborhood that can be determined for the given status of knowledge of "a" in that node, this is denoted by x. The two cases $x = 0$ and $x = 1$ give rise to two new nodes. A leap tree has the property that a leap is realized in all of its terminal nodes.

$$\begin{tikzcd}
\frac{*x}{x} \arrow[r] \arrow[dr] & \frac{\ell 0}{\ell} & \frac{x \triangleleft i \triangleright *1 \triangleleft i \triangleright 0}{x} \arrow[r] \arrow[dr] & \frac{0 \triangleleft i \triangleright \bar{\ell}1 \triangleleft i \triangleright 0}{\ell} \\
 & \frac{*1 \triangleleft i \triangleright x}{x} \arrow[ur] \arrow[r] & \frac{\ell 1 \triangleleft i \triangleright 1}{\ell} & \frac{1 \triangleleft i \triangleright \ell 1 \triangleleft i \triangleright 0}{\ell}
\end{tikzcd}$$

A remarkable set of invertible transformations contains the ϕ's specified by the sets $\Gamma_m : \{0 \triangleleft i \triangleright \ \triangleleft i \triangleright *10 \mid 0 \leq i < m\}$. For more examples of locally invertible shift-invariant transformations we refer to Appendix A.3.

6.6.2 Global invertibility

Consider the shift-invariant transformation defined by $\ell = *01$. Because of its practical importance in our work we give this transformation a name: χ. A seed and a leap can easily be found:

1. **Seed:** If a_i is in $*1$, $\phi^{-1}(a)|_i = a_i$.

2. **Leap:** If $\phi^{-1}(a)|_i$ is known, $\phi^{-1}(a)|_{i-2}$ can be found.

$$\begin{tikzcd}
\frac{*\text{-}x}{x} \arrow[r] \arrow[dr] & \frac{\ell\text{-}0}{\ell} & \frac{\bar{\ell}01}{\ell} \\
 & \frac{*x1}{x} \arrow[ur] \arrow[r] & \frac{\ell 11}{\ell}
\end{tikzcd}$$

In contrast with the leaps found in the previous section, the leap for χ gives a new value that has a distance of 2 cells to the seed. In general, two seeds are needed for the full reconstruction of the state, one for the components with even indices and one for the components with odd indices. This has important implications for periodic states. If the period is odd, the presence of a seed on an even indexed position necessarily implies that there must be a seed on an odd indexed position. Therefore $\chi^{-1}(a)$ can be determined for all states with odd period. By a simple counting argument, there can be no state a different from 0^* with odd period such that $0^* = \chi(a)$. Hence, we have

Proposition 6.3 *χ is invertible for $\mathcal{A}_n$ with n odd or, equivalently, for χ we have $\xi = \{2\}$.*

If n is even, there are states without seeds for the even (odd) indexed components. Of the 2^n states in $\mathcal{A}_n$, $2^{n/2}-2$ have two preimages, $2^{n/2}$ have no preimages and the all-zero state 0^* has three preimages: the two states with period 2 and 0^* itself.

For larger neighborhoods the invertibility can sometimes be proved for a set of transformations at once. For instance, all ϕ's that are specified by one of the 16 subsets of $\{*0001, *0101, *0111, *01\text{-}0\}$ are invertible for odd state lengths. For the proof, we refer to Appendix A.2.

Globally invertible ϕ's can be found by checking landscapes and sets of landscapes for "seeds" and "leaps." For larger neighborhoods, the number of globally invertible ϕ's grows exponentially. The sets ξ of the transformations depend on the "length" of the leaps. The simplest invertible ϕ's are listed in Appendix A.3.

Unlike with locally invertible ϕ, the inverse of globally invertible ϕ's cannot be calculated locally. It can therefore not be implemented in a straightforward parallel manner. The algorithm for computing the inverse based on the seed and leap method is inherently sequential.

6.7 Cyclic multiplication

A state $a \in \mathcal{A}$ that has no nonzero components above a given index λ can be interpreted as the representation of a positive real number, i.e., $a_\lambda 2^\lambda + a_{\lambda-1}2^{\lambda-1}\dots$ A translation τ_r is interpreted as a multiplication by 2^r. Multiplication by a constant $c \in \mathbb{R}$ is shift-invariant because multiplication in $\mathbb{R}$ is commutative: $c{\cdot}2^r{\cdot}a = 2^r{\cdot}c{\cdot}a$.

If this constant is an integer (possibly divided by a power of 2), its binary representation contains a finite number of 1's. Although a periodic state does not represent a real number, the "multiplication" by an integer acting upon periodic states, or cyclic multiplication, can be given a simple interpretation and implementation. A state with period n can be represented by $a = A \cdot \sum_{i=-\infty}^{\infty} 2^{ni}$ with $A \in \mathbb{Z}_{2^n}$. If a state with period n is multiplied by 2^n-1, the result must be the state representing the number 0 since $(2^n-1)a = \tau_n(a) - a = a - a = 0$. For $a \neq \bar{0}$ we have

$$C \cdot a = C \cdot A \sum_{i=-\infty}^{\infty} 2^{ni} = (C \cdot A \bmod (2^n-1)) \sum_{i=-\infty}^{\infty} 2^{ni} \ .$$

Hence, multiplication by a constant C restricted to $\mathcal{A}_n$ can be modeled by multiplication modulo 2^n-1. If $b = \phi(a)$,

$$B = C \cdot A \bmod (2^n-1) \ . \tag{6.9}$$

The all-1 state $\bar{0}$ corresponds to the number 2^n-1 and can therefore be seen as an alternative representation of the number 0. Since $\phi(0) = 0$ for all multiplication factors, invertibility imposes that $\phi(\bar{0}) = \bar{0}$.

ϕ defined by a multiplication factor C is invertible if $\gcd(C, 2^n-1) = 1$. The inverse can be described by the multiplication modulo 2^n-1 with a constant D

with $CD \equiv 1 \pmod{2^n - 1}$. D can be computed with the (extended) algorithm of Euclid. Since $\gcd(2^p - 1, 2^q - 1) = 2^{\gcd(p,q)} - 1$ [61], the invertibility of these transformations can again be represented by a set ξ.

6.8 Diffusion

In the wide trail strategy, the resistance against LC and DC is achieved by the iterated alternation of a nonlinear transformation and a transformation with high diffusion. In this section we will study the diffusion properties of shift-invariant transformations. This will be followed by a treatment of the nonlinearity properties of shift-invariant transformations.

Definition 6.6 *The Hamming distance between two binary vectors is the number of nonzero components of their bitwise difference (modulo 2).*

The Hamming distance to 0 is denoted by *Hamming weight*:

Definition 6.7 *The Hamming weight* $\mathrm{w_h}$ *is the number of nonzero components of a vector.*

The *diffusion factor* describes the amount of *local* diffusion that is realized by a transformation:

Definition 6.8 *The diffusion factor $\mathcal{D}$ of a Boolean transformation h is the average Hamming distance between $h(a)$ and $h(a + \delta_i)$ over all a and i.*

In this definition δ_i denotes the binary vector with a single nonzero component at position i.

For a shift-invariant transformation $\mathcal{D}(\phi)$ can be calculated directly from the local map $f_l(a|_\nu)$. If $\#\nu = k$ we have

$$\mathcal{D}(\phi) = 2^{-k} \sum_{i \in \nu} (\sum_x \delta(f_l(x) + f_l(x + \delta_i)) \ . \tag{6.10}$$

With some effort it can be shown that the following definition is equivalent:

Definition 6.9 *The diffusion factor $\mathcal{D}(\phi)$ is equal to*

$$\sum_w \mathrm{C}^2(\phi(a)|_0, w^{\mathrm{t}} a)\, \mathrm{w_h}(w) = \sum_w \hat{F}_l(w)^2 \mathrm{w_h}(w) \ .$$

For a linear ϕ the diffusion factor is equal to the number of terms in the characteristic polynomial. In Appendix A.3 it can be seen that nonlinear invertible ϕ's have in general a smaller $\mathcal{D}$ than even the simplest linear invertible ϕ. For larger neighborhoods $\mathcal{D}$ even decreases. Invertible nonlinear ϕ's seem to be inefficient information *diffusers*.

On the one hand, the diffusion factor describes the *average* amount of information propagation resulting from isolated single-component input differences. On the other hand, it describes a kind of *average* effective neighborhood size. Cryptanalysis often takes advantage of the worst case behavior of round transformations. In nonlinear transformations the amount of propagation can depend strongly on the absolute values of the components in the neighborhood of the complemented component. In almost all ϕ's listed in Appendix A.3 these absolute values can be chosen in such a way that changing a single input component only affects a single output component. Alternatively, for all these ϕ's there is a large correlation between a_i and $\phi(a)|_i$. This means that in the worst case these nonlinear transformations do not accomplish any diffusion at all.

For linear ϕ's, a more detailed picture of the diffusion can be given by also looking at linear combinations of output bits or differences in more than a single input bit at once.

6.8.1 Diffusion in invertible linear ϕ

For linear ϕ, the difference propagation can be expressed in terms of the multiplication polynomial:

$$b'(x) = c(x)a'(x) \bmod (1 + x^n) \ . \tag{6.11}$$

If the Hamming weight of $a'(x)$ is equal to 1, $b'(x)$ is just a shifted version of $c(x)$ and its Hamming weight is equal to the diffusion factor. If the Hamming weight of $a'(x)$ is higher than 1, the Hamming weight of $b'(x)$ depends on the actual $a'(x)$ at hand.

The correlation can likewise be expressed in terms of the multiplication polynomial. A selection vector u corresponds to a polynomial $u(x)$ and the linear combination $u^{\mathrm{t}}b$ corresponds to the coefficient of x^0 in the polynomial given by $u(x^{-1})b(x) \bmod (1 + x^n)$. Using $b(x) = c(x)a(x) \bmod (1 + x^n)$ it can be derived that $C_{uw} = 1$ for

$$w(x) = c(x^{-1})u(x) \bmod (1 + x^n) \ , \tag{6.12}$$

and 0 otherwise.

The diffusion effect of a linear mapping θ can be studied by analyzing pairs $(a, \theta(a))$. A useful tool in this context is the *Hamming weight distribution table*. This is a table that partitions the couples $(a, \theta(a))$ according to the Hamming weights of a and $\theta(a)$. Table 6.1 is an example of such a Hamming weight distribution table. The element in row i and column j denotes the number of couples $(a, \theta(a))$ with $\mathrm{w_h}(a) = i$ and $\mathrm{w_h}(\theta(a)) = j$.

The single nonzero entry in row 0 (and column 0) corresponds to the couple $(0, 0)$. The single nonzero entry in row 11 (and column 11) corresponds to the couple $(\bar{0}, \bar{0})$. For invertible linear shift-invariant mappings these rows and columns clearly do not contain any information. Therefore, they will be omitted in the following.

In Table 6.1 it can be seen that there is a nonzero entry in row 2 and column 2. This indicates output differences with only two nonzero bits that correspond to input differences with only two nonzero bits, or alternatively, linear equations involving only two input components and two output components. The minimum

	0	1	2	3	4	5	6	7	8	9	10	11
0	1	-	-	-	-	-	-	-	-	-	-	-
1	-	-	-	-	-	11	-	-	-	-	-	-
2	-	-	11	-	11	-	11	-	11	-	11	-
3	-	-	-	-	-	110	-	55	-	-	-	-
4	-	-	44	-	99	-	110	-	77	-	-	-
5	-	-	-	77	-	220	-	165	-	-	-	-
6	-	-	-	-	165	-	220	-	77	-	-	-
7	-	-	-	77	-	110	-	99	-	44	-	-
8	-	-	-	-	55	-	110	-	-	-	-	-
9	-	11	-	11	-	11	-	11	-	11	-	-
10	-	-	-	-	-	-	11	-	-	-	-	-
11	-	-	-	-	-	-	-	-	-	-	-	1

Table 6.1: Hamming weight distribution table of multiplication by $1+x+x^2+x^6+x^7$ modulo $1+x^{11}$.

total Hamming weight of $(a, \theta(a))$ is a measure for the minimum amount of diffusion that is realized by a linear mapping, and is called the *branch number*.

Definition 6.10 *The branch number $\mathcal{B}$ of a linear mapping θ is given by*

$$\mathcal{B}(\theta) = \min_{a \neq 0}(\mathrm{w_h}(a) + \mathrm{w_h}(\theta(a))) \ . \tag{6.13}$$

For modular multiplication by a polynomial $c(x)$ the couple $(1, c(x))$ imposes an upper bound of $\mathcal{D}+1$ on the branch number. Table 6.2 gives an example of a polynomial multiplication with $\mathcal{B} = \mathcal{D}+1$.

Let $\mathcal{B}(c(x), n)$ denote the branch number corresponding to multiplication by $c(x)$ modulo $1+x^n$. The knowledge of $\mathcal{B}(c(x), n)$ allows us to determine the branch numbers of a large set of related polynomials.

	1	2	3	4	5	6	7	8	9	10
1	-	-	-	-	11	-	-	-	-	-
2	-	-	-	-	-	55	-	-	-	-
3	-	-	55	-	-	-	110	-	-	-
4	-	-	-	220	-	-	-	110	-	-
5	11	-	-	-	369	-	-	-	55	-
6	-	55	-	-	-	369	-	-	-	11
7	-	-	110	-	-	-	220	-	-	-
8	-	-	-	110	-	-	-	55	-	-
9	-	-	-	-	55	-	-	-	-	-
10	-	-	-	-	-	11	-	-	-	-

Table 6.2: Hamming weight distribution table of multiplication by $1+x+x^2+x^4+x^7$ modulo $1+x^{11}$.

We have

- $\forall i : \mathcal{B}(c(x), n) = \mathcal{B}(x^i c(x), n)$: the branch numbers of two congruent linear transformations are equal;
- $\forall i$ with $\gcd(i, n) = 1 : \mathcal{B}(c(x), n) = \mathcal{B}(c(x^i), n)$: the branch numbers of two linear transformations that can be converted into each other by expansion are equal;
- $\mathcal{B}(c(x), n) = \mathcal{B}(c(x)^{-1} \bmod (1 + x^n), n)$: the branch number of a linear transformation and its inverse are equal.

Moreover, the branch number corresponding to an invertible polynomial modulo $1+x^n$ must be even. This is due to the fact that multiplication modulo $1+x^n$ by a polynomial with an odd number of terms preserves the parity.

For a given n an upper bound on the branch number of *any* linear mapping θ over $\mathbb{Z}_2^n$ is given by the *sphere-packing bound* for error-correcting codes [67, p. 19]. Consider a $(2n, n)$ linear code where the 2^n codewords are given by the couples $(a, \theta(a))$. The branch number is the smallest Hamming weight of the nonzero codewords. Since this is a linear code, $\mathcal{B}$ is also the minimum Hamming distance between the codewords. Hence, the sets of *neighbors* at a Hamming distance of less than $\frac{\mathcal{B}}{2}$ bits from each codeword must be disjoint. We have

$$\sum_{0 \le i < \frac{\mathcal{B}}{2}} \binom{2n}{i} \le 2^n \ , \tag{6.14}$$

with $\binom{y}{x}$ denoting the number of possible combinations of x elements from a set of y elements.

This equation imposes a lower bound upon the dimension n for a linear transformation if some branch number has to be attained. The minimum dimensions for odd branch numbers smaller than 20 are listed in Table 6.3. For an even branch number $2k$, (6.14) imposes the same minimum dimension as for $2k-1$.

Branch number	3	5	7	9	11	13	15	17	19
Minimum dimension	3	7	11	15	20	24	29	33	37

Table 6.3: Minimum dimension n versus branch numbers.

6.9 Nonlinearity properties of χ

The largest occurring prop ratio is a measure for the minimum amount of nonlinear difference propagation realized by ϕ and is denoted by $\mathcal{R}$. The largest occurring correlation in the correlation matrix of ϕ is a measure for the minimum amount of *decorrelation effect* through ϕ and is denoted by $\mathcal{C}$.

In Appendix A.3 it can be seen that, despite its simple local map, the shift-invariant mapping denoted by χ has very good nonlinear properties. Other ϕ's with comparable $\mathcal{C}$ and $\mathcal{R}$ have much larger implementation complexities. Therefore, the nonlinear shift-invariant transformations that are used in our own designs are variants of χ. In this section the nonlinear propagation properties of χ are treated in detail.

The local map of $b = \chi(a)$ is given by

$$b_i = a_i + (a_{i+1} + 1)a_{i+2} \ . \tag{6.15}$$

It can be seen that the local map is quadratic, i.e., has algebraic degree two. This has interesting consequences for the nonlinear behavior of χ.

6.9.1 Difference propagation

The components of $b' = \chi(a) + \chi(a + a')$ are given by

$$b'_i = \chi(a')|_i + a'_{i+1}a_{i+2} + a'_{i+2}a_{i+1} \ . \tag{6.16}$$

For a given input difference the bits a'_i are fixed and the bits a_i are variables. It can be seen that b'_i depends *in a linear way* on the components of a. All possible difference propagations ($a' \dashv \chi \vdash b'$) have the same prop ratio. The space of all b' that are compatible with a given a' can be described as an affine variety:

$$b' \in \{x = \chi(a') + \sum_k a_k u^k \mid \forall k \in \mathbb{Z}_n, a \in \mathbb{Z}_2^n\} \ ,$$

with the generating vectors u^k specified by $u^k = a'_{k+1}\delta_{k-1} + a'_{k-1}\delta_{k-2}$. In matrix notation we have

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ \vdots \\ b'_{n-2} \\ b'_{n-1} \end{bmatrix} = [\chi(a')] + \begin{bmatrix} 0 & a'_2 & a'_1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & a'_3 & a'_2 & \cdots & 0 & 0 \\ 0 & 0 & 0 & a'_4 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a'_{n-1} & 0 & 0 & 0 & \cdots & 0 & a'_0 \\ a'_1 & a'_0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ \vdots \\ a_{n-2} \\ a_{n-1} \end{bmatrix}$$

The total number of b' vectors compatible with a' depends on the dimension of the affine variety, i.e., the number of independent vectors in $\{u^k\}$. This dimension is equal to the restriction weight of $(a' \dashv \chi \vdash b')$. Since it is independent of b' this restriction weight can be denoted by $\mathrm{w_r}(a')$. The number of b' values compatible with a' is equal to $2^{\mathrm{w_r}(a')}$. A given difference propagation $(a' \dashv \chi \vdash b')$ imposes exactly $\mathrm{w_r}(a')$ linear Boolean conditions on the bits of a. It can easily be shown that for $a' \neq \bar{0}$, the restriction weight $\mathrm{w_r}(a')$ is equal to the Hamming weight of a' plus the number of 001-patterns in a'. For $a' = \bar{0}$ and dimension n, we have $\mathrm{w_r}(\bar{0}) = n - 1$.

6.9.2 Correlation

All input selections w with $\mathrm{C}(u^{\mathrm{t}}\chi(a), w^{\mathrm{t}}a) \neq 0$ are called *(χ-)compatible* with u. We have

$$\begin{aligned} u^{\mathrm{t}}\chi(a) &= \sum_i u_i(a_i + (a_{i+1}+1)a_{i+2}) \qquad (6.17) \\ &= \sum_i u_i(a_i + a_{i+2}) + \sum_i u_i a_{i+1} a_{i+2} \ . \end{aligned}$$

The right-hand side of (6.18) is split up into a sum of linear terms and a sum of quadratic product terms. We will first investigate the effect of adding the product terms. As shown on p. 77, the effect of adding the linear sum corresponds to a dyadic shift in the transform domain. For the quadratic product terms we distinguish four cases:

- $a_0a_1 + a_1a_2 = (a_0 + a_2)a_1$: addition of two adjacent products can be reduced to a single product,
- a_0a_1 and a_2a_3 are disjunct (see p. 77),
- $(a_0 + a_2)a_1$ and a_2a_3 are disjunct,
- $(a_0 + a_2)a_1$ and $(a_2 + a_4)a_3$ are disjunct.

The product terms corresponding to components of u that are not adjacent are disjunct. Therefore, the quadratic functions obtained by grouping the product terms corresponding to the all-one substrings of u separated by zeros are disjunct. Such a quadratic function is of the form $\sum_{0<i\leq s} a_i a_{i+1}$. This expression can be converted into a sum of disjunct products. The reduction depends on the parity of the length s. If s is even, we have

$$\sum_{0<i\leq s/2} (a_{2i-1} + a_{2i+1})a_{2i} \ , \qquad (6.18)$$

and if s is odd

$$\sum_{0<i\leq (s-1)/2} (a_{2i-1} + a_{2i+1})a_{2i} + a_s a_{s+1} \ . \qquad (6.19)$$

Hence, the number of disjunct product terms corresponding to an all-1 substring of length s is $\lfloor \frac{s+1}{2} \rfloor$. This gives us a general procedure to split (6.18) up into a single linear term and a number of quadratic product terms that are mutually disjunct. Let the result of this reduction be denoted by

$$u^{\mathrm{t}}\chi(a) = z^{\mathrm{t}}a + \sum_{0<i\leq r} (v^{2i-1^{\mathrm{t}}}a)(v^{2i^{\mathrm{t}}}a) \ . \tag{6.20}$$

The vectors v^i correspond to a basis with respect to which the quadratic Boolean function $u^{\mathrm{t}}\chi(a)$ has the normal form according to Dickson's theorem [84, p. 240].

From (5.17) it can be seen that the product of two linear terms $(\mathrm{u}^{\mathrm{t}}a)(\mathrm{v}^{\mathrm{t}}a)$ is correlated to the linear functions $0, \mathrm{u}^{\mathrm{t}}a$ and $\mathrm{v}^{\mathrm{t}}a$ with correlation coefficient $1/2$, and to the linear function $(\mathrm{u}+\mathrm{v})^{\mathrm{t}}a$ with correlation coefficient $-1/2$. Its Walsh-Hadamard transform is described by

$$\mathcal{W}((\mathrm{u}^{\mathrm{t}}a)(\mathrm{v}^{\mathrm{t}}a)) = \frac{1}{2}(\delta(w) + \delta(w+\mathrm{u}) + \delta(w+\mathrm{v}) - \delta(w+\mathrm{u}+\mathrm{v})) \ . \tag{6.21}$$

Hence, the support space of $(\mathrm{u}+\mathrm{v})^{\mathrm{t}}a$ is generated by u and v.

Since the product terms in (6.20) are mutually disjunct, (5.18) can be applied here. From this, it immediately follows that the amplitudes of the Walsh-Hadamard transform values of (6.20) are given by

$$|\hat{F}(w)| = 2^{-r} \quad \text{if} \quad w \in \{z + \sum_i x_i v^i \mid \forall i \in \mathbb{Z}_n, x \in \mathbb{Z}_2^{2r}\} \ , \tag{6.22}$$

and 0 otherwise. This has also been proven in [84, p. 246]. Hence, a linear combination of output components is correlated to 2^{2r} different linear combinations of input components. The correlation is equal to 2^{-r} for all of these input combinations. The correlation weight of $(w \triangleleft \chi \triangleright u)$ is therefore given by r. Since r is completely determined by u, it is denoted by $\mathrm{w_c}(u)$.

We conclude this section with an example that illustrates the effect of adding quadratic product terms in the transform domain.

Example 6.8 *Say $u = \delta_0 + \delta_1 + \delta_2$. The expression $u^{\mathrm{t}}\chi(a)$ is given by*

$$a_0 + (a_1+1)a_2 + a_1 + (a_2+1)a_3 + a_2 + (a_3+1)a_4 \ .$$

This can be converted into

$$(a_0 + a_1 + a_3 + a_4) + (a_1 + a_3)a_2 + a_3 a_4 \ .$$

The effect of the addition of the product terms and the linear terms is illustrated in Fig. 6.1. The domain is represented by Karnaugh maps [51]. It can be seen that adding a quadratic product has a flattening effect on the Walsh-Hadamard transform.

It is clear that for χ there is a correspondence between the two types of nonlinear behavior. This correspondence is described in Table 6.4.

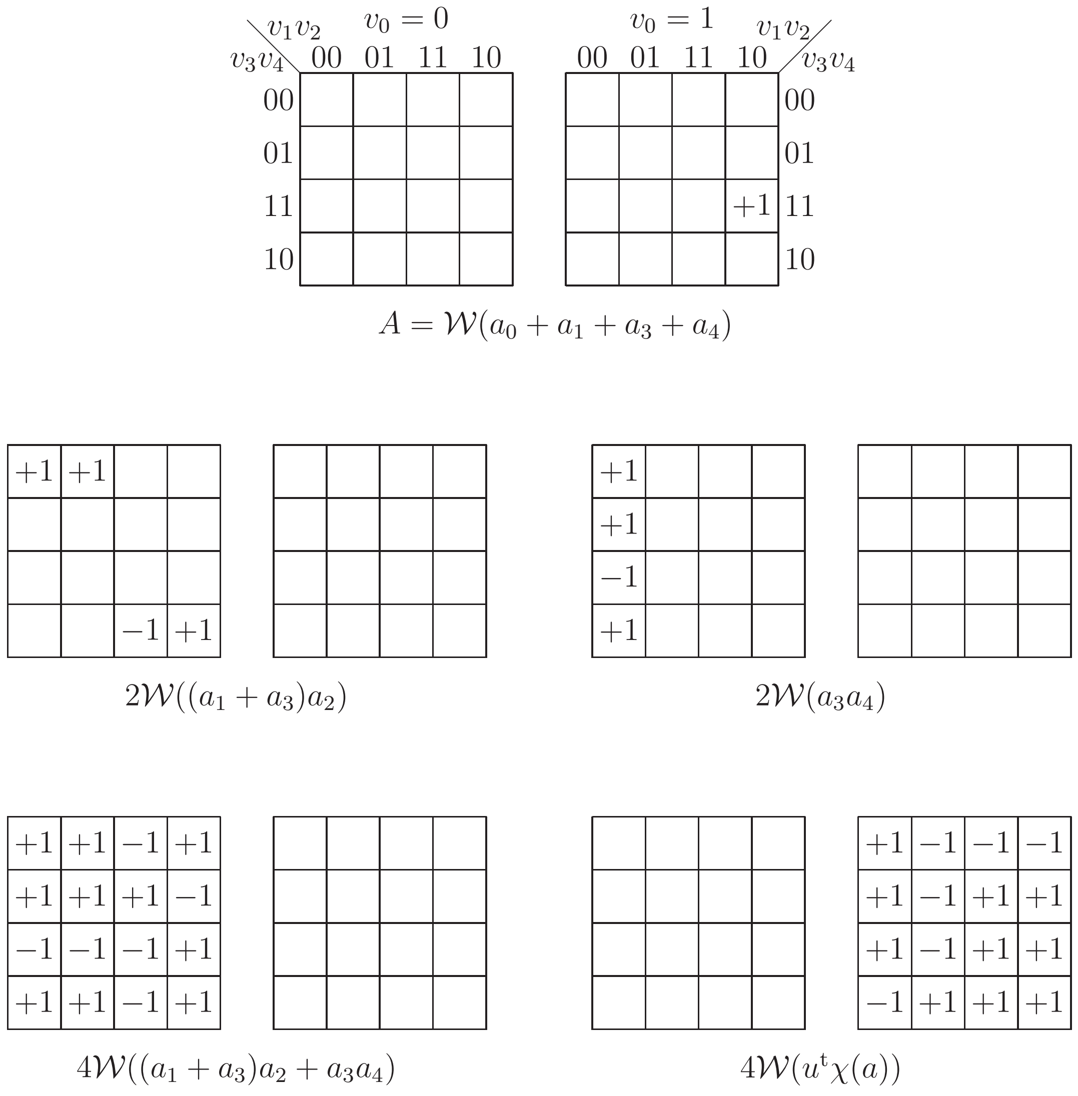

v_1v_2
v_3v_4
$v_0 = 0$
$v_0 = 1$
$A = \mathcal{W}(a_0 + a_1 + a_3 + a_4)$
$2\mathcal{W}((a_1 + a_3)a_2)$
$2\mathcal{W}(a_3a_4)$
$4\mathcal{W}((a_1 + a_3)a_2 + a_3a_4)$
$4\mathcal{W}(u^{\mathrm{t}}\chi(a))$

Figure 6.1: Graphical representation of the calculation of the Walsh-Hadamard transform of $u^{\mathrm{t}}\chi(a)$ from example 6.8. Zero entries have been omitted.

Differential Cryptanalysis	**Linear Cryptanalysis**
input difference a'	output selection u
restriction weight $\mathrm{w_r}(a')$	correlation weight $\mathrm{w_c}(u)$
$2^{\mathrm{w_r}(a')}$ comp. output differences	$2^{2\mathrm{w_c}(u)}$ comp. input selections
with prop ratio $2^{-\mathrm{w_r}(a')}$	with correlation $2^{-\mathrm{w_c}(u)}$

Table 6.4: Summary of the described nonlinear properties of χ.

6.10 Conclusions

This chapter has been devoted to the invertibility and the propagation and correlation properties of shift-invariant transformations. Our own contributions in this chapter are:

- the idea of global invertibility and the corresponding proof method based on seeds and leaps,
- the treatment of multiplication modulo $2^n - 1$ as a shift-invariant transformation,
- the Hamming weight distribution table, the branch number and its upper bound imposed by the *sphere-packing bound*,
- the study of the specific propagation and correlation properties of χ.

Globally invertible shift-invariant transformations will be the most important building blocks in our cryptographic finite state machine designs.

Chapter 7

Block Cipher Design

7.1 Introduction

In this chapter we further elaborate our block cipher design strategy that was introduced in Chapter 4. We start with describing a new type of self-reciprocal cipher structure that is a widely applicable alternative for the Feistel round structure. The round transformation is composed of a small number of simple basic transformations that must satisfy certain algebraic conditions.

We describe a number of transformations that are especially suited for the proposed cipher structure. For each of these transformations the difference propagation and correlation properties are treated. We show that, for the proposed cipher structure and basic transformations, the behavior of differential and linear trails are governed by the same equations. This allows the evaluation and optimization of the resistance against LC and DC in a single effort. We also discuss the risk of weaknesses due to symmetry and the measures that have to be taken with respect to these aspects.

This chapter contains two fully specified block ciphers with high portability and a short and elegant description. For both we give our findings with respect to LC and DC. We conclude with describing an example filtered counter stream encryption scheme that can be built using one of the two proposed block ciphers. A large part of this chapter has already been published in our paper [22].

7.2 A self-reciprocal cipher structure

The basic building blocks of our block ciphers are a number of different invertible transformations, each with its own specific contribution:

γ: a local **nonlinear** transformation,

θ: a local linear transformation for **diffusion**,

σ: bitwise addition of a round **key** κ_j,

π_i: blockwise bit permutations for **bit dispersion**.

These transformations must be arranged into a portable and simple block cipher. For block encryption and decryption to be executable by a single cryptographic finite state machine, this cipher must be structurally self-reciprocal. In this section we show how such a nontrivial self-reciprocal structure can be built by introducing a simple bit permutation μ and imposing that the basic transformations interact with μ in a specific way.

The block cipher consists of a certain number m of iterations of a round transformation $\rho[\kappa_j]$, followed by an output transformation $\omega[\kappa_m]$ and the bit permutation μ. The round keys κ_i are derived from the cipher key κ by the key schedule. The round transformation ρ and the output transformation ω are composed from the basic transformations in the following way:

$$\rho[\kappa_j] = \pi_2 \circ \gamma \circ \pi_1 \circ \theta \circ \sigma[\kappa_j] \ , \tag{7.1}$$
$$\omega[\kappa_m] = \theta \circ \sigma[\kappa_m] \ . \tag{7.2}$$

Consider the simple single-round block cipher

$$\mathrm{B}_1[\kappa_0, \kappa_1] = \mu \circ \omega[\kappa_1] \circ \rho[\kappa_0] \tag{7.3}$$

or

$$\mathrm{B}_1[\kappa_0, \kappa_1] = \mu \circ \theta \circ \sigma[\kappa_1] \circ \pi_2 \circ \gamma \circ \pi_1 \circ \theta \circ \sigma[\kappa_0] \ . \tag{7.4}$$

Its inverse is given by

$$\mathrm{B}_1[\kappa_0, \kappa_1]^{-1} = \sigma[\kappa_0] \circ \theta^{-1} \circ \pi_1^{-1} \circ \gamma^{-1} \circ \pi_2^{-1} \circ \sigma[\kappa_1] \circ \theta^{-1} \circ \mu^{-1} \ . \tag{7.5}$$

Since θ is linear we can switch θ and σ resulting in

$$\mathrm{B}_1[\kappa_0, \kappa_1]^{-1} = \theta^{-1} \circ \sigma[\theta(\kappa_0)] \circ \pi_1^{-1} \circ \gamma^{-1} \circ \pi_2^{-1} \circ \theta^{-1} \circ \sigma[\theta(\kappa_1)] \circ \mu^{-1} \ . \tag{7.6}$$

We choose the basic transformations in such a way that they satisfy the following conditions:

$$\theta^{-1} = \mu^{-1} \circ \theta \circ \mu \ , \tag{7.7}$$
$$\gamma^{-1} = \mu^{-1} \circ \gamma \circ \mu \ , \tag{7.8}$$
$$\pi_2^{-1} = \mu^{-1} \circ \pi_1 \circ \mu, \tag{7.9}$$
$$\pi_1^{-1} = \mu^{-1} \circ \pi_2 \circ \mu \ . \tag{7.10}$$

Using these relations, we can eliminate all occurrences of inverse transformations in (7.6), with the exception of μ^{-1}, i.e.,

$$\begin{aligned}
&\mathrm{B}_1[\kappa_0, \kappa_1]^{-1} \\
&= \theta^{-1} \circ \sigma[\theta(\kappa_0)] \circ \pi_1^{-1} \circ \gamma^{-1} \circ \pi_2^{-1} \circ \theta^{-1} \circ \sigma[\theta(\kappa_1)] \circ \mu^{-1} \\
&= \theta^{-1} \circ \sigma[\theta(\kappa_0)] \circ \pi_1^{-1} \circ \gamma^{-1} \circ \pi_2^{-1} \circ \mu^{-1} \circ \theta \circ \sigma[\mu(\theta(\kappa_1))] \\
&= \theta^{-1} \circ \sigma[\theta(\kappa_0)] \circ \pi_1^{-1} \circ \gamma^{-1} \circ \mu^{-1} \circ \pi_1 \circ \theta \circ \sigma[\mu(\theta(\kappa_1))] \\
&= \theta^{-1} \circ \sigma[\theta(\kappa_0)] \circ \pi_1^{-1} \circ \mu^{-1} \circ \gamma \circ \pi_1 \circ \theta \circ \sigma[\mu(\theta(\kappa_1))] \\
&= \theta^{-1} \circ \sigma[\theta(\kappa_0)] \circ \mu^{-1} \circ \pi_2 \circ \gamma \circ \pi_1 \circ \theta \circ \sigma[\mu(\theta(\kappa_1))] \\
&= \mu^{-1} \circ \theta \circ \sigma[\mu(\theta(\kappa_0))] \circ \pi_2 \circ \gamma \circ \pi_1 \circ \theta \circ \sigma[\mu(\theta(\kappa_1))] \ .
\end{aligned} \tag{7.11}$$

By imposing the additional condition

$$\mu^{-1} = \mu \ , \tag{7.12}$$

we have

$$\mathrm{B}_1[\kappa_0, \kappa_1]^{-1} = \mathrm{B}_1[\kappa'_0, \kappa'_1] \ , \tag{7.13}$$

with the inverse round keys given by $\kappa'_j = \mu(\theta(\kappa_{1-j}))$. If line 5 of (7.11) is rewritten in terms of the round transformation and the output transformation, we have for $j \geq 1$,

$$\rho[\kappa_{j-1}]^{-1} \circ \omega[\kappa_j]^{-1} \circ \mu^{-1} = \omega[\kappa_{j-1}]^{-1} \circ \mu^{-1} \circ \rho[\mu(\theta(\kappa_j))] \ . \tag{7.14}$$

A cipher B_m with m rounds is defined by

$$\mathrm{B}_m[\kappa_0, \kappa_1, \kappa_2, \ldots, \kappa_m] = \mu \circ \omega[\kappa_m] \circ \rho[\kappa_{m-1}] \circ \ldots \circ \rho[\kappa_1] \circ \rho[\kappa_0] \ .$$

By iteratively applying (7.14) we obtain

$$\mathrm{B}_m[\kappa_0, \kappa_1, \kappa_2, \ldots, \kappa_m]^{-1} = \mathrm{B}_m[\kappa'_0, \kappa'_1, \kappa'_2, \ldots, \kappa'_m] \ , \tag{7.15}$$

with $\kappa'_j = \mu(\theta(\kappa_{m-j}))$.

We propose to use a cipher key κ of length equal to the block length. The round keys κ_j are derived from this cipher key by the bitwise addition of so-called *round constants* c^j:

$$\kappa_j = \kappa + \mathrm{c}^j \ . \tag{7.16}$$

If $\kappa' = \mu(\theta(\kappa))$ is considered to be the inverse cipher key, the inverse round keys κ'_j can be derived from κ' in a similar manner: $\kappa'_j = \kappa' + \mathrm{c}'^j$ with

$$\mathrm{c}'^j = \mu(\theta(\mathrm{c}^{m-j})) \ . \tag{7.17}$$

This cipher structure lends itself to a straightforward implementation as a cryptographic finite state machine. Encryption and decryption can be performed by the same finite state machine with ρ as updating transformation. An encryption operation consists of initializing the state register by loading the plaintext and doing m state updating iterations. During these iterations the key schedule is executed by a separate round-constant generating module. The additional application of the

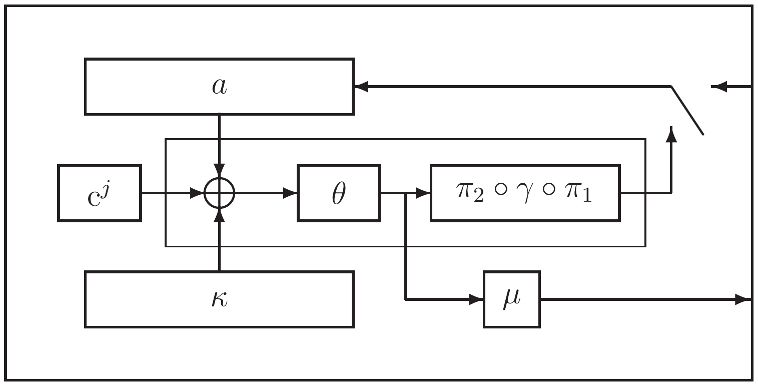


Figure 7.1: Cryptographic finite state machine for the proposed cipher structure.

output transformation can be accomplished by reading out the intermediate stage after θ of the round transformation logic, instead of the internal state itself. The bit permutation μ can be hardwired in the connections to the output pins. The block scheme of the resulting cryptographic finite state machine is shown in Fig. 7.1. Additionally, it can be observed from this scheme that if the cipher key is loaded into the state register and the key register is set to 0, the inverse cipher key appears at the output. Hence, the calculation of the inverse cipher key from the cipher key can be executed by the module itself.

If the specific permutation μ is hard to implement in software and the inverses of θ, γ, π_1 and π_2 are easy, a variant of the proposed structure may be more suitable. In this variant the final application of μ in the encrypting operation is omitted. This must be compensated by an additional application of μ at the beginning of decryption. We have

$$\mathrm{B}_m[\kappa_0, \kappa_1, \ldots, \kappa_m] = \omega[\kappa_m] \circ \rho[\kappa_{m-1}] \circ \ldots \circ \rho[\kappa_1] \circ \rho[\kappa_0] \ , \tag{7.18}$$

and

$$\mathrm{B}_m[\kappa_0, \kappa_1, \ldots, \kappa_m]^{-1} = \mu \circ \mathrm{B}_m[\kappa'_0, \kappa'_1, \ldots, \kappa_m] \circ \mu \ , \tag{7.19}$$

with $\kappa'_j = \mu(\theta(\kappa_{m-j}))$.

The cryptographic finite state machine implementation scheme of this variant differs from that given in Fig. 7.1 by the presence of two functional blocks responsible for μ in decryption mode. One of these is placed at the input, the other at the output. These blocks are switched "off" for encryption and "on" for decryption.

In software implementations of this variant the encrypting transformation does not contain μ. In the decrypting transformation the need for μ can be avoided by implementing it using the inverses of θ, γ, π_1 and π_2.

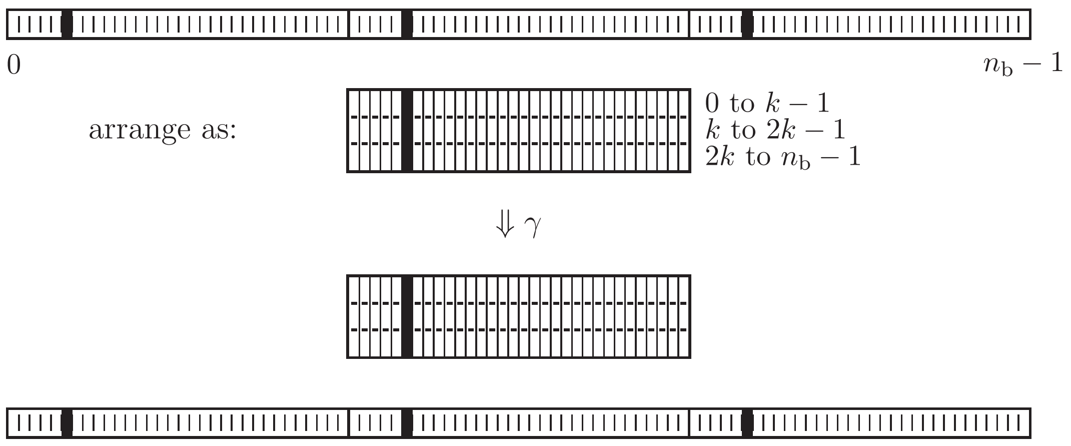


Figure 7.2: Arrangement of the bits in the transformation γ. The bits of an individual triplet are marked in black.

7.3 The Nonlinear transformation γ

The transformation γ is defined for vectors with a dimension n_b divisible by 3. Let $b = \gamma(a)$ with a and b vectors of length $n_b = 3k$. We have

$$b_i = a_i + (a_{i+k} + 1)a_{i+2k} + 1 \ . \tag{7.20}$$

This is in fact a simple variant of the invertible shift-invariant transformation χ. Every 3-tuple, or *triplet* (b_i, b_{i+k}, b_{i+2k}) is completely determined by the triplet (a_i, a_{i+k}, a_{i+2k}), i.e., γ is a juxtaposed transformation consisting of k equal 3-bit substitution boxes. The arrangement of the bits in triplets is illustrated in Fig. 7.2. In this figure it can also be seen that in software implementations a number of substitution boxes equal to the processor word length can be handled simultaneously by using bitwise Boolean operations. The block length n_b must be a multiple of 3. Since computer word lengths are typically powers of 2, we restrict the block length to $n_b = 2^\ell 3$ for some ℓ.

x	000	001	010	100	110	101	011	111
$\gamma(x)$	111	010	100	001	011	110	101	000

Table 7.1: The 3-bit γ substitution box.

The γ substitution box is given in Table 7.1. Its effect can be described as: a single 1 at the input is shifted one position to the left, a single 0 one position to the right and three equal input bits are complemented. The description of the inverse of γ is obtained by simply interchanging the words “left” and “right”. Hence, for

$$\gamma^{-1} = \mu^{-1} \circ \gamma \circ \mu \tag{7.21}$$

to hold, μ must respect the division in triplets and invert the order of the components within the triplets.

	000	001	010	100	011	101	110	111
000	1	-	-	-	-	-	-	-
001	-	1/4	-	-	1/4	1/4	-	1/4
010	-	-	1/4	-	1/4	-	1/4	1/4
100	-	-	-	1/4	-	1/4	1/4	1/4
011	-	1/4	1/4	-	-	1/4	1/4	-
101	-	1/4	-	1/4	1/4	-	1/4	-
110	-	-	1/4	1/4	1/4	1/4	-	-
111	-	1/4	1/4	1/4	-	-	-	1/4

Table 7.2: Prop ratios for the γ substitution box.

	000	001	010	100	011	101	110	111
000	1	-	-	-	-	-	-	-
001	-	$-1/2$	-	-	$-1/2$	$1/2$	-	$-1/2$
010	-	-	$-1/2$	-	$1/2$	-	$-1/2$	$-1/2$
100	-	-	-	$-1/2$	-	$-1/2$	$1/2$	$-1/2$
011	-	$1/2$	$-1/2$	-	-	$1/2$	$1/2$	-
101	-	$-1/2$	-	$1/2$	$1/2$	-	$1/2$	-
110	-	-	$1/2$	$-1/2$	$1/2$	$1/2$	-	-
111	-	$-1/2$	$-1/2$	$-1/2$	-	-	-	$1/2$

Table 7.3: Correlation matrix of the γ substitution box.

7.3.1 Propagation and correlation properties

Table 7.2 lists the prop ratios of difference propagation in the γ substitution box. The input differences are listed above and the output differences at the left. Every nonzero input (output) difference is compatible with exactly four output (input) differences. All nontrivial difference propagations have a prop ratio of 1/4 and a restriction weight of 2. An input difference triplet and an output difference triplet are *compatible* if they have an odd number of 1-bits in common, i.e., if the parity of their bitwise product (AND) is odd.

An input difference and an output difference to γ are compatible if all their component triplets are compatible. If an input difference has ℓ nonzero triplets it is compatible with $2^{2\ell}$ different output differences. All possible difference propagations from this input difference have restriction weight 2ℓ. Hence, $\mathrm{w_r}(a')$, the restriction weight of a difference vector a' with respect to γ, is equal to twice the number of nonzero component triplets in a'.

Table 7.3 gives the correlation matrix of the γ substitution box. Every nonzero output selection triplet is correlated with exactly four input selection triplets, all with correlation $\pm 1/2$. By comparing Tables 7.3 and 7.2 it can be seen that the condition for compatibility between input and output selection triplets is the same as that between input and output differences. This is not the case in general but a

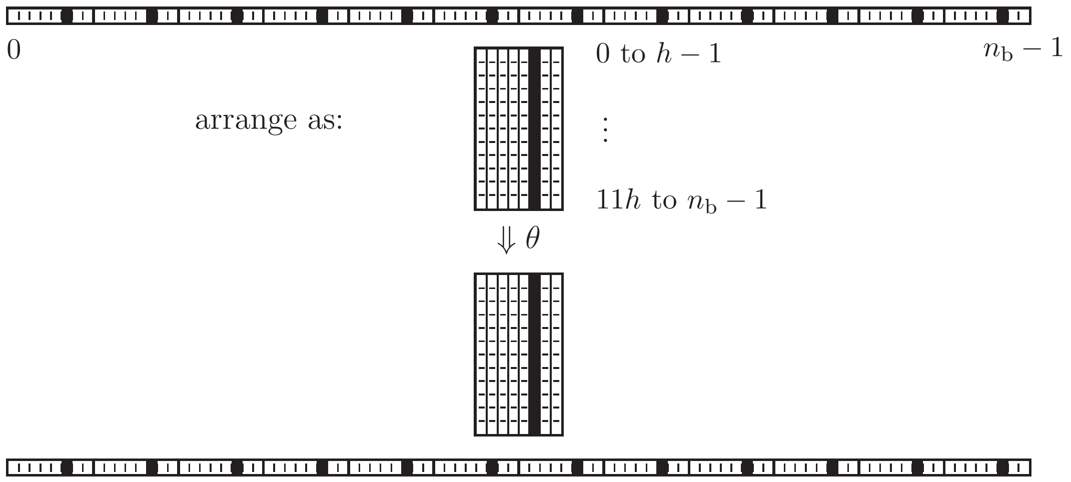


Figure 7.3: Arrangement of the bits in the transformation θ. The bits of an individual 12-tuple are marked in black.

consequence of the symmetry properties of γ.

An input selection and an output selection to γ are compatible if all their component triplet selections are compatible. Since linear combinations corresponding to different triplets are disjunct, $\mathrm{C}(u^{\mathrm{t}}\gamma(a), w^{\mathrm{t}}a)$ is equal to $\pm 2^{-\ell}$ with ℓ the number of nonzero component triplets in u. Every output selection vector is compatible to exactly $2^{2\ell}$ input selection vectors. Hence, the correlation weight of a selection vector $\mathrm{w_c}(u)$ is equal to the number of nonzero component triplets in u. We introduce the *triplet weight* $\mathrm{w_t}(a)$ to be the number of nonzero triplets in a vector. We have

$$\mathrm{w_c}(a) = \mathrm{w_t}(a) \quad \text{and} \quad \mathrm{w_r}(a) = 2\mathrm{w_t}(a) \; . \tag{7.22}$$

7.4 The linear transformation θ

The transformation θ is defined for vectors with a dimension n_b divisible by 12. Let $b = \theta(a)$, with a and b vectors of length $n_b = 12h$. We have

$$b(x) = e(x^h)a(x) \bmod (1 + x^{12h}) \; , \tag{7.23}$$

with

$$e(x) = 1 + x + x^2 + x^3 + x^5 + x^6 + x^{10} \; . \tag{7.24}$$

Every 12-tuple $(b_i, b_{i+h}, b_{i+2h}, \ldots, b_{i+11h})$ is completely determined by the 12-tuple $(a_i, a_{i+h}, a_{i+2h}, \ldots, a_{i+11h})$, i.e., θ can be seen as a juxtaposed transformation with h linear 12-bit substitutions boxes. This is illustrated in Fig. 7.3. Similar to γ, in software implementations a number of linear θ substitutions can be handled simultaneously using bitwise Boolean addition.

The inverse of θ consists of multiplication by the inverse polynomial of $e(x^h)$ modulo $1 + x^{n_b}$. The selected polynomial has been chosen from the subclass of polynomials with the following property:

$$e(x)^{-1} \equiv e(x^{-1}) \pmod{1 + x^{12}} \ , \tag{7.25}$$

hence, the inverse of θ is given by

$$a(x) = e(x^{-h})b(x) \bmod 1 + x^{12h} \ . \tag{7.26}$$

The effect of substituting x by x^{-1} in the argument of a polynomial $a(x)$ corresponds to a bit permutation in the vector a. It can be seen as a reflection around component 0, interchanging the components in pairs $(i, n_b - i)$. We can rewrite (7.26) as

$$a(x^{-1}) = e(x^h)b(x^{-1}) \bmod 1 + x^{12h} \ , \tag{7.27}$$

i.e., the inverse of θ applied to the reflected vectors corresponds to θ itself. Hence, for

$$\theta^{-1} = \mu^{-1} \circ \theta \circ \mu \tag{7.28}$$

to hold, μ must respect the division in 12-tuples and invert the order of the components within the 12-tuples.

The selection of $e(x)$ was governed by two additional design decisions. To have high diffusion we decided that $e(x)$ should have a Hamming weight of 7. This has the additional benefit that in hardware the combination of θ and $\sigma[\kappa_j]$ can be implemented with balanced-tree circuits consisting of three stages with respectively 4, 2 and 1 EXOR gates. The modulus exponent m was chosen to be the smallest value larger than 7 of the form $2^\ell 3$, since m must divide the block length n_b.

7.4.1 Propagation and correlation properties

As explained in Chapter 6, the difference propagation through θ can be expressed as

$$b'(x) = e(x^h)a'(x) \bmod (1 + x^{n_b}) \ , \tag{7.29}$$

and a linear combination of output bits specified by u is equal to the linear combination of input bits specified by w with

$$w(x) = e(x^{-h})u(x) \bmod (1 + x^{n_b}) \ . \tag{7.30}$$

Using (7.25) this can be converted to

$$u(x) = e(x^h)w(x) \bmod (1 + x^{n_b}) \ . \tag{7.31}$$

Hence, for θ difference propagation and correlation are governed by essentially the same equation. This is not the case in general, nor is it a coincidence. It is the consequence of the design decision specified in (7.25).

	1	2	3	4	5	6	7	8	9	10	11
1	-	-	-	-	-	-	12	-	-	-	-
2	-	-	-	-	-	60	-	-	-	6	-
3	-	-	-	-	180	-	-	-	40	-	-
4	-	-	-	255	-	-	-	240	-	-	-
5	-	-	180	-	-	-	600	-	-	-	12
6	-	60	-	-	-	804	-	-	-	60	-
7	12	-	-	-	600	-	-	-	180	-	-
8	-	-	-	240	-	-	-	255	-	-	-
9	-	-	40	-	-	-	180	-	-	-	-
10	-	6	-	-	-	60	-	-	-	-	-
11	-	-	-	-	12	-	-	-	-	-	-

Table 7.4: Hamming weight distribution table of polynomial multiplication by $e(x)$ modulo $1 + x^{12}$.

The combination of the shift-invariance and (7.25) causes the matrix M_θ corresponding to θ to be orthogonal with respect to $\mathbb{Z}_2^{n_\mathrm{b}}$, i.e.,

$$M_\theta{}^{-1} = M_\theta{}^{\mathrm{t}} \ . \tag{7.32}$$

The Hamming weight distribution table (see p. 106) of modular multiplication by $e(x)$ is given in Table 7.4. It can be observed that the branch number $\mathcal{B}$ of this linear shift-invariant transformation is 8, the maximum attainable value both for a polynomial with 7 terms and for a polynomial multiplication modulo $1 + x^{12}$ (see p. 108).

7.5 The bit permutations μ and π

μ is a bit permutation with $\mu^{-1} = \mu$ that respects the grouping of the triplets and 12-tuples, but inverts the order of the components within them. The simplest example of such a bit permutation is μ_1, that simply inverts the order of the components of the vector. Hence, if $b = \mu_1(a)$ we have

$$b_i = a_{n_\mathrm{b}-1-i} \text{ for } 0 \le i < n_\mathrm{b} \ . \tag{7.33}$$

This bit permutation requires the handling of individual bits and is not very well suited for software implementations. This is not the case for the bit permutation μ_h that inverts the order of h-bit subblocks. If $b = \mu_h(a)$ we have

$$b_{i+jh} = a_{i+(11-j)h} \ , \tag{7.34}$$

for $0 \le i < h$ and $0 \le j < 12$.

The bit permutations π_1 and π_2 treat the vector in 3 or 12 subblocks. The bits of each subblock are cyclically shifted by a specified number of positions. The bit permutations can be specified by an array that contains these rotation constants.

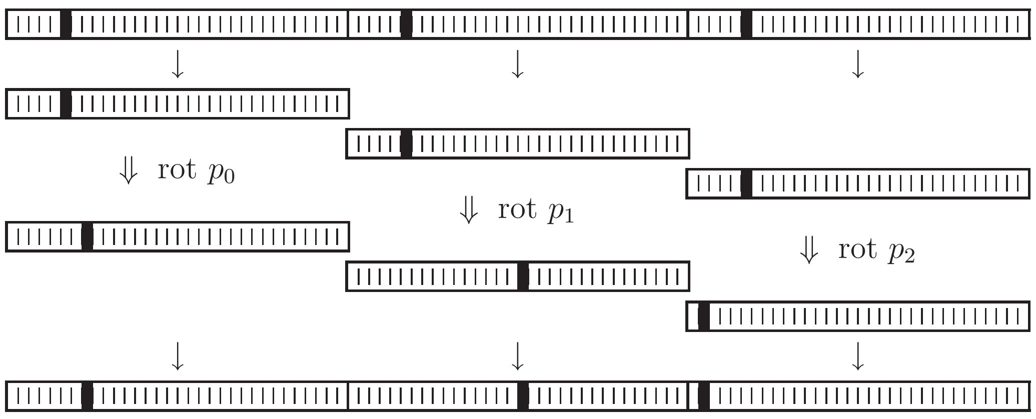


Figure 7.4: The arrangements of bits in the π bit permutation with 3 blocks specified by $(p_0, p_1, p_2) = (2, 11, 28)$. The effect is shown on the three bits of a triplet.

The effect of a bit permutation $b = \pi(a)$ specified by the 3-tuple (p_0, p_1, p_2) is described by

$$\left.\begin{array}{lcl} b_i & = & a_{i-p_0 \bmod k} \\ b_{i+k} & = & a_{(i-p_1 \bmod k)+k} \\ b_{i+2k} & = & a_{(i-p_2 \bmod k)+2k} \end{array}\right\} \text{for } 0 \leq i < k \; .$$

Figure 7.4 illustrates the arrangement of the bits in such a bit permutation. In software, the transformations π can be implemented using bitwise shift operations.

The choice of one of the π bit permutations and μ determines the other π bit permutation through (7.9). For μ_1 we have

$$\pi_1 : (p_0, p_1, \ldots) \Leftrightarrow \pi_2 : (\ldots, p_1, p_0) \; . \tag{7.35}$$

For μ_h it can easily be checked that the bit permutations must necessarily split the vector into 12 subblocks. We have

$$\pi_1 : (p_0, p_1, \ldots, p_{11}) \Leftrightarrow \pi_2 : (-p_{11}, \ldots, -p_1, -p_0) \; . \tag{7.36}$$

7.5.1 Propagation and correlation properties

Since bit permutations are linear, the propagation of differences through π_1 and π_2 is governed by

$$b' = \pi_i(a') \; . \tag{7.37}$$

For the specification of linear combinations of input bits w in terms of output bits u, we write the bit permutation in a matrix M_π. We have

$$w = M_\pi^{\mathrm{t}} u \; . \tag{7.38}$$

Using the fact that the permutation of components is an orthogonal transformation and therefore ${M_\pi}^{-1} = M_\pi^{\mathrm{t}}$, we have $u = M_\pi w$, or equivalently

$$u = \pi_i(w) \; . \tag{7.39}$$

As in the case of γ and θ it can be seen that the difference propagation and the input-output selection correlation are governed by the same equation. This is an inherent property of bit permutations.

7.6 Propagation analysis

In this section we describe the behavior of differential and linear trails in the proposed block cipher structure.

The block cipher can be described as the repeated application of alternating nonlinear transformations γ and linear (in fact affine) transformations $\lambda[\kappa_j] = \pi_1 \circ \theta \circ \sigma[\kappa_j] \circ \pi_2$. For the nonlinear step the correlation and difference propagation properties are described by the compatibility conditions and the triplet weight. The propagation of differences through λ is governed by

$$b' = \lambda[0](a') \ . \tag{7.40}$$

A linear combination of input bits specified by w is correlated to a linear combination of output bits specified by u given by

$$u = \lambda[0](w) \ , \tag{7.41}$$

with correlation $(-1)^{u^{\mathrm{t}}\pi_1(\theta(\kappa_j))}$. In the following we will omit the $[0]$ in $\lambda[0](a)$.

In the context of propagation analysis we consider a round to be $\gamma \circ \lambda[\kappa_j]$. A differential *step* consists of a couple of difference vectors (a', b') with $\lambda(a')$ γ-compatible with b'. Its restriction weight is $2\mathrm{w_t}(b')$. A linear *step* consists of a couple of selection vectors (w, u) with $\lambda(w)$ compatible with u. Its correlation weight is $\mathrm{w_t}(u)$. Hence, a couple (a, b) with $\lambda(a)$ γ-compatible with b can be interpreted both as a differential and as a linear step and is called a *propagation* step. These steps can be chained to form *propagation* trails.

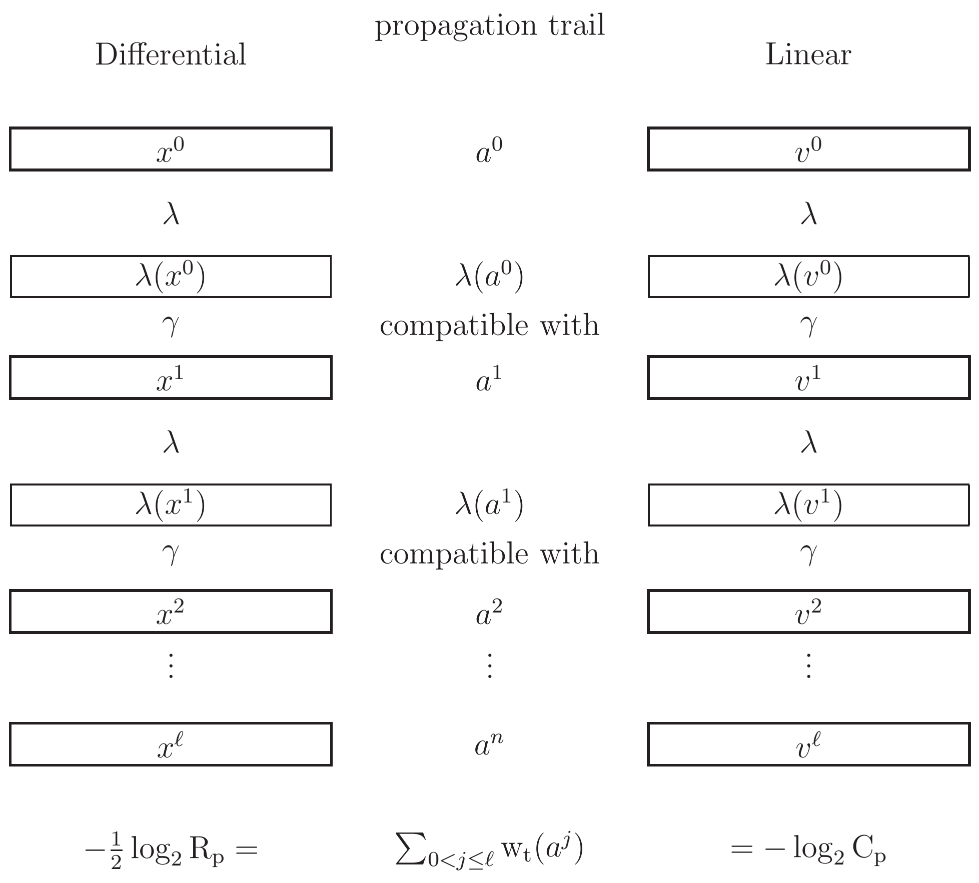


Figure 7.5: The differential and linear interpretations of propagation trails for the self-reciprocal block cipher structure.

An ℓ-round propagation trail Ω is specified by an $\ell+1$-tuple

$$(\omega^0, \omega^1, \ldots, \omega^\ell) ,$$

with $\lambda(\omega^{i-1})$ compatible with ω^i for $0 < i \leq \ell$. Its triplet weight is given by

$$\mathrm{w_t}(\Omega) = \sum_{0<i\leq\ell} \mathrm{w_t}(\omega^i) . \tag{7.42}$$

This propagation trail represents both a differential trail with restriction weight $2\mathrm{w_t}(\Omega)$ and a linear trail with correlation weight $\mathrm{w_t}(\Omega)$. Figure 7.5 shows the two interpretations of a propagation trail.

Once the block size n_b and the permutation μ have been fixed, all that remains is the specification of π_1. The shift constants must be chosen to eliminate the occurrence of propagation trails with low triplet weight, addressing the resistance against DC and LC in a single effort. The selection of the array of rotation constants involves finding and comparing the critical propagation trails for a small number of rounds by computer.

After the specification of the rotation constants, it has to be verified that the triplet weight of the propagation trails appropriately reflects the resistance against

LC and DC. The most important effect that has to be investigated is the potential systematic clustering of low-weight trails.

7.7 Symmetry considerations

Even if a block cipher is resistant against LC and DC, symmetry in its structure can be the cause of serious weaknesses. The best known example of a block cipher with such weaknesses is DES. The first symmetry-based weakness is the existence of weak and semi-weak keys for DES [30]. For the four weak keys, encryption is an involution. For the six pairs of semi-weak keys, encryption with one key of a pair is the same as decryption with the other key of the pair. The second symmetry-based weakness of DES is the complementation property [6, 48]. This property can be exploited to reduce exhaustive key search of DES by a factor of 2. More recent examples can be found in [7], where the regularity in key schedules is used to construct efficient chosen-key attacks and to speed up exhaustive key search.

Many undesirable symmetry properties are special cases of one of the two following properties:

- There are affine mappings $\lambda_\mathrm{k}, \lambda_\mathrm{p}$ and λ_c, such that for some keys $\lambda_\mathrm{c} \circ \mathrm{B}[\lambda_\mathrm{k}(\kappa)] \circ \lambda_\mathrm{p}$ is equal to $\mathrm{B}[\kappa]$ or $\mathrm{B}[\kappa]^{-1}$;
- There are keys for which the last $r-p$ rounds of the cipher (or its inverse) under one key perform the same transformation as the first $r-p$ rounds of the cipher (or its inverse) under another key, with p small.

The round constants c^j must be chosen in such a way that all symmetry-related weaknesses are eliminated. This choice is not affected in any way by propagation trail considerations. The round constants are derived from the state q of a linear feedback shift register with length 8. In polynomial representation we have

$$q^j(x) = (1 + x + x^3)x^j \bmod (1 + x^4 + x^8) . \tag{7.43}$$

The order of the feedback polynomial is 12, hence, $x^{12} = 1 \bmod (1 + x^4 + x^8)$. The calculation of the round constants c^j in polynomial representation is

$$\mathrm{c}^j(x) = (x^{2h} + x^{3h} + x^{8h} + x^{9h})q^j(x) , \tag{7.44}$$

with $12h = n_\mathrm{b}$.

The round constants are chosen in such a way that the difference between the round constants of any pair of subsequent encryption or decryption rounds is different. The decryption round constants depend on the block length and on the bit permutation μ.

7.8 3-Way

7.8.1 Specification

3-Way is a block cipher with the self-reciprocal structure. It is designed to be hermetic and K-secure with respect to sound initialization mappings. It is specified by

1. $n_{\mathrm{b}} = 96$,
2. $\mu = \mu_1$,
3. $\pi_1 : (10, 0, -1)$ and $\pi_2 : (-1, 0, 10)$,
4. encryption: 11 rounds and a single output transformation,
5. decryption: 11 rounds and a single output transformation, preceded and followed by μ.

μ_1 has been chosen to allow the π permutations to act on 32-bit words. The rotation constants $(10, 0, -1)$ have been selected in the following way. 0 and -1 have been fixed in advance because of their economy. 10 was selected from the candidate constants $\{3, 5, 6, 7, 9, 10, 11, 13, 14, 15\}$ as realizing the best propagation properties in the short term. It would of course be better to select the rotation constant with the best propagation properties in the long term, but this turns out to be computationally infeasible.

7.8.2 Implementation aspects

The number of rounds 11 is motivated by its convenience in a cryptographic finite state machine implementation. The encryption of a single block takes 12 clock cycles: 11 state updating iterations and 1 simultaneous plaintext load and ciphertext read operation. This results in an encryption (and decryption) rate of 8 bits per clock cycle. The total gate delay of the cryptographic finite state machine can be made as small as that of 4 exors, 1 nand and 1 mux (multiplexor), allowing clock speeds of over 100 MHz and encryption rates of over 800 Mbit/s, even with conventional technology. The small number of basic operations also allows for extremely compact hardware implementations with a small 8-bit processor with instructions bitwise addition, and and shift, some program memory (rom) and some data memory (ram).

In software, the steps γ and θ can be efficiently programmed using bitwise exor, or, complementation and shifting. A straightforward C implementation allows an encryption speed of over 2 Mbit/s on a 66 MHz 80486 processor. We expect that optimization and the use of coding in assembler language allow a speedup by at least a factor of 5.

7.8.3 Decryption

For the inverse round constants in 3-Way we have $\mathrm{c}'^j = \mu_1(\theta(\mathrm{c}^{11-j}))$. It can be seen that

$$\mathrm{c}'^j(x) = (x^{2h} + x^{3h} + x^{8h} + x^{9h})q'^j(x) \ , \tag{7.45}$$

with $q'^j(x)$ given by

$$q'^j(x) = (1 + x^4 + x^5 + x^7)x^j \bmod (1 + x^4 + x^8) \ . \tag{7.46}$$

In a cryptographic finite state machine the encryption and decryption round constants can be generated and applied with the same circuitry. The only difference is the initial value q^0 of the 8-bit linear feedback shift register.

7.8.4 Propagation analysis

Propagation analysis mainly consists of the search for propagation trails with low weight. For this purpose we have written and ran programs that scan the space of propagation trails in a recursive pruned tree search.

Table 7.5 is the (partial) triplet weight distribution table of $\lambda = \pi_1 \circ \theta \circ \pi_2$ for 3-Way. The element in row i and column j denotes the number of couples $(a, \lambda(a))$ with $\mathrm{w_t}(a) = i$ and $\mathrm{w_t}(\lambda(a)) = j$. This table illustrates the high quality of the short-term propagation properties of the 3-Way round transformation. It can be seen that for any couple $(a, \lambda(a))$ the sum of their triplet weight is at least 8. It follows that there are no 2-round propagation trails with triplet weight below 8 and consequently that the minimum triplet weight for propagation trails of even length is 4 per round. The triplet weight distribution table inherits this property from the Hamming weight distribution table of θ. This is a consequence of the application of the π bit permutations before and after θ. These simple bit permutations contribute to the single-round diffusion by spreading the bits in a single triplet over several other triplets.

The number of vectors with a triplet weight $\mathrm{w_t}$ is given by

$$7^{\mathrm{w_t}} \binom{n_\mathrm{b}/3}{\mathrm{w_t}} \ . \tag{7.47}$$

For $n_\mathrm{b} = 96$ and $\mathrm{w_t} = 6$ we have 3.4×10^9 vectors. This turned out to be too large for our exhaustive program to end within a reasonable time span, hence the question marks in Table 7.5. The values that are actually listed in column 6 and 7 are known because the partial triplet weight distribution table is symmetrical. This is a consequence of the fact that λ is an orthogonal linear transformation in $\mathbb{Z}_2^{n_\mathrm{b}}$.

The bit permutations play an important role in the multiple-round propagation properties in preventing the clustering of propagation trails. This can be illustrated by considering the hypothetical case of all three rotation constants in π_1 being 0. In that case the bits of the 12-tuples are not mixed and the propagation trails are restricted to stay within the 12-tuples, inevitably giving rise to clustering. Moreover, the bit permutations π prevent the iterative chaining of propagation steps (w_{j-1}, w_j)

	1	2	3	4	5	6	7
1	-	-	-	-	-	-	96
2	-	-	-	-	-	480	-
3	-	-	-	-	1440	-	-
4	-	-	-	2040	-	7	168
5	-	-	1440	-	25	313	12480
6	-	480	-	7	313	?	?
7	96	-	-	168	12480	?	?
8	-	-	55	5335	71138	?	?
9	-	19	1122	28012	265865	?	?
10	-	195	6381	90042	431964	?	?
11	39	836	18775	119868	457174	?	?
12	25	1883	20751	113010	776241	?	?
13	32	2017	17408	159098	2682584	?	?
14	13	1677	21418	469917	6262878	?	?

Table 7.5: Partial triplet weight distribution table of λ for 3-Way.

with $\mathrm{w_t}(w_{j-1})+\mathrm{w_t}(w_j) = 8$, by destroying the alignment required for these low triplet weights.

If 3-Way has no 9-round propagation trails with a triplet weight below 48, there are no 9-round differential trails with prop ratio above 2^{-96} and no linear trails with correlation above 2^{-48}. Both are too insignificant to be exploited in an attack.

From observing Table 7.5 it can easily be seen that there are no (even-length) differential trails with a prop ratio below 2^{-8} per round, neither (even-length) linear trails with a correlation contribution below 2^{-4} per round. This is more than a factor 2 better than the round function of DES, with its iterative differential trails with a prop ratio of $2^{-3.6}$ per round [5] and its 14-round linear trail with a correlation contribution of $2^{-20.2}$ or $2^{-1.4}$ per round [71].

By listing the critical propagation weights of 1,2,... rounds, a *propagation weight profile* can be specified. We have been able to determine this profile for up to 5 3-Way rounds: $(1, 8, 11, 16, 22)$. For 6 rounds no propagation trails were found with a propagation weight below 36. The relatively low weights of the critical propagation trails for a small number of rounds are due to the inability of π to destroy certain occurrences of local alignment. As the number of rounds grows, these alignment conditions become increasingly restrictive and for 6 rounds we were already unable to exploit it. For this reason we believe the triplet weight of the critical 9-round propagation trails to be much higher than 48.

We once more indicate that the function of our propagation investigations is to support the choice of the rotation constants and the verification that 11 rounds are sufficient, not to give any proof of security. The security will eventually be based on the inability of cryptologists to find exploitable weaknesses.

Attacks can be devised where part of the key is known. The knowledge of some key material can be exploited to fix part of a differential trail. The large diffusion ensures that, already after two rounds, the unknown part of the key is diffused over the complete encryption state. Squeezing off more than a single round requires the knowledge of too many key bits to be a threat to K-security.

7.9 BaseKing

7.9.1 Specification

BaseKing is a block cipher with the self-reciprocal structure and is designed to be hermetic and K-secure with respect to sound initialization mappings. It is specified by

1. $n_{\mathrm{b}} = 192$,
2. $\mu = \mu_{16}$,
3. $\pi_1 : (0, 8, 1, 15, 5, 10, 7, 6, 13, 14, 2, 3)$ and
   $\pi_2 : (13, 14, 2, 3, 10, 9, 6, 11, 1, 15, 8, 0)$,
4. encryption: 11 rounds, a single output transformation and μ,
5. decryption: 11 rounds, a single output transformation and μ.

The block and key length impose that the triplet weight of 9-round propagation trails must exceed 96. From some early experiments it was concluded that this could not be realized by π permutations with only three rotation constants. Therefore, the π permutations act on 12 16-bit words, allowing the adoption of μ_{16} which is easily implementable in software. The rotation constants have been determined after a coarse analysis of their most elementary interaction (combined addition) and may be susceptible of improvement. The hardware and software implementation aspects of BaseKing are almost identical to those of 3-Way. In most implementations, encryption with BaseKing requires the same effort per bit as with 3-Way. In hardware, the doubling of the block length with respect to 3-Way allows a multiplication of the encryption speed by a factor close to two.

The most important advantage of BaseKing over 3-Way is its large block length. Because of this, BaseKing can be used as the main building block in the elegant and efficient block cipher based cryptographic hash function and checksum schemes described and/or proposed by Bart Preneel in [84] and standardized by ISO in [54, 53]. In this way the different modes of BaseKing cover the complete spectrum of single-key encryption and hashing.

7.9.2 Decryption

For the inverse round constants in BaseKing we have $\mathrm{c}'^{j} = \mu_{16}(\theta(\mathrm{c}^{11-j}))$. It can be seen that

$$\mathrm{c}'^{j}(x) = (x^{2h} + x^{3h} + x^{8h} + x^{9h})q'^{j}(x) \ , \tag{7.48}$$

with ${q'}_j(x)$ given by

$$ {q'}_j(x) = (1 + x^2 + x^3 + x^7)x^{-j} \bmod (1 + x^4 + x^8) \ . \tag{7.49} $$

In a cryptographic finite state machine the encryption and decryption round constants can be generated with two simple linear feedback shift registers and be applied using the same connection circuitry.

7.9.3 Propagation analysis

Table 7.6 is the (partial) reduced triplet weight distribution table of λ for BaseKing. Since λ has a rotational symmetry over the 16-bit subblocks, all entries of Table 7.6 would be a multiple of 16. This common factor has been removed.

Because of its key and block length of 192 bits, BaseKing is in principle a much more ambitious design as 3-Way. The required triplet weight of a 9-round propagation trail is 96, or almost 11 per round. This is more than a factor 7 better than DES, i.e., a single application of the round function of BaseKing must be as effective as *seven* rounds of DES. However, in experiments with propagation trails for only a few rounds, the 12-component permutations π appeared to be very powerful in their task of disrupting locally propagating structures. These observations have given us a high degree of confidence in the assumption that the triplet weight of the critical 9-round propagation trails is significantly larger than 96.

7.9.4 Alternative software implementations

Both 3-Way and BaseKing require the handling of words that have a length smaller than 32, the common processor word length in modern computers. However, by a simple rearranging of the blocks both can be implemented using only 32-bit instructions (or any larger power of 2). We illustrate this for the case of BaseKing.

For BaseKing the operations on 16-bit words can be turned into operations on 32-bit words by encrypting the messages in blocks of 384 bits. In this alternative scheme γ and θ are substituted by their $n_{\mathrm{b}} = 384$ versions and μ_{16} is substituted by μ_{32}. The subblock size and the rotation constants of the π bit permutations are doubled. The key and the round constants are doubled in length by doubling every bit. The resulting cipher can be considered to be the parallel application of BaseKing to the 192 bits on the odd positions and the 192 bits on the even positions.

7.10 Filtered counter stream encryption

The ease of changing the cipher key for 3-Way and BaseKing allows the specification of a very simple filtered counter stream encryption scheme.

The state updating is governed by a linear feedback shift register of length n_{b}. The feedback polynomial must be primitive and needs to have a Hamming weight of only 3. The initial counter state and cipher key both depend on the parameter

	1	2	3	4	5	6	7
1	-	-	-	-	-	-	12
2	-	-	-	-	-	60	-
3	-	-	-	-	180	-	-
4	-	-	-	255	-	-	-
5	-	-	180	-	-	-	624
6	-	60	-	-	-	?	?
7	12	-	-	-	624	?	?
8	-	-	-	255	1282	?	?
9	-	-	44	509	17547	?	?
10	-	6	89	6087	88972	?	?
11	-	4	1254	27588	136398	?	?
12	-	150	5498	36569	4284	?	?
13	3	618	6060	689	35274	?	?
14	9	562	59	5474	167937	?	?
15	-	1	491	25142	690142	?	?
16	-	13	2064	103494	2228150	?	?
17	-	50	8886	324518	4206530	?	?
18	-	295	27539	563180	4083993	?	?
19	1	775	42592	479688	2533248	?	?
20	1	942	31118	233210	4435549	?	?
21	2	357	11037	346777	12752974	?	?

Table 7.6: Partial reduced (divided by 16) triplet weight distribution table of the linear transformation λ for BaseKing.

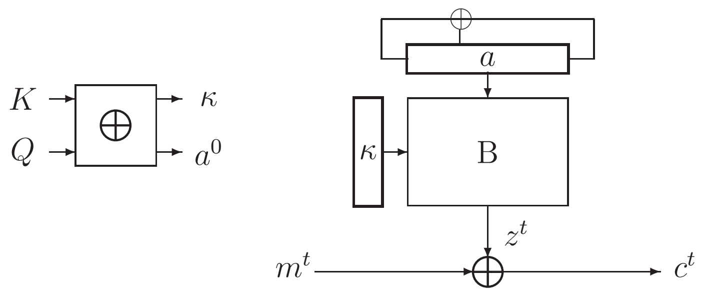


Figure 7.6: Proposed filtered counter encryption scheme using 3-WAY or BASEKING as a component.

Q and the key K. The dependence of the cipher key on the parameter Q prevents the choice of two parameter values that give rise to partly overlapping sequences. All n_{b} bits can be used for encryption, hence, $n_{\mathrm{s}} = n_{\mathrm{b}}$. This scheme is illustrated in Fig. 7.6 and described by

$$\begin{aligned} \kappa &= K + Q \ , \\ a^0 &= \tau_1(K) + Q \ , \\ a^{t+1}(x) &= x \cdot a^t(x) \bmod m(x) \ , \\ z^t &= \mathrm{B}[\kappa](a^t) \ . \end{aligned}$$

7.11 Conclusions

In this chapter we have presented a new self-reciprocal structure for block ciphers. This structure is quite general since it gives the designer a high degree of freedom in the choice of the specific transformations.

By adopting some specific step transformations, we have shown that it is possible to build block ciphers that have the unique property that differential and linear trails are governed by exactly the same equations.

We have shown that applying the wide trail strategy yields efficient and portable round transformations that are superior to the DES round function with respect to LC and DC.

The specific designs 3-WAY and BASEKING have been investigated with respect to their propagation behavior. We have not been able to determine the critical propagation trails for more than a few rounds. We think that it is an interesting opportunity for further research to improve the efficiency of the exhaustive propagation trail search procedures.

Chapter 8

Design of Stream/Hash Modules

8.1 Introduction

In Chapter 4 it was explained that both synchronous stream encryption and cryptographic hashing can be executed by a cryptographic finite state machine. The requirements for the updating transformation are so similar that the same finite state machine can be used as the core of both a synchronous stream cipher *and* of a hash function. This led us to the conception of a new type of cryptographic module that efficiently executes both cryptographic bulk operations necessary in the providing of security. After explaining the proposed general architecture for these stream/hash modules, we describe the hardware-oriented design Subterranean with its predecessors and derived schemes. The last part of this chapter is devoted to StepRightUp, a software-oriented design with very high efficiency.

8.2 Cipher architecture

The core of the stream/hash modules is a cryptographic finite state machine. The cryptographic hash function and the stream cipher are specified in terms of operations of this machine.

There are two internal registers, the n_{a}-bit state register containing the (hashing) state a and the buffer register containing b. There is an n_{i}-bit input denoted by p and an output denoted by z. During a cycle the contents of both registers are subject to one of several externally specified *operations*. We have

State operations: hold, reset and iterate;

Buffer operations: hold, reset and load.

The state **iterate** operation updates the state a according to a buffer-dependent state updating transformation denoted by $\rho[b]$. The buffer consists of a simple shift register with n_{i}-bit stages. The buffer **load** operation shifts an n_{i}-bit input into this register and is denoted by $\lambda[p]$. For both registers there are also self-explanatory **hold** and **reset** (to 0) operations.

The updating transformation $\rho[b]$ consists of the succession of a number of simple invertible transformations. As in the case of our block ciphers, each of these transformations has a specific contribution:

γ**:** a local **nonlinear** transformation, a variant of χ,

θ**:** a local linear transformation for **diffusion**,

π**:** a bit permutation for **bit dispersion**,

σ**:** bitwise addition of **buffer** bits,

ς**:** bitwise addition of a constant vector for **asymmetry**.

Unlike for block ciphers, these basic transformations are not restricted to have easily implementable inverses. In fact, the inverse of the simple and highly parallel transformation χ is most efficiently executed by a recursive sequential procedure. If θ is a linear shift-invariant transformation specified by a low-weight polynomial, the polynomial of θ^{-1} typically has high Hamming weight.

8.2.1 The cryptographic hash function

The *preparation phase* of the hash function is the segmentation and padding of the message (and the possible key) to the *input sequence* of n_i-bit blocks. The *iteration phase* is the initialization of the state register and the buffer and subsequent block by block loading of the input sequence into the buffer while the state is iterated. It concludes with some possible additional iterations and the generation of the hash result at the output. The iteration phase can be specified in a *sequence diagram*, listing the exact sequence of state and buffer operations.

Our main design concern with respect to the hash function is that of collision resistance. In our approach this is governed by the study of differential trails. A difference pattern in the input sequence gives rise to a differential trail in the internal state. The difference propagation with respect to two input sequences with different length can be modeled by including an initial difference vector in the state at the time step where the iteration phase of the shortest input sequence starts. The differential trail analysis consists of investigating whether:

1. The restriction weight of the potential differential trails is significantly higher than the degrees of freedom (in bits) available in the message blocks for any number of rounds. This must be the case for any initial difference pattern in state and buffer, resulting from a pair of partial hashing processes. This implies that it is impossible to *control* differential trails by manipulating the message blocks.

2. The restrictions imposed by the steps of differential trails with low-weight can be considered uncorrelated.

3. Clustering of differential trails cannot be exploited.

In our case the design process mainly consists of observing that these conditions fail to hold for some intermediate prototypes and convincing ourselves that they do hold for others.

The differential trails emerge in the interaction of the state updating transformation and the buffer load operation through the selected buffer blocks as defined in $\sigma[b]$. Hence, all these three components are very important in this respect.

Additionally, it must be investigated whether there are detectable correlations between the hash result and the input sequence. This analysis is governed by the study of linear trails.

8.2.2 The stream cipher

In stream encryption, the initialization mapping converts the key K and the public parameter Q into a number of initialization blocks. The cryptographic finite state machine is initialized by resetting buffer and state and subsequently loading the initialization blocks into the buffer. As in the case of the hash function, this can be specified in a sequence diagram. During cryptographic sequence generation an encrypting symbol z, consisting of q state bits, is presented at the output for every iteration of the state updating transformation. In some designs the buffer contents is constant during this process, in others the buffer may be updated, with its input consisting of part of the internal state.

The state bits are naturally subdivided into the bits that are part of the encrypting symbol z, called the *output* bits, and the complementary bits, called the *hidden* bits. For a given state updating transformation and a buffer load operation, the positions of the output bits in the internal state must be chosen very carefully to avoid reconstruction of hidden bits or buffer bits and detectable correlations between encrypting symbols. The most important tools in the analysis of these properties are linear trails.

By manipulating the public parameter Q, a cryptanalyst can apply a variant of differential cryptanalysis. With a linear initialization mapping, a difference in Q completely determines a difference in the initial state and buffer contents. From observing the differences in the encrypting symbols, state or buffer bits may be derived.

8.3 Subterranean

Subterranean is a stream/hash module designed to be implemented as a cryptographic coprocessor chip. It is not suited for software implementation and has therefore only a limited application domain. Subterranean has been realized as a prototype chip at IMEC. The Subterranean hash function is called Subhash, its stream cipher Substream.

8.3.1 Specification

The Subterranean cryptographic finite state machine has a 257-bit state register, a 256-bit buffer b, a 32-bit input p and a 16-bit output z. The buffer is divided into eight 32-bit blocks denoted by b^i with $0 < i \leq 8$. The buffer load operation $\lambda[p]$ is specified by

$$\begin{aligned} \lambda(b)|^1 &= p && \text{and} \\ \lambda(b)|^i &= b^{(i-1)} && \text{for } 1 < i \leq 8 \ , \end{aligned} \tag{8.1}$$

hence, the buffer consists of a shift register of width 32 and length 8.

The state updating transformation $\rho[b]$ is specified by:

$$\rho[b] = \pi \circ \sigma[b] \circ \theta \circ \varsigma \circ \gamma \ . \tag{8.2}$$

θ and γ are shift-invariant transformations defined by

$$\theta(a) = (1 + x^{-3} + x^{-8})a(x) \bmod 1 + x^{257} \ , \tag{8.3}$$

$$\gamma(a)|_i = a_i + (a_{i+1} + 1)a_{i+2} + 1 \ . \tag{8.4}$$

γ and θ are both invertible for $n_{\mathrm{a}} = 257$ since $\xi_\gamma = \{2\}$ and $\xi_\theta = \{7, 31\}$. The bit permutation π and the transformation ς are specified by

$$\pi(a)|_i = a_{12*i \bmod 257} \ , \tag{8.5}$$

$$\varsigma(a)|_i = a_i + \delta_0 \ , \tag{8.6}$$

for $0 \leq i < 257$. ς simply consists of complementing a single state bit. For the bitwise addition of buffer bits $\sigma[b]$ we have

$$\sigma(a)|_0 = a_0 \ , \tag{8.7}$$

$$\sigma(a)|_i = a_i + b^j_{i-1 \bmod 32} \text{ with } j = 1 + \left\lfloor \frac{i-1}{32} \right\rfloor \text{ for } 0 < i < 257. \tag{8.8}$$

Fig. 8.1 gives the computational graph of a single component of the state updating transformation, clearly indicating the five basic transformations. The 16-bit output symbol z^{t+1} consists of the 16 bits of the state a^t with indices

$$(11, 24, 37, 48, 60, 73, 84, 98, 117, 130, 143, 154, 168, 200, 235, 249) \ .$$

These indices have been chosen in such a way that:

- the $\rho[b]|_i$ of the output bits do not depend on output bits,
- the $\rho[b]|_i$ of the output bits depend on non-overlapping subsets of state bits,
- the $\rho[b]|_i$ of hidden bits depend on not more than a single output bit.

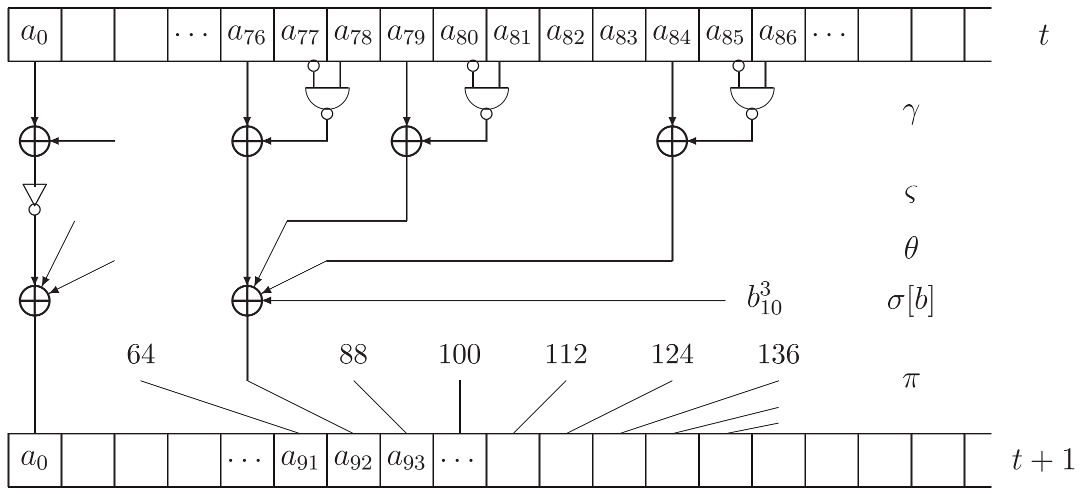


Figure 8.1: Computational graph of $\rho[b](a)|_{92}$.

Time step t	State	Buffer	Input	Output
$-6,\ldots,0$	hold	load	0	–
1	reset	load	p^t	–
$2,\ldots,\ell$	iterate	load	p^t	–
$\ell+1,\ldots,\ell+8$	iterate	load	0	–
$\ell+9,\ldots,\ell+8+\omega$	iterate	hold	–	$h_{t-\ell-9}$

Table 8.1: The sequence diagram of the iteration phase of SUBHASH. The hash result consists of the concatenation of h_0 to $h_{\omega-1}$.

SUBHASH

The arguments of SUBHASH are a message M, an optional key K and a parameter ω with $0 < \omega \leq 16$. The length of the key must be a multiple of 32 bits and may not exceed 256 bits. The length of the hash result is a multiple ω of 16 bits.

In the preparation phase M is converted into a string M' with a length that is a multiple of 32 by appending a single 1 followed by a number d of 0 bits with $0 \leq d < 32$. Subsequently, P is formed by

$$P = K \| M' \| K \ , \tag{8.9}$$

with $\|$ denoting concatenation. P can be seen as an array of ℓ 32-bit blocks p^i, i.e., $P = p^1 p^2 \ldots p^\ell$. The iteration phase of SUBHASH is specified in Table 8.1. The first 7 iterations are needed to reset the state and buffer to 0. Subsequently, the first block of P is loaded and the state updating iterations start. When the last block of P has left the buffer, ω additional iterations are performed. The output symbols of these iterations are concatenated to form the hash result.

The number of machine cycles to hash an ℓ-block input sequence is $\ell + \omega + 15$.

The invertibility of the state updating transformation allows the generation of a message to a given result by a meet-in-the-middle attack with a work factor of approximately 2^{128} applications of SUBHASH. This restricts the length of the hash

Time step t	State	Buffer	Input	Output
-31	reset	load	p^t	–
$-30, \ldots, -24$	hold	load	p^t	–
-23	iterate	load	p^t	–
$-22, \ldots, -16$	hold	load	p^t	–
$-15, \ldots, 0$	iterate	hold	–	–
$1, \ldots$	iterate	hold	–	z^t

Table 8.2: The sequence diagram of Substream.

result to 128 bits for a hermetic hash function, hence, no claims are made for $\omega > 8$. For $\omega \leq 8$, Subhash is claimed to be hermetic and, if there is a secret key, also K-secure.

Substream

Substream is initialized by first loading sixteen 32-bit initialization blocks p^{-31} to p^{-16} followed by sixteen additional *blank* iterations, i.e., without delivering encrypting symbols at the output. During cryptographic sequence generation the buffer operation is 'hold'.

Turning Substream into a stream encryption scheme takes the specification of an initialization mapping. For example, a 128-bit key K and a 128-bit parameter Q can be converted into the initialization blocks by

$$\begin{array}{rcccl} p^{-23}p^{-22}p^{-21}p^{-20} & = & p^{-19}p^{-18}p^{-17}p^{-16} & = & K \ , \\ p^{-31}p^{-30}p^{-29}p^{-28} & = & p^{-27}p^{-26}p^{-25}p^{-24} & = & Q \ . \end{array} \tag{8.10}$$

The sequence diagram of Substream is given in Table 8.2. It can be seen that the first 8 initialization blocks determine the initial state at time step $t = -23$ and the following 8 initialization blocks determine the buffer contents. After 16 additional "blank" iterations, the generation of encrypting symbols starts.

Substream is claimed to be hermetic and K-secure with respect to all sound initialization mappings with a keylength of 128 bits and a parameter length of 128 bits.

8.3.2 Structural analysis

In Fig. 8.1 it can be seen that every bit of a state a^t depends on 9 bits of its predecessor state a^{t-1}. Thanks to the particular choice of π these bits depend in their turn on non-overlapping subsets of bits of a^{t-2}, hence, every bit of a^t depends on 81 bits of a^{t-2}. Alternatively, every state bit occurs in 9 components of $\rho[b]$ and in 81 components of $\rho[b] \circ \rho[b]$.

The inverse of the state updating transformation differs strongly from the transformation itself. The polynomial of the inverse of θ has a Hamming weight of 127. This severely complicates reverse calculation.

The state updating transformation can be split up into the nonlinear transformation γ and an affine transformation consisting of the other four transformations. The effect of this affine transformation in differential and linear trails is completely determined by $\pi \circ \theta$. The absence of a self-reciprocity condition for θ and π allows the realization of much better long-term diffusion than in the case of a block cipher. The relatively small diffusion factor 3 of θ is compensated by the high effectiveness of the bit permutation π in dispersing the individual trail bits all over the state.

In the following sections we give our design motivations for both cryptographic schemes of SUBTERRANEAN .

Motivations for SUBSTREAM

The correlations between linear combinations of output bits and linear combinations of state bits $w^t a$ in previous time steps can be investigated using linear trails terminating in specific selection vectors. The $2^{16}-1$ different nonzero linear combinations of output bits of a given time step correspond to the selection vectors that are restricted to 0 outside the 16 output bit indices.

The combination of the high multiple-round diffusion of $\rho[b]$ and the specific choice of the output bit indices should cause all linear trails terminating in one of the $2^{16}-1$ selection vectors to have very high weight, even for a few (4 or 5) number of rounds. This results in only small correlations with initial selections that all have high Hamming weight.

Nonzero correlations between linear combinations of output bits of different time steps require linear trails with initial and terminal selection vectors that belong to the allowed set of $2^{16}-1$ selection vectors. This imposes severe restrictions that should enforce the resulting linear trails to have intermediate selection vectors with very high correlation weight. This results in very small correlations that only become nonzero for linear combinations of output bits that are separated by a substantial number of time steps.

Assume for simplicity that the buffer contents is known and the state is unknown to the cryptanalyst. The knowledge of the 16 bits of the encrypting symbol z^t restricts a^t to 2^{257-16} values. The knowledge of z^{t+1} can be converted into 16 equations for the bits of a^t using $\rho[b]$. In this stage a^t is restricted to 2^{257-32} values. By doing this iteratively for 15 more rounds, a sufficient number of equations is collected to uniquely determine a^t. The problem with this approach is that the complexity of these equations grows so dramatically with the number of rounds that even trying to store them is problematic. We believe it to be highly unlikely that these complex sets of highly nonlinear Boolean equations could be solved by a method more efficient than exhaustive search over the space of 128-bit keys.

Instead of working with the exact equations, one could approximate them by a simpler, more manageable set of equations that only hold in a majority of cases. By adopting the set of linear (in fact affine) equations in this approach, this corresponds to linear cryptanalysis. The small values of the multiple-round correlations illustrate the volatility of the knowledge of specific state bits at given time steps. One could imagine that other manageable sets of equations must exist. However,

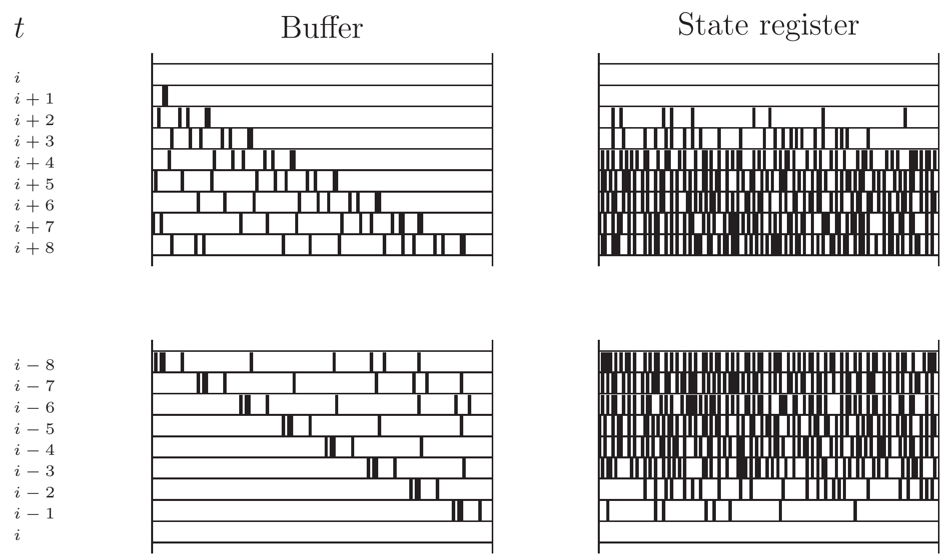


Figure 8.2: Difference propagations in Subhash (1:black,0:white). Top: initial effect of input differences. Bottom: difference pattern of an internal collision.

we believe that such sets do not exist for high-diffusion bit-level designs such as Subterranean.

The non-secrecy of the public parameter Q allows the cryptanalyst to observe different encrypting sequences resulting from initial states and buffer contents that have a known relation. If the initialization mapping is linear, as in our example, the bitwise difference in Q completely determines the bitwise difference in the initialization blocks p and therefore in a^{-16} and b. If the cryptanalyst can actually choose the public n_{p}-bit parameter Q, he can choose from $2^{n_{\mathrm{p}}} - 1$ possible difference patterns in a^{-16} and b.

In the 16 blank iterations the difference vector in a^{-16} follows a differential trail to a difference pattern in a^0. Because of the large diffusion, the probability that there are 16-round differential trails with a restriction weight below 257 is extremely small. Hence, the difference pattern in a^{-16} does not give any usable information on the difference pattern in a^0. This excludes the exploitation of relations between two different initializations to find correlations between the two encrypting streams or to derive the value of buffer or hidden state bits.

The clustering of linear or differential trails should be prevented by the powerful dispersion caused by π.

Motivations for Subhash

Most of this section is devoted to the collision-resistance of Subhash.

From the sequence diagram of Subhash, Table 8.1, it follows that the hash result is completely determined by $a^{\ell+8}$, called the *terminal* hashing state. The converse is however not true, and pairs of input sequences may be found with different terminal hashing states both consistent with the same hash result. The collision actually occurs in the derivation of the hash results from the hashing states and is called *terminal*. Collisions in which two different input sequences give rise to equal states and buffer contents at a certain point are called *internal*.

In SUBHASH every time step a new input block is shifted into the buffer. An input block resides in the buffer during 8 time steps, every time affecting the hashing state through the state updating transformation. This is reflected in the difference propagation in buffer and hashing state. Two examples of difference propagation patterns in SUBHASH are given in Fig. 8.2.

The top of this figure displays the difference propagation resulting from a pair of input sequences that are identical in the first i blocks and have differences starting from block p^{i+1}. The bottom displays a difference propagation that corresponds to an internal collision taking place at time step i. In this collision it is not necessary that the two input sequences of the pair have equal length. If the difference in length consists of d blocks, the index i in the difference propagation diagram corresponds to the index $i' = i - d$ for the longest of the two input sequences. At $i = 0$, $i' = d$ and the first d blocks of the longest input sequence have already been shifted into the buffer. This is modeled by a nonzero initial ($i = 0$) difference pattern in the buffer and hashing state.

The difference patterns in the buffer induce a differential trail in the hashing state. This differential trail depends in two ways on the input. Directly, the difference pattern in the input has an instantaneous impact on the difference pattern in the successor of the hashing state. Indirectly, the absolute values of the input bits influence the absolute values of the state bits and hence the evolution of the differential trail.

From the given point of view, the problem of generating collisions, both internal and terminal, is one of differential trail control by input block manipulation. An internal collision corresponds to a differential trail with a *dead-end*. This control, while theoretically possible, is assumed to be infeasible because of the large diffusion and distributed nonlinearity of the state updating transformation combined with the fact that every input block resides in the buffer for 8 time steps.

A possible strategy for finding internal collisions would be to look for an input difference pattern that can give rise to a dead-end differential trail Ω with a low restriction weight. Such a differential trail would have a small number of rounds and/or difference vectors with low restriction weight. If the restrictions can be considered independent, a collision would be found after trying $2^{\mathrm{w_r}(\Omega)-1}$ different pairs of input sequences with the specific difference. For SUBHASH the structure of the state updating transformation and its interaction with the buffer should eliminate the existence of such differential trails with low restriction weight.

Since the state updating transformation is invertible, it is possible to reverse the hashing process, starting from the terminal hashing state and terminating in an intermediate hashing state and buffer contents. This could be used to find colliding input *tails*. This "reduces" the problem of generating a collision to finding an input *head* that converts the initial state into a given intermediate hashing state with matching intermediate buffer contents. The control required in this operation is comparable to that needed in finding collisions themselves.

Another potential weakness would be in *fixed points*. A fixed point consists of a hashing state a and a buffer contents b that are left unchanged by the state updating transformation and the buffer loading operation for some input block p_f. Clearly, the buffer contents must consist of 8 identical blocks that are equal to p_f. An input that exhibits this fixed point during the hashing process must have some sequence of at least 8 blocks p_f. Adding some extra blocks p_f to this subsequence of the input sequence does not affect the hash result, and hence, gives rise to a collision. This concept of fixed points can be generalized to multiple time steps, where the application of more than a single iteration leaves the internal state and the buffer contents unchanged for some input sequence. The difficulty in generating this kind of collisions lies in finding an input that converts the initial hashing state into a fixed point. In our view this is of a difficulty comparable to finding an input sequence that gives rise to a prespecified terminal hashing state.

8.3.3 Chip Realization

The Subterranean module has been implemented as an integrated circuit in the thesis project of student Gert Peeters [83] of the institute KIH De Nayer in Mechelen. The work was supervised by Prof. Luc Claesen and Marc Genoe of IMEC and myself. Additionally, Luc Claesen built a real-time video encryption/decryption demonstrator. In this demonstrator a PAL color composite video signal, generated by a camera, is digitized, encrypted by a Subterranean chip, sent over a “wire channel”, decrypted with another Subterranean chip, converted back to analog and shown on a monitor. If the two Subterranean chips have been loaded with the same keyword and are in synchronism, the monitor displays the image recorded by the camera. If synchronism is lost or the keywords of the two chips differ, the only thing that can be seen is white noise.

The implementation technology has been the 2.4μm CMOS standard cell technology of MIETEC. The active area is 5.00mm x 7.01mm (35mm^2). The area including bonding pads is 6.00mm x 7.85mm (47mm^2). The correct operation of this chip has been measured up to a clock period of 56 nsec on a Tektronix LV-500 tester. This is a clock frequency of 17.8 MHz and corresponds to a stream encryption/decryption throughput of 286 Mbit/s, and a hashing speed of 572 Mbit/s. To our knowledge, this makes Subhash the cryptographic hash function with the fastest implementation in the world. The applied technology is in fact conservative and the use of a more advanced technology will without a doubt push encryption and hashing rates into the Gbit/s range.

The design has been realized in such a way that it fits into a 40 pin package. From the sequence diagrams of Substream and Subhash it can be seen that no simultaneous *input* and *output* occurs. All input and output to the chip goes through a 32-bit bus. It provides a parallel 32-bit input at the rate of the overall clock for the (buffer) load operation. If the buffer operation is hold, it provides a 16-bit input x and a 16-bit output y, likewise at the rate of the overall clock. The output consists of the bitwise addition of the 16-bit input of the previous time step and the encrypting symbol z, hence, the stream encryption is performed on chip. The encrypting symbol z can be obtained at the output by applying 0 at the input.

To avoid the possibility of reading out the state and buffer registers, there is no additional internal test circuitry in the form of scan pads. The only external access to the internal registers are the output bits. In general this would have serious consequences for the testability, which is very important in the elimination of malfunctioning chips after fabrication. However, in the case of Subterranean this is no problem. Thanks to the high diffusion of the finite state machine, the effects of *stuck-at* (0 or 1) faults propagate very fast over all bits in the state and consequently manifest themselves at the output of the chip. A set of test sequences (with a maximum length of 43 iteration cycles) has been determined that allow for a 100% testability of all "stuck-at" faults at the inputs and outputs of the standard cells.

8.4 Designs preceding Subterranean

The design of Subterranean was preceded and influenced by two other design proposals. Our first proposal has been presented in [10] and is a synchronous stream cipher consisting of a specific cellular automaton configuration. Our second design proposal has been presented in [11] and is a cryptographic hash function called Cellhash. Subhash can be seen as an improved version of this hash function.

8.4.1 Sequence generation by cellular automata

In [104] Stephen Wolfram proposed to generate cryptographic sequences using cellular automata. A cellular automaton is an automaton with a shift-invariant state updating transformation and can therefore be described by a local map. In his paper Wolfram concentrated on two one-dimensional binary cellular automata both specified by nonlinear local maps with neighborhood $\nu = \{-1, 0, 1\}$. He proposed to build a cryptographic sequence generator from a finite cellular automaton with periodic boundary conditions, with the output consisting of a single specific state bit per iteration.

The combination of simplicity, efficiency and potential for cryptographic strength made this proposal an attractive subject for further research. After the suggestion of our promotors to study this subject, we embarked with running a number of statistical tests.

From our experiments we discovered that for one of the two local maps the cellular automata with odd length have an expected cycle length in the order of magnitude of the total number of states. From this we deduced, and later proved, that the corresponding shift-invariant transformation must be invertible for odd lengths, a fact that apparently was overlooked by Wolfram. This local map corresponds to a variant of the shift-invariant transformation that we denote by χ. Still, exhaustive cycling tests showed that the state-transition diagrams of these cellular automata exhibit large numbers of undesirable short cycles. We discovered that this could be attributed to the small diffusion in the state updating transformation.

This triggered our search for cellular automata with a better cyclic behavior. An exhaustive search was performed for nonlinear globally invertible shift-invariant transformations with neighborhoods $\nu = \{-1, 0, 1, 2\}$ and $\nu = \{-1, 0, 1, 2, 3\}$. The results of this search were 7 cellular automata with experimental cycle distributions completely in accordance with the distribution expected from a uniformly chosen permutation. Of these 7 shift-invariant transformations, the first 5 have a simple description in terms of complementing landscapes and their invertibility can easily be proved using seeds and leaps. The two remaining ones have a considerably more complex description and initially we were unable to prove their invertibility. One of them, denoted by ϕ_7, is described by local map

$$\phi_7(a)|_0 = a_{-1} + (a_1 + 1)(a_0 + a_2) + (a_2 + 1)a_3 \ . \tag{8.11}$$

Later we found that this shift-invariant transformation could be factored in two shift-invariant transformations with neighborhood size 3, hence $\phi_7 = \phi_\alpha \circ \phi_\beta$ with the two component transformations given by

$$\phi_\alpha|_0 = a_{-1} + a_0 + a_1 \ , \tag{8.12}$$

$$\phi_\beta|_0 = a_0 + (a_1 + 1)a_2 \ . \tag{8.13}$$

Clearly, $\phi_\beta = \chi$ and ϕ_α is a linear mapping with polynomial $x^{-1} + 1 + x$. The local map of ϕ_6, the other complicated mapping, simply consists of the complement of the local map of ϕ_7.

During our research, Willi Meier and Othmar Staffelbach described a powerful attack on the schemes proposed by Wolfram in [76]. Their attack consists of a simple and efficient algorithm to reconstruct the initial internal state from the output sequence. For example, the expected work factor for reconstructing the internal state of a cellular automaton with length 200 is equivalent to that of exhaustive key search with a key length of only 15 bits for the "best" of the two local maps.

Upon investigation we found that their attack also works for each of the first five local maps we proposed. For ϕ_6 and ϕ_7 the attack loses its efficiency, thanks to the much larger diffusion of these rules. We found no way of generalizing the attack for these local maps.

The Meier and Staffelbach attack depends crucially on the overlap of dependencies of subsequent output bits, i.e., every output bit is partially determined by the previous output bit. This is no longer the case if the state updating transformation is altered by also including an extra cyclic shift τ_r. This can also be modeled by a cyclic shift of the output bit position for every iteration. If r is chosen near the square root of the length of the cellular automaton, the output bits are distant both in time and space. Figure 8.3 illustrates this for an automaton length of 127 and a rotation constant of 11.

In [10] these ideas resulted in a concrete proposal for a synchronous stream cipher. Its finite state machine consists of a cellular automaton with length 127 and state updating transformation $\tau_{11} \circ \phi_7$. The output of an iteration consists of a single state bit at position 0.

The combination of the large diffusion of the state updating transformation and the large distances between the output bits should prevent attacks significantly more

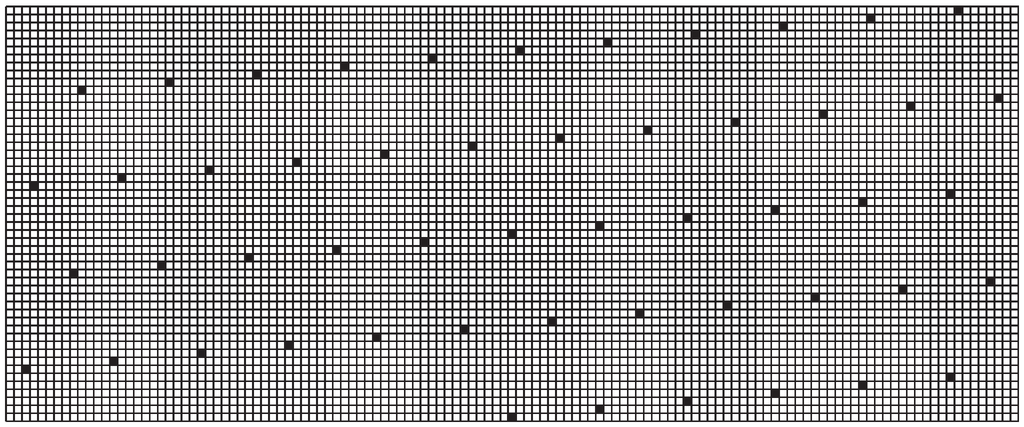


Figure 8.3: The samples of the space-time pattern that form the output sequence in a configuration with length 127 and $r = 11$.

efficient than exhaustive key search of the space of initial states. No attacks have been reported on this cipher.

In retrospect the cellular automaton proposal already contained most of the elements present in our later designs. Its state updating transformation is invertible and composed of different simple transformations. Each of these transformations has its own specific contribution:

- χ for nonlinearity,
- a linear shift-invariant transformation for diffusion,
- a bit permutation (cyclic shift) for the bit dispersion.

8.4.2 Cellhash

Subterranean is in fact both an extension and an improvement of an earlier hash function design, called Cellhash. The improvements consist mainly of an adaptation of the state updating transformation and a simplification of the global hashing process.

In Cellhash the state updating transformation is also given by the succession of 5 transformations, as expressed in (8.2). The only differences with Subterranean are in θ and π. For Cellhash the polynomial corresponding to θ is given by

$$x^{-3} + 1 + x^3 \ . \tag{8.14}$$

This is the simplest polynomial which causes every component of the state updating transformation to depend on 9 bits. The bit permutation π for Cellhash is specified by

$$\pi(a)|_i = a_{10*i \bmod 257} \ . \tag{8.15}$$

The factor 10 was the result of the following process. With respect to 2-round dispersion as discussed on p. 8.3.2, it should be larger than 8. In a comparison

	1	2	3	4	5
1	-	-	1	-	-
2	-	1	-	1	-
3	-	-	3	-	$\approx n$
4	-	1	-	$\approx n/2$	-
5	-	-	5	-	$\approx 3n$

	1	2	3	4	5
1	-	-	1	-	-
2	-	-	-	3	-
3	-	-	1	-	9
4	-	-	-	7	-
5	-	-	2	-	30

Table 8.3: Partial reduced Hamming weight distribution table of multiplication by $x^{-3} + 1 + x^3$ (left) and $1 + x^3 + x^8$ (right) modulo $1 + x^n$.

between the (simple) candidates 9, 10, 11 and 12, the factor 10 gave rise to the least amount of short-term regularity.

The adaptation of θ was motivated by the better local diffusion properties of the new polynomial. The improvement becomes apparent upon inspection of the partial reduced Hamming weight distribution tables given in Table 8.3. With the new polynomial the factor 10 in π had to be augmented. The factor 12 was chosen from a small set of simple candidates, realizing the least amount of short-term regularity.

Cellhash is only defined as a hash function without a key. In the padding and segmentation operation the message is converted into a sequence of at least eight 32-bit blocks. Subsequently, the first 8 input blocks are loaded into the buffer and the hashing state is reset to 0. During the hashing process a new block is loaded every iteration. After the last input block has been loaded, the 7 first input blocks are repeated. At this point the 257-bit contents of the state register is taken to be the hash result.

The repeating of the first 7 input blocks at the end of the hashing process requires their temporary storage and complicates the description. Therefore this has been replaced by a simpler scheme in Subhash. In a hardware implementation of Cellhash the reading out of the hash result requires the need for a *state read* operation. For Subterranean abuses of the presence of such an operation could compromise the security of Substream. Therefore the hash result has been defined as a sequence of outputs obtained during a number of additional iterations, removing the need for this operation.

To our knowledge there has been no successful cryptanalysis of Cellhash since its publication. Still, the invertibility of the state updating transformation allows the generation of an input sequence with a prespecified hash result with an expected work factor of about 2^{128} applications of Cellhash. Although this is not a real threat, to be hermetic this work factor should correspond to approximately 2^{256} applications of Cellhash.

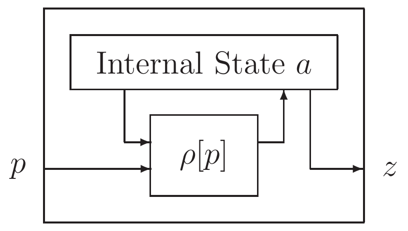


Figure 8.4: Block diagram of the Jam finite state machine.

8.5 Designs derived from Subterranean

In this section we present two designs that can be seen as simplifications of Subterranean. The first design is a hardware-oriented cryptographic sequence generator, called Jam. This component is designed in the context of the Eureka project VADIS. ESAT-COSIC was engaged in this project in the working group systems, division conditional access. In this group we proposed Jam to be used as the cryptographic sequence generator of the video/audio scrambling module. Jam was first specified in [25] but was never officially published.

The second design is a cryptographic hash function, called Boognish. It was an attempt to design a portable cryptographic hash function with the same structure as Subhash. Boognish was presented in the rump-session of Eurocrypt '93. There is an internal report [26] describing the function.

8.5.1 Jam

The Jam finite state machine can be considered a downscaled and simplified version of the Subterranean finite state machine. It is given here mainly to illustrate the scalability of the Subterranean design.

Since its internal state a consists of only 61 bits, it cannot be used in applications that require a high level of security. There is a 4-bit input p, a 4-bit output w and no buffer. A block scheme of the Jam finite state machine is given in Fig. 8.4.

The state updating transformation $\rho[p]$ is composed of 5 transformations, i.e.,

$$\rho[p] = \pi \circ \theta \circ \sigma[p] \circ \varsigma \circ \gamma \ , \tag{8.16}$$

γ and ς are the same as for Subterranean and are specified respectively by (8.4) and (8.6). The linear shift-invariant transformation θ is specified by the polynomial

$$1 + x^{-3} + x^{-6} \ , \tag{8.17}$$

and the bit permutation π by

$$\pi(a)|_i = a_{9*i \bmod 61} \ . \tag{8.18}$$

The transformation $\sigma[p]$ is specified by

$$\begin{array}{lcll} \sigma_1 & = & a_1 + p_0 \ , & \\ \sigma_{18} & = & a_{18} + p_1 \ , & \\ \sigma_{33} & = & a_{33} + p_2 \ , & \\ \sigma_{50} & = & a_{50} + p_3 \ , & \\ \sigma_i & = & a_i & \text{for } i \notin \{1, 18, 33, 50\} \ . \end{array} \tag{8.19}$$

Finally, the positions of the output bits are

$$\{1, 18, 33, 50\} \ . \tag{8.20}$$

There are three operations for the internal state: reset (to 0), hold and iterate. Initialization consists of resetting the internal state and subsequently iterating the state updating transformation 15 times while loading 4-bit initialization blocks. After 16 additional blank iterations the initialization process is finished and the generator is ready for use.

We propose to use the Jam cryptographic sequence generator with a 60-bit key K and a 15-bit parameter Q. In the initialization mapping the initialization vector is obtained by bitwise addition of the key K and the concatenation of 4 15-bit strings equal to Q.

The Jam finite state machine is designed to fit on a small area. Its components are 61 flip-flops, 61 nand (or nor) gates and 157 exor gates and its expected area in 1.2 μ CMOS technology is smaller than a single square mm.

8.5.2 Boognish

The Boognish hash function is defined in terms of operations on 32-bit *words*. The Boognish finite state machine has a single-word input p, a single-word output z and a state register containing 5 words $a_{(0)}$ to $a_{(4)}$. The buffer is a shift register containing 25 words b^1 to b^{25}.

The state updating transformation $\rho[b]$ is composed of 5 transformations:

$$\rho[b] = \sigma[b] \circ \pi \circ \theta \circ \pi \circ \gamma \ . \tag{8.21}$$

These transformations are most easily described in terms of wordwise operations. We have (indices i must be taken modulo 5):

$$\gamma(a)|_i = a_i + (a_{i-1} + \bar{0})a_{i-2} \ , \tag{8.22}$$

with $\bar{0}$ the all-1 32-bit word. For θ, we have

$$\theta(a)|_i = a_i + a_{i-1} + a_{i-2} \ , \tag{8.23}$$

i.e., the polynomial corresponding to θ is $1 + x + x^2$. The bit permutation π consists of cyclic word shifts and can be specified (as π_1 and π_2 in Chapter 7) by the 5-tuple of shift constants:

$$(0, 1, -3, 6, 8) \ . \tag{8.24}$$

Time step t	State	Buffer	Input	Output
$-23, \ldots, 0$	hold	load	0	–
1	reset	load	p^t	–
$2, \ldots, \ell$	iterate	load	p^t	–
$\ell+1, \ldots, \ell+25$	iterate	load	0	–
$\ell+26, \ldots, \ell+26+\omega$	iterate	hold	–	$h_{t-\ell-26}$

Table 8.4: The sequence diagram of the iteration phase of Boognish. The hash result consists of the concatenation of h_0 to $h_{\omega-1}$.

$\sigma[b]$ consists of the bitwise addition of both the buffer bits and the constant (for asymmetry):

$$\begin{aligned} \sigma(a)|_0 &= a_0 + 10204081_{\text{hex}} \ , \\ \sigma(a)|_1 &= a_1 + b^1 \ , \\ \sigma(a)|_2 &= a_2 + b^9 \ , \\ \sigma(a)|_3 &= a_3 + b^{17} \ , \\ \sigma(a)|_4 &= a_4 + b^{25} \ . \end{aligned} \tag{8.25}$$

The output z is given by a_0.

The arguments of Boognish are a message M, an optional key K and a parameter ω with $0 < \omega \leq 5$. The length of the key must be a multiple of 32 bits and may not exceed 160 bits. The hash result consists of the concatenation of ω words. This is formally expressed by

$$h = \text{Boognish}(M, K, \omega) \ . \tag{8.26}$$

In the preparation phase M and K are concatenated, message first. The resulting string is padded by appending one 1-bit, followed by a number d of 0-bits with $0 \leq d < 32$. This results in the input sequence P, that can be seen as an array of ℓ 32-bit words $p^1p^2 \ldots p^\ell$. The iteration phase of Boognish is specified in Table 8.4. The total number of machine iterations is $\ell + \omega + 48$.

The basic operations of the round transformation can be implemented by bitwise Boolean operations and bitwise shift operations. The π shift constants have been chosen to optimize the short-term diffusion with respect to difference propagation. Every word p^i of the input sequence affects the state updating transformation 4 times, spread over 25 iterations. The collision-resistance should be realized by the fact that a single input bit affects the internal state over a range of 25 iterations of the highly effective state updating transformation.

Weak points

In retrospect, we came to the conclusion that the length of the internal state of Boognish is too small, giving rise to some potentially weak points.

The most important consequence is of an external nature. Because of the invertibility of the state updating transformation, the hash function cannot be hermetic if the hashing state is shorter than twice the length of the hash result. Hence, BOOGNISH can only be hermetic for $\omega < 3$. For $\omega = 5$ the expected work factor of generating an input with a given hash result by a general meet-in-the-middle attack is approximately equal to that of generating a collision with a birthday attack. Both are of the order of 2^{80} executions of BOOGNISH .

The small number of words in the hashing state limits the diffusion properties of the state updating transformation. In the case of SUBTERRANEAN the polynomial corresponding to the inverse of θ has a large Hamming weight and the description of the inverse of γ is complicated. This seriously hampers calculations that involve the inverse of $\rho[b]$ and the construction of low-weight differential and linear trails. For BOOGNISH the polynomial corresponding to the inverse of θ is given by $1 + x^3 + x^4$ and the inverse of γ has a simple expression. Hence, the inverse of $\rho[b]$ only has a complexity that is comparable to that of $\rho[b]$ itself.

8.6 STEPRIGHTUP

STEPRIGHTUP is a stream/hash module that is designed to be very efficient in software implementations. Its basic operations are on 32-bit words, as in BOOGNISH. In comparison with BOOGNISH, the work factor per encrypted or hashed bit has been divided by 2 while its claimed security level is much higher. To achieve this, it was necessary to have a very long internal state and buffer. Another important new element is the introduction of linear feedback in the buffer shift register, similar to that applied in the compression function of NIST-SHA [39, 40].

The price paid for the low per-bit work factor is a relatively high fixed computational overhead for every execution of the hash function. This makes the STEPRIGHTUP hash function less suited for the hashing of messages shorter than the equivalent of a typewritten page. For the stream cipher it results in a relatively long initialization procedure. Hence, in applications where speed is critical, too frequent resynchronization should be avoided.

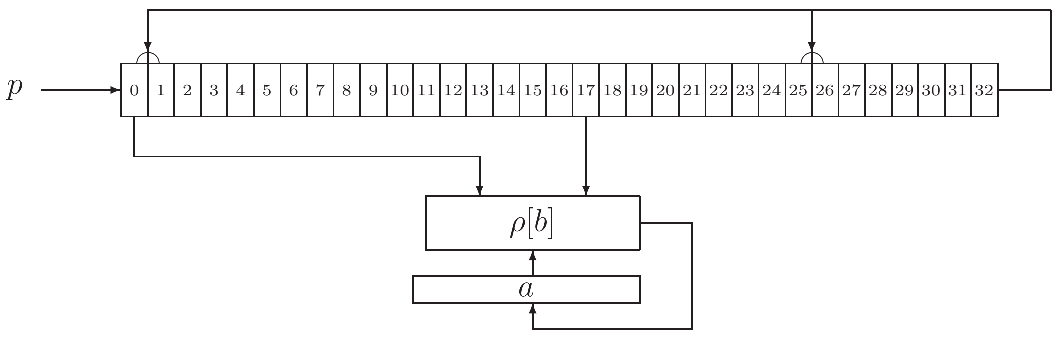
After specifying the STEPRIGHTUP hash function and stream cipher, we discuss the particular design strategy and the implementation aspects. The STEPRIGHTUP design has not been published before.

8.6.1 Specification

The 544-bit state a of the STEPRIGHTUP finite state machine consists of seventeen 32-bit words a_0 to a_{16}. The 8448-bit buffer is a linear feedback shift register with 33 *stages*, each consisting of eight 32-bit words. An 8-word stage is denoted by b^j and its words by b^j_i. Both stages and words are indexed starting from 0.

There are three possible *modes* for the STEPRIGHTUP finite state machine: *reset, input* and *auto(nomous)*. In reset mode all state and buffer bits are set to 0. The input and auto modes differ only slightly and are specified in this section. Both consist of a simultaneous buffer load operation and a state update operation.

For the buffer load operation λ, in general $\lambda(b)|^j = b^{j-1}$ except for $j \in \{0, 1, 26\}$.



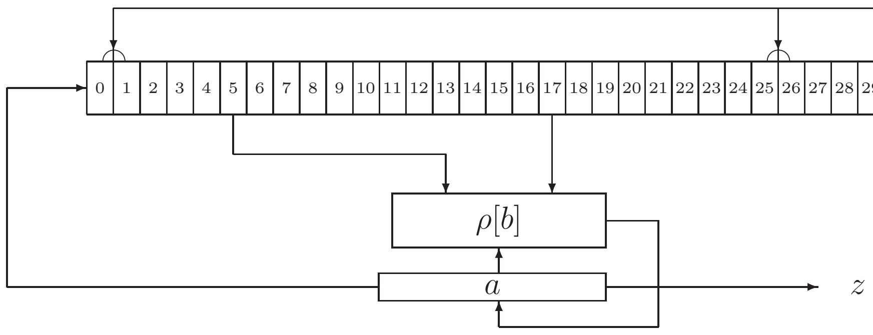


Figure 8.5: Input (above) and auto (below) modes of StepRightUp.

For these stages we have

$$\begin{aligned}\lambda(b)|^0 &= p \ ,\\ \lambda(b)|^1 &= b^0 + b^{32} \ ,\\ \lambda(b)|_i^{26} &= b_i^{25} + b_{i+1 \bmod 8}^{32} \ ,\end{aligned} \tag{8.27}$$

For $0 \leq i < 8$. In auto mode the input block p consists of a part of the internal state a. Its 8 component words are given by

$$p_i = a_{2i+2} \ , \tag{8.28}$$

for $0 \leq i < 8$.

The state updating transformation $\rho[b]$ is specified by:

$$\rho[b] = \sigma[b] \circ \theta \circ \pi \circ \gamma \ . \tag{8.29}$$

θ and γ are shift-invariant transformations defined by

$$\theta(a)|_i = a_i + a_{i+1} + a_{i+4} \ , \tag{8.30}$$

$$\gamma(a)|_i = a_i + (a_{i+1} + \bar{0})a_{i+2} + \bar{0} \ . \tag{8.31}$$

The permutation π combines cyclic word shifts and a permutation of the words relative to one another. We have

$$\pi(a)|_i = \tau_k(a_j) \ , \tag{8.32}$$

with

$$\begin{aligned} j &= 7i & \mathrm{mod} 17 \quad & \text{and}\\ k &= i(i+1)/2 & \mathrm{mod} 32 \ . & \end{aligned} \tag{8.33}$$

The bitwise addition of the buffer bits and the constant is given by

$$\begin{array}{lcl}\sigma(a)|_0 & = & a_0 \quad + \quad 00000001_{\text{hex}} \ , \\ \sigma(a)|_{2i+1} & = & a_{2i+1} \quad + \quad b_i^k \ , \\ \sigma(a)|_{16-2i} & = & a_{16-2i} \quad + \quad b_i^{17} \ , \end{array} \tag{8.34}$$

for $0 \leq i < 8$. In the input mode $k = 0$ and in the auto mode $k = 5$. In the auto mode the output z consists of 8 words given by

$$z_i = a_{2i+1} \ . \tag{8.35}$$

The input and auto modes of the StepRightUp module are illustrated in Fig. 8.5.

The StepRightUp hash function

The arguments of the StepRightUp hash function are a message M and an optional 256-bit key K. The length of the hash result is 256 bits. In the preparation phase M is converted into a string M' with a length that is a multiple of 256 by appending a single 1 followed by a number d of 0-bits with $0 \leq d < 256$. Subsequently, the input sequence $P = p^1 p^2 \dots p^\ell$ is formed by

$$P = K \| M' \| K \ , \tag{8.36}$$

with $\|$ denoting concatenation. The iteration phase is specified in Table 8.5, with the hash result denoted by h. It can be seen that after all input blocks have been loaded, an additional 33 (blank) auto iterations are performed. The number of input and auto cycles to hash an ℓ-block input sequence is $\ell + 33$.

The StepRightUp hash function is designed to be K-secure and hermetic.

The StepRightUp stream cipher

The stream cipher is initialized by first loading two 8-word initialization blocks p^{-33} and p^{-32}, followed by 32 additional *blank* auto iterations. During cryptographic sequence generation an 8-word block z is delivered at the output for every iteration.

Turning this stream cipher into a stream encryption scheme takes the specification of an initialization mapping. For example, a 256-bit key K and a 256-bit parameter Q can be mapped to the initialization blocks by

$$\begin{array}{lcl} p^{-33} & = & Q \ , \\ p^{-32} & = & K \ . \end{array} \tag{8.37}$$

Time step t	Mode	Input	Output
0	reset	–	–
$1, \dots, \ell$	input	p^t	–
$\ell + 1, \dots, \ell + 32$	auto	–	–
$\ell + 33$	auto	–	h

Table 8.5: The sequence diagram of the iteration phase of the StepRightUp hash function.

Time step t	Mode	Input	Output
-34	reset	–	–
$-33, \ldots, -32$	input	p^t	–
$-31, \ldots, 0$	auto	–	–
$1, \ldots$	auto	–	z^t

Table 8.6: The sequence diagram of the StepRightUp stream cipher.

The sequence diagram of the StepRightUp stream cipher is given in Table 8.6.

This stream cipher is designed to be hermetic and K-secure with respect to all sound initialization mappings with a key length of 256 bits and a parameter length of 256 bits.

8.6.2 Discussion

The state updating transformation

The nonlinear transformation γ is a simple variant of χ. The multiplication polynomial of the linear shift-invariant transformation θ is selected from the polynomials with Hamming weight 3. It is the simplest of the class with the most favorable Hamming weight distribution table and has $\xi = \{15\}$. Since the number of words in the state is 17 and neither 2 or 15 divide 17, both γ and θ are invertible, and hence also the state updating transformation.

The cyclic shift coefficients of π, described by the simple expression in (8.33), form an array of 17 different constants. The word permutation factor 7 is chosen to let every component of ρ depend on 9 state bits. For the chosen π parameters it has been verified that $\rho \circ \rho$ has propagation and correlation properties that are close to optimal with respect to the space of possible π parameters.

In the transformation $\sigma[b]$, one stage of the buffer is injected in the odd-indexed words and another in the even-indexed words to obtain an immediate nonlinear interference. For the constant that is added to a_0 we have chosen 00000001_{hex} for its simplicity.

The hash function

For the StepRightUp hash function, a meet-in-the-middle attack for the construction of a message with a given hash result, as was described for Subhash, has an expected work factor of the order of 2^{272} applications of the hash function. As opposed to Subhash and Boognish, this is no threat to the “hermetic” claim for the StepRightUp hash function.

Our belief in the collision-resistance is based on arguments similar to those given in the case of Subhash. There are however some important differences. In Subhash a difference pattern in a block of the input sequence shifts unaltered through the buffer and disappears after 8 time steps. In the buffer of StepRightUp this is no longer the case. Because of the linear feedback, a difference pattern in a single input

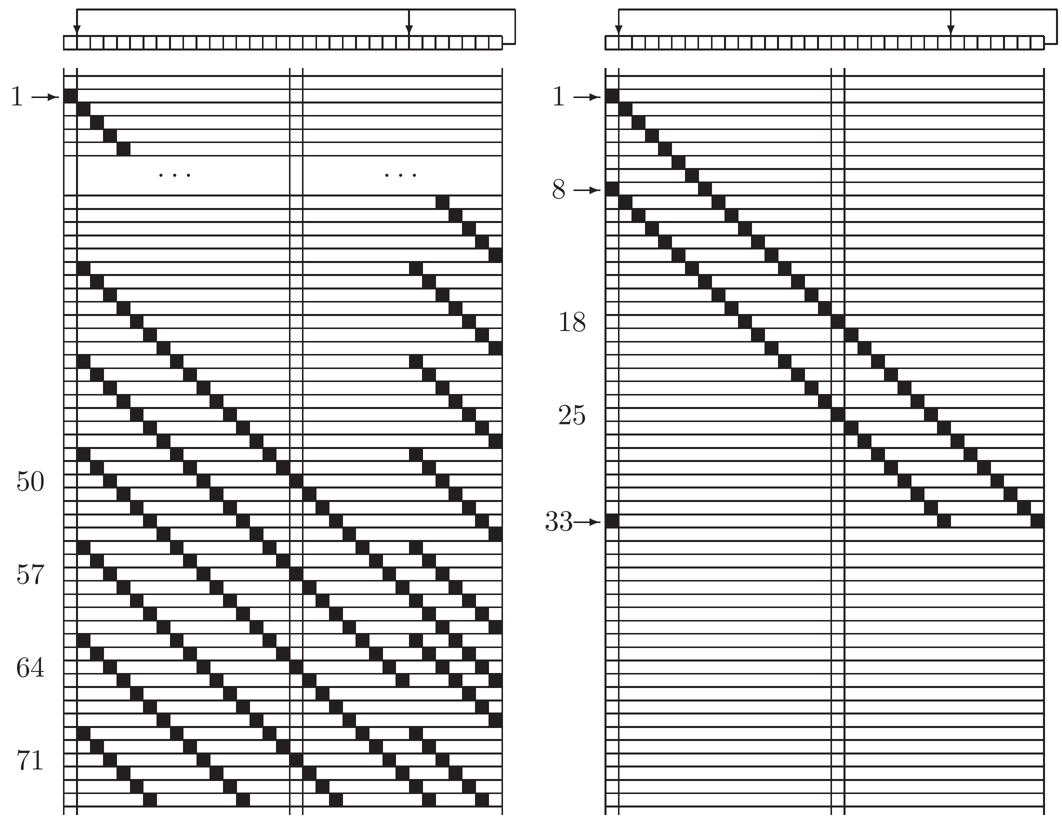


Figure 8.6: Difference propagation in the buffer of StepRightUp.

$t = 1$	-	d_0	-	d_1	-	d_2	-	d_3	-	d_4	-	d_5	-	d_6	-	d_7	-
$t = 8$	-	d_1	-	d_2	-	d_3	-	d_4	-	d_5	-	d_6	-	d_7	-	d_0	-
$t = 18$	-	-	d_7	-	d_6	-	d_5	-	d_4	-	d_3	-	d_2	-	d_1	-	d_0
$t = 25$	-	-	d_0	-	d_7	-	d_6	-	d_5	-	d_4	-	d_3	-	d_2	-	d_1
$t = 33$	-	d_0	-	d_1	-	d_2	-	d_3	-	d_4	-	d_5	-	d_6	-	d_7	-

Figure 8.7: Difference vectors in σ for a simple finite difference propagation.

block gives rise to an ever-continuing difference propagation in the buffer. This is illustrated in the left-hand side of Fig. 8.6. Only difference patterns in the input sequence that meet a particular condition give rise to a finite difference propagation in the buffer. This can easily be illustrated by a simple example.

Suppose we have a binary linear feedback shift register (LFSR) with internal state denoted by $b(x)$ and input p. The updating transformation of such a LFSR is specified by

$$\lambda[p_t](b(x)) = (p_t + x \cdot b(x)) \bmod m(x) \ , \tag{8.38}$$

with $m(x)$ the feedback polynomial. If the input sequence $p_{1-n}, \ldots, p_1, p_0$ is denoted by a polynomial $p(x) = p_{1-n}x^{n-1} + \ldots + p_{-2}x^2 + p_{-1}x + p_0$, multiple iterations are described by

$$\lambda^n[p(x)](b(x)) = (p(x) + x^n \cdot b(x)) \bmod m(x) \ . \tag{8.39}$$

The propagation of differences is described by the same equations where $b(x)$ and $p(x)$ must be interpreted as bitwise differences. Input difference sequences that, starting from an initial difference equal to 0, give rise to a terminal difference equal to 0 must satisfy

$$p(x) \bmod m(x) = 0 \ . \tag{8.40}$$

Hence, the polynomial corresponding to the input difference sequence must be a multiple of the feedback polynomial of the linear feedback shift register. A similar condition can be found for the buffer of StepRightUp. The simplest difference pattern that gives rise to a finite difference propagation in the buffer is illustrated in the right-hand side of Fig. 8.6. All other input differences that give rise to a finite difference propagation are superpositions (linear combinations) of shifted (in time and space) instances of this difference pattern.

In Fig. 8.6 it can be seen that even the simplest difference pattern affects the state updating transformation during 5 different iterations, spread over 32 time steps. Fig. 8.7 shows the difference vectors in σ resulting from a similar difference pattern in p that gives rise to a finite difference propagation and that has $p^{1'} = d_0, d_1, \ldots, d_7$. It can be seen that 4 of the 5 difference vectors are in general different. For more complicated difference patterns in the input sequence, the number of iterations that the state updating transformation is affected and the number of different difference vectors in σ are both larger.

If the difference pattern in the internal state is not equal to 0 after the loading of the last input block, the evolving difference pattern in the buffer and state will affect the internal state heavily during the 33 final auto iterations. This excludes internal collisions during these final iterations. Because of the large number of blank iterations and the strong difference propagation properties of ρ, we assume that terminal collisions cannot be obtained efficiently enough to be a threat to a "hermetic" claim.

In internal collisions the difference pattern in the state and buffer is 0 before the final hashing state has been reached. Clearly, this restricts the difference pattern in the input sequence to superpositions of instances of the simple difference pattern given in the left-hand side of Fig. 8.6. These input difference patterns give rise to nonzero difference vectors in σ over a span of at least 32 time steps. We believe that controlling the differential trails in the internal state over these large numbers of iterations to obtain a collision is too hard to be a threat to a "hermetic" claim. There is however a strategy that avoids the need for control over large numbers of iterations.

The basic idea of this strategy is that of the elimination of differences in the state, immediately after they have been created. Consider for example the difference pattern described by $(d_0, d_1, \ldots, d_7)$ that gives rise to the difference vectors shown in Fig. 8.7. Suppose this is followed by a similar difference pattern described by $(e_0, e_1, \ldots, e_7)$ that gives rise to nonzero difference vectors in σ at time steps 2, 9, 19, 26 and 34. It is conceivable that, for a given d vector, an e vector can be found such that for some values of the internal state the difference pattern after iteration 2 is 0. Hence, the difference vector in σ at time step 1 is compensated by the difference vector at time step 2. This can in principle be further extended to time steps 8 and 9, 18 and 19, 25 and 26, and 33 and 34. Hence, a couple of difference vectors d and e for which there exists a differential trail in the internal state that is 0 for time steps 2, 9, 19, 26 and 34 can be used to construct a collision. This can be further generalized to 3-tuples or 4-tuples of difference vectors. The usability of these difference patterns depends on the restriction weight of the corresponding differential trails in the internal state. The smaller these weights, the easier it will be to exploit them to generate collisions.

The described strategy for the generation of collisions has shaped two components of the StepRightUp design. First of all, the feedback term in the expression for $\gamma(b)|^{26}$ in (8.27) does an additional in-block cyclic shift of the words. In this way it is ensured that the difference vectors shown in Fig. 8.7 at $t = 1$ and $t = 8$ (and consequently $t = 18$ and $t = 25$) are different, resulting in 4 different vectors. For more complicated difference patterns in the input sequence there are always more than 4 different vectors.

The parameters of the bit permutation π have been chosen to make it strongly position-dependent. In this way the conditions for the different difference propagations in the described strategy give rise to conflicting requirements for simple low-weight difference patterns.

We did some tests for simple difference patterns and no weaknesses were found. Still, we think there is a need to explore the possibilities of the attack if more complicated difference patterns are taken.

The stream cipher

For every auto iteration 16 words of the buffer are injected into the state and 8 state words are given at the output. In the short term, the number of buffer bits that are injected into the state is twice as large as the number of bits that are given at the output. It can be checked that this is the case for any number of iterations smaller than 12. The feedback from the state to the buffer causes the buffer contents to be renewed every 32 iterations. These factors cause correlations between output bits and linear combinations of state and buffer bits to be of no practical use for cryptanalysis.

Resynchronization attacks should be made infeasible by the 32 blank auto iterations after loading the initialization blocks. Because of the feedback from the state to the buffer in the auto mode, the state and almost all buffer stages depend in a complicated way on these blocks at the end of the initialization phase. We expect that there are no exploitable 32-round differential trails.

Finally, it can easily be checked that the auto mode corresponds to an invertible updating transformation of both buffer and state.

8.6.3 Implementation aspects

Clearly, its heavy use of 32-bit words makes the StepRightUp biased towards 32-bit architectures. Still, all bitwise Boolean operations can also be implemented easily on processors with smaller word lengths. For the cyclic shifts this poses a somewhat larger problem. They can however easily be implemented by combining simple shifts with bitwise addition. Because of the high length of the state and the buffer, it is advantageous to have a processor with an on-chip data cache of at least 2 Kbyte.

The work factor of the auto and the input mode are equal. We express the work factor in 32-bit word operations. The buffer updating takes only 16 bitwise additions, the state updating 16 cyclic shifts, 68 bitwise additions, 17 bitwise and operations and 17 complementations. Hence, the work factor of an iteration is 118 bitwise Boolean operations (BB) and 16 cyclic shifts (CS). For hashing this corresponds to a work factor of 4012 BB and 544 CS and an additional 3.7 BB and 0.5 CS per input byte. Hashing becomes efficient for inputs longer than 2000 bytes, roughly the equivalent of a single typewritten page. For stream encryption the initialization phase takes 4012 BB and 544 CS and the generation of a byte 3.7 BB and 0.5 CS. These are very low work factors.

In dedicated hardware, the StepRightUp module is too large to be directly implemented as a finite state machine with contemporary technology. It can however be implemented as a set of dedicated very simple 32-bit processors and some memory, giving rise to a compact and cheap ultra-high speed module.

8.7 Conclusions

In this chapter we have given a global architecture and a number of concrete designs of a new type of cryptographic primitive, the stream/hash module. The hardware-oriented design Subterranean has been implemented as an integrated circuit that is capable of both stream encryption and cryptographic hashing at very high speeds.

The StepRightUp module is a more software-oriented proposal with a very low work factor per encrypted or hashed bit.

In our opinion, a more thorough investigation of the linear and differential trail propagation in Subterranean and StepRightUp can be the subject of some challenging and relevant further research.

Chapter 9

Design of Self-Synchronizing Stream Ciphers

9.1 Introduction

Because of its simple add-on property, single-bit self-synchronizing stream encryption is probably the most widely adopted encryption method in industrial, diplomatic and military communications. Still, self-synchronizing stream ciphers have not received the attention in cryptologic literature that block ciphers and synchronous stream ciphers have. The silence was broken by Ueli Maurer at Eurocrypt '91 [74].

One of the central ideas in [74] is the concept of building a cipher function from a (finite input memory) finite state machine with a cryptographically secure state updating transformation *and* a cryptographically secure output function. In this chapter we show that this is a vacuous concept. The other main idea in [74] is that of building a finite state machine with an internal state larger than the input memory n_{m}, by means of "serial" and "parallel" composition of smaller finite state machines. We show that a straightforward application of this design strategy leads to cipher functions with easily detectable and potentially exploitable weaknesses.

Subsequently, we perform a propagation analysis of a structure proposed in [74]. The conclusions of this analysis are used to further refine our design strategy as introduced in Chapter 4 and to propose a new structure, called a *conditional complementing shift register*. Our strategy is illustrated by a fully specified hardware-oriented single-bit self synchronizing stream cipher.

Some of the ideas in this chapter have already been published in our paper [15].

9.2 Machines with finite input memory

In Chapter 4 it was proposed to build the cipher function as a pipeline of a number (denoted by b_{s}) of relatively simple stages with small gate delay. The input to the first stage consists of the last n_{m} cryptogram bits, contained in a shift register. This structure guarantees that the encrypting bit z^t only depends on the cipher key κ and cryptogram bits $c^{t-n_{\mathrm{m}}-b_{\mathrm{s}}}$ to $c^{t-1-b_{\mathrm{s}}}$.

Replacing the shift register by a finite state machine with finite input memory n_{m} *can* improve the propagation properties without violating this dependence restriction. By imposing that the gate delay of this finite state machine must be smaller than the gate delay of the stages, the obtainable encryption speed is not affected.

The idea of replacing the shift register by a finite state machine with finite memory is related to that introduced in [74] of building the cipher function from a finite state machine with finite input memory and a combinatorial (memoryless) output function. The difference is that in our case the output function with respect to the finite state machine that replaces the shift register is not memoryless but consists of a number of pipelined stages.

A finite state machine with finite input memory has some specific propagation properties. Let q be the internal state and G the state updating transformation. We have

$$q^{t+1} = \mathrm{G}(q^t, c^t) \ , \tag{9.1}$$

with c^t the cryptogram bit at time t.

With every component of the internal state q can be associated an input memory, equal to the number of past cryptogram bits that it depends on. The internal state, confined to the components with input memory j is denoted by $q^{(j)}$, with $q_i^{(j)}$ its ith component. Although it is not a part of the internal state, c can be considered as a component with input memory zero: $q^{(0)}$. The input memory of the finite state machine is equal to the largest occurring component input memory.

Clearly, $q_i^{(j)}$ must be independent of all components with equal or higher input memory through G and *must* depend on $q^{(j-1)}$. From this, it follows that the input memory partitions the components of the internal state into non-empty subsets with input memory 1 to n_{m}. The components of the state updating transformation are of the form:

$$q_i^{(j)^{t+1}} = \mathrm{G}[\kappa]_i^{(j)}(c^t, q^{(1)^t}, \ldots, q^{(j-1)^t}) \ , \tag{9.2}$$

for $0 < j \leq n_{\mathrm{m}}$.

9.3 N-reductionist design principles

In [74] it is proposed to design the cipher function with a cryptographically secure state updating transformation G as well as a cryptographically secure output function h. In this context the term "cryptographically secure" is explained as follows:

- **Output function $z^{t+1} = \mathrm{s}[\kappa](q^t)$:** it should be infeasible for an adversary to determine the output of the finite state machine, even if the (actually hidden) state sequence is given.

- **State updating transformation:** a state updating transformation can be defined to be cryptographically secure if it is infeasible to determine the state for a given cryptogram sequence of n_{m} bits, even when the state would be provided for any other cryptogram bit sequence of n_{m} bits.

To demonstrate the gravity of the restrictions this imposes, we describe a straightforward procedure to reconstruct the subkeys $\kappa_i^{(j)}$ in these circumstances. The subkey $\kappa_i^{(j)}$ is that part of the key that $\mathrm{G}_i^{(j)}$ explicitly depends on.

The component functions $\mathrm{G}_i^{(j)}$ can be considered as subkey-dependent tables. If the adversary is given a number of output values corresponding to a number of inputs of $\mathrm{G}_i^{(j)}$ that is larger than the length of $\kappa_i^{(j)}$, this subkey can be determined by exhaustive key search over the subkey space. In the given circumstances the adversary can obtain outputs corresponding to as many inputs as he likes by applying cryptogram bit sequences with length $n_\mathrm{m} + 1$ and observing the internal state. The internal state after loading the first n_m bits of this sequence is the input to the component functions $\mathrm{G}_i^{(j)}$ that results in the internal state one time step further.

The work factor of this attack depends critically on the length ℓ of the longest subkey $\kappa_i^{(j)}$. The attack requires the application of ℓ cryptogram bit sequences. The effort consists mainly of the exhaustive key search of the ℓ-bit subkey. For this attack to be infeasible, at least one subkey $\kappa_i^{(j)}$ should be very long.

In general, the access that the adversary has to the keyed Boolean functions $\mathrm{G}_i^{(j)}$ is that of getting the output corresponding to known input. The state updating transformation can only be cryptographically secure by the above definition if at least one of its components $\mathrm{G}_i^{(j)}$ can resist attacks by an adversary with this type of access. This resistance includes hiding the key $\kappa_i^{(j)}$. This can only be realized if $\mathrm{G}_i^{(j)}$ has excellent propagation properties. This also holds for the output function, since the access of the adversary to the output function is the same as to the components of the state updating transformation.

Keyed Boolean functions with the required resistance against structural cryptanalysis can in practice only be implemented as the succession of a considerable number of implementable round mappings. Since the state updating transformation of a finite state machine is per definition memoryless, this iterated structure must be realized as a combinatorial circuit. The gate delay of this circuit grows with the number of rounds. Hence, the design approach gives rise to an output function and a state updating transformation with large gate delay, while the main objective of [74] was the design of self-synchronizing stream ciphers that are substantially *faster* than block ciphers in CFB mode.

A remarkable aspect of the attacks in the context of the proposed security definitions is that they only work if the adversary has access to the internal state of the self-synchronizing stream cipher. This is a highly artificial situation with no connection to any realistic threat or attack whatsoever. The described design principle seems to impose that the design contains big "lumps", i.e., complex combinatorial blocks that are "cryptographically secure".

These combinatorial blocks could be developed into a number of stages using pipelining in such a way that the external behavior would be the same except for some extra delay in the encrypting bit. This new structure can again be represented by a simple finite state machine with combinatorial state updating transformation if the internal state is supplemented with all the intermediate values of these additional stages. For this structure, it is however no longer the case that combinatorial blocks with some kind of cryptographic security can be identified. One could circumvent this by also allowing the cryptographically secure output function and components of the state updating transformation to have memory. However, this imposes a division of the bits of the physical internal state into two classes: the logical internal state and the memory of the components of the state updating transformation and the output function. This would make the design principle even more artificial.

The second major N-reductionist design principle that is presented in [74] is that of building a self-synchronizing encryption scheme with several keyed cipher functions in parallel. The encrypting bit is obtained as the bitwise sum of the outputs of the cipher functions. A theorem is given stating that if all the keys are independent, the resulting cipher is at least as cryptographically secure as any of the component ciphers. By basing the design of every component cipher on an entirely different principle, the risk that the cipher will be broken is equal to the risk that all design strategies fail simultaneously. The enthusiasm should be quenched by the fact that all *effective* design strategies available today consist of the study of propagation based on differential and linear cryptanalysis.

If we look at this construction within our framework, we get an even worse picture. Limiting the number of cipher functions to two, the resulting encryption scheme is described by

$$\begin{aligned} (\kappa_1, \kappa_2, IV) &= \mathrm{J}(K, Q) , && (9.3)\\ c_{-n_\mathrm{m}} \ldots c_{-1} &= IV , && (9.4)\\ z^t &= \mathrm{f_{c1}}[\kappa_1](c^{t-n_\mathrm{m}} \ldots c^{t-1}) + \mathrm{f_{c2}}[\kappa_2](c^{t-n_\mathrm{m}} \ldots c^{t-1}) . && (9.5) \end{aligned}$$

If κ_1 and κ_2 are independent, the resulting encryption scheme is only K-secure if both cipher functions are K-secure with respect to the initialization mapping. This is a consequence of the fact that the definition of K-secure does not impose the uniformity of the a priori key distribution. One of the cipher keys, say κ_1, can be given a certain value. This gives the cryptanalyst full access to the output of cipher function $\mathrm{f_{c1}}$. Hence, building a K-secure self-synchronizing stream encryption scheme from several component cipher functions with independent cipher keys requires *all* of the component cipher functions to be K-secure with respect to the initialization mapping. If the cipher keys are not independent, this no longer holds, but neither does the theorem in [74].

Moreover, seen in the context of design, it is obvious that splitting the available resources into several independently operating cipher functions is a practice that severely limits the potential for internal propagation.

9.4 Structural cryptanalysis

In this section we treat the differential and linear cryptanalysis specific for single-bit self-synchronizing stream ciphers. If the symbol size is larger than a single bit, the situation generally becomes more complex. For examples of differential attacks on self-synchronizing stream ciphers with a symbol size larger than 1, we refer to

[90, 85]. These publications contain some powerful attacks of DES variants in the 8-bit CFB mode.

9.4.1 Differential cryptanalysis

For every pair of n_{m}-bit input (cryptogram) sequences with a specific difference a', a pair of outputs each consisting of only a single bit is returned. The usability of $\mathrm{R_p}(a' \dashv \mathrm{f_c} \vdash 1)$ is determined by its deviation from 1/2. If this prop ratio is $1/2(1 \pm \ell^{-1})$, the number of input pairs needed to detect this deviation is approximately ℓ^2.

Consequently, a cipher function should not have difference propagations with prop ratios that deviate significantly more than $2^{-n_{\mathrm{m}}/2}$ from 1/2. The input differences a' with the highest deviations should depend in a complex way on the cipher key.

The differential attacks can be generalized in several ways. One generalization that proved to be powerful in the cryptanalysis of some weak proprietary designs can be labeled as *second order* differential cryptanalysis. Here the inputs to the cipher function are applied in 4-tuples. The 4 inputs denoted by a_0, a_1, a_2 and a_3 have differences $a' = a_0 + a_1 = a_2 + a_3$ and $a'' = a_0 + a_2 = a_1 + a_3$. By examining the 4 corresponding output bits it can be observed whether complementing certain input bits (a'') affects the propagation of a difference (a'). This can be used to determine useful internal state bits or even key bits. Typically these attacks exploit properties very specific to the design under analysis. This can be generalized to even higher order DC in a straightforward way.

9.4.2 Linear cryptanalysis

The number of inputs needed to detect a correlation C of the encrypting bit with a linear combination of input bits is C^{-2}. Hence, a cipher function should not have input-output correlations significantly larger than $2^{-n_{\mathrm{m}}/2}$. The selection vectors v_a with the highest correlations should depend in a complex way on the cipher key. By imposing a number k of affine relations on the input bits, the cipher function is effectively converted to a Boolean function in $n_{\mathrm{m}} - k$ variables. These functions should have no correlations significantly larger than $2^{-(n_{\mathrm{m}}-k)/2}$ for any set of affine relations.

A special case of a selection vector is the zero vector. An output function that is correlated to the constant function is unbalanced. A correlation of C to the constant function gives rise to an information leakage of approximately $C^2/\ln 2$ bits per encrypted bit for $C < 2^{-2}$.

In linear cryptanalysis correlations between the output and linear combinations of input bits are exploited. This can be generalized by allowing for a wider range of functions. An example of this is given by the table reconstruction attack as described in [74]. The idea is to find a low degree function that approximates the cipher function and can be used to diminish the uncertainty about the plaintext. It is shown that the construction of such an approximation using a limited number of samples can be seen as a (de)coding problem.

The feasibility of this attack is limited by the amount of high order terms in the algebraic normal form of the cipher function. Although this restriction severely limits the applicability of the attack, its basic idea is powerful and can inspire dedicated attacks. The process of fixing the class of approximating functions and deducing conditions for the cipher functions can be reversed. Depending on the specific design, the internal structure of the cipher function can possibly be used to define a manageable class of functions that is likely to contain a usable approximation for the cipher function.

9.5 Cipher function architectures

A finite state machine with finite input memory n_m realizes a mapping R from a length-n_m sequence of cryptogram bits $c^{t-n_\mathrm{m}}, \ldots, c^{t-1}$ to an internal state q^t. Its components are described by

$$q_i^{(j)^t} = \mathrm{R}[\kappa]_i^{(j)}(c^{t-j}, \ldots c^{t-1}) \ . \tag{9.6}$$

The finite state q serves as the input to the stages that map it to the enciphering bit z. We have

$$z^{b_\mathrm{s}+t} = \mathrm{s}[\kappa](q^t) \ . \tag{9.7}$$

In this context the cipher function $z^{b_\mathrm{s}+t} = \mathrm{f_c}[\kappa](c^{t-n_\mathrm{m}}, \ldots, c^{t-1})$ is decomposed into

$$\mathrm{f_c} = \mathrm{s} \circ \mathrm{R} \ . \tag{9.8}$$

This section is devoted to the study of the propagation properties of the mapping R as realized by different state updating transformations.

Because of the asymmetry inherent in the finite input memory requirement, the symmetry-based design approach of Chapters 7 and 8 cannot be applied here. This results in structures with a more cumbersome description. Moreover, the asymmetry prevents that difference propagations in R are dominated by a single differential trail or that correlations between the cryptogram bits c^i and q are dominated by a single linear trail.

First we will describe a specific architecture to illustrate some typical undesired properties that are present in many designs. This is followed by the introduction of *conditional complementing shift registers* and *pipelined stages*, structures that are much better adapted to their role in the cipher function.

9.5.1 A proposed recursive architecture

In [74] it is proposed to build the finite state machine with finite input memory by the application of parallel and serial composition of cipher functions. Parallel composition denotes bitwise addition of the output bits of two cipher functions fed with the same input. In serial composition the output of one of the automata is fed into the input of the other. In Fig. 9.1 an elegant recursive design is depicted that

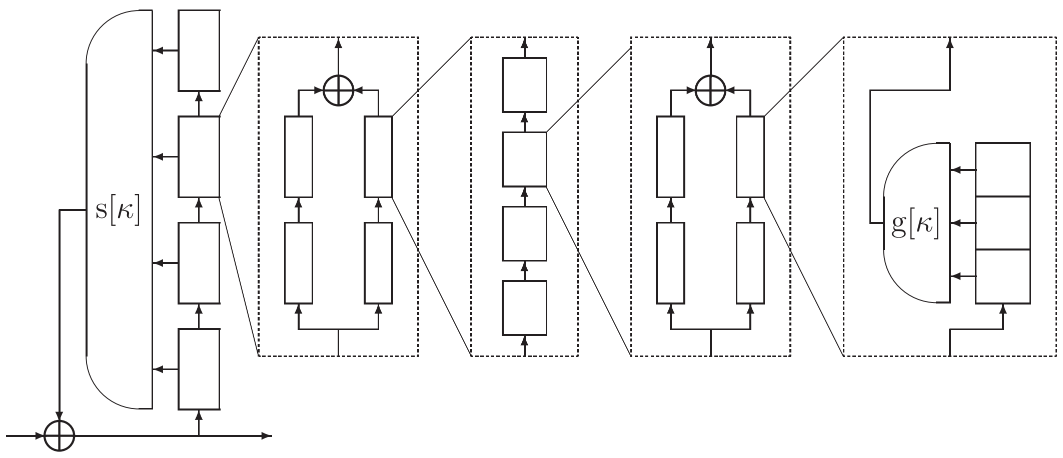


Figure 9.1: The Ψ architecture.

was given in [74] to illustrate the applicability of these composition modes. We will refer to this architecture as Ψ.

The rightmost box depicts the basic component: a 3-bit shift register with a keyed Boolean output function g[κ]. Four of these components are arranged in the parallel composition of two serial compositions of two components. On the next level, four of the resulting blocks are arranged by serial composition. At the two following levels these arrangements are repeated. The resulting finite state machine has an input memory of 192, with exactly 4 component bits for every input memory value. The internal state of this finite state machine serves as the input to the keyed output function s[κ]. It is not specified how the keyed Boolean functions g[κ] should be constructed. The most obvious way would be to expand the cipher key κ into 256 vectors of 8 bits that can serve as the tables for the functions g[κ]. More sophisticated key schedules typically impose that the tables fulfill certain criteria such as balancedness or completeness (output is not independent of any of the input bits).

We did some difference propagation experiments with the Ψ structure. A cryptographic sequence generator, loaded with a specific initial state, was used to generate the 8-bit tables of the functions g[κ]. Two input strings, differing in a specific pattern of bits, were applied and the difference propagation in the internal state was observed. This was repeated for a number of different keys and sequences. Figure 9.2 gives a typical differential trail for Ψ with an initial difference pattern with a Hamming weight of 1. It can be seen that this differential trail has already completely dissolved about 160 cycles after it has started. Our experiments show that flipping cryptogram bits c^{-j} with j between 192 and 180 leaves q^0, and therefore z^{b_s}, virtually always unaffected.

For a super self-synchronizing stream cipher every input difference c' has an expected prop ratio $\mathrm{R_p}(c' \dashv \mathrm{f_c} \vdash 0)$ of 1/2. For Ψ the input differences c' that are 0 in the last 180 bits have $\mathrm{R_p}(c' \dashv \mathrm{R} \vdash 0) \approx 1$. Since always $\mathrm{R_p}(0 \dashv \mathrm{s} \vdash 0) = 1$, this propagation defect of Ψ cannot be compensated by any output function. We have experienced that this type of propagation defect can serve as a lever in cryptanalysis to pry open the cipher.

The observed defective difference propagation is due to the bad difference propagation inherent in serial composition of cipher functions. For the vast majority of super self-synchronizing ciphers with input memory n_m, the prop ratio $\mathrm{R_p}(c' \dashv \mathrm{f_c} \vdash 1)$, with c' a vector that is only 1 in component c^{-n_m} and 0 elsewhere, is 1/2. For the serial composition of ℓ uniformly chosen cipher functions with input memories adding up to n_m, this prop ratio is only $2^{-\ell}$. Serial composition as proposed in [74] should therefore be avoided.

An unlucky choice of the component cipher functions with the lowest input memory can cause even more serious problems. In Ψ there are four 3-bit shift registers that are fed with the input (cryptogram) bits themselves. Let the four corresponding key-dependent Boolean functions be denoted by $\mathrm{g}_1, \mathrm{g}_2, \mathrm{g}_3$ and g_4. Assume that for all these functions

$$\mathrm{g}_i(000) = \mathrm{g}_i(001) = \mathrm{g}_i(010) = \mathrm{g}_i(100) \ .$$

In that case pairs of cryptogram bit sequences that only differ in a single bit preceded and followed by two zeroes (i.e., $\ldots 00000 \ldots$ and $\ldots 00100 \ldots$) give rise to pairs of output sequences with pairwise identical members for all four functions. This implies that the resulting differential trail does not reach beyond the four shift registers with lowest input memory. Totally there are 16 different cases corresponding to the possible combinations of values in the two following and preceding bits. If the functions g are uniformly generated, the differential trail stops at least for one of these 16 cases for exactly 1 cipher key in 256. For balanced functions g_i, the proportion of cipher keys with prematurely dissolving differential trails is larger than 1 in 6000.

9.5.2 Conditional complementing shift registers

In this section we introduce a new architecture for finite state machines with finite input memory. In this architecture the discussed propagation defects are prevented by imposing (partial) linearity on the components of the state updating transformation. For simplicity we impose the preliminary restriction that all $q^{(j)}$ have only one component, i.e., that there is only one bit for every input memory value. The components of the state updating transformations are of the form

$$ {q^{(j)}}^{t+1} = {q^{(j-1)}}^{t} + \mathrm{E}[\kappa]^{(j)}({q^{(j-2)}}^{\mathrm{t}}, \ldots, {q^{(1)}}^{t}, c^{t}) \ . \tag{9.9}$$

Since the new value of $q^{(j)}$ is equal to the sum of the old value of $q^{(j-1)}$ and some Boolean function, we call this type of finite state machine a *conditional complementing shift register* (CCSR).

Proposition 9.1 *The mapping R corresponding to a CCSR is an injection.*

Proof : We describe an algorithm for the reconstruction of $c^t {q^{(1)}}^t \ldots {q^{(j-1)}}^t$ from ${q^{(1)}}^{t+1} \ldots {q^{(j)}}^{t+1}$. The components are reconstructed starting from c and finishing with $q^{(j-1)}$. For $q^{(1)}$ (9.9) becomes

$$ {q^{(1)}}^{t+1} = {q^{(0)}}^{t} + \mathrm{E}[\kappa]^{(1)} = c^t + \mathrm{E}[\kappa]^{(1)} \ ,$$

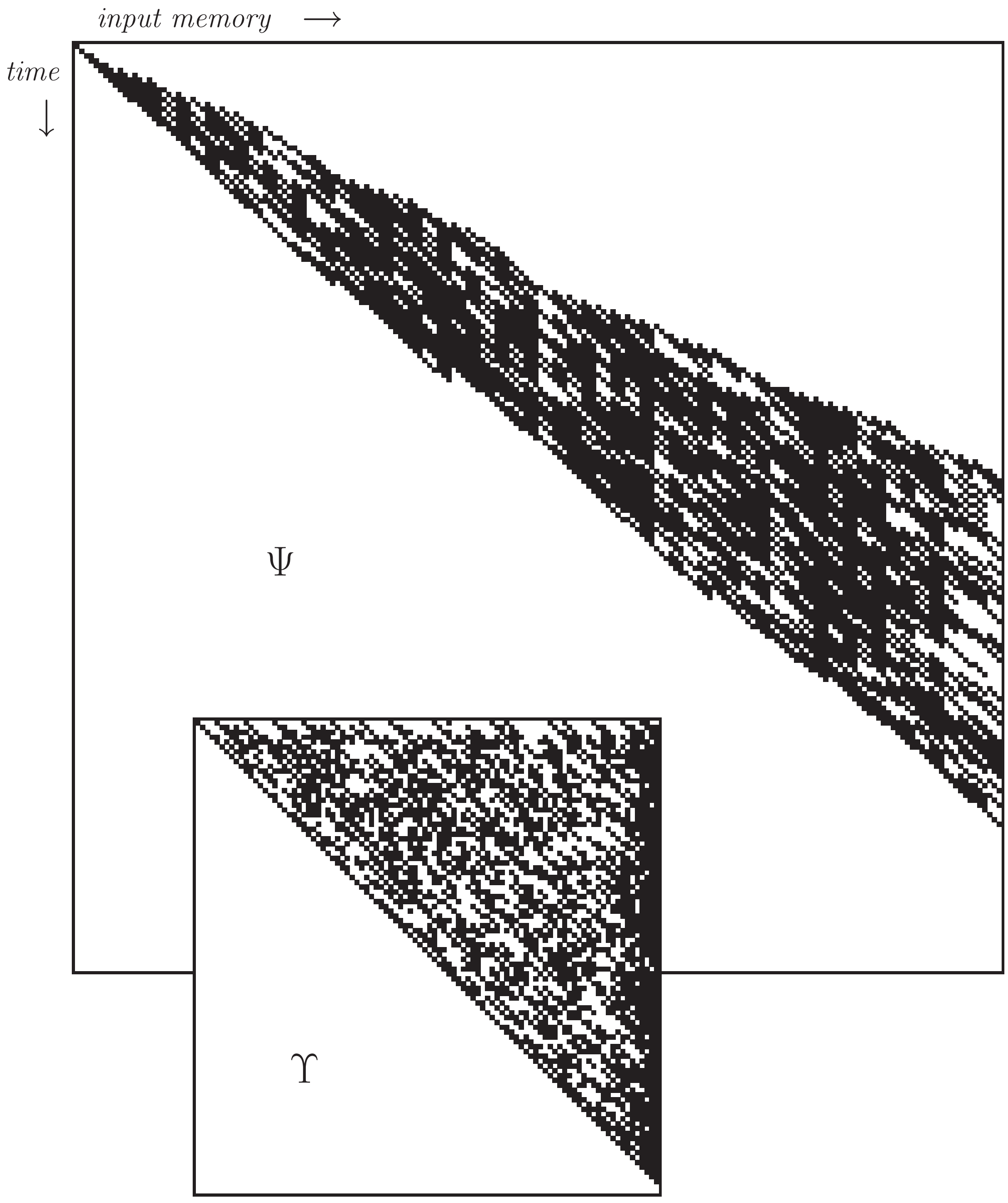


Figure 9.2: Difference propagation patterns in Ψ and ϒ. There are in general several state bits for every input memory value. A black mark at time coordinate t and input memory coordinate i denotes that at time t there is a difference in at least one of the bits of input memory i.

since $\mathrm{E}[\kappa]^{(1)}()$ depends only on κ. From this we can calculate c^t. The values of $q^{(k-1)^t}$ for k from 2 to j can be calculated iteratively from the previously found values by

$$q^{(k-1)^t} = q^{(k)^{t+1}} + \mathrm{E}[\kappa]^{(j)}(q^{(k-2)^t}, \ldots, q^{(1)^t}, c^t) \ .$$

$c^{t-n_\mathrm{m}} \ldots c^{t-1}$ can be calculated uniquely from $q^{(1)^t} \ldots q^{(n_\mathrm{m})^t}$ by iteratively applying the described algorithm. Hence, R is an injection. $\square$

Since R is an injection, a nonzero difference in $c^{t-n_\mathrm{m}} \ldots c^{t-1}$ must give rise to a nonzero difference in q^t. Therefore in a CCSR there is no premature dissolving of differential trails as for instance in Ψ.

The CCSR has the undesired property that a difference in $c^{-n_\mathrm{m}-t}$ propagates to $q^{(n_\mathrm{m})^t}$ with a prop ratio of 1. This can be avoided by "expanding" the high input memory end of the CCSR, i.e., taking more than a single state bit per input memory value near memory value n_m. We will illustrate this in our design example.

An elaborated CCSR design example

In this section we describe the design of a finite state machine Υ of the CCSR architecture. The main design goal is the elimination of difference propagations with a prop ratio larger than 2^{-15} for the mapping R corresponding to Υ, while keeping the gate delay very small and the description as simple as possible.

Υ has an input memory of 96. The CCSR is expanded at the high input memory end, with 2 bits per input memory starting from $j = 89$, four starting from 93, eight of 95 and sixteen of 96, resulting in 128 state bits. The cipher key κ corresponding to Υ consists of 96 bits: $\kappa_0 \ldots \kappa_{95}$.

For reasons of simplicity, area and gate delay, the majority of the components of the state updating transformation consist of a very simple Boolean function of the form

$$\mathrm{G}[\kappa]_i^{(j)}(q, c) = q_i^{(j-1)} + \kappa_{j-1} + (q_i^{(v)})(q_i^{(w)} + 1) + 1 \ , \qquad (9.10)$$

with $0 \leq v, w < j-1$. We have chosen for key application by simple bitwise addition and let the remaining part of E consist of the simplest nonlinear function possible. A combinatorial circuit that implements one component of the state updating transformation is depicted in Fig. 9.3. It can be seen that this circuit has a gate delay of only 2 EXOR gates. The figure also shows the expansion of the CCSR at the high input memory end and the corresponding indexing. The v and w values for almost all components $\mathrm{G}_i^{(j)}$ are specified in Table 9.1. For $j \leq 4$, the $q^{(v)}$ and $q^{(w)}$ entries are taken to be 0. In some cases terms $q_i^{(j)}$ specified by Table 9.1 do not correspond to existing components. In that case the i-index must be diminished by iteratively subtracting the largest power of 2 contained in i until the term corresponds to an existing component (e.g., $q_{14}^{(93)} \to q_6^{(93)} \to q_2^{(93)}$).

The 15 components $\mathrm{G}_i^{(96)}$ with $i > 0$ are of a different type. These are specified by

$$\mathrm{G}[\kappa]_i^{(96)}(q, c) = q_i^{(95)}(q_0^{(95-i)} + 1) + q_i^{(94)}(q_1^{(94-i)} + 1) \ . \qquad (9.11)$$

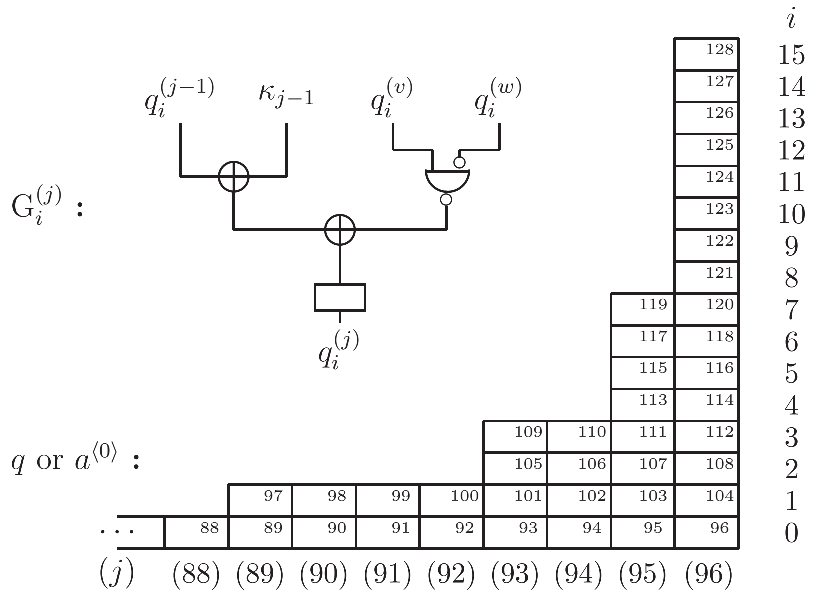


Figure 9.3: A circuit implementing a component of the state updating transformation and expansion of the CCSR in the high memory region. The q indexing (CCSR) is given at the right and the bottom, the $a^{\langle 0 \rangle}$ indexing (pipelined stages, see p. 173) inside the boxes.

	v	w
$(i+j) \bmod 3 = 0$	$j - 4 + (i \bmod 2)$	$j - 2$
$(i+j) \bmod 3 = 1$	$j - 6 + (i \bmod 2)$	$j - 2$
$(i+j) \bmod 6 = 2$	$j - 5 + (i \bmod 2)$	0
$(i+j) \bmod 6 = 5$	0	$j - 2$

Table 9.1: v and w values for the components $G_i^{(j)}$ with $4 < j < 96$ and $G_0^{(96)}$.

Observe that these are unbalanced functions and will cause a bias in the corresponding components.

In Fig. 9.2 a typical propagation pattern is shown for Υ. The diagonal trailing edge is caused by the linear forward propagation of the CCSR ensuring that differential trails do not prematurely dissolve. The difference pattern at time 0 resulting from a difference pattern in the cryptogram symbols ending in c^{-96} is restricted to $q^{(96)}$. The partial linearity in (9.10) guarantees that the difference pattern is 1 in $q_0^{(96)}$. The difference value of each of the 15 remaining components is determined by balanced key-dependent functions of the absolute values of the cryptogram symbols c^{-95} to c^{-1}. This results in 2^{15} equiprobable nonzero difference patterns. The difference patterns in q^0 resulting from difference patterns that end in cryptogram symbol c^{96-e} for small e are not uniformly distributed. However, all prop ratios are well below the 2^{-15} limit.

Care was taken that difference propagations $(c' \dashv \mathrm{R} \vdash 0)$ with c' different from 0 cannot occur because these typically give rise to detectable propagation defects. This can be seen as follows. Assume that for the output function $\mathrm{R_p}(q' \dashv \mathrm{s} \vdash 0) = 1/2$ for all nonzero difference patterns q'. In that case a difference propagation $(c' \dashv \mathrm{R} \vdash 0)$ with a prop ratio of p gives rise to a difference propagation $(c' \dashv \mathrm{f_c} \vdash 0)$ with a prop ratio of approximately $(1 + p)/2$.

As opposed to Ψ, in Υ the input difference pattern diffuses immediately to components all over the internal state. This is a consequence of the fact that c^t is not only injected in $q^{(1)}$, but in many components at once. These are represented by the zero v and w entries in Table 9.1 (keep in mind that $q^{(0)} = c$). Depending on the value of $q_0^{(j-3)}$, a difference in c propagates to either $q_0^{(j-1)}$ or $q_0^{(j+2)}$. Since there are more than 15 of these "double injections", the prop ratios are below 2^{-15}. In subsequent iterations this pattern is subject to the nonlinearity of the Υ state updating transformation.

9.5.3 Pipelined stages

In our architecture, the cipher function consists of a CCSR followed by a number of pipelined stages. The stage transformations are similar to the round transformation in a block cipher but are less restricted.

A round transformation of an iterated block cipher must be invertible, and its inverse must be easily implementable. A stage transformation does not have this restriction and the length of its output can be different from that of its input. The output of the last stage transformation is a Boolean function of the components of the state q some cycles ago. This Boolean function can be forced to be balanced by imposing that all the stage transformations are *semi-invertible*. We call an n-bit to m-bit mapping $b = f(a)$ semi-invertible if there exists an n-bit to $(n - m)$-bit mapping $b' = f'(a)$ so that a is uniquely determined by the couple (b, b'). In that case the output bit can be seen as a component of the output of an invertible function of the state q.

The last round transformation of an iterated block cipher must be followed by a key application or include a key dependence. This is necessary to prevent the cryptanalyst to calculate an intermediate encryption state thereby making the last round transformation useless. For the cipher function of a single-bit self-synchronizing stream cipher the calculation of intermediate values is impossible since only a single output bit z^t is given per input. Therefore, key dependence is not a strict necessity for the stage transformations.

An elaborated design example

In this section we describe a staged output function Γ for use with Υ. Γ consists of 7 stages in total. The output of stage $\langle i \rangle$ is stored in a register denoted by $a^{\langle i \rangle}$. Register $a^{\langle 0 \rangle}$ corresponds to q and has a length of 128. For ease of notation $a^{\langle 0 \rangle}$ has been given a different indexing than q. This is specified in Fig. 9.3.

The components of the registers $\langle 1 \rangle$ to $\langle 7 \rangle$ are indexed starting from 0. Registers $a^{\langle 1 \rangle}$ to $a^{\langle 5 \rangle}$ have length 53. For the component state updating transformation of stage 1 we have

$$\mathrm{G}^{\langle 1 \rangle}_{4i \bmod 53} = a^{\langle 0 \rangle}_{128-i} + a^{\langle 0 \rangle}_{i+18} + a^{\langle 0 \rangle}_{113-i}(a^{\langle 0 \rangle}_{i+1} + 1) + 1 \ , \tag{9.12}$$

for $0 \leq i < 53$. The stages 2 to 5 are specified by

$$\mathrm{G}^{\langle j \rangle}_{4i \bmod 53} = a^{\langle j-1 \rangle}_{i} + a^{\langle j-1 \rangle}_{i+3} + a^{\langle j-1 \rangle}_{i+1}(a^{\langle j-1 \rangle}_{i+2} + 1) + 1 \ , \tag{9.13}$$

for $0 \leq i < 53$. If a lower index in the right-hand side of these equations becomes larger than 52, the corresponding term is taken to be 0, e.g., $a^{\langle j-1 \rangle}_{53} = 0$. Register $a^{\langle 6 \rangle}$ has a length of 12. Stage 6 is defined by

$$\mathrm{G}^{\langle 6 \rangle}_{i} = a^{\langle 5 \rangle}_{4i} + a^{\langle 5 \rangle}_{4i+3} + a^{\langle 5 \rangle}_{4i+1}(a^{\langle 5 \rangle}_{4i+2} + 1) + 1 \ . \tag{9.14}$$

Register $a^{\langle 7 \rangle}$ consists of 3 bits. We have

$$\mathrm{G}^{\langle 7 \rangle}_{i} = a^{\langle 6 \rangle}_{4i} + a^{\langle 6 \rangle}_{4i+1} + a^{\langle 6 \rangle}_{4i+2} + a^{\langle 6 \rangle}_{4i+3} \ . \tag{9.15}$$

Finally the encrypting bit is given by

$$z = a^{\langle 7 \rangle}_{0} + a^{\langle 7 \rangle}_{1} + a^{\langle 7 \rangle}_{2} \ . \tag{9.16}$$

The input to the first stage consists of the state bits of the CCSR. Special care has been taken with respect to difference patterns restricted to the high-memory region of Υ and those resulting from a difference in the most recent cipher bit. The purpose of stages $\langle 1 \rangle$ to $\langle 6 \rangle$ is the elimination of low-weight linear and differential trails. The components of these stage transformation combine diffusion, nonlinearity and dispersion respectively in the linear term, in the quadratic term and in the arrangement of inputs and outputs. Their effectiveness is reinforced by the diffusion in stage $\langle 7 \rangle$ and the output function that computes the encrypting bit as the bitwise addition of all 12 bits of $a^{\langle 6 \rangle}$ to the output.

Finally, it can easily be checked that all the stages are semi-invertible.

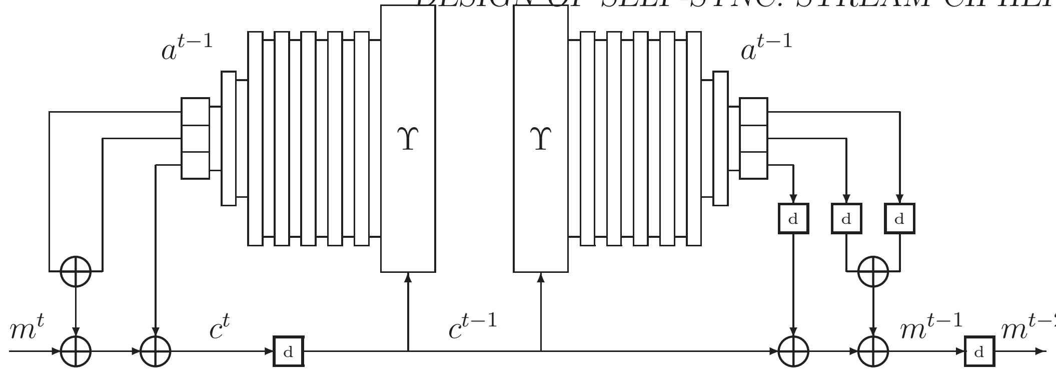


Figure 9.4: Encryption and decryption with the $\Upsilon\Gamma$ architecture.

9.6 The $\Upsilon\Gamma$ cipher

Figure 9.4 shows a self-synchronizing stream cipher consisting of the combination of Υ and Γ. The gate delay of the combination of the output function and the encryption (or decryption) is 2 EXOR gates, equal to the gate delay of the state updating transformation. This necessitates the introduction of extra intermediate storage cells, denoted in Fig. 9.4 by boxes containing a **d**. In the encryptor this cell is located between the encryption and the input of Υ. For correct decryption this necessitates an extra delay of 1 clock cycle after the last stage of Γ in the decryptor. In Fig. 9.4 it can be seen that the operation of encryption and subsequent decryption takes only two clock cycles. The clock speed is limited by the gate delay of only 2 EXOR gates, allowing throughput rates in the range of 100 Mbit/s with present-day technology.

This cipher can be seen as an improvement of our earlier design called KNOT that was presented in our paper [15]. We discovered that the output function of KNOT has a detectable imbalance. This has been avoided in the present design by imposing that the stages of Γ are semi-invertible. The $\Upsilon\Gamma$ cipher has the additional benefit that it has a more compact description than KNOT.

The $\Upsilon\Gamma$ cipher can be turned into a self-synchronizing stream encryption scheme by the specification of an initialization mapping. As an example of a simple and effective initialization mapping we have:

$$\begin{aligned} \kappa &= K \, , \\ IV &= Q \, . \end{aligned} \tag{9.17}$$

9.6.1 The output feedback mode

In Chapter 2, a method was described for building a cryptographic sequence generator with an invertible state updating transformation from a self-synchronizing stream cipher. As can be seen in Fig. 2.5, this requires a feedback from the last cell of the shift register that contains the previous n_{m} ciphertext symbols. This cannot be applied in a straightforward way in the case of $\Upsilon\Gamma$ since the shift register is replaced by a CCSR and the stages introduce additional delay. The scheme in

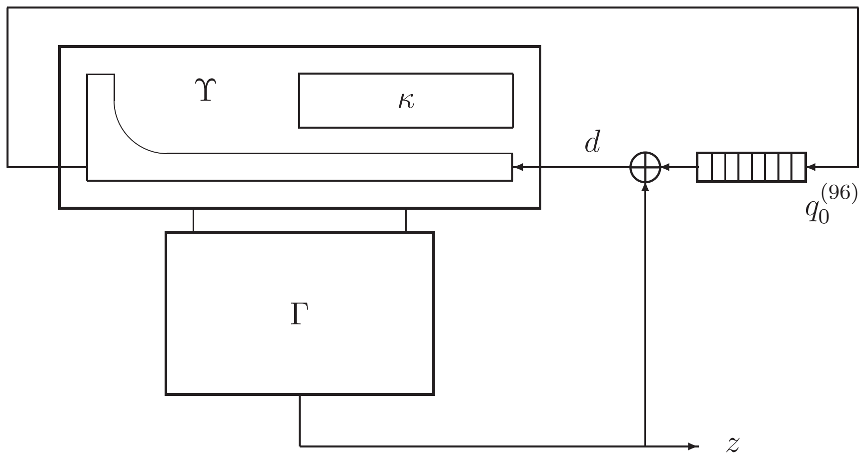


Figure 9.5: OFB mode for $\Upsilon\Gamma$ with invertible state updating.

Fig. 2.5 can however be adapted for $\Upsilon\Gamma$ by simply replacing the single-bit register by a $b_{\mathrm{s}} + 1$-bit shift register. This is illustrated in Fig. 9.5. The invertibility of this state updating transformation can easily be proved using the injection property of the CCSR. The expected cycle length of this structure is $2^{n_{\mathrm{m}}+b_{\mathrm{s}}}$, equal to 2^{104} in this case.

As initialization mapping for this synchronous encryption scheme we propose the following procedure. First the internal states of Υ and Γ are reset to 0. Then κ and the initial state are specified by

$$\begin{aligned} \kappa &= K \ , \\ d^{-104} \ldots d^{-1} &= Q \ , \end{aligned} \tag{9.18}$$

9.7 Conclusions

It has been shown that there are serious problems with the design approach for self-synchronizing stream ciphers presented in [74]. We have presented our own design strategy for single-bit self-synchronizing stream ciphers containing new, better adapted architectures. We have illustrated this design strategy by an actual design that can be implemented in hardware to give a high-speed cipher.

We believe that there is an interesting opportunity for further research in extending our design strategy to dedicated n_{s}-bit self-synchronizing stream ciphers with $n_{\mathrm{s}} > 1$.

Chapter 10

Supporting Cryptanalytic Results

10.1 Introduction

This chapter contains some of our cryptanalytic results that have played an important role in the conception of our design strategies and goals.

First we describe our *resynchronization attacks*. These attacks show the destructive impact that resynchronization can have on the security of synchronous stream ciphers. The point is that the initialization mapping, a necessary component of any practical synchronous encryption scheme, must be seen as part of the encryption scheme and therefore included in the cryptanalytic phase of the design.

Subsequently, we show how to generate collisions for a cryptographic hash function based on cellular automata designed by Ivan Damgård. This attack has been included to illustrate that the application of theoretical design principles does not remove the need for basic correlation and propagation evaluation prior to publication.

Finally, we treat the security of a block cipher construction that was proposed by Shimon Even and Yishay Mansour. The actual consistency of the results of our cryptanalysis with the computational complexity theoretic analysis of the designers clearly illustrates the limitations of the computational complexity theoretic point of view.

10.2 Resynchronization attacks

In this section we present some examples of a new class of powerful attacks for synchronous stream encryption schemes, called resynchronization (or resync) attacks. These attacks exploit the relations between encrypting bits resulting from different initializations with the same key K and different values of the public parameter Q and were originally published in our papers [17] and [19].

The objects of all attacks presented here are filtered counter stream encryption schemes. We concentrate on schemes with a linear state updating transformation and a linear initialization mapping. The initialization and the state updating transformation can be described by matrix equations:

$$s_i^0 = \mathrm{M}K + \mathrm{r}_i \ , \tag{10.1}$$
$$s_i^{t+1} = \mathrm{F}s_i^t \ , \tag{10.2}$$

with M and F binary matrices and F nonsingular. In these equations s_i^t denotes the internal state t time steps after the ith initialization. The r_i are vectors determined by the public parameter in the different initializations.

The attacks focus on the full reconstruction of s_p^τ, the internal state of the counter at a certain time step τ. From this internal state, s_p^0 can be computed and subsequently the initial states s_i^0 corresponding to all other initializations:

$$s_p^0 = \mathrm{F}^{-\tau}s_p^\tau \ , \tag{10.3}$$
$$s_i^0 = s_p^0 + \mathrm{r}_i + \mathrm{r}_p \ . \tag{10.4}$$

The bitwise sum of any two states with equal time indices, s_i^t and s_j^t, is completely determined by the r_i vectors:

$$s_i^t + s_j^t = \mathrm{F}^t(\mathrm{r}_i + \mathrm{r}_j) \ . \tag{10.5}$$

The output function (or filter) computes the encrypting symbol from the internal state. In order to simplify the notation, only binary encrypting symbols are considered. If the symbols consist of more than a single bit, the output function can be decomposed in its binary components.

In most practical proposals, the encrypting bit only depends on a small set of φ (independent) linear combinations of state bits. Suppose the values of these linear combinations of state bits form the components of the binary vector u with φ components. The relation between the vector u and the internal state s is expressed in a matrix equation

$$u_i^t = \mathrm{G}s_i^t \ . \tag{10.6}$$

We can write $z = \mathrm{f_o}(u)$ with $\mathrm{f_o}$ the output function.

The linear relations between the internal states can be exploited to reinforce existing attacks or to construct new ones. In the following we describe four different attacks, ranging from a general known-plaintext attack to a scheme for on-line cracking of a commercial video encryption system.

10.2.1 A simple general attack

We start with describing a procedure to reconstruct u_p^τ for some τ (in the rest of this discussion the superscript τ has been omitted). Suppose z_p is known to the cryptanalyst. The correct u_p has to fulfill $\mathrm{f_o}(u_p) = z_p$. Every z_i that is known to the cryptanalyst gives rise to a similar equation: $\mathrm{f_o}(u_i) = z_i$. Substitution of u_i by $u_p + \mathrm{GF}^t(\mathrm{r}_i + \mathrm{r}_p)$ yields a condition for u_p:

$$\mathrm{f_o}(u_p + \mathrm{GF}^t(\mathrm{r}_i + \mathrm{r}_p)) = z_i \ . \tag{10.7}$$

This results in a set of nonlinear Boolean equations with φ variables. If the number of equations is larger than φ, the number of solutions is expected to converge to 1. If φ is small enough, the solution can be found by performing exhaustive search over the space of all possible u-vectors. This involves on the average 2^{φ} evaluations of $\mathrm{f_o}()$.

Substitution of the known values of the bits of u_p^{τ} in $u_i^{\tau} = \mathrm{G}s_i^{\tau}$ gives rise to φ affine equations in s_p^{τ}. These can in their turn be converted to affine equations in s_p^0 using the relation $\mathrm{F}^{\tau}s_p^0 = s_p^{\tau}$.

The procedure described above can be repeated for a number of different time steps τ until the number of linearly independent affine equations in the bits of s_p^0 is larger than the number of state bits n_{a}. The resulting set of equations can be efficiently solved by methods of linear algebra. Hence, the reconstruction of approximately $\lceil \frac{n_{\mathrm{a}}}{\varphi} \rceil$ different u_p^{τ} vectors is sufficient to completely determine s_p^0.

For this attack to be possible, the number of initializations must be larger than φ. The cryptanalyst must know at least φ encrypting bits for a number of $\lceil \frac{n_{\mathrm{a}}}{\varphi} \rceil$ time steps. The expected work factor is

$$\left\lceil \frac{n_{\mathrm{a}}}{\varphi} \right\rceil 2^{\varphi} \tag{10.8}$$

evaluations of $\mathrm{f_o}()$ and some additional linear algebra computations. The attack can easily be parallelized by distributing the search over a number of different processors. It can be observed that the work factor of this attack only grows linearly with the size of the internal state. For this general attack to be infeasible, it is essential that φ is large and consequently that the output function depends on a large number of state bits.

This general attack works independent of the specific properties of the output function. In the following section we show that in some cases a significant speedup can be attained by exploiting the structure of $\mathrm{f_o}()$.

10.2.2 A simple attack for the multiplexor generator

A so-called *multiplexor generator* [95, p. 108] consists of a set of linear feedback shift registers filtered by a multiplexor output function. A multiplexor with β address inputs has 2^{β} data inputs. If the address inputs are denoted by $a^0, a^1, \ldots, a^{\beta-1}$, the data inputs by $d[0], d[1], \ldots, d[2^{\beta}-1]$ and the output by z, the operation of the multiplexor can be described by

$$z = d[a] \text{ with } a = \sum_{0 \le i < \beta} a^i 2^i \ . \tag{10.9}$$

Clearly, a is the interpretation of $a^{\beta-1} \ldots a^1 a^0$ as an integer. The vector u_p is split into a_p and d_p and $\varphi = \beta + 2^{\beta}$. Since $(s_i^t + s_j^t)$ is completely determined by $\mathrm{r}_i + \mathrm{r}_j$, so are $(a_i + a_j)$ and $(d_i[k] + d_j[k])$ for any k.

Suppose the cryptanalyst knows the encrypting bits z_p and z_i for a given time step t in which $(a_p + a_i) = 0$. If $z_p = z_i$, the integer a_p (and equivalently a_i) must select a bit in d for which $d_p[a_p] + d_i[a_p] = 0$. If $z_p \neq z_i$, we have $d_p[a_p] + d_i[a_p] = 1$.

In both cases the cryptanalyst can eliminate about half of the possible values for a_p. This can be repeated for every two encrypting bits z_e, z_f for which $(a_e + a_f) = 0$. The correct value for a_p will be found after investigating β pairs (or a few more). After the correct value of a_p has been found in this way, the remaining $2^\beta - \beta$ values of d_p can systematically be scanned by considering a set of 2^β encrypting bits z_{g_i} with the indices g_i determined by $a_{g_i} = a_p + i$.

The complete process of fixing the u-vector takes about $\beta + 2^\beta = \varphi$ evaluations of the multiplexor function. This has to be repeated $\lceil \frac{n_\mathrm{a}}{\varphi} \rceil$ times. The complexity of the complete attack can be approximated by

$$\left\lceil \frac{n_\mathrm{a}}{\varphi} \right\rceil \varphi \approx n_\mathrm{a} \tag{10.10}$$

multiplexor evaluations and some additional linear algebra computations. In fact, the linear algebra will be responsible for the largest part of the work factor in realistic cases.

Because of the particular demands for the difference patterns in a, the number of initializations should be approximately 2^β or larger for this attack to be possible.

10.2.3 A powerful attack for the multiplexor generator

In this section we present a more complicated attack on the multiplexor generator that takes advantage of even a small number of initializations to reconstruct the internal state in an efficient way.

In this attack we manipulate binary vectors with components that are partly unknown. Apart from the symbols 0 and 1, – (unknown) can occur. The set of indices corresponding to unknown components of a vector q is denoted by $\nu(q)$. If such a partly known vector is the argument of a Boolean function, any output depending on unknown components of the input is taken to be unknown.

$x|_\nu$ (with x a binary vector and ν a set of component indices) denotes the 'projection' of the vector x on a vector with components in ν. If ν is a singleton with element k, this is denoted by $x|_k$. A subset ν of the components of a vector a can be given a value by the expression $a|_\nu = \mathrm{c}$ with c a constant vector with $\#\nu$ components.

The set of indices of the components that are used as address inputs of the multiplexor is denoted by α. If for a state s the part $s|_\alpha$ is completely known, the address index, i.e., the index of the component selected by the multiplexor, can be calculated and is denoted by $\mathrm{sel}(s)$.

The cryptanalytic algorithm tries to reconstruct the internal state in a recursive way using available encrypting bits. The central function in the attack is the recursive function **reconstruct**. A C-like description of this function is given in Fig. 10.1.

The function 'reconstruct' takes as input a partly known internal state q and a recursion depth. The internal state is reconstructed by considering the encrypting bits at time steps dictated by the time step sequence $\tau_0\tau_1\ldots$ The initial call of 'reconstruct' has depth 0 and takes a (completely unknown) state q. In 'reconstruct',

```
**reconstruct**$(q, d)$
{
$\alpha' = \alpha \cap \nu(q)$ ;
for (all $c_1 \in \mathbb{Z}_2^{\#\alpha'}$ ) {
    $q|_{\alpha'} = c_1$ ; $^\star$
    $i = 0$ ;
    contradiction = FALSE ;
    while ( $i < \ell$ and NOT contradiction) {
            if ( $z_i^{\tau_d}$ = NOT unknown ) {
              $k = \mathrm{sel}(q + F^{\tau_d} r_i)$ ;
              if ( $k \in \nu(q)$ ) { $q|_k = z_i^{\tau_d} + F^{\tau_d} r_i|_k$ ; }
              else { if ( $q_k \neq z_i^{\tau_d} + F^{\tau_d} r_i|_k$) { contradiction = TRUE ; } }
              }
            $i = i + 1$ ; }
    if ( NOT contradiction) {
      if ( $\nu(q) = \emptyset$ and $d > \mathrm{d_{lim}}$) { print($s^0 = F^{-\tau_d} q$) ; **STOP ;**}
      else {
           for (all $c_2 \in \mathbb{Z}_2^{\#\lambda}$) {
               $q' = q$ ;
               $q'|_{\lambda^{\star\star}} = c_2$ ;
               **reconstruct**$(F^{\tau_{d+1} - \tau_d} q', d + 1)$ } } } }
}
```

* The unknown components of q with indices in α are assigned a value such that sel(q) can be calculated.

** $\lambda \subseteq \nu(q)$ should be the smallest set such that $\#\nu(F^{\tau_{d+1}-\tau_d} q') \leq \#\nu(q)$.

Figure 10.1: The recursive function that forms the core of the attack.

α' is the set of indices in α corresponding to unknown components and λ is typically the subset of $\nu(q)$ corresponding to components that affect many bits of the successor state of q. The sizes of α' and λ are important factors in the divergence and convergence of the recursive procedure. The algorithm is considered successful if it is able to reconstruct all components of the internal state at a certain time τ_d with d large enough to guarantee that the correct internal state is obtained.

In Fig. 10.2 the effect of 'reconstruct' is demonstrated on a typical multiplexor generator with toy dimensions. In this example, $\alpha = \{18, 21, 24, 27\}$ and $\lambda = \{28\}$ or $\emptyset$.

The work factor of the attack can be expressed by the total number of 'reconstruct()' function calls. The algorithm is a recursive tree search, where the tree is pruned by two mechanisms:

- **knowledge from previous steps**: as the recursion depth grows, $\#\nu(p)$ shrinks and with it $\#\alpha'$ and $\#\lambda$,
- **contradictions**: the probability a contradiction grows as $\nu(p)$ shrinks.

The most important factor for the efficiency of the attack is the number of initializations. If this is small the efficiency of the attack depends strongly on the specific feedback polynomials of the linear feedback shift registers and the arrangement of the multiplexor inputs. The choice of the time step sequence is critical in this case. If it is approximately 2^β, the simple and very efficient attack described in the previous section can be applied.

10.2.4 The EBU MAC/packet scrambling system

In [33] the European Broadcasting Union (EBU) proposes a video-audio scrambling system that has already been implemented in a number of systems (some operational, some in development). In this system the video component of a TV-signal is scrambled by line rotation. There are two options: in *double cut mode* both the luminance component and the color difference component are rotated, and in *single cut mode* only the color difference component is rotated. The range of possible rotations for a line consists of 256 equidistant rotation intervals. The 8-bit number specifying the rotation interval is generated by a multiplexor generator consisting of a 29-bit and a 31-bit linear feedback shift register, filtered by a 32 to 1 multiplexor.

Every 256 frames a fresh key K is supplied. At the beginning of each frame the encryption scheme is initialized by loading an initial state that is a linear combination of the 60-bit key K and the 8-bit serial number i of the frame. If a certain portion of the image contents is known, the attack described in the previous section is directly applicable. In the case of a completely automated *descrambling* device this can however not be assumed. In this case we can exploit the correlations between corresponding lines in subsequent frames.

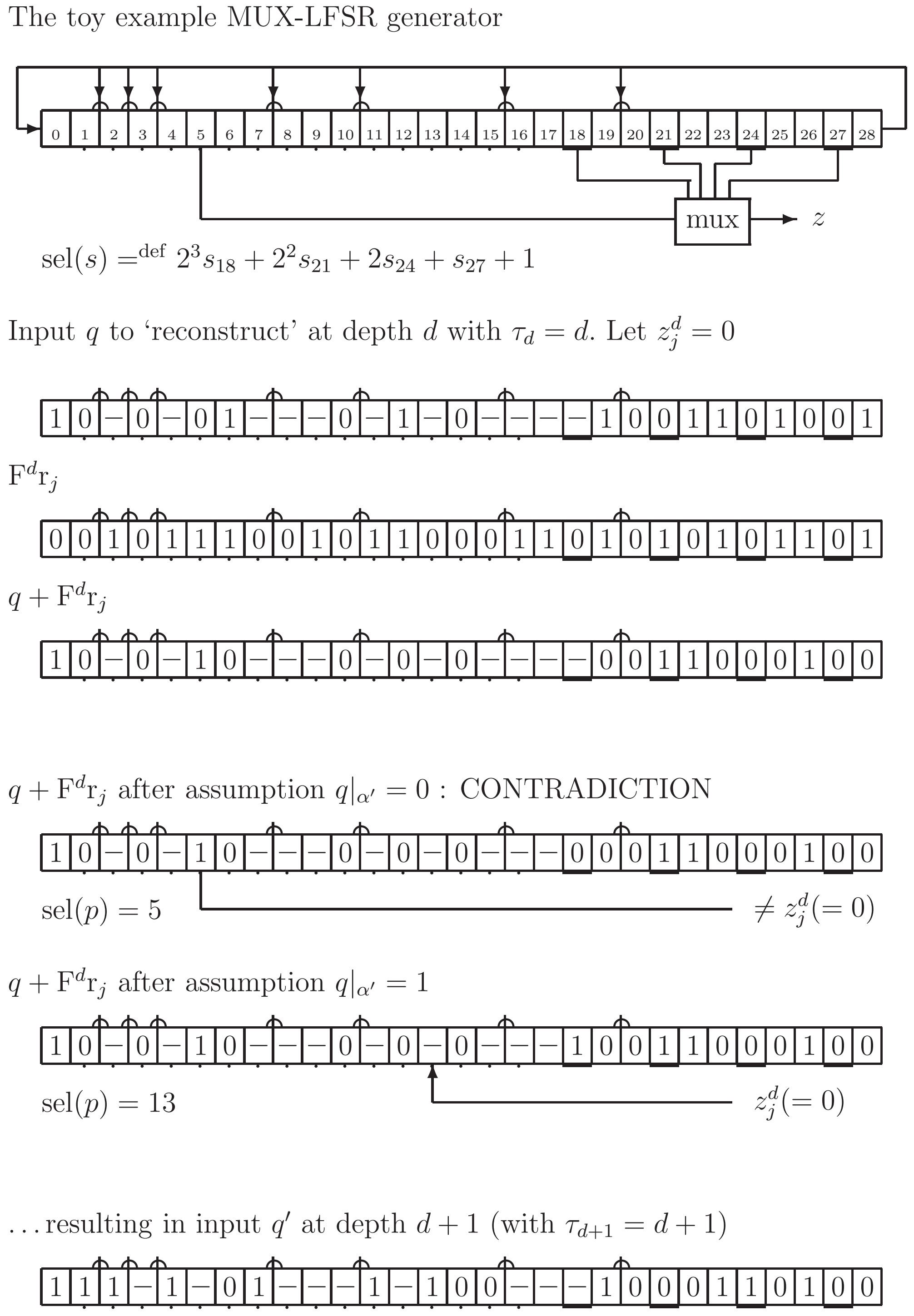


Figure 10.2: Effect of 'reconstruct' demonstrated on a toy example MUX-LFSR generator.

If r_i is the 8-bit vector that corresponds to the binary representation of i, we have

$$s_i^0 = K + \mathrm{M}\mathrm{r}_i \ , \tag{10.11}$$

with M is a 60×8 matrix. The bitwise difference between the initial states corresponding to two consecutive frames with numbers $2j$ and $2j+1$ is equal to

$$s_{2j+1}^0 + s_{2j}^0 = \mathrm{M}(\mathrm{r}_{2j+1} + \mathrm{r}_{2j}) = \mathrm{M}\mathrm{r}_1 = \mathrm{m} \ , \tag{10.12}$$

with m the first column of M. The difference between states with index $2j$ and $2j+1$ at time t is

$$s_{2j+1}^t + s_{2j}^t = \mathrm{F}^t\mathrm{m} \ . \tag{10.13}$$

The output bits are used in chunks of 16 bits per line. In the double cut mode 8 bits are used for luminance rotation and 8 for color difference rotation. In single cut mode there is no luminance rotation and only 8 of the 16 bits are used. The ℓth bit of the 16 bits used for scrambling line p in frame i is equal to $z_i^{16p+\ell}$.

Let τ be chosen in such a way that z_i^τ is the MSB of an 8-bit word to scramble a line in frame i, and that

$$\mathrm{F}^\tau \mathrm{m}|_\alpha = 0 \ . \tag{10.14}$$

In this case we have

$$s_{2j+1}^\tau|_\alpha = s_{2j}^\tau|_\alpha \ . \tag{10.15}$$

In other words, in both cases the multiplexor selects the same bit position. Let $z_{2j+1}^\tau + z_{2j}^\tau$ be denoted by e_j. We have

$$e_j = \mathrm{F}^\tau m|_{a_j} \text{ with } a_j = \mathrm{sel}(s_{2j+1}^\tau) = \mathrm{sel}(s_{2j}^\tau) \ . \tag{10.16}$$

If e_j was known, all values of a_j with $e_j \neq \mathrm{F}^\tau m|_{a_j}$ could be excluded as candidates for the correct solution.

In fact, e_j is not known but a prediction can be made based on the relative rotation between the line in frame $2j$ and the line in frame $2j+1$. If the z^τ are the MSB's of the rotation intervals of the lines, and ρ is the reduced (to the interval $(-\frac{1}{2}, +\frac{1}{2}]$) difference in rotation between the two lines, it can easily be shown that

$$\Pr(e_j = 1) = |2\rho| \ . \tag{10.17}$$

It follows that values of ρ near 0 or $\frac{1}{2}$ give reliable predictions.

This leaves us with the problem of finding the relative rotation between two lines in consecutive frames. If we suppose that the contents of the two unscrambled lines only differ slightly, ρ is indicated by a peak in the crosscorrelation between the two lines. The crosscorrelation can efficiently be calculated by A/D converting the lines to (256 or 512-component) arrays and using the Fast Fourier Transform. The magnitude of the highest crosscorrelation peak is a measure for the reliability of the prediction. The reliability can further be augmented by applying nonlinear filtering and/or edge detection. A threshold can be fixed for discarding the least reliable predictions.

The result of this process is a set of predictions of relative rotations ρ_i that can be converted to a set of probabilities $p_i = \Pr(e_i = 1)$. These probabilities can be used to calculate the most probable value for a_0. The vectors a_0 and a_i are linked by

$$a_0 + a_i = \mathrm{F}^{\tau}\mathrm{M}\,\mathrm{r}_i|_{\alpha} \ . \tag{10.18}$$

For ease of notation the right-hand expression will be denoted by y_i.

The values p_i can be used to iteratively modify a probability distribution for a_0, denoted by π_{a_0}. Let $\zeta(a)$ be defined as $\zeta(a) = \mathrm{F}^{\tau}\mathrm{m}|_a$. The contribution of each p_i can be seen as an adaptation of the distribution π_{a_0} to π'_{a_0} according to

$$\pi'_{a_0}(v) = \frac{\zeta(v+\mathrm{y}_i)p_i+(1-\zeta(v+\mathrm{y}_i))(1-p_i)}{\sum_w(\zeta(w+\mathrm{y}_i)p_i+(1-\zeta(w+\mathrm{y}_i))(1-p_i))\pi_{a_0}(w)}\pi_{a_0}(v) \ . \tag{10.19}$$

A similar adaptation is performed for all obtained p_i. It can be shown that the order of the adaptations does not affect the final distribution. The initial distribution is flat. For a typical video signal this will converge very fast to a distribution with a single peak. If there would be no uncertainty about the e_ℓ, 5 or 6 values would be sufficient to fix u_0 with certainty.

Execution of the described procedure gives 5 state bits of s_0^{τ}. This can be converted to 5 linear equations for the state bits of s_0^0. This state is completely fixed if 60 linear equations are available. Therefore, the procedure has to be performed for at least 12 lines in the frame. Since the equations obtained for s_0^0 and s_1^0 are always equal and s_0^0 and s_1^0 are known to be different, in principle only 59 equations can be found. The remaining bit can easily be fixed by checking the correlation between subsequent lines in one frame for both possibilities.

The descrambling information for a certain 256-frame block is only known after reception of the complete block. Therefore, the descrambler will have to be equipped with a video storage device such as a VCR. All other calculations are straightforward and can easily be executed by simple (existing) dedicated hardware devices, controlled by a personal computer.

The sound and data scrambling cannot be attacked in this way. However, in the assumption that more than 70 consecutive plaintext bits are known to the cryptanalyst, this part can be cracked by an attack of the type described in Sect. 10.2.3. The determination of every key that is used for only 10 seconds takes about 2^{38} function calls of the 'reconstruct' function. On-line cracking would require a parallel machine with 30,000 dedicated 'reconstruct' chips working at a million iterations per second. Although this attack is no real threat for the system, it is considerably more efficient than exhaustive key search (key length: 60 bits).

10.3 Ivan Damgård's CA hash function

In [29] Ivan Damgård presented his well-known design principle for cryptographic hash functions. The same paper also contained three concrete hash function designs

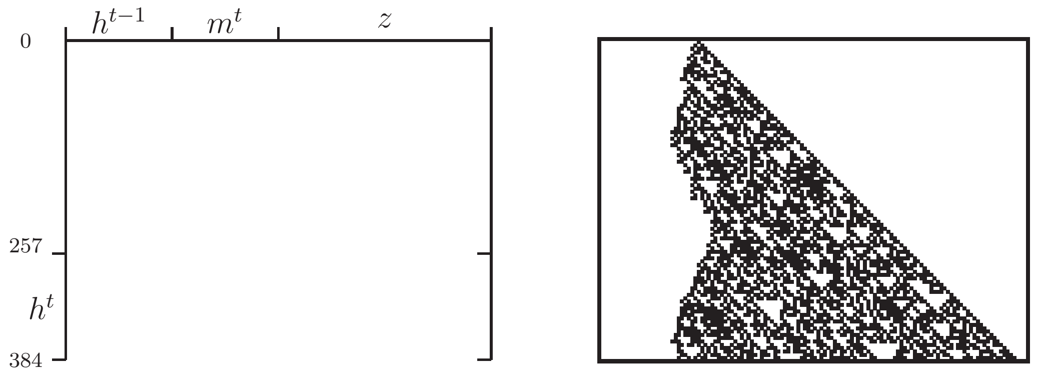


Figure 10.3: Left: Space-time diagram of the cellular automaton of $\mathrm{G}(m, h)$. Right: typical example of a differential trail in this cellular automaton.

based on this principle. One of these has a chaining transformation based on a cellular automaton (CA) and is inspired by the work of Stephen Wolfram in [104]. In this section we show that for this hash function collisions can be generated ad libitum. A modification in the first few (< 20) bits of a message block leaves the hash result unaffected with high probability ($> 50\%$). These results were originally published in our paper [11].

For the cryptographic hash function under attack the emphasis on N-reductionist design principles has led to a neglect of even the most basic correlation and propagation evaluation. The detected weaknesses would have been discovered in doing some simple general propagation simulations of the type described in [104]. Apparently, the designer did not carefully read [104] and has neglected to do any propagation or correlation analysis at all.

The chaining transformation of the proposed hash function computes a 128-bit chaining state h^t from the previous chaining state h^{t-1} and the message block m^t and is denoted by $h^t = \mathrm{G}(m^t, h^{t-1})$. This chaining transformation is specified in terms of a finite cellular automaton of length 512 with periodic boundary conditions. The initial state, denoted by $a^{(0)}$, consists of the concatenation of m^t, h^{t-1} and a constant 256-bit string z. The state updating transformation of the cellular automaton is specified by:

$$a_i^{(j+1)} = a_{i-1}^{(j)} + (a_i^{(j)} + 1)(a_{i+1}^{(j)} + 1) \ . \tag{10.20}$$

The computation of h^t involves 384 iterations of this cellular automaton. The output h^t consists of the concatenation of the bits with index 0 of the internal states $a^{(257)}$ to $a^{(384)}$. We have $h^t = a_0^{(257)} a_0^{(258)} \ldots a_0^{(384)}$.

The work factor of this hash function is approximately 3 iterations of the 512-bit CA per input bit. For Substream this is only a single iteration of the Subterranean finite state machine per 16 input bits. In the assumption that the attainable clock rate for the CA would be a factor 3 higher than for Subterranean, a hardware implementation of Subhash would still be a factor 16 faster than a hardware implementation of the CA hash function. In software, a single iteration of the CA

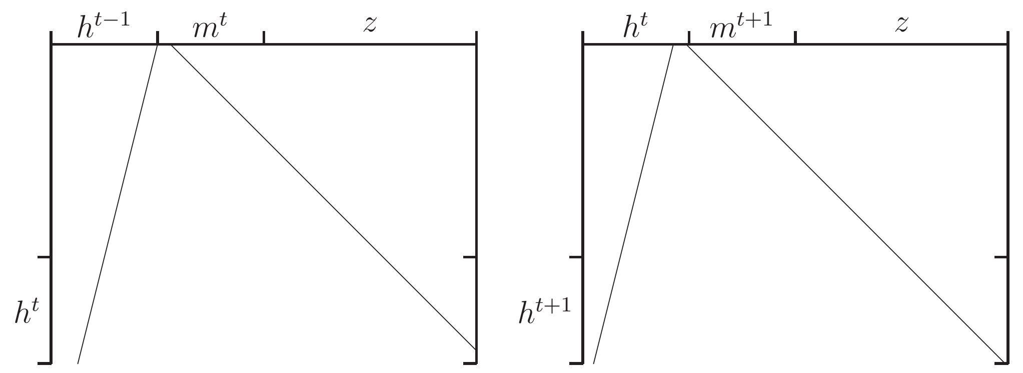


Figure 10.4: The two phases in generating collisions.

takes approximately 64 bitwise Boolean and 64 shift operations on a 32-bit processor. Hence, the hashing of a single byte takes more than 1500 bitwise Boolean and 1500 shift operations. In contrast, our dedicated design StepRightUp takes less than 4 bitwise Boolean operations and a single shift per byte.

Figure 10.3 gives the space-time diagram of the cellular automaton computation. The top line of the diagram is the initial state a^0 consisting of the concatenation of h^{t-1}, m^t and z. Because of the periodic boundary conditions, the right and left edge are in fact adjacent and the diagram can be seen as an unfolded cylinder with a vertical axis. In this diagram h^t is a temporal sequence and h^{t-1} and m^t are spatial sequences.

The right-hand side of Fig. 10.3 depicts a typical differential trail for the specific cellular automaton. It can be seen that the small initial difference pattern expands in time, confined between an irregular left edge and a right edge consisting of a straight line with slope 1 (bit in space per bit in time). For the differential trail in Fig. 10.3 the left edge can be approximated by a straight line with a very small slope. In [104], it is stated that empirical measurements suggest that the average asymptotic value of this slope is about 0.24, but that in individual cases the actual slope can deviate significantly from this average. This was confirmed in our experiments. Generating collisions now becomes a matter of some simple geometry.

Suppose we have a difference pattern in the first 16 bits of m^t. This gives rise to a difference pattern between bit 128 and 144 of a^0. This has expanded to bit 512 ($= 0$) at the right after $512 - 144 = 368$ iterations. Given a slope of 0.24, it has expanded at the left to bit $128 - 0.24 \cdot 384 = 35$ after 384 iterations. Clearly, the differential trail affects only the last $384 - 368 = 16$ bits of h^t. In the subsequent application of the chaining transformation the difference pattern in h^t gives rise to a difference pattern in a^0 between bit 112 and 127. After 384 iterations this difference pattern has expanded to bit $127 + 384 = 511$ at the right and bit $112 - 0.24 \cdot 384 = 19$ at the left, leaving a^0 and therefore h^{t+1} undisturbed. Hence, complementing a few bits at the beginning of a message block m^t affects the chaining state of two iterations later, h^{t+1}, only with small probability. This mechanism is illustrated in Fig. 10.4.

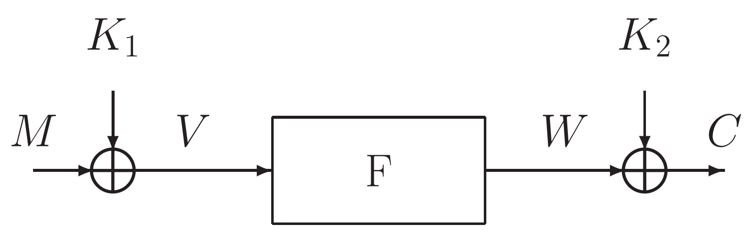


Figure 10.5: The Even-Mansour block cipher construction.

10.4 The Even-Mansour construction

In [34] it is proposed to build a block cipher from a key-independent invertible transformation F and two bitwise key additions. The transformation F is publicly known and it is easy to compute $\mathrm{F}(x)$ and $\mathrm{F}^{-1}(x)$ for any given input $x \in \mathbb{Z}_2^{n_\mathrm{b}}$. The cipher key consists of two n_b-bit subkeys K_1 and K_2. The relation between a plaintext block M and its ciphertext block C is given by $C + K_2 = \mathrm{F}(M + K_1)$. This is illustrated in Fig. 10.5.

We present two very simple differential attacks that result in a remarkably low upper bound for the security attainable by block ciphers with this construction. These attacks were presented at the same conference as the Even-Mansour construction itself and were originally published in [13].

10.4.1 Differential cryptanalysis

A known-plaintext attack

The cryptanalyst knows two plaintexts and their corresponding ciphertexts denoted by M, M^*, C and C^*. Let the bitwise difference of two blocks be denoted by $X' = X + X^*$ and let $V = M + K_1$ and $W = C + K_2$. The attack consists of calculating $W' = \mathrm{F}(V) + \mathrm{F}(V + M')$ for all possible values of V. If $V = M + K_1$ or $V = M^* + K_1$, we have $W' = C'$. For almost all possible permutations F only a few V values will be found for which this holds. Wrong values for V are discarded by checking that $\mathrm{F}(V) \neq M + K_1$ and $\mathrm{F}(V^*) \neq M^* + K_1$. When the correct value of V is known, the two subkeys can easily be calculated. This attack takes on the average $2^{n_\mathrm{b}-1}$ applications of F. The work factor of this attack corresponds to that of exhaustive key search with an effective key length of only 1/2 of the actual key length.

A chosen-plaintext attack

The cryptanalyst chooses ℓ plaintext pairs M_i, M_i^* with the same bitwise difference M' and obtains the corresponding ciphertexts C_i, C_i^*. For $\ell < 2^{n_\mathrm{b}/2}$ there are close to ℓ different C_i' values, speeding up the search for a $W' = C_i'$ for some i by a factor of ℓ as compared to the known-plaintext attack. The magnitude of ℓ is limited by the memory requirement of $n_\mathrm{b}\ell$ bits. The expected number of evaluations of F in this attack is $\ell + 2^{n_\mathrm{b}}/\ell$. This is minimized for $\ell = 2^{n_\mathrm{b}/2}$. Hence, in the absence of memory restrictions the work factor of this attack is comparable to that of exhaustive key search with an effective key length of only 1/4 of the actual key length.

Both attacks can easily be parallelized. In the described attacks no advantage is taken from possible internal structure of the permutation F.

Example : Let Ψ be an Even-Mansour type block cipher with block length 64 and key length 128. Suppose an F-processor is a dedicated chip that computes F in 100 microseconds. A cryptanalyst performing a 2 MByte chosen-plaintext attack ($\ell \approx 10^5$) on Ψ using a parallel machine with 10^6 F-processors will recover the key in a matter of hours.

10.4.2 Discussion

In [34] computational complexity theory is employed to recommend a construction that can equally be considered as a *restriction* on the key schedule of a block cipher.

Even and Mansour prove that a polynomially bounded adversary has a negligible probability of success when attacking their scheme if permutation F is pseudorandomly 'chosen'. The work factor of an attack with non-negligible probability of success is a superpolynomial function of the block length n_{b} for large enough n_{b}. The two presented attacks do not contradict this proof since their work factor is an exponential function of the security parameter n_{b}. We think this is a nice indication of the relevance of the proof given in [34], and of computational complexity theoretic proofs of security in general.

Additionally, the complexity theoretic proof of security is only applicable if the permutation F is 'chosen' in a 'pseudorandom' way. This means that F must in some way be based on a computationally hard problem. One method to obtain this is by forming F with the construction of Michael Luby and Charles Rackoff presented in [66]. The "random functions" used in this scheme can be constructed as outlined by Oded Goldreich, Shafi Goldwasser and Sylvio Micali in [43], using a cryptographic sequence generator with a security based on some general complexity theoretic assumption. This can be used to define a concrete block cipher by the unavoidable fixing of the block length. Despite the involved construction, *no guarantee* on the security of this block cipher can be given. Moreover, it will definitely be very, very slow.

A number of statements in the Even-Mansour paper illustrate how awkward their point of view actually is. In their paper they state that the construction "removes the need to *store*, or generate a multitude of permutations." Furthermore it reads "The scheme may lead to a system *more efficient* than systems such as the DES and its siblings, since the designer has to worry about one thing only: How to implement one pseudorandomly chosen permutation? This may be easier than *getting one for each key*." (the italics are ours).

10.5 Conclusions

Resynchronization attacks are a new type of attack on synchronous stream encryption schemes. We have shown that most LFSR-based stream ciphers are very weak if frequent resynchronization is applied and that it is necessary to consider the initialization mapping as part of the encryption scheme.

The two other attacks have been included to illustrate the limitations of certain theoretical design principles.

Chapter 11

Schemes Based on Modular Arithmetic

11.1 Introduction

Arithmetic operations have only moderate portability and are not very well suited for dedicated hardware implementations. They have however a simple description and are widely available as fast (co-)processor instructions. An additional stimulus for the use of arithmetic operations lies in the fact that dedicated cryptographic coprocessors that also have to perform large-integer based public-key schemes are especially designed to efficiently execute arithmetic operations.

A disadvantage of arithmetic operations is that their algebraic properties can possibly be exploited in cryptanalysis. In this chapter we present two examples of cryptographic designs that have important weaknesses due to the algebraic structure of their building blocks. First we present a procedure to generate collisions for the cryptographic hash function FFT-hash, a design of Claus Schnorr [96]. Then we show that the block cipher IDEA, designed by Xuejia Lai and Jim Massey [63], has large classes of weak keys.

The weaknesses in FFT-hash and IDEA are due to the fact that arithmetic operations are used with propagation and correlation properties that are good on the average but very poor in the worst case. We propose to use cyclic multiplication as a component and show that there exist multiplication factors that have good worst-case difference propagation properties. We conclude this chapter with the treatment of our block cipher design MMB. To avoid confusion, in this chapter bitwise addition will be denoted by "$\oplus$," (modular) addition by "$+$" and (modular) multiplication by "$\cdot$".

11.2 Collisions for FFT-Hash

At the rump session of Crypto '91 Claus Schnorr presented FFT-Hash, a cryptographic hash function with a hash result of 128 bits [96]. During the Summer of '91 Antoon Bosselaers and I wrote a program that generates sets of 384-bit messages with elements that all hash to the same result using FFT-Hash. The CPU time consumed is of the order of a few hours on a microVAX. Our collisions were announced at the rump session of Asiacrypt '91 [12]. Later, collisions were also found independently by Thierry Baritaud, Henri Gilbert and Marc Girault [2]. In this section we give a description of our procedure to generate collisions for FFT-Hash.

11.2.1 The chaining transformation

The chaining transformation converts a 128-bit chaining state h^{t-1} and a 128-bit message block m^t into a new chaining state h^t. This is denoted by

$$h^t = \mathrm{G}(h^{t-1}, m^t) \ . \tag{11.1}$$

The input consists of the concatenation of h^{t-1} and m^t. In the computation of the chaining transformation this 256-bit string is treated as a 16-component vector $(a_0, \ldots, a_{15})$ with the components a_i representing integers modulo $2^{16}+1$. The so-called *FFT-step*, denoted by $b = \textsc{fft}(a)$, is defined by

$$b_{2i} = \sum_{j=0}^{7} 2^{4ij} a_{2j} \bmod 2^{16}+1 \ , \tag{11.2}$$

for the even components and by $b_{2i+1} = a_{2i+1}$ for the odd components. The so-called *recursion step*, denoted by $b = \textsc{rec}(a)$, consists of an invertible nonlinear transformation. It can best be defined by a pseudo-C program :

```
$\textsc{for}(i = 0; i < 16; i++)$
  $b_i = a_i$ ;
$\textsc{for}(i = 0; i < 16; i++)$
  $b_i = (b_i + b_{i-1}b_{i-2} + b_{b_{i-3}} + 2^i) \bmod 2^{16}+1$ ,        (11.3)
```

with all indices taken modulo 16. In the chaining transformation the input is subject to

$$\textsc{rec} \circ \textsc{fft} \circ \textsc{rec} \circ \textsc{fft} \ . \tag{11.4}$$

The output of G consists of the last 8 components of the resulting vector, i.e., $h^t = a_8 \| a_9 \ldots \| a_{15}$, with all occurrences of 2^{16} ($= 10000_{\mathrm{hex}}$) substituted by 0000_{hex}.

11.2.2 Generating collisions

The generation of collisions involves the exploitation of some particular propagation properties of the chaining transformation.

The FFT step only affects the components with even index. For odd-indexed components no propagation takes place during this step. Moreover, its linearity can be exploited to impose certain values upon a number of output components. If for certain subsets of no more than 8 components, belonging to either the output or the input, the values are fixed, values for the remaining components can be computed such that (11.2) holds. This computation involves only linear algebra.

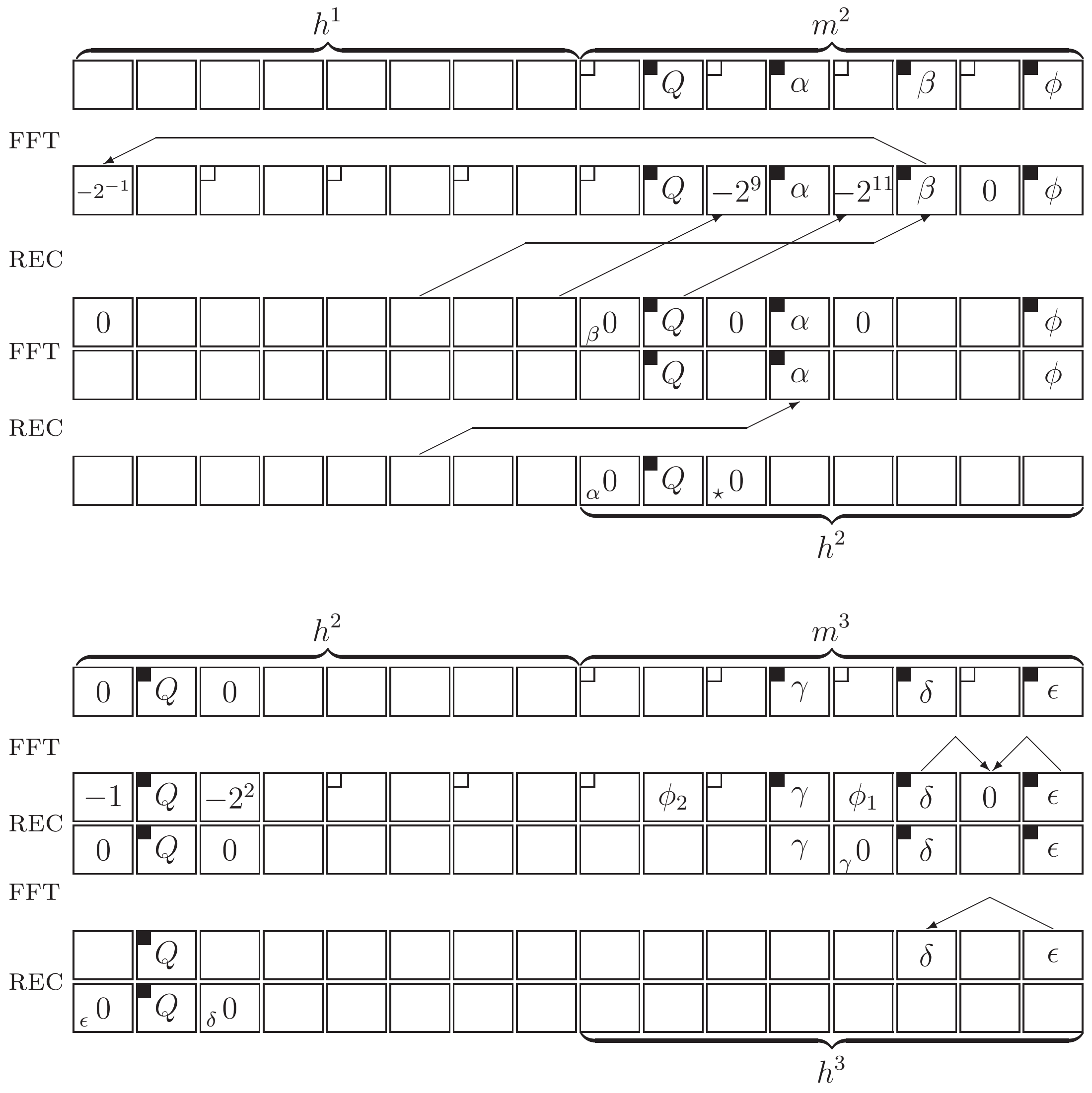


Figure 11.1: Schematic overview of the collisions of FFT-Hash. The state $(a_0, \ldots, a_{15})$ is depicted before and after every step.

In the recursion step, the diffusion can be eliminated locally by imposing that certain components are 0. Let $b = \text{REC}(a)$ and suppose that $b_5 = b_7 = 0$. In that case both product terms $b_{i-1}b_{i-2}$ in (11.3) that contain b_6 are 0. Suppose also that b_6 has not been addressed in the indirect indexing term $b_{b_{i-3}}$ in (11.3) for any i. This is the case if $a_i \bmod 16 \neq 6$ for $i > 12$ and $b_i \bmod 16 \neq 6$ for $i \leq 12$. Under the described conditions, a change in the 12 MSB's of a_6 only affects the 12 MSB's of b_6. a_6 is said to be *isolated*. This can be applied to any component. Hence, isolation of (the 12 MSB's of) a component in a recursion step requires that the two neighboring components are 0 *and* that it is not addressed in the term $b_{b_{i-3}}$ for any i.

The collision generating attack is a probabilistic procedure that exploits the possibility of isolating a component during all four steps of the chaining transformation. The colliding messages consist of 3 blocks: m^1, m^2 and m^3. All effort goes into the search for appropriate m^2 and m^3 values. These steps are illustrated in Fig. 11.1. A subset of message bits are given randomly chosen values thereby fixing the remaining bits through a number of imposed relations. Starting from intermediate hash result $h^1 = \mathrm{G}(IV, m^1)$ we have:

1. Calculation of m^2. The values are chosen in a way that the second component of m^2 $(= a_9)$ has a high probability of staying isolated throughout the calculation of $G()$. Certain changes in the 12 MSB's of this component affect the intermediate hash value h^2 only in the second component. On the average 2^{23} different h^1, obtained by trying different m^1 values, have to be tested. Only about 2^{11} of these survive a first check. For each of these remaining m^2 values 2^{15} trials have to be performed by varying ϕ (see Fig. 11.1).

2. Calculation of m^3. The values of m^3 are chosen in such a way that the second component of h^2 $(= a_1)$ has maximum probability of being isolated and thus does not affect h^3. About 2^{22} different values of ϕ_1 and ϕ_2 have to be tried.

In Fig. 11.1 Q indicates the component that is isolated throughout the complete calculation. An arrow from a_i to a_j means that $a_{a_i} = a_j$ or, equivalently, $a_i \bmod 16 = j$. Boxes containing a constant indicate that the given value is imposed upon the component. Boxes containing a Greek letter indicate components that are isolated (denoted by "■ ") until they are *used* (as indicated in the down left corner) to impose a certain value on a component. A "⋆" in the lower left corner indicates that we depend on chance (Pr : 2^{-16}). The symbol "⊡" indicates that the component is fixed by an FFT relation. An empty box denotes a component that is fixed by initial values and/or internal relations.

The first result obtained by this method was a set of 805 colliding messages (printed as hexadecimal values)

```
00a1 0000 0000 0000 0000 0000 000c 5b18
9156 XXXd 9e89 67e8 35f8 e2b0 12ec 26c0
570b 06ee ba21 8da5 6ec4 c27e 5d5d e6be ,
```

where `XXX` ranges over `1b5` to `4d9` that all hash to

```
527d c019 d8cb 1d92 162b f04c cfff 26c6 .
```

11.3 Weak keys of IDEA

In May 1990, a block cipher called PES (Proposed Encryption Standard) was introduced by Xuejia Lai and Jim Massey [62]. PES has a block length of 64 bits and a key length of 128 bits. The design is based on the concept of "mixing operations from different algebraic groups." The group operations involved are addition modulo 2^{16}, multiplication modulo $2^{16}+1$ and bitwise addition of 16-bit words. Triggered by the emergence of differential cryptanalysis the designers (joined by Sean Murphy) presented a modification of PES in April 1991, called IPES (Improved PES) [63].

r	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6
1	0–15	16–31	32–47	48–63	64–79	80–95
2	96–111	112–127	25–40	41–56	57–72	73–88
3	89–104	105–120	121–8	9–24	50–65	66–81
4	82–97	98–113	114–1	2–17	18–33	34–49
5	75–90	91–106	107–122	123–10	11–26	27–42
6	43–58	59–74	100–115	116–3	4–19	20–35
7	36–51	52–67	68–83	84–99	125–12	13–28
8	29–44	45–60	61–76	77–92	93–108	109–124
9	22–37	38–53	54–69	70–85	—	—

Table 11.1: Derivation of the encryption round keys of the global 128-bit key. The key bits are indexed starting from 0. The most significant bits (MSB) of the round keys are the bits with the lowest global index.

This cipher has been commercialized under the name IDEA (International Data Encryption Algorithm).

In the Fall of '92 we discovered that there are large classes of keys for which IDEA has detectable weaknesses. These results were first announced at SPRC '93 in Rome [18] and were presented at Crypto '93 [21]. For a general chosen-plaintext attack that efficiently breaks 2.5 round versions of IDEA, we refer to our paper [27].

11.3.1 Description of the cipher

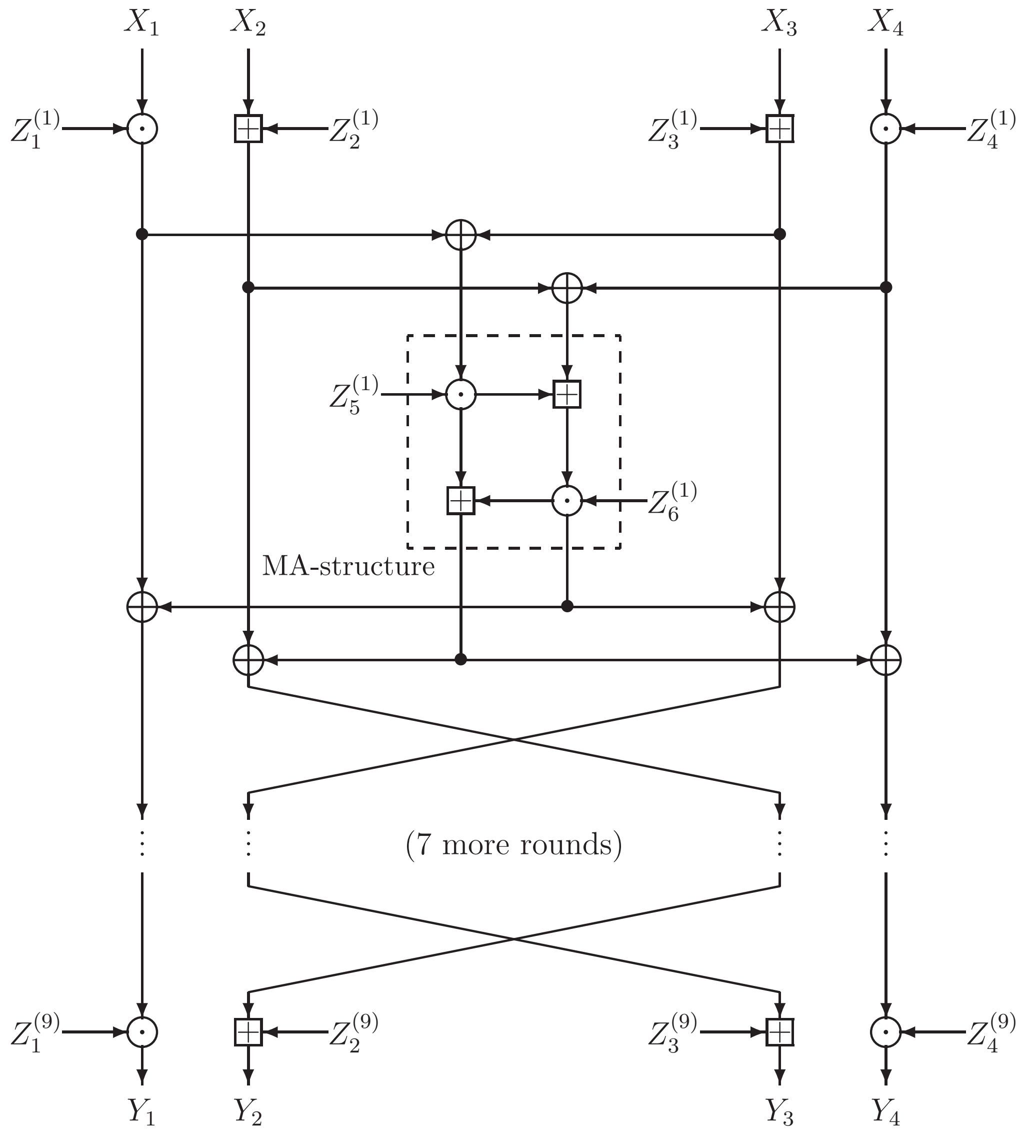
IDEA is an iterated block cipher consisting of 8 similar rounds and a single output transformation. With exception of the key schedule, the IDEA decryption process is the same as its encryption process. For the specification of IDEA we refer to Fig. 11.2. The encryption round keys are 16-bit substrings of the global key and are specified in Table 11.1. The decryption round keys must be derived from the encryption round keys.

11.3.2 Unfavorable properties of the building blocks

In this section we show that the quality of the propagation and correlation properties of the building blocks strongly depends on the particular value of the key. We also show that addition modulo 2^n and multiplication modulo $2^n + 1$ by specific values have exploitable linearities.

Key-dependence of the propagation properties

The propagation and correlation properties of the IDEA round transformation depend strongly on the value of the round keys. The effect of key multiplications with keys that represent a power of 2 modulo $2^{16} + 1$ can be approximated very well by an (easy to find) affine transformation in the vector space $< \mathbb{Z}_2^{16}, \oplus >$. This category comprises all multiplicative keys with a single nonzero bit, i.e., of the form δ_i.



$X_i, Y_i, Z_i^{(r)}$: 16-bit plaintext, ciphertext and key subblocks
$\oplus$: bitwise addition $\boxplus$: addition mod 2^{16}
$\odot$: multiplication mod $2^{16}+1$ with $0000_{\text{hex}} \equiv 2^{16}$

Figure 11.2: The encryption process of IDEA.

For all the 32 keys that fall into the category, it can easily be checked that the expected Hamming distance between the correct output and the affine approximation is smaller than 1 bit and that the probability that the approximation matches the correct output is larger than 1/2.

The nonlinearity of the key additions is on the average much smaller than that of the key multiplications. If an additive key is chosen uniformly from $\mathbb{Z}_{2^{16}}$, the expected Hamming distance between the modular sum and the best affine approximation over $< \mathbb{Z}_2^{16}, \oplus >$ is 1.75 bits.

The round keys are substrings of the global key, specified by the key schedule. All global keys in the (albeit very small) set of 128-bit strings consisting of only few 1's separated by sufficiently long blocks of 0's give rise to only 'weak' round keys. For these keys the resistance against differential and linear cryptanalysis of the cipher must be realized by the nonlinearity of the additions modulo 2^{16} in the *MA-structure* (see Fig. 11.2, MA: multiplicative-additive) of the round transformation.

Linearities in the modular arithmetic

Let x_i denote the ith bit in the binary representation of the number X, i.e., $X = \sum 2^i x_i$. The bits of $Y = X + W \bmod 2^n$ are given by

$$y_i = x_i \oplus z_i \oplus c_i \ , \tag{11.5}$$

with c_i a carry bit that only depends on bits with indices *smaller than* i. The LSB of Y (y_0) is simply equal to $x_0 \oplus z_0$. Difference propagation of the MSB's of X and Z into Y is restricted to linear propagation (over $< \mathbb{Z}_2^{16}, \oplus >$) into the MSB of Y.

For the multiplication by -1 (0000_{hex}) modulo $2^{16} + 1$ as defined in the IDEA block cipher it can easily be checked that

$$-1 \odot A = \bar{A} + 2 \bmod 2^{16} \ , \tag{11.6}$$

with $\bar{A}$ the bitwise complement of A. Therefore, multiplication by -1 inherits the linearity properties of the addition modulo 2^n.

11.3.3 Classes of weak keys detectable by LC

The occurrence of multiplicative subkeys with value 1 or -1 gives rise to a linear step with a correlation contribution equal to 1. These linear steps can be revealed by expressing the linear combination of LSB's of the output subblocks of an IDEA round in terms of input and key bits.

As an example, we will express the bitwise sum of the LSB's of the first and second output subblock of a round: $y_1 \oplus y_2$ (with the indices denoting the subblock number). From Fig. 11.2 it can be seen that $y_1 \oplus y_2 = (X_1 \odot Z_1)|_0 \oplus 1 \oplus x_3 \oplus z_3$. If $Z_1 = (-)1$, i.e., if the 15 MSB's of the Z_1 are 0,

$$y_1 \oplus y_2 = x_1 \oplus x_3 \oplus z_1 \oplus z_3 \oplus 1 \ . \tag{11.7}$$

This is a linear step with correlation contribution 1 and is denoted by $(1, 0, 1, 0) \rightarrow (1, 1, 0, 0)$. Similar linear steps and their corresponding conditions on subkey blocks can be found for all 15 combinations of output LSB's and are listed in Table 11.2.

linear factor	Z_1	Z_4	Z_5	Z_6
$(0,0,0,1) \rightarrow (0,0,1,0)$	-	$(-)1$	-	$(-)1$
$(0,0,1,0) \rightarrow (1,0,1,1)$	-	-	$(-)1$	$(-)1$
$(0,0,1,1) \rightarrow (1,0,0,1)$	-	$(-)1$	$(-)1$	-
$(0,1,0,0) \rightarrow (0,0,0,1)$	-	-	-	$(-)1$
$(0,1,0,1) \rightarrow (0,0,1,1)$	-	$(-)1$	-	-
$(0,1,1,0) \rightarrow (1,0,1,0)$	-	-	$(-)1$	-
$(0,1,1,1) \rightarrow (1,0,0,0)$	-	$(-)1$	$(-)1$	$(-)1$
$(1,0,0,0) \rightarrow (0,1,1,1)$	$(-)1$	-	$(-)1$	$(-)1$
$(1,0,0,1) \rightarrow (0,1,0,1)$	$(-)1$	$(-)1$	$(-)1$	-
$(1,0,1,0) \rightarrow (1,1,0,0)$	$(-)1$	-	-	-
$(1,0,1,1) \rightarrow (1,1,1,0)$	$(-)1$	$(-)1$	-	$(-)1$
$(1,1,0,0) \rightarrow (0,1,1,0)$	$(-)1$	-	$(-)1$	-
$(1,1,0,1) \rightarrow (0,1,0,0)$	$(-)1$	$(-)1$	$(-)1$	$(-)1$
$(1,1,1,0) \rightarrow (1,1,0,1)$	$(-)1$	-	-	$(-)1$
$(1,1,1,1) \rightarrow (1,1,1,1)$	$(-)1$	$(-)1$	-	-

Table 11.2: Linear factors in the round transformation with conditions on the subkeys.

These linear steps can be combined into multiple-round linear trails with correlation contributions equal to 1. For every round this gives conditions on subkeys that can be converted to conditions on global key bits using Table 11.1. An example is given in Table 11.3 for the 8-round linear trail with initial selection vector $(1,0,1,0)$ and terminal selection vector $(0,1,1,0)$. The global key bits corresponding to indices that appear in this table must be 0. Since key bits with indices in 26-28, 72-74 or 111-127 do not appear, there are 2^{23} global keys that have this linear trail. This is called a *class of weak keys* since membership can easily be checked by observing some corresponding plaintext-ciphertext combinations.

round	input term	Z_1	Z_5
1	$(1,0,1,0)$	0–14	-
2	$(1,1,0,0)$	96–110	57–71
3	$(0,1,1,0)$	-	50–64
4	$(1,0,1,0)$	82–96	-
5	$(1,1,0,0)$	75–89	11–25
6	$(0,1,1,0)$	-	4–18
7	$(1,0,1,0)$	36–50	-
8	$(1,1,0,0)$	29–44	93–107
9	$(0,1,1,0)$	-	-

Table 11.3: Conditions on key bits for linear factor $(1,0,1,0) \rightarrow (0,1,1,0)$.

characteristic	Z_1	Z_4	Z_5	Z_6
$(0,0,0,\eta) \Rightarrow (\eta,\eta,\eta,0)$	-	$(-)1$	-	$(-)1$
$(0,0,\eta,0) \Rightarrow (\eta,0,0,0)$	-	-	$(-)1$	$(-)1$
$(0,0,\eta,\eta) \Rightarrow (0,\eta,\eta,0)$	-	$(-)1$	$(-)1$	-
$(0,\eta,0,0) \Rightarrow (\eta,\eta,0,\eta)$	-	-	-	$(-)1$
$(0,\eta,0,\eta) \Rightarrow (0,0,\eta,\eta)$	-	$(-)1$	-	-
$(0,\eta,\eta,0) \Rightarrow (0,\eta,0,\eta)$	-	-	$(-)1$	-
$(0,\eta,\eta,\eta) \Rightarrow (\eta,0,\eta,\eta)$	-	$(-)1$	$(-)1$	$(-)1$
$(\eta,0,0,0) \Rightarrow (0,\eta,0,0)$	$(-)1$	-	$(-)1$	$(-)1$
$(\eta,0,0,\eta) \Rightarrow (\eta,0,\eta,0)$	$(-)1$	$(-)1$	$(-)1$	-
$(\eta,0,\eta,0) \Rightarrow (\eta,\eta,0,0)$	$(-)1$	-	-	-
$(\eta,0,\eta,\eta) \Rightarrow (0,0,\eta,0)$	$(-)1$	$(-)1$	-	$(-)1$
$(\eta,\eta,0,0) \Rightarrow (\eta,0,0,\eta)$	$(-)1$	-	$(-)1$	-
$(\eta,\eta,0,\eta) \Rightarrow (0,\eta,\eta,\eta)$	$(-)1$	$(-)1$	$(-)1$	$(-)1$
$(\eta,\eta,\eta,0) \Rightarrow (0,0,0,\eta)$	$(-)1$	-	-	$(-)1$
$(\eta,\eta,\eta,\eta) \Rightarrow (\eta,\eta,\eta,\eta)$	$(-)1$	$(-)1$	-	-

Table 11.4: Difference propagation in the round transformation with conditions on the subkeys.

11.3.4 Classes of weak keys detectable by DC

The use of multiplicative subkeys with value 1 or -1 gives rise to differential steps with a prop ratio equal to 1.

Let η be the 16-bit block 8000_{hex}, i.e., $\eta = \delta_{15}$. Apply an input difference $(X'_1, X'_2, X'_3, X'_4) = (0,0,0,\eta)$. If $Z_4 = (-)1$, the difference pattern after the application of Z_1 to Z_4 is still $(0,0,0,\eta)$. The difference pattern at the input of the MA structure is $(0,\eta)$. η propagates unchanged through the top right addition to the bottom right multiplication by Z_6. If this subkey is equal to $(-)1$, the corresponding output difference is again η. This difference propagates unchanged through the bottom left addition and the difference pattern at the output of the MA structure is (η,η). This results in an output difference Y' of the round equal to $(\eta,\eta,\eta,0)$. Hence, if the 15 MSB's of both Z_4 and Z_6 are 0, the input difference $(0,0,0,\eta)$ fixes the output difference to $(\eta,\eta,\eta,0)$. This is a differential step with a prop ratio equal to 1 and is denoted by $(0,0,0,\eta) \Rightarrow (\eta,\eta,\eta,0)$. A similar analysis can be made for any of the 15 other possible nonzero input differences where only the MSB's of the subblocks are allowed to be 1. The results are listed in Table 11.4.

These differential steps can be combined into multiple-round differential trails with a prop ratio equal to 1. The conditions on the subkeys can be read in Table 11.4.

An example of an 8-round differential trail with initial difference vector $(0,\eta,0,\eta)$ is given in Table 11.5. It can be seen that for keys with only nonzero bits on positions 26–40, 72–76 and 108–122 the output difference must be equal to $(0,\eta,\eta,0)$. This is the largest class we found, comprising a total of 2^{35} keys. Membership can be

round	input difference	Z_4	Z_5
1	$(0,\eta,0,\eta)$	48–62	-
2	$(0,0,\eta,\eta)$	41–55	57–71
3	$(0,\eta,\eta,0)$	-	50–64
4	$(0,\eta,0,\eta)$	2–16	-
5	$(0,0,\eta,\eta)$	123–9	11–25
6	$(0,\eta,\eta,0)$	-	4–18
7	$(0,\eta,0,\eta)$	84–98	-
8	$(0,0,\eta,\eta)$	77–91	93–107
9	$(0,\eta,\eta,0)$	-	-

Table 11.5: Propagation of plaintext difference $(0,\eta,0,\eta)$ in IDEA.

checked by performing two encryptions where the plaintexts have a chosen difference and observing the difference in the ciphertexts. Similar differential trails can be constructed for the 14 other possible input differences.

11.3.5 Expanding the classes of weak keys

Classes of weak keys can sometimes be significantly expanded at the cost of some more effort in the checking for membership. Omitting in Table 11.5 the conditions for the subkeys of round 8 gives rise to the class of 2^{51} keys with nonzero bits on positions 26–40, 72–83 and 99–122. We will show that both checking for membership and calculation of the specific key can be performed efficiently.

The membership test

The input difference of round 8 is equal to the output difference of round 7 and is guaranteed to be equal to $(0,0,\eta,\eta)$ by the conditions on the subkeys of the first 7 rounds. Using the fact that $Z_3^{(9)}$, consisting of global key bits 54–69, is 0 for these keys it can easily be derived that

$$Y_3' \oplus \eta = ({Z_1^{(9)}}^{-1} \odot Y_1^*) \oplus ({Z_1^{(9)}}^{-1} \odot Y_1)\ . \tag{11.8}$$

This can be verified by inspecting Fig. 11.3. In (11.8) only $Z_1^{(9)}$ is unknown. This subkey consists of global key bits 22–37. For the given class only the 12 LSB's may differ from 0. If the global key does not belong to the class of weak keys, the probability that (11.8) has a solution is 1/16. These solutions can be eliminated by some additional encryptions. Every pair of encryptions yields an equation for $Z_1^{(9)}$ similar to (11.8).

The determination of the unknown key bits

The values of the 12 unknown bits of $Z_1^{(9)}$ have already been determined by the membership test. The following step is the determination of the 3 unknown bits of

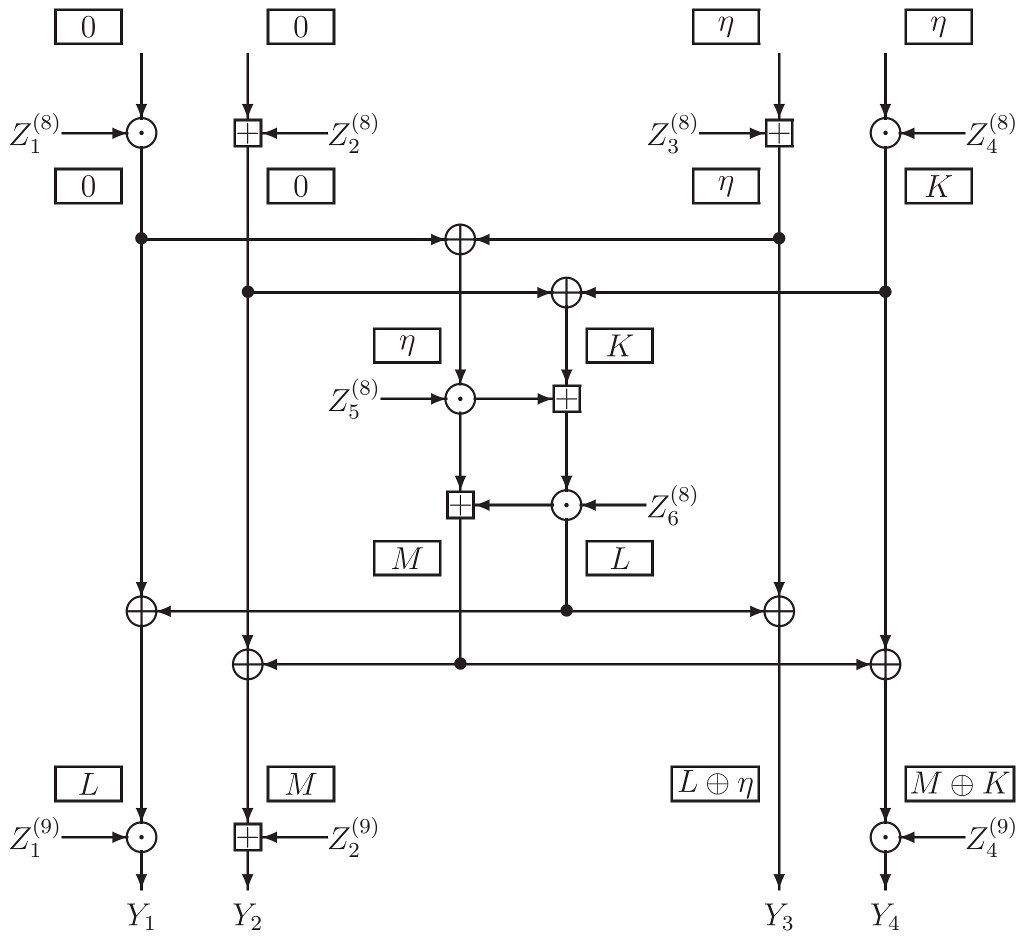


Figure 11.3: Difference propagation of $X' = (0, \eta, 0, \eta)$ through the last round of IDEA for keys with only nonzero bits on positions 26–40, 72–83 and 99–122. The bitwise differences are indicated in boxes.

$Z_2^{(9)}$, the 12 unknown bits of $Z_4^{(9)}$ and the 7 unknown bits of $Z_4^{(8)}$. A consistency check can be executed on these bits in the following way. Suppose $Z_2^{(9)}$ and $Z_4^{(9)}$ are known. In this case it is possible to compute the difference that is denoted by K in Fig. 11.3. For this value K there must be a vector A (with MSB 0) such that

$$\begin{aligned} K &= (Z_4^{(8)} \odot A) \oplus (Z_4^{(8)} \odot (A \oplus \eta)) \\ &= (Z_4^{(8)} \odot A) \oplus ((Z_4^{(8)} \odot A) + (Z_4^{(8)} \odot 2^{15}))\ . \end{aligned} \tag{11.9}$$

For a given vector K it is easy to find the possible values of $Z_4^{(8)}$. Only values of $Z_4^{(8)}$ with the 9 LSB's equal to 0 are valid. This information can be calculated in advance for every value of K and stored in an array of 2^{16} lists. The average number of possible $Z_4^{(8)}$ per K value turns out to be smaller than 1. Through this table, the observed value of K specifies a set of possible $Z_4^{(8)}$ values. If the set is empty, the chosen values for $Z_2^{(9)}$ and $Z_4^{(9)}$ must have been wrong. If the set is not empty, the K value resulting from another pair of encryptions (with input difference at round 8 equal to $(0, 0, \eta, \eta)$) can be observed. The correct value for $Z_4^{(8)}$ must be in the list for both observed values of K. This can be repeated until there is no value for $Z_4^{(8)}$ left. The correct values for $Z_2^{(9)}$ and $Z_4^{(9)}$ are found if there is a value for $Z_4^{(8)}$ that is consistent for all the (say a maximum of 8) encryption pairs. Now 34 bits are fixed. The remaining 17 bits can easily be found by exhaustively trying all remaining 2^{17} possibilities and comparing it with any plaintext-ciphertext pair obtained during the attack.

The complete work factor of the key determination consists of 16 chosen plaintext-difference encryptions, about 2^{15} modular additions, multiplications and table-lookups, and 2^{17} key search encryptions.

11.3.6 A modified IDEA without weak keys

In the present specification of IDEA the conditions for weak multiplicative round keys are converted to the condition that global key bits must be 0. In Table 11.3 and 11.5 it can be seen that many global key bits appear more than once in the conditions.

Now let $\hat{Z}_i^{(r)} = \Delta \oplus Z_i^{(r)}$ with Δ a fixed nonzero binary vector. This can be seen as a kind of subkey *biasing*. If in IDEA the subkeys $Z_i^{(r)}$ are replaced by $\hat{Z}_i^{(r)}$, the conditions for weak multiplicative keys are converted to the condition that some global key bits must be 0 and some must be 1. The vector Δ must be chosen in such a way that for all potential multiple-round linear and differential trails, the conditions on the subkeys give conflicting conditions on global key bits. Because of the large overlap between subkeys, the exact value of Δ is not critical. For instance, for $\delta = 0\text{DAE}_{\text{hex}}$ no weak keys were found.

11.4 Cyclic multiplication

Multiplication of two integers A and B, denoted by $C = A \cdot B$, achieves very high average diffusion at the bit level. If however $A = 0$, C is independent of B. We have exploited this kind of worst-case behavior, both in the generation of collisions for FFT-Hash and in the determination of classes of weak keys of IDEA. Moreover, the bit-level propagation and correlation properties of the multiplication of two integers are very complicated. The application of modular reduction to reduce C to the size of A or B makes the analysis even harder.

In Chapter 6 it has been shown that multiplication by a constant modulo $2^n - 1$, denoted by the term cyclic multiplication, is a shift-invariant transformation. Cyclic multiplication has the advantage that it is fully specified by the word length n and a single n-bit number, making it an attractive building block for ciphers. In the following sections we describe the difference propagation and implementation properties of cyclic multiplication. Most of the material in this section has already been published in our paper [14].

The multiplication constant is denoted by G and the resulting mapping by γ. The n-bit words are denoted by small characters $a, b, \ldots$ The bits of a word a are denoted by a_i. The numbers represented by these words are denoted by capital characters $A, B, \ldots$, hence, $A = \sum_i a_i 2^i$. Multiplication modulo $2^n - 1$ is denoted by $\times$. For $a \neq \bar{0}$ we have

$$b = \gamma(a) \Leftrightarrow B = G \cdot A \bmod (2^n - 1) = G \times A \ , \tag{11.10}$$

and $\gamma(\bar{0}) = \bar{0}$.

11.4.1 Difference propagation properties

We have studied the difference propagation properties of cyclic multiplication. Our most important results are a number of properties that can be used to find the critical prop ratio of a given multiplication factor in an efficient way.

From $\bar{a} \oplus a = \bar{0}$ it can be derived that $\bar{A} \equiv -A \pmod{2^n - 1}$. Therefore,

$$\gamma(a \oplus \bar{0}) = \gamma(-A) = -\gamma(A) = \gamma(a) \oplus \bar{0} \ . \tag{11.11}$$

Hence, we have $\mathrm{R_p}(\bar{0} \dashv \gamma \vdash \bar{0}) = 1$. Together with $(0 \dashv \gamma \vdash 0)$, this difference propagation is called *trivial*.

Suppose we have two words a, a^* and their γ-images $b = \gamma(a)$ and $b^* = \gamma(a^*)$. Since modular multiplication is distributive with respect to modular addition, the modular difference $B' = B - B^* \bmod 2^n - 1$ is given by

$$B' = G \times A' \ . \tag{11.12}$$

Modular multiplication is however not distributive with respect to bitwise addition. The bitwise difference $b' = b \oplus b^*$ is not determined by $a' = a \oplus a^*$ but depends on the specific value of a (or equivalently a^*).

There is a relation between the bitwise difference of two words and the modular difference of the corresponding numbers. Let $a' = a \oplus a^*$ and $A' = A - A^* \bmod 2^n - 1$. We have

$$A' = \sum_i (a_i - a_i^*) 2^i \ . \tag{11.13}$$

Let $d_i = a_i - a_i^*$. Since $d_i \in \{-1, 0, 1\}$, the word d can be seen as a (redundant) *ternary* representation of the number A'. The bits of the bitwise difference a' are given by $a'_i = a_i \oplus a_i^*$. Clearly, $a'_i = 0 \Leftrightarrow d_i = 0$ and $a'_i = 1 \Leftrightarrow d_i = \pm 1$. If the Hamming weight of a ternary word is defined as the number of nonzero symbols, we have $\mathrm{w_h}(d) = \mathrm{w_h}(a')$.

Definition 11.1 *A word x is said to be* diff-compatible *with a number U, denoted by $x \sim U$, if there exists a couple a, a^* with $x = a \oplus a^*$ and $U = (A - A^*) \bmod 2^n - 1$.*

It can be seen that a word x and a number U are diff-compatible if there exists a ternary word d with components in $\{-1, 0, 1\}$ such that

$$U = \sum_i d_i x_i 2^i \ . \tag{11.14}$$

Clearly, $(x \sim U) \Rightarrow (x \sim -U)$.

Definition 11.2 *$\mathcal{N}(x, U)$ is the set of couples with the bitwise difference x and the modular difference U and $\mathrm{N}(x, U) = \#\mathcal{N}(x, U)$. $\mathcal{N}(-, U)$ and $\mathcal{N}(x, -)$ are given by*

$$\mathcal{N}(-, U) = \bigcup_{x, x \sim U} \mathcal{N}(x, U) \ , \tag{11.15}$$
$$\mathcal{N}(x, -) = \bigcup_{U, x \sim U} \mathcal{N}(x, U) \ . \tag{11.16}$$

$\gamma(\mathcal{N}(x, U))$ is the set of couples $(\gamma(a), \gamma(a^*))$ with $(a, a^*) \in \mathcal{N}(x, U)$.

For any $x \in \mathbb{Z}_2^n$, we have $\mathrm{N}(x, -) = 2^n$. By taking into account that both 0 and $\bar{0}$ represent the number 0, we obtain $\mathrm{N}(-, U) = 2^n + 1$ for all $U \neq 0$ and $\mathrm{N}(-, 0) = 2^n + 2$.

Proposition 11.1 *The combination of x and U fixes a ternary word d with components $d_i = 0$ for $x_i = 0$ and nonzero components fixed by (11.14).*

Proof: If in (11.14) the nonzero d_i are specified by bits e_i with $d_i = 2e_i - 1$, we have

$$U = \sum_i (2 * e_i - 1) x_i 2^i = \sum_i e_i x_i 2^{(i+1) \bmod n} - \sum_i x_i 2^i \ . \tag{11.17}$$

This equation fixes the values of all occurring e_i and therefore of all d_i that occur in (11.14). Hence, x and U uniquely determine the ternary word d.

□

Proposition 11.2 *If $x \sim U$ and $U \neq 0$, we have*

$$\mathrm{N}(x, U) = 2^{n-\mathrm{w_h}(x)} \ . \tag{11.18}$$

Proof: If $(a, a^*) \in \mathcal{N}(x, U)$, the bits of a and a^* must obey

$$a_i - a_i^* = d_i \ , \tag{11.19}$$

with d the unique ternary word corresponding to x and U. (11.19) fixes all bits of a and a^* with $d_i \neq 0$ and determines that $a_i = a_i^*$ if $d_i = 0$. This gives us $n - \mathrm{w_h}(d)$ degrees of freedom in the choice of a, and therefore $\mathrm{N}(x, U) = 2^{n-\mathrm{w_h}(x)}$ different couples (a, a^*). □

Anatomy of the difference propagations

Given a word length n, determining the multiplication factor G with the lowest critical prop ratio by exhaustive search would take approximately 2^{3n} cyclic multiplications on n-bit words. Using the shift-invariance and the complementation property it can be derived that

$$\begin{aligned} \mathrm{R_p}(x' \dashv \gamma \vdash y') &= \mathrm{R_p}(\tau_i(x') \dashv \gamma \vdash \tau_i(y')) \\ &= \mathrm{R_p}(x' \oplus \bar{0} \dashv \gamma \vdash y' \oplus \bar{0}) \ . \end{aligned} \tag{11.20}$$

Using these relations, the work factor of the exhaustive search can be reduced to approximately $2^{3(n-1)}/n^3$ cyclic multiplications. This is still only feasible for relatively small values of n (say ≤ 16). In this section we demonstrate some structural difference propagation properties that enable us to find the critical difference propagations in a much more efficient way.

For a difference propagation $(a' \dashv \gamma \vdash b')$ with $a' = \delta_0$ (see p. 11), we have

$$a' = \delta_0 \Longrightarrow A' = \pm 1 \Longrightarrow B' = \pm G \ . \tag{11.21}$$

It follows that only b' can occur with $b' \sim G$. From (11.15) and Prop. 11.2 it can be seen that for all b' with $b' \sim G$,

$$\mathrm{N}(b', G)/\mathrm{N}(-, G) = \mathrm{N}(b', -G)/\mathrm{N}(-, -G) \approx 2^{-\mathrm{w_h}(b')} \ . \tag{11.22}$$

Proposition 11.3 *For a given factor G, the prop ratios $\mathrm{R_p}(\delta_0 \dashv \gamma \vdash b')$ with $b' \sim G$ can be approximated by*

$$\mathrm{R_p}(\delta_0 \dashv \gamma \vdash b') \approx 2^{-\mathrm{w_h}(b')} \ . \tag{11.23}$$

Proof: Since b' must be diff-compatible with G, $\gamma(\mathcal{N}(\delta_0, -))$ is a subset of $\mathcal{N}(-, G) \cup \mathcal{N}(-, -G)$. We have

$$\#(\mathcal{N}(b', G) \cup \mathcal{N}(b', -G))/\#(\mathcal{N}(-, G) \cup \mathcal{N}(-, -G)) = 2^{-\mathrm{w_h}(b')} \ .$$

Hence, if $\gamma(\mathcal{N}(\delta_0, -))$ is a representative subset of $\mathcal{N}(-, G) \cup \mathcal{N}(-, -G)$, this is the value of the prop ratio $\mathrm{R_p}(\delta_0 \dashv \gamma \vdash b')$. □

Experiments confirm that this is generally a good approximation for the bitwise differences b' with small Hamming weight. Large deviations only occur for *simple* factors G that have high critical prop ratios anyway, such as 3 or 5.

If the input difference a' has a Hamming weight larger than 1, the situation is more complicated:

1. For a given difference pattern a', there are $2^{\mathrm{w_p}(a')}$ different values A' that are diff-compatible with a'.

2. For each of these A' values, there are exactly $2^{n-\mathrm{w_h}(a')}$ couples in $\mathcal{N}(a', -)$ with modular difference A'. A modular difference A' gives rise to a difference $G \times A'$.

3. If $\gamma(\mathcal{N}(a', A'))$ is a representative subset of $\mathcal{N}(-, G \times A')$, a relative portion $\mathrm{N}(b', G \times A')/\mathrm{N}(-, G \times A') \approx 2^{-\mathrm{w_h}(b')}$ of $\mathcal{N}(a', A')$ will give rise to a bitwise difference b' for $b' \sim (G \times A')$.

4. Hence, the number of couples in $\mathcal{N}(a', A')$ that give rise to a bitwise output difference b' with $b' \sim G \times A'$ is approximately equal to $2^{n-(\mathrm{w_h}(a')+\mathrm{w_h}(b'))}$.

5. The total number of couples (a, a^*) with input bitwise difference a' and output bitwise difference b' can be approximated by

$$\sum_{A'|a'\sim A' \text{ and } b'\sim G\times A} 2^{n-(\mathrm{w_h}(a')+\mathrm{w_h}(b'))} \ . \tag{11.24}$$

If $(\sum_{A'|a'\sim A' \text{ and } b'\sim G\times A} 1)$ is denoted by $\mathrm{N_p}(a', b')$, we have

$$\mathrm{R_p}(a' \dashv \gamma \vdash b') \approx \mathrm{N_p}(a', b') 2^{-(\mathrm{w_h}(a')+\mathrm{w_h}(b'))} \ . \tag{11.25}$$

It can be seen that a modular input difference A' contributes to a difference propagation $(a' \dashv \gamma \vdash b')$ if $a' \sim A'$ and $b' \sim G \times A$. The number of couples it contributes is approximately $2^{n-(\mathrm{w_h}(a')+\mathrm{w_h}(b'))}$.

Finding the critical difference propagations

The difference propagations $(\delta_0 \dashv \gamma \vdash b')$ with the highest prop ratios correspond to the b' words with the smallest Hamming weight and $b' \sim G$. To every combination (b', G) corresponds a word d that is a ternary representation of G. We define

Definition 11.3 *The minimum ternary representation of a number A is a ternary representation with no neighboring nonzero symbols. Its Hamming weight is called minimum ternary Hamming weight and is denoted by $\mathrm{w_m}(A)$.*

The minimum ternary representation and weight of a number can be calculated very efficiently. It can be proved that the minimum ternary representation is unique and has minimum Hamming weight. Moreover $\mathrm{w_m}(A) \leq \frac{n}{2}$.

n	5–6	7–10	11–14	15–17	18–22	23–27	28–31	32–?
	2	3	4	5	6	7	8	9

Table 11.6: Maximum critical restriction weights.

It can be seen that the highest prop ratio for a difference propagation with $a' = \delta_i$ is approximately $2^{-\mathrm{w_m}(G)}$. Alternatively, the highest prop ratio for a difference propagation with $b' = \delta_i$ is $2^{-\mathrm{w_m}(G^{-1})}$. This gives a lower bound for the critical prop ratio for a given factor G of

$$2^{-\min(\mathrm{w_m}(G),\mathrm{w_m}(G^{-1}))} .$$

In general, this imposes a lower bound of $2^{-\lfloor \frac{n}{2} \rfloor}$ for the critical prop ratio for cyclic multiplication with word length n.

In experiments for $n \leq 16$ we did not encounter cases in which the factor $\mathrm{N_p}$ in (11.25) for the critical difference propagations differs from 2. This implies that for the propagation $(a' \dashv \gamma \vdash b')$, there is only a single pair of numbers $A', -A'$ such that $a' \sim A'$ and $b' \sim G \times A'$. These critical propagations can be found by determining the number A' that minimizes $\mathrm{w_m}(A') + \mathrm{w_m}(G \times A')$. Therefore, the critical prop ratio is approximated by

$$2^{1-\min_A(\mathrm{w_m}(A)+\mathrm{w_m}(G\times A))} . \tag{11.26}$$

If $\mathrm{N_p}$ in (11.25) is larger than 2, this approximation is an underestimation. It is however very unlikely that this is the case.

Suppose the difference propagation $(a' \dashv \gamma \vdash b')$ corresponding to a number A' is the "candidate" critical propagation. There is a set α of $2^{\mathrm{w_h}(a')}$ numbers with elements X for which $a' \sim X$ holds and a set β of $2^{\mathrm{w_h}(b')}$ numbers with elements Y for which $b' \sim Y$ holds. If the elements of α and β are uniformly selected from $\mathbb{Z}_{2^n-1}$, the probability that there exists a couple (X, Y) with $Y = G \times X$ only becomes significant if $\mathrm{w_h}(a') + \mathrm{w_h}(b') > n$. Since

$$\mathrm{w_h}(a') + \mathrm{w_h}(b') < \left\lfloor \frac{n}{2} \right\rfloor ,$$

this probability is very small. However, the sets α and β are not uniformly selected. The difference propagation $(a' \dashv \gamma \vdash b')$ that determines α and β was selected given the number A'. It follows that necessarily $(-)A' \in \alpha$ and $(-)G \times A \in \beta$. There is however no reason that the argument should not apply for all other numbers in α and β.

The validity of (11.26) has been verified by comparing the predicted value of the critical prop ratio with the exact value for a large number of factors for $n \leq 16$.

The critical restriction weight is minus the binary logarithm of the critical prop ratio. Table 11.6 illustrates the power of cyclic multiplication with respect to difference propagation by listing the highest occurring critical restriction weights for word lengths below 33.

Correlation properties

A similar analysis may be possible with respect to the correlation properties of cyclic multiplication. After the publication of linear cryptanalysis in [71] we did some simple correlation experiments. We found that there definitely is a relation between the selection vectors corresponding to large correlations and the ternary representations of G. The relations seemed however to be much more complicated than in the case of difference propagation. Having more urgent priorities, we decided to abandon the subject.

Recently we did a search for the "best" multiplication factor for a word length 16. This multiplication factor combines a low (nearly minimum) critical prop ratio with a low (nearly minimum) critical correlation. It is given by 635_{hex}, has a critical prop ratio of exactly 521/16384 ($\approx 2^{-5}$) and a critical correlation of exactly 869/4096 ($\approx 2^{-2}$).

11.4.2 Implementation aspects

The portability of cyclic multiplication, as of all other arithmetic operations, is not excellent. In this section we show however that its symmetry properties enable reasonably efficient implementations on a wide range of platforms.

Cyclic multiplication with a word length of n can be executed by one ordinary multiplication with $2n$-bit result, one ordinary addition, a shift operation and a possible incrementation by 1. This follows from the property

$$G \cdot A \bmod 2^n - 1 \equiv G \cdot A \bmod 2^n + \left\lfloor \frac{G \cdot A}{2^n} \right\rfloor \ . \tag{11.27}$$

The first right-hand term of this equation is obtained by taking the n least significant bits of $G \cdot A$, the second term by taking the remaining bits of $G \cdot A$ and shifting them n positions to the left. The modular reduction can be performed by deleting the MSB (at position n) and adding it to the intermediate result. In this way it is guaranteed that γ maps $\bar{0}$ to $\bar{0}$. We have

$$G \cdot (2^n - 1) = 2^n \cdot G - G = 2^n \cdot G + \bar{G} \ .$$

The substitution of the right-hand side expression in (11.27) results in $\bar{0}$. The bit in position n is 0. Hence, the reduction does not convert $\bar{0}$ into 0.

The cyclic multiplication can be implemented using look-up tables. This can be useful in dedicated hardware implementations or to speed up software implementations on processors where multiplication is not available or too slow. Let n and e be powers of 2. The n-bit multiplication can be performed by e table look-ups, $e-1$ n-bit additions and a single reduction modulo $2^n - 1$. This takes e tables of size $2^{n/e}$ containing the values of $G \times A$ for $A \in \{a \cdot 2^{kn/e} \mid 0 \leq a < 2^{n/e}\}$. Because of the shift-invariance, table k simply contains the elements of table 0 shifted cyclically to the right over $k(n/e)$ positions. Hence, at the expense of $e-1$ additional cyclic shifts, the total table size can be reduced to $n2^{n/e}$ bits instead of $ne2^{n/e}$.

We illustrate this with some numbers for a cyclic multiplication with a word length n of 32. With a table size of 512 Kbyte, the work factor is two table lookups, one addition and one reduction modulo $2^{32} - 1$. If the table size is only 256 Kbyte, the work factor is augmented by a single cyclic shift over 16-bits. If $e = 4$, the total table size is 4 Kbyte and the number of additions is 3. This table size can be reduced to 1 Kbyte at the cost of 3 additional cyclic shifts per multiplication. In the extreme case that $e = 32$, the work factor is on the average 16 additions and a single reduction modulo $2^{32} - 1$ (and possibly 31 cyclic shifts over a single position).

11.5 The block cipher MMB

MMB (Modular Multiplication based Block cipher) is a block cipher design that makes use of cyclic multiplication as its most important component. We first published MMB in our paper [18] as an alternative for IDEA.

The basic operations of MMB are on 32-bit words since this is the word length in most modern processors. It is however easy to see that variants can be constructed for any word length.

11.5.1 Specification

MMB is a block cipher with both block and key length 128 bits. For the specification the 128-bit encryption state a is split into four 32-bit words (a_0, a_1, a_2, a_3). The cipher key κ consists of $(\kappa_0, \kappa_1, \kappa_2, \kappa_3)$.

The round transformation $\rho[\kappa^j]$ is composed of four transformations:

$$\rho[\kappa^j] = \theta \circ \varsigma \circ \gamma_4 \circ \sigma[\kappa^j] \ . \tag{11.28}$$

The nonlinear transformation γ_4 consists of cyclic multiplication of the four 32-bit words respectively by factors G_0, G_1, G_2 and G_3. We have

$$\begin{aligned} G_0 &= 025\text{F1CDB}_{\text{hex}} \ , \\ G_1 &= 2 \times G_0 \ , \\ G_2 &= 2^3 \times G_0 \ , \\ G_3 &= 2^7 \times G_0 \ . \end{aligned}$$

The linear transformation θ is a shift-invariant transformation defined by

$$\theta(a)|_i = a_{i-1} \oplus a_i \oplus a_{i+1} \ , \tag{11.29}$$

with all indices modulo 4. The round key application $\sigma[\kappa^j]$ is specified by

$$\sigma[\kappa^j](a)|_i = a_i \oplus \kappa_i^j \ , \tag{11.30}$$

with the round keys defined by the key schedule. Finally, we specify the asymmetrical transformation ς:

$$\begin{aligned} \varsigma(a)|_0 &= a_0 \oplus \Delta \text{ if } \mathrm{LSB}(a_0) = 1 \text{ and } a_0 \text{ otherwise} \ , \\ \varsigma(a)|_1 &= a_1 \ , \\ \varsigma(a)|_2 &= a_2 \ , \\ \varsigma(a)|_3 &= a_3 \oplus \Delta \text{ if } \mathrm{LSB}(a_3) = 0 \text{ and } a_3 \text{ otherwise} \ , \end{aligned} \tag{11.31}$$

with

$$\Delta = 2\text{AAAAAA}_{\text{hex}} \ . \tag{11.32}$$

Encryption consists of 6 applications of the round transformation $\rho[\kappa^j]$, followed by an application of $\sigma[\kappa^6]$:

$$\text{MMB}[\kappa] = \sigma[\kappa^6] \circ \rho[\kappa^5] \circ \rho[\kappa^4] \circ \rho[\kappa^3] \circ \rho[\kappa^2] \circ \rho[\kappa^1] \circ \rho[\kappa^0] \ , \tag{11.33}$$

with the round keys given by

$$\kappa_i^j = \kappa_{i+j \bmod 4} \ . \tag{11.34}$$

MMB decryption

MMB does not have a self-reciprocal structure. It is therefore not trivial to build a simple cryptographic finite state machine that can handle both block encryption and block decryption. In software implementations the lack of self-reciprocity poses no problems. The round transformation ρ' corresponding to decryption is given by

$$\rho'[\kappa^j] = \gamma_4^{-1} \circ \varsigma^{-1} \circ \theta^{-1} \circ \sigma[\kappa^j] = \gamma_4^{-1} \circ \varsigma \circ \theta \circ \sigma[\kappa^j] \ . \tag{11.35}$$

The transformation γ_4^{-1} consists of cyclic multiplication by the multiplicative inverses of the G_i. We have

$$G_0^{-1} = 0\text{DAD4694}_{\text{hex}} \ . \tag{11.36}$$

11.5.2 Discussion

The selected G_i have a critical prop ratio of approximately 2^{-9}, the minimum for word lengths of 32. The branch number $\mathcal{B}$ of θ is 4. From this it can be seen that any two-round differential trail affects at least four 32-bit cyclic multiplications. This gives a lower limit of 18 for the restriction weight per round for differential trails with even length. The actual minimum restriction weight is expected to be significantly higher because of the additional alignment conditions in the differential steps. Clearly MMB is not the result of the wide trail design strategy. In MMB we have a narrow trail consisting of very wide S-boxes.

The step ς has been inserted to prevent the existence of differential trails consisting only of trivial difference propagations. We have $\text{R}_\text{p}(\bar{0} \dashv \gamma \vdash \bar{0}) = 1$. If ς is removed from the round transformation, there are 15 differential steps with prop ratio 1 composed of blocks 0 and $\bar{0}$. An example:

$$(\bar{0}, 0, 0, \bar{0}) \dashv \rho \vdash (0, \bar{0}, \bar{0}, 0) \ . \tag{11.37}$$

These could be chained to form 6-round differential trails with prop ratio 1. The transformation ς converts the differences $\bar{0}$ into $\bar{0} \oplus \Delta$ thereby preventing the chaining of the differential steps with a prop ratio of 1.

11.5.3 Problems of MMB and potential solutions

The most important problem of MMB is the fact that the correlation properties of cyclic multiplication are not well understood. We suspect that the correlation properties of cyclic multiplication by the selected multiplication factors is more than sufficient to provide adequate protection against linear cryptanalysis. Still, the multiplication factors have carefully been selected only with DC in mind, while LC is in practice equally, if not even more, important. The solution to this problem is developing the tools to efficiently investigate the correlation properties of cyclic multiplication, as we did for difference propagation. Using these new tools in combination with ours, new multiplication factors can be selected.

The different rounds of MMB only differ in the fact that the cipher key is rotated. This can give rise to exploitable symmetry properties. Eli Biham told us that this property could be exploited in a chosen-key attack of the type described in [7] to efficiently calculate the actual cipher key. This attack can easily be prevented by applying round key biasing, i.e., adding constants to the round keys to remove the symmetry. For example,

$$\kappa_0^j = \kappa_{j \bmod 4} \oplus 2^j \mathrm{B} \ , \tag{11.38}$$

with $\mathrm{B} = 0\mathrm{DAE}_{\mathrm{hex}}$.

11.6 Conclusions

It has been shown that the injudicious use of modular arithmetic in cryptographic schemes can lead to serious weaknesses. In MMB we have avoided these problems by concentrating on the worst-case behavior in the design phase.

As an interesting subject for further research, we propose the investigation of the relation between the correlation properties and the multiplication factor in cyclic multiplication.

Chapter 12

Conclusions

In this chapter we summarize our most important conclusions. They are divided into three distinct parts. We start with contrasting our design approach with some widespread cryptographic design and research practices. Subsequently, we run over what we consider our most important contributions to design. We conclude with a short enumeration of some research subjects that naturally follow from our design approach.

12.1 Our design approach

In many publications it is considered an ultimate goal to have a provably secure cryptographic scheme, with the actual cryptographic design going on just a temporary way of dealing with the situation. Information theory provides us with sharp upper bounds for the achievable security. The only provably secure scheme, the one-time pad, reduces the problem of protecting the secrecy of a message to that of protecting the secrecy of a key with equal length and is therefore useless in most applications.

Much has been expected from the application of computational complexity theory in cryptographic design. It would bring us an essential step closer to the realization of provable security. The example of the Even-Mansour block cipher construction in Chapter 10 is a good illustration of the severe limitations of this theory in the field of cryptographic design.

A difficulty inherent in provable security is that of formulating the security proof. This must necessarily be succinct and clear, otherwise it cannot serve to persuade another individual of the provable security. Apparently, actual historical proofs or arguments of security only say something about the security with respect to *loyal cryptanalysis*. This is cryptanalysis by adversaries (often the designers themselves) who respect the implicit or explicit assumptions or access restrictions that seem to be inherent in the formulation of a proof. In practice however, cryptographic schemes are typically broken through the application of *hostile cryptanalysis* by "sly skeptics that maliciously exploit seemingly innocent design flaws".

In our opinion provable security is nothing more than a *phantom*, similar to the perpetuum mobile in thermodynamics. We propose to abandon the quest for provable security and recognize that security is related to the concepts of trust and commitment. A cryptographic design is accompanied by an explicit or implicit claim of security that has two distinct functions. With respect to potential cryptanalysts, this claim serves as a well-defined challenge that instigates the *public evaluation* of the design. In the absence of refutation, the claim serves as the specification of the offered security for users or system designers. The trust of the users in the cryptographic scheme is based on their belief in the validity of the claim. Directly, this belief can be built up by the observation that, despite large efforts, no weaknesses have been found that refute the claim. Indirectly, this belief can be based on the commitment of a trusted organization or individual to the claim. The terms K-secure and hermetic have been introduced to be used in these claims. Both correspond to very strict definitions of cryptographic security, and express the security relative to the majority of schemes with the same dimensions.

A very powerful strategy for problem solving in science, engineering or any human activity is that of trying to reduce problems that seem too complex or too large to solve directly to an equivalent set of smaller, simpler problems. In cryptographic design this strategy has often been applied in an injudicious way. We have denoted this by the term *N-reductionism.* As can be read in Chapters 4, 9 and 10, this approach always seems to impose serious restrictions on the design, while in most cases it only reduces one hard design problem to another. The benefits of N-reductionistic design principles are only apparent within the narrow framework in which they have been put forward. If all aspects of the design are taken into consideration, these principles turn out to be meaningless pseudo-goals that seriously hamper the design of efficient schemes. In our design approach we have tried to eliminate these pseudo-goals as much as possible. This is especially the case in our design strategy for cryptographic hash functions as explained in Chapter 4 and that for self-synchronizing stream ciphers as explained in Chapter 9. Additionally, we have shown in Chapters 3 and 9 that with our definitions of security, N-reductionistic constructions are discouraged.

In cryptographic research, there is a tendency to study and describe certain properties of Boolean functions and S-boxes, or to give constructions for Boolean functions or S-boxes that satisfy certain criteria. This has resulted in a vast theoretical body of work that can provide components for cryptographic designs. In practice however, there seems to be a missing link between this theory and the components that are actually used in real cryptographic designs. In our opinion this is not surprising, since many of the properties and criteria that have been defined have no longer a direct connection with existing attacks. In other cases, the implications of actual attacks are converted to criteria in a short-sighted and simplistic way. This practice has led to nonsensical statements such as "we have constructed an S-box that is optimally secure against both DC and LC" and "large S-boxes can offer better resistance against LC and DC than small S-boxes".

In our design approach, *we start from a global structure that is simple to describe and easy to implement.* For block ciphers and stream/hash modules this is the *cryptographic finite state machine.* Within this framework, the design of the actual functions and transformations is governed by *symmetry considerations* and the two very powerful types of cryptanalysis of *LC and DC*. In this context we have developed some tools for the description and analysis of correlation and difference propagation properties essential in LC and DC. After a thorough discussion of both LC and DC, we have explicitly formulated our specific *wide trail* design strategy. It is this design strategy that gives rise to our criteria for linear and nonlinear transformations. In the choice of the component transformations, the concern for simplicity, symmetry and parallelism has led us to shift-invariant transformations. In our actual designs, these transformations are combined and arranged in such a way that the specific schemes appear to be resistant against known attacks.

In several recent cryptographic designs, cryptanalysis is supposed to be impossible because the propagation and correlation properties of the component transformations depend strongly on the key or intermediate state bits. This is also the case for most designs based on modular arithmetic. In these designs, analysis exploiting the properties of the arithmetic operations is thwarted by combining several operations that are non-distributive. The problem with this approach is that the *quality* of the propagation and correlation properties can depend strongly on the specific key or intermediate state. In Chapter 11 we have shown that this can lead to serious weaknesses.

In our design approach, the resistance against cryptanalysis is based on transparency rather than on obscurity. The components that are chosen have very simple propagation and correlation properties. Cipher keys are always applied by means of bitwise addition and the propagation and correlation properties of the round transformations are independent of the key. This greatly simplifies the analysis of the multiple-round propagation and correlation properties in the design phase. Still, we have to make the unproved and in our opinion unprovable assumption that within our design approach cryptographic security can be realized by ensuring resistance against LC and DC and eliminating all exploitable symmetry. Our belief in the validity of this assumption is based on the fact that (bitwise) correlation and difference propagation appear to be fundamental properties of Boolean mappings. In our opinion, all attacks on schemes designed by our approach must involve at least one these two aspects.

In block cipher design we have replaced the classical Feistel structure by a new self-reciprocal structure. For stream ciphers we have stressed the importance of the initialization mapping in real-world applications and shown that the presence of a simple initialization mapping has a destructive impact on the security of most linear feedback shift register based schemes. Together with the rejection of the Damgård-Merkle design principle for cryptographic hash functions, this has led us to the conception of a cryptographic building block that is completely new. In self-synchronizing stream cipher design, we have pointed out the misconceptions in the scarce publications on the subject and given an alternative strategy illustrated by a specific design.

12.2 Our specific contributions to design

In Chapter 2 we have introduced the distinction between ciphers and encryption schemes. This distinction enables us to give a consistent categorization of the different ciphers and encryption schemes with respect to external behavior and internal structure.

In Chapter 3 we have pointed out the importance of considering non-uniform key selection and the role of the cryptographic claim. We have introduced the concepts K-secure and hermetic to be used in these claims. An overview has been given of the limits to manipulation detection by the encryption of redundant data for three different types of redundancy.

In Chapter 4 we have introduced the concept of the cryptographic finite state machine, useful in the design of block ciphers, stream ciphers and cryptographic hash functions.

In Chapter 5 we have given a new formalism for the description of difference propagation and correlation properties of Boolean mappings and iterated transformations. This includes the introduction of correlation matrices and prop ratios and the derivation of a number of fundamental properties and relations. Motivated by this analysis, we have formulated the wide trail design strategy.

In Chapter 6 we have given new methods of finding locally and globally invertible nonlinear binary shift-invariant transformations. We have introduced the Hamming weight distribution table and the branch number of a linear mapping. We have treated in detail the difference propagation and correlation properties of the specific shift-invariant transformation denoted by χ.

In Chapter 7 we have introduced a new self-reciprocal cipher structure and a number of specific transformations to be used in this structure. The resulting block cipher constructions have the remarkable property that differential and linear trails are governed by the same equations. This has led to two concrete proposals that are very well suited for hardware implementations and have high portability: 3-Way and BaseKing.

In Chapter 8 we have introduced the stream/hash module as a new type of cryptographic building block. This is basically a cryptographic finite state machine that is designed for both efficient hashing and efficient cryptographic sequence generation. We have presented the hardware-oriented design Subterranean that has been implemented as a chip in 2.4μ cmos standard cell technology in collaboration with IMEC. These Subterranean coprocessors attain encryption rates of over 280 Mbit/s and hashing rates of over 560 Mbit/s. Finally, we have presented the software-oriented design StepRightUp that has a work factor of only 3.7 bitwise Boolean operations and 0.5 cyclic rotations on 32-bit words per encrypted or hashed byte.

In Chapter 9 we have given a construction for self-synchronizing stream ciphers consisting of pipelined stages in combination with a new structure called a conditional complementing shift register. This is illustrated by a concrete hardware-oriented design example.

In Chapter 10 we have introduced the idea of a resynchronization attack. The power of this type of attack has been illustrated by four specific examples, ranging from a general attack with theoretical implications to the description of a system for on-line unauthorized descrambling of a specific commercial Pay-TV system.

In Chapter 11 we have given an analysis of the difference propagation properties of cyclic multiplication. This operation is the most important component in our block cipher proposal MMB.

12.3 Further research

We conclude with giving some directions for useful research in the philosophy of the presented design approach:

- attacking the specific designs described in this thesis,
- design and analysis of portable stream/hash modules, including the analysis of the StepRightUp design,
- improving the efficiency of the algorithms to find the critical propagation chains in block ciphers such as 3-Way and BaseKing,
- investigating the correlation properties of cyclic multiplication,
- design of n-bit self-synchronizing stream ciphers with $n > 1$,
- extending and applying the theory of correlation matrices.

Appendix A

On Shift-Invariant Transformations

In this appendix we treat some aspects of shift-invariant transformations that have only affected our cryptographic design in an indirect way, but are nevertheless interesting. We start with a short treatment of the cardinalities of the congruence classes of the state-space. This is followed by a proof of invertibility for a class of nonlinear shift-invariant transformations. Finally, we give an overview of the simplest nonlinear shift-invariant transformations and their diffusion and nonlinearity properties.

A.1 Structure of the state-space

The numbers of elements in $\mathcal{T}_n$, the quotient space of period n, and $\mathcal{Z}_n$, the quotient space corresponding to $\mathcal{A}_n$, can easily be calculated using the Euler totient function $\varphi(n)$ and the Möbius function $\mu(n)$ [47]. Let $\mathrm{T}(n) = \#\mathcal{T}_n$ and $\mathrm{Z}(n) = \#\mathcal{Z}_n$.

Every integer can be factored into a unique product of powers of distinct primes, and the Möbius function and the Euler totient function can be defined in terms of this factorization. If

$$n = \prod_{i=1}^{\ell} p_i^{\alpha_i \cdot} \text{ with all } p_i \text{ prime,} \tag{A.1}$$

we have

$$\mu(n) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } \prod_{i=1}^{\ell} \alpha_i > 1 \\ (-1)^{\ell} & \text{otherwise.} \end{cases} \tag{A.2}$$

In words, the Möbius function is equal to 1 if n is the product of an even number of distinct primes, to -1 if n is the product of an odd number of distinct primes, and to 0 if the factorization of n contains prime powers. For $\varphi(n)$ we have

$$\varphi(n) = \begin{cases} 1 & \text{if } n = 1 \\ \prod_{i=1}^{\ell} p_i^{\alpha_i - 1}(p_i - 1) & \text{if } n > 1 \ . \end{cases} \tag{A.3}$$

The Euler totient function equals the number of integers j smaller than and coprime to n. Using the principle of inclusion and exclusion it can be proved that the number of different quotient states with period n is given by

$$\mathrm{T}(n) = \frac{1}{n}\sum_{d|n} \mu(d) 2^{n/d} \ . \tag{A.4}$$

The number of different quotient states in $\mathcal{Z}_n$ is equal to the number of distinct cycles of the rotating shift register of length n. This has been studied by Solomon Golomb in [45] and is given by

$$\mathrm{Z}(n) = \frac{1}{n}\sum_{d|n} \varphi(d) 2^{n/d} \ . \tag{A.5}$$

Since $\mathcal{Z}(n) = \bigcup_{d|n} \mathcal{T}_d$, these two numbers are linked by the following equations:

$$Z(n) = \sum_{d|n} T(d) \ , \tag{A.6}$$

$$T(n) = Z(n) - \sum_{(d<n)\&(d|n)} Z(d) \ . \tag{A.7}$$

A.2 Proof of invertibility

Proposition A.1 *All ϕ that are specified by one of the 16 subsets of*

$$\{*0001, *0101, *0111, *01\texttt{-}0\}$$

are invertible for odd state lengths.

We prove this proposition by giving a seed and a leap. These two mechanisms can be used to calculate a unique preimage for every state with odd length.
Let $\phi(b) = a$.

Seed: If a_0 is in landscape $*\texttt{-}01$, b_0 and b_2 can be fixed. The only states without a seed are 1^* and 0^*. However, for all considered ϕ we have $\phi(1^*) = 1^*$ and $\phi(0^*) = 0^*$.

Proof : If a_i is in landscape $*1$, we have $b_i = a_i$, since b_i cannot be in a complementing landscape.
If a_0 is in landscape $*101$, $b_0 = a_0$ and $b_2 = 0$.
If a_0 is in landscape $*001$, $b_2 = 0$. If it is assumed that b_0 is in a complementing landscape (CL), this must be $*0001$, the only CL that is compatible with $*\texttt{-}0$. The consequence that $b_3 \neq a_3$ implies that b_3 must be in a CL. This is however impossible since the assumption imposes that $b_4 = 1$.
Hence, we have a_0 in $*\texttt{-}01 \Rightarrow a_0 = b_0 \ \& \ a_2 = 0$. □

Leap : If b_0 and b_2 are known, b_{-2} can be fixed. Since seeds on even (odd) positions only give rise to knowledge of bits on even (odd) positions with this leap, $\xi = \{2\}$.

Proof : There are four possibilities for the landscape of b_0:
`*-1-1` : the relevant landscape of b_0 is completely known,
`*-0-1` : b_3 is known. If equal to 1 : $b_0 = a_0$. Otherwise, b_1 is known, revealing the full landscape,
`*-1-0` : b_1 is known. If equal to 1 : $b_0 = a_0$, the landscape is unambiguously `*01-0`,
`*-0-0` : there is no CL for b_0 compatible with this and therefore, $b_0 = a_0$. □

A.3 Tables of invertible nonlinear ϕ

During our research a large number of new invertible nonlinear binary shift-invariant transformations were found. In tables A.1 and A.2 we list the most important, i.e. simple, instances.

The transformations are specified by their set of complementing landscapes. For every entry, we have specified the invertibility properties by ξ, the average diffusion by the diffusion factor $\mathcal{D}$ and the nonlinearity by $\mathcal{C}$ and $\mathcal{R}$. $\mathcal{C}$ corresponds with the largest correlation that occurs between linear combinations of output bits and of input bits. $\mathcal{R}$ corresponds to the largest occurring prop ratio.

CL	ξ	$\mathcal{D}$	$\mathcal{C}$	$\mathcal{R}$
*01 (χ)	{2}	2	1/2	1/4
0*01	{2}	1.75	3/4	27/64
*001	{3}	1.75	3/4	7/16
*100	{3}	1.75	3/4	7/16
0*-10	{2}	1.75	3/4	27/64
*01-1	{2}	1.75	3/4	27/64
*0-01	{2}	1.75	3/4	27/64
*01-0	{4}	1.75	3/4	27/64
0 $\lhd i \rhd$ *1 $\lhd i \rhd$ 0	$\emptyset$	1.75	3/4	$\geq$ 27/64
*0100	{2}	1.5	3/4	35/64
0*010	{2}	1.5	7/8	75/128
0*001	{3}	1.5	7/8	151/256
0*100	{3}	1.5	7/8	9/16
0*101	{3}	1.5	7/8	9/16
*0111	{4}	1.5	7/8	39/64
*0001	{4}	1.5	7/8	39/64
*0011	{4}	1.5	7/8	37/64
0*01-0	$\emptyset$	1.5	7/8	75/128
0*011	$\emptyset$	1.5	7/8	19/32
0-*-11-0	$\emptyset$	1.5	7/8	301/512
00*10	$\emptyset$	1.5	7/8	75/128
0*1 $\lhd i \rhd$ 10	$\emptyset$	1.5	7/8	$\geq$ 143/256
01*01	$\emptyset$	1.5	7/8	17/32
0 $\lhd i \rhd$ -*1 $\lhd i \rhd$ 10	$\emptyset$	1.5	7/8	$\geq$ 9/16
0-*100	$\emptyset$	1.5	7/8	301/512

Table A.1: Invertible ϕ specified by a single landscape.

CL	ξ	$\mathcal{D}$	$\mathcal{C}$	$\mathcal{R}$
`*001 ∨ 1*10`	$\{2,3\}$	2.25	1/2	15/64
`*01-0 ∨ *0-01`	$\{2\}$	2.25	1/2	9/32
`*01-0 ∨ *0-11`	$\{2\}$	2.25	1/2	7/32
`*01 ∨ 01*0`	$\{2\}$	2.25	5/8	37/128
`*01 ∨ *0-01`	$\{2\}$	2.25	5/8	5/16
`*01 ∨ *0-11`	$\{2\}$	2.25	5/8	19/64
`*01 ∨ 1*001`	$\{2\}$	2.25	5/8	93/256
`*001 ∨ 1*100`	$\{3\}$	2.25	5/8	1/4
`*110 ∨ 0*001`	$\{3\}$	2.25	5/8	31/128
`*01-0 ∨ *0001`	$\{2\}$	2.25	5/8	1/4
`0*-10 ∨ 1*100`	$\{2\}$	2.25	5/8	31/128
`0*01 ∨ *011`	$\{3\}$	2	5/8	41/128
`*010 ∨ 0*01`	$\{2,3\}$	2	5/8	33/128
`*011 ∨ 0*01`	$\{2,3\}$	2	5/8	41/128
`*001 ∨ 1*-01`	$\{2,3\}$	2	5/8	43/128
`*010 ∨ *01-0`	$\{2\}$	2	5/8	17/64
`1*10 ∨ 0-*10`	$\{2\}$	2	5/8	11/32
`1*10 ∨ 1-*10`	$\{2\}$	2	5/8	39/128
`*011 ∨ *01-0`	$\{4\}$	2	5/8	3/16
`*100 ∨ 0*101`	$\{5\}$	2	5/8	19/64
`0*10 ∨ 0--*10`	$\emptyset$	2	5/8	305/1028
`*01-1 ∨ *0001`	$\{2\}$	2	5/8	11/32
`0*-10 ∨ 0*1-0`	$\{2,3\}$	2	5/8	73/256
`*0011 ∨ *0101`	$\{2\}$	2	3/4	11/32
`*0111 ∨ *0001`	$\{2\}$	2	3/4	41/128
`0*001 ∨ 0*110`	$\{3\}$	2	3/4	81/256
`0*101 ∨ 1*001`	$\{3\}$	2	3/4	9/32
`1*110 ∨ 0*001`	$\{3\}$	2	3/4	21/64
`0*011 ∨ 1*100`	$\{6\}$	2	3/4	21/64
`0*010 ∨ 1*011`	$\{2,3\}$	2	3/4	7/16
`0*001 ∨ 1*101`	$\{2,3\}$	2	3/4	9/32
`0*110 ∨ 10*10`	$\emptyset$	1.875	3/4	9/32
`10*0 ∨ 10*-1 ∨ 00*10`	$\{2\}$	2.5	1/2	109/512
`*100 ∨ *1-01 ∨ *0100`	$\{3\}$	2.5	1/2	1/4
`*011 ∨ *01-0 ∨ *0001`	$\{6\}$	2.5	1/2	7/32
`0*10 ∨ 0--*10 ∨ 0----*10`	$\emptyset$	2.125	9/16	305/1024

Table A.2: Invertible ϕ specified by multiple landscapes.

Bibliography

[1] C. Adams and S. Tavares, The Structured Design of Cryptographically Good S-Boxes, *Journal of Cryptology*, Vol. 3, No. 1, 1990, pp. 27–42.

[2] T. Baritaud, H. Gilbert and M. Girault, FFT-Hash is not Collision-free, in *Advances in Cryptology, Proc. Eurocrypt '92, LNCS 658*, R.A. Rueppel, Ed., Springer-Verlag, 1992, pp. 35–44.

[3] C. Bennett, F. Bessette, G. Brassard and L. Salvail, Experimental Quantum Cryptography, *Journal of Cryptology*, Vol. 5, No. 1, 1992, pp. 3–28.

[4] M. Bertilsson, E.F. Brickell and I. Ingemarsson, Cryptanalysis of Video Encryption Based on Space-Filling Curves, *Advances in Cryptology, Proc. Eurocrypt '89, LNCS 434*, J.-J. Quisquater and J. Vandewalle, Eds., Springer-Verlag, 1990, pp. 403–411.

[5] E. Biham and A. Shamir, Differential Cryptanalysis of DES-like Cryptosystems, *Journal of Cryptology*, Vol. 4, No. 1, 1991, pp. 3–72.

[6] E. Biham and A. Shamir, *Differential Cryptanalysis of of the Data Encryption Standard*, Springer-Verlag, 1993.

[7] E. Biham, New Types of Cryptanalytic Attacks Using Related Keys, in *Advances in Cryptology, Proc. Eurocrypt '93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 398–409.

[8] E.F. Brickell and A.M. Odlyzko, Cryptanalysis: a survey of recent results, in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, Ed., IEEE Press, 1991, pp. 501–540.

[9] L. Claesen, J. Daemen, M. Genoe, G. Peeters, Subterranean,: a 600 Mbit/sec Cryptographic VLSI Chip, *Proc. ICCD '93: VLSI in Computers and Processors*, R. Camposano, A. Domic, Eds., IEEE Computer Society Press, 1993, pp. 610–613.

[10] J. Daemen, R. Govaerts and J. Vandewalle, Efficient Pseudorandom Sequence Generation by Cellular Automata, in *Proc. 12th Symposium on Information Theory in the Benelux*, F.M.J. Willems and Tj.J. Tjalkens, Eds., Werkgemeenschap voor Informatie- en Communicatietheorie, 1991, pp. 17–24.

[11] J. Daemen, R. Govaerts and J. Vandewalle, A Framework for the Design of One-Way Hash Functions Including Cryptanalysis of Damgård's One-Way Function based on a Cellular Automaton, in *Advances in Cryptology, Proc. of Asiacrypt '91, LNCS 739*, H. Imai, R. Rivest and T. Matsumoto, Eds., Springer-Verlag, 1993, pp. 82–96.

[12] J. Daemen, A. Bosselaers, R. Govaerts and J. Vandewalle, Collisions for Schnorr's Hash Function FFT-Hash, in *Advances in Cryptology, Proc. of Asiacrypt '91, LNCS 739*, H. Imai, R. Rivest and T. Matsumoto, Eds. , Springer-Verlag, 1993, pp. 477–480.

[13] J. Daemen, Limitations of the Even-Mansour Construction, in *Advances in Cryptology, Proc. of Asiacrypt '91, LNCS 739*, H. Imai, R. Rivest and T. Matsumoto, Eds., Springer-Verlag, 1993, pp. 495–498.

[14] J. Daemen, L. Van Linden, R. Govaerts and J. Vandewalle, Propagation Properties of Multiplication Modulo $2^n - 1$, in *Proc. 13th Symposium on Information Theory in the Benelux*, G.H.L.M. Heideman, F.W. Hoeksema and H.E.P. Tattje, Eds. , Werkgemeenschap voor Informatie- en Communicatietheorie, 1992, pp. 111–118.

[15] J. Daemen, R. Govaerts and J. Vandewalle, On the Design of High Speed Self-Synchronizing Stream Ciphers, in *Singapore ICCS/ISITA '92 Conference Proceedings* P.Y. Kam and O. Hirota, Eds., IEEE, 1992, pp. 279–283.

[16] J. Daemen, R. Govaerts and J. Vandewalle, A Hardware Design Model for Cryptographic Algorithms, in *Computer Security – Esorics '92, Proc. 2nd European Symposium on Research in Computer Security, LNCS 648*, Y. Deswarte, G. Eizenberg and J.-J. Quisquater, Eds., Springer-Verlag, 1992, pp. 419–434.

[17] J. Daemen, R. Govaerts and J. Vandewalle, Cryptanalysis of MUX-LFSR Based Scramblers, in *Proc. of the 3rd Symposium on the State and Progress of Research in Cryptography*, W. Wolfowicz, Ed., Fondazione Ugo Bordoni, Roma, 1993, pp. 55–61.

[18] J. Daemen, R. Govaerts and J. Vandewalle, Block Ciphers Based on Modular Arithmetic, in *Proc. of the 3rd Symposium on the State and Progress of Research in Cryptography*, W. Wolfowicz, Ed., Fondazione Ugo Bordoni, Roma, 1993, pp. 80–89.

[19] J. Daemen, R. Govaerts and J. Vandewalle, Resynchronization Weaknesses in Synchronous Stream Ciphers, in *Advances in Cryptology, Proc. Eurocrypt '93, LNCS 765*, T. Helleseth, Ed., Springer–Verlag, 1994, pp. 159–169.

[20] J. Daemen, L. Claesen, M. Genoe, G. Peeters, R. Govaerts and J. Vandewalle, A Cryptographic Chip for ISDN and High Speed Multi-Media Applications, in *Proceedings of VLSI Signal Processing VI*, L.D.J. Eggermont, P. Dewilde, E. Deprettere and J. van Meerbergen, Eds., IEEE, 1993, pp. 12–20.

[21] J. Daemen, R. Govaerts and J. Vandewalle, Weak Keys of IDEA, in *Advances in Cryptology, Proc. Crypto'93, LNCS 773*, D.R. Stinson, Ed., Springer–Verlag, 1994, pp. 224–231.

[22] J. Daemen, R. Govaerts and J. Vandewalle, A New Approach towards Block Cipher Design, in *Fast Software Encryption, LNCS 809*, R. Anderson, Ed., Springer–Verlag, 1994, pp. 18–33.

[23] J. Daemen, R. Govaerts and J. Vandewalle, Invertible Shift-invariant Transformations on Binary Arrays, *Journal of Applied Mathematics and Computation*, to appear.

[24] J. Daemen, R. Govaerts and J. Vandewalle, An Efficient Nonlinear Shift-Invariant Transformation, in *Proceedings of the 15th Symposium on Information Theory in the Benelux*, B. Macq, Ed., Werkgemeenschap voor Informatie- en Communicatietheorie, 1994, pp. 108–115.

[25] J. Daemen, JAM, a Cryptographic Pseudorandom Sequence Generator, *ESAT-COSIC Report 92-1*, Department of Electrical Engineering, Katholieke Universiteit Leuven, March 1992. (2 pp.)

[26] J. Daemen, R. Govaerts and J. Vandewalle, Fast Hashing Both in Hard- and Software, *ESAT-COSIC Report 92-2*, Department of Electrical Engineering, Katholieke Universiteit Leuven, April 1992. (7 pp.)

[27] J. Daemen, R. Govaerts and J. Vandewalle, Cryptanalysis of 2,5 Rounds of IDEA, *ESAT-COSIC Report 94-1*, Department of Electrical Engineering, Katholieke Universiteit Leuven, March 1994. (10 pp.)

[28] I.B. Damgård, The application of claw free functions in cryptography, *PhD Thesis*, Aarhus University, Mathematical Institute, 1988.

[29] I.B. Damgård, Design Principles for Hash Functions, in *Advances in Cryptology, Proc. Crypto'89, LNCS 435*, G. Brassard, Ed., Springer-Verlag, 1990, pp. 416–427.

[30] D.W. Davies, Some Regular Properties of the DES, in *Advances in Cryptology, Proc. Crypto'82*, D. Chaum, R. Rivest and A. Sherman, Eds., Plenum Press, 1983, pp. 89–96.

[31] W. Diffie and M.E. Hellman, New directions in cryptography, *IEEE Trans. on Information Theory*, Vol. IT–22, No. 6, 1976, pp. 644–654.

[32] W. Diffie, The First Ten Years of Public Key Cryptography, in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, Ed., IEEE Press, 1991, pp. 135–175.

[33] Specification of the Systems of the MAC/Packet Family. *EBU Technical Document 3258–E*, Oct 1986.

[34] S. Even, Y. Mansour, A Construction of a Cipher From a Single Pseudorandom Permutation, in *Advances in Cryptology, Proc. Asiacrypt '91, LNCS 739*, H. Imai, R. Rivest and T. Matsumoto, Eds., Springer-Verlag, 1993, pp. 210–224.

[35] H. Feistel, Cryptography and Computer Privacy, *Scientific American*, Vol. 228, No. 5, 1973, pp. 15–23.

[36] H. Feistel, W.A. Notz, and J.L. Smith, Some cryptographic techniques for machine-to-machine data communications, *Proc. IEEE*, Vol. 63, No. 11, 1975, pp. 1545–1554.

[37] *Data Encryption Standard*, Federal Information Processing Standard (FIPS), Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C. , January 1977.

[38] *DES Modes of Operation*, Federal Information Processing Standard (FIPS), Publication 131, National Bureau of Standards, US Department of Commerce, Washington D.C. , December 1980.

[39] FIPS 180, *Secure Hash Standard,* Federal Information Processing Standard (FIPS), Publication 180, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., May 1993.

[40] FIPS 180, *Proposed Revision of FIPS 180, Secure Hash Standard,* Federal Register, July 11, 1994.

[41] P. Flajolet and A.M. Odlyzko, Random Mapping Statistics, in *Advances in Cryptology, Proc. Eurocrypt '89, LNCS 434*, J.-J. Quisquater and J. Vandewalle, Eds., Springer-Verlag, 1990, pp. 450–468.

[42] M.R. Garey and D.S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979.

[43] O. Goldreich, S. Goldwasser and S. Micali, How to Construct Random Functions, in *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, October 24–26, 1984.

[44] D. Gollman and W.G. Chambers, Clock-Controlled Shift-Registers: a Review, *IEEE J. Selected Areas Commun.*, Vol. 7, 1989, pp. 525–533.

[45] S.W. Golomb, *Shift Register Sequences*, Holden–Day Inc., San Francisco, 1967.

[46] J.D. Golić, On the Security of Shift Register Based Keystream Generators, in *Fast Software Encryption, LNCS 809*, R. Anderson, Ed., Springer–Verlag, 1994, pp. 90–100.

[47] G.H. Hardy and E.M. Wright, *An Introduction to the Theory of Numbers (5th edition)*, Oxford University Press, 1979.

[48] M. Hellman, R. Merkle, R. Schroeppel, L. Washington, W. Diffie, S. Pohlig, and P. Schweitzer, Results of an initial attempt to cryptanalyze the NBS Data Encryption Standard, *Information Systems Lab., Dept. of Electrical Eng., Stanford Univ.*, 1976.

[49] M. Hellman, A Cryptanalytic Time-Memory Trade-Off, *IEEE Trans. Informat. Theory*, Vol IT-15, 1980, pp. 401–406.

[50] M. Hellman and S. Langford, Differential-Linear Cryptanalysis, in *Advances in Cryptology, Proc. Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 26–39.

[51] F.J. Hill and G.R. Petersen, *Introduction to Switching Theory and Logical Design*, John Wiley & Sons, 1981.

[52] *Information Technology – Security Techniques – Modes of Operation of an n-bit Block Cipher Algorithm*, IS 10116, ISO/IEC, 1991.

[53] ISO/IEC 9797, *Information technology – Data cryptographic techniques – Data integrity mechanisms using a cryptographic check function employing a block cipher algorithm,* IS 9797, ISO/IEC, 1993.

[54] *Information technology - Security techniques - Hash-functions, Part 1: General and Part 2: Hash-functions using an n-bit block cipher algorithm,"* DIS 10118, ISO/IEC, 1992.

[55] R. Impagliazzo and M. Naor, Efficient cryptographic schemes provably as secure as subset sum, in *Proc. 30th IEEE Symposium on Foundations of Computer Science*, 1989, pp. 236–241.

[56] C.J.A. Jansen and D.E. Boekee, Modes of Blockcipher Algorithms and Their Protection Against Active Eavesdropping, in *Advances in Cryptology, Proc. Eurocrypt '87, LNCS 304*, D. Chaum and W.L. Price, Eds., Springer-Verlag, 1988, pp. 327–347.

[57] C.J.A. Jansen, Investigations on nonlinear streamcipher systems: construction and evaluation methods, *Doctoral Dissertation*, Technische Universiteit Delft, 1989.

[58] D. Kahn, *The Codebreakers. The Story of Secret Writing*, MacMillan, 1967.

[59] J.B. Kam and G.I. Davida, Structured design of substitution-permutation encryption networks, *IEEE Trans. on Computers*, Vol. C–28, 1979, pp. 747–753.

[60] L.R. Knudsen, Iterative Characterisitics of DES and s^2-DES, in *Advances in Cryptology, Proc. Crypto'92, LNCS 740*, E.F. Brickell, Ed., Springer-Verlag, 1993, pp. 497–512.

[61] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, 1987.

[62] X. Lai and J.L. Massey, A Proposal for a New Block Encryption Standard, in *Advances in Cryptology, Proc. Eurocrypt' 90, LNCS 473*, I.B. Damgård, Ed., Springer-Verlag, 1991, pp. 389–404.

[63] X. Lai, J.L. Massey and S. Murphy, Markov Ciphers and Differential Cryptanalysis, in *Advances in Cryptology, Proc. Eurocrypt' 91, LNCS 547*, D. Davies, Ed., Springer-Verlag, 1991, pp. 17–38.

[64] X. Lai and J.L. Massey, Hash Functions Based on Block Ciphers, in *Advances in Cryptology, Proc. Eurocrypt '92, LNCS 658*, R.A. Rueppel, Ed., Springer-Verlag, 1992, pp. 55–70.

[65] R. Lidl, H. Niederreiter, *Finite Fields*, *Encyclopaedia of Mathematics and its Applications* Vol. 20, Reading, Mass, Addison-Wesley, 1983.

[66] M. Luby and C. Rackoff, How to Construct Pseudorandom Permutations from Pseudorandom Functions, *SIAM Journal on Computing*, Vol. 17, No. 2, 1988, pp. 373–386.

[67] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Publishing Company, 1978.

[68] J.L. Massey, Contemporary Cryptology: An introduction, in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, Ed., IEEE Press, 1991, pp. 1–39.

[69] J.L. Massey, Shift-Register Synthesis and BCH Decoding, *IEEE Trans. Informat. Theory*, Vol. IT-15, 1969, pp. 122–127.

[70] Y. Matias and A. Shamir, A Video Scrambling Technique Based On Space Filling Curves, in *Advances in Cryptolog, Proc. Crypto'87, LNCS 293*, C. Pomerance, Ed., Springer-Verlag 1987, pp. 398–417.

[71] M. Matsui, Linear Cryptanalysis Method for DES Cipher, *Advances in Cryptology, Proc. Eurocrypt '93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1993, pp. 386–397.

[72] M. Matsui, The First Experimental Cryptanalysis of the Data Encryption Standard, in *Advances in Cryptology, Proc. Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 1–11.

[73] U.M. Maurer, *Provable Security in Cryptography*, Doctoral Dissertation ETH No. 9260, Zürich, 1990.

[74] U.M. Maurer, New Approaches to the Design of Self-Synchronizing Stream Ciphers, in *Advances in Cryptology, Proc. Eurocrypt '91, LNCS 547*, D. Davies, Ed., Springer-Verlag 1991, pp. 458–471.

[75] U.M. Maurer and J.L. Massey, Cascade Ciphers: The Importance of Being First, *Journal of Cryptology*, Vol. 6, No. 1, 1993, pp. 55–61.

[76] W. Meier and O. Staffelbach, Analysis of Pseudo Random Sequences Generated by Cellular Automata, in *Advances in Cryptology, Proc. Eurocrypt '91, LNCS 547*, D.W. Davies, Ed., Springer-Verlag 1991, pp. 186–199.

[77] R.C. Merkle, One Way Hash Functions and DES, in *Advances in Cryptology, Proc. Crypto'89, LNCS 435*, G. Brassard, Ed., Springer-Verlag, 1990, pp. 428–446.

[78] R.C. Merkle, A Fast Software One-Way Hash Function, *Journal of Cryptology*, Springer-Verlag Vol. 3 No. 1, 1990, pp. 43–58.

[79] C.H. Meyer and S.M. Matyas, *Cryptography*, John Wiley & Sons, 1982.

[80] C.J. Mitchell, F. Piper and P. Wild, Digital Signatures, in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, Ed., IEEE Press, 1991, pp. 325–378.

[81] J. Nechvatal, Public Key Cryptography, in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, Ed., IEEE Press, 1991, pp. 177–288.

[82] L. O'Connor, On the Distribution of Characteristics in Bijective Mappings, in *Advances in Cryptology, Proc. Eurocrypt '93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 360–370.

[83] G. Peeters, *Implementatie van een Cryptografisch Algoritme (Implementation of a Cryptographic Algorithm – in Dutch)*, Katholieke Industriële Hogeschool De Nayer-Mechelen, Thesis ind. eng., 1993.

[84] B. Preneel, *Analysis and Design of Cryptographic Hash Functions*, Doct. Dissertation KULeuven, 1993.

[85] B. Preneel, M. Nuttin, V. Rijmen and J. Buelens, Cryptanalysis of the CFB Mode of the DES with a Reduced Number of Rounds, *Advances in Cryptology, Proc. Crypto'93, LNCS 773*, D.R. Stinson, Ed., Springer-Verlag, 1994, pp. 212–223.

[86] J.-J. Quisquater and J.-P. Delescaille, Other Cycling Tests for DES, in *Advances in Cryptology, Proc. Crypto'87, LNCS 293*, C. Pomerance, Ed., Springer-Verlag 1987, pp. 255–256.

[87] J.-J. Quisquater and J.-P. Delescaille, How Easy Is Collision Search? Application to DES, *Advances in Cryptology, Proc. Eurocrypt'89, LNCS 434*, J.-J. Quisquater and J. Vandewalle, Eds., Springer-Verlag, 1990, pp. 429–434.

[88] J.-J. Quisquater and J.-P. Delescaille, How Easy Is Collision Search. New Results and Applications to DES, *Advances in Cryptology, Proc. Crypto'89, LNCS 435*, G. Brassard, Ed., Springer-Verlag, 1990, pp. 408–413.

[89] D. Richardson, Tessellation with Local Transformations, *J. Comp. Syst. Sci.*, Vol. 6, 1972, pp. 373–388.

[90] V. Rijmen, *Cryptanalysis of DES – in Dutch, Cryptanalyse van DES*, ESAT Katholieke Universiteit Leuven, Thesis grad. eng., 1993.

[91] R.L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications ACM*, Vol. 21, February 1978, pp. 120–126.

[92] R.L. Rivest, The MD4 Message Digest Algorithm, *Advances in Cryptology, Proc. Crypto'90, LNCS 537*, S. Vanstone, Ed., Springer-Verlag, 1991, pp. 303–311.

[93] R.L. Rivest, The MD5 Message Digest Algorithm, *presented at the rump session of Crypto'91.*

[94] J. Rompel, One-way functions are necessary and sufficient for secure signatures, in *Proc. 22nd ACM Symposium on the Theory of Computing*, 1990, pp. 387–394.

[95] R.A. Rueppel, Stream Ciphers, in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, Ed., IEEE Press, 1991, pp. 65–134.

[96] C. Schnorr, FFT-Hash, An Efficient Cryptographic Hash Function,*Presented at the rump Session of Crypto'91.*

[97] C. Schnorr, On the Construction of Random Number Generators and Random Function Generators, in *Advances in Cryptology, Proc. Eurocrypt '88, LNCS 330*, C.Günther, Ed., Springer-Verlag, 1988, pp. 225–232.

[98] C.E. Shannon, A Mathematical Theory of Communication, *Bell Syst. Tech. Journal*, Vol. 27, No. 3, 1948, pp. 379–423 and pp. 623–656.

[99] C.E. Shannon, Communication Theory of Secrecy Systems, *Bell Syst. Tech. Journal*, Vol. 28, 1949, pp. 656–715.

[100] G.J. Simmons, A Survey of Information Authentication, in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, Ed., IEEE Press, 1991, pp. 379–419.

[101] H. Taub and D.L. Schilling, *Principles of Communication Systems*, Mc. Graw-Hill, 1971.

[102] T. Toffoli, N. Margolus, Invertible Cellular Automata: A Review, *Physica D*, Vol. 45, 1990, pp. 229–253.

[103] G.S. Vernam, Cipher Printing Telegraph Systems for Secret Wire and Radio Telegraphic Communications, *J. Am. Inst. Elec. Eng.*, Vol. 55, 1926, pp. 7–10.

[104] S. Wolfram, Random Sequence Generation by Cellular Automata, *Advances in Applied Mathematics*, Vol. 7, 1986, pp. 123–169.

Nederlandse Samenvatting

1 Inleiding

In onze moderne maatschappij wordt steeds meer gebruik gemaakt van computers en telecommunicatienetwerken. Dit heeft aanleiding gegeven tot een revolutie op het vlak van informatieverwerking en -communicatie. Enerzijds heeft de moderne informatietechnologie een groot aantal nieuwe diensten mogelijk gemaakt. Anderzijds is het wenselijk dat papieren documenten, dragers van informatie maar ook van overeenkomsten of waarde, worden vervangen door “abstracte” tegenhangers. Deze abstracte documenten staan los van hun fysische drager en bestaan louter uit reeksen van symbolen.

In commerciële en politieke campagnes spiegelt men graag het beeld voor van de *multimedia informatie supersnelweg* waarin de consument, de onderzoeker of de manager alle denkbare diensten, uiteraard tegen betaling, binnen handbereik heeft. In het aanbod van deze diensten wordt een grote mobiliteit nagestreefd: een abonnee is niet gebonden aan een bepaalde plaats of een bepaald tijdstip voor het verkrijgen van een bepaalde dienst. Deze mobiliteit impliceert een grote fysische toegankelijkheid van het communicatienetwerk. De combinatie van deze toegankelijkheid en de hoge gevoeligheid van vele toepassingen maakt een degelijke beveiliging noodzakelijk voor de sociale aanvaardbaarheid van dergelijke netwerken. Deze beveiliging moet hoofdzakelijk op het logische vlak worden gerealiseerd.

De verzameling wiskundige en logische functies en methoden die ingezet kunnen worden in deze logische beveiliging wordt doorgaans aangeduid met de term *cryptografie*, hun analyse met de term *cryptanalyse* en de gezamenlijke wetenschap van cryptografie en cryptanalyse met *cryptologie*. Tot voor enkele decennia stond cryptografie gelijk aan geheimschrift verkregen door middel van *encryptie* onder invloed van een *geheime sleutel*. Dit geheimschrift kan terug leesbaar gemaakt worden door *decryptie* met diezelfde geheime sleutel. Alle kennis betreffende deze *encryptieschema's* en het kraken ervan was voorbehouden aan een kleine elite in dienst bij militaire hoofdkwartieren, geheime diensten of een enkel produktiebedrijf [58]. Onder invloed van de revolutie in telecommunicatie- en computertechnologie is de cryptologie gedurende de laatste decennia echter uitgegroeid tot een volwaardige wetenschappelijke discipline. Deze nieuwe openheid is een vruchtbare bodem gebleken voor het ontstaan van een aantal zeer nuttige nieuwe cryptografische schema's en concepten [68].

Binnen de cryptografie hebben wij ons geconcentreerd op het ontwerp van schema's die grote hoeveelheden gegevens te verwerken krijgen. Deze schema's zijn in hoge mate verantwoordelijk voor de haalbare verwerkings- of communicatiesnelheid in cryptografisch beveiligde toepassingen. Het is duidelijk dat encryptieschema's, die de geheimhouding van communicatie kunnen beschermen, tot deze klasse behoren. Encryptie biedt echter weinig tot geen bescherming tegen ongeoorloofde externe manipulatie van berichten. Deze bescherming kan wel verkregen worden met een cryptografisch controlesom mechanisme. In zo'n mechanisme is er maar één component die grote hoeveelheden gegevens verwerkt, de *cryptografische hashfunctie*. Deze beeldt een bericht van willekeurige lengte af op een hashresultaat met een bepaalde lengte, al of niet onder invloed van een geheime sleutel. Alle eventuele andere bewerkingen gebeuren op dit relatief korte resultaat.

Cryptografische hashfuncties hebben in feite een zeer breed toepassingsdomein, gaande van digitale handtekeningen tot bepaalde identificatieprotocols. Voor een overzicht van de verschillende toepassingen refereren wij naar de doctoraatsverhandeling van Bart Preneel [84]. De eigenschappen die van een cryptografische hashfunctie verwacht worden hangen af van de toepassing. In de meeste toepassingen is het voldoende dat het ondoenbaar moet zijn *botsende* berichten te vinden. Dit zijn verschillende berichten die op hetzelfde resultaat afgebeeld worden.

Op dit moment bestaan er al een groot aantal concrete voorstellen voor encryptieschema's en cryptografische hashfuncties. Men kan zich dus afvragen of de studie van het ontwerp van deze bouwblokken nog wel zinvol is. Volgens ons is dit wel degelijk het geval. In ons proefschrift trachten wij aan te duiden wat er schort aan de gangbare ontwerpmethoden en stellen wij een nieuwe aanpak voor. Ons doel is bijdragen aan ontwerpmethoden voor encryptieschema's en cryptografische hashfuncties waarin twee aspecten centraal staan. Enerzijds moeten de ontwerpen zich lenen tot goedkope en snelle implementaties, liefst op een zo groot mogelijke verscheidenheid van platformen. Anderzijds mag er van deze ontwerpen verwacht worden dat het beveiligingsniveau zo hoog ligt als de dimensies suggereren.

Na het bespreken van de verschillende soorten schema's waarvan we het ontwerp willen bestuderen, behandelen we cryptografische veiligheid. Dit wordt gevolgd door een korte bespreking van onze algemene ontwerpstrategie. We geven een overzicht van propagatie en correlatie, de twee belangrijkste aspecten van transformaties in cryptografische schema's, en bespreken vervolgens de klasse van de translatie-invariante transformaties waartoe de meeste basiscomponenten van onze ontwerpen behoren. Daarna behandelen we achtereenvolgens het ontwerp van zogenaamde blokcijfers, stroom/hash modules en zelf-synchroniserende stroomcijfers. Na het bespreken van schema's die gebaseerd zijn op rekenkundige bewerkingen, formuleren we tenslotte onze belangrijkste besluiten en enkele suggesties voor verder onderzoek.

2 Encryptie, cijfers en hashfuncties

Door middel van encryptie wordt een bericht omgezet in een *cryptogram*. Meestal gebeurt dit door sequentiële *substitutie*: de symbolen van het bericht worden één voor één vervangen door cryptogramsymbolen. In een ander type van schema's gebeurt encryptie door *transpositie*: het wijzigen van de volgorde van de symbolen in het bericht. Vroeger werden substitutie en transpositie dikwijls gecombineerd. Door zijn relatief grote implementatiecomplexiteit en gebrekkige bescherming wordt transpositie tegenwoordig alleen nog maar gebruikt in een klein aantal zeer specifieke toepassingen. Daarom hebben we ons volledig toegespitst op het ontwerp van substitutieschema's.

In alle substitutieschema's kan men encryptie als volgt beschrijven. Een berichtsymbool wordt omgezet in een cryptogramsymbool door een omkeerbare *encryptietransformatie* die afhangt van de inwendige toestand van een eindige toestandsmachine. Deze eindige toestandsmachine past voor elk nieuw te encrypteren symbool zijn inwendige toestand aan. De nieuwe toestand hangt af van de vorige toestand en het laatste cryptogramsymbool. De inwendige toestand hangt als zodanig af van alle verleden cryptogramsymbolen. In praktijk wordt de eindige toestandsmachine echter op zo'n manier ontworpen dat de inwendige toestand slechts afhangt van de m laatste cryptogramsymbolen. Het getal m wordt het *ingangsgeheugen* genoemd.

Bij het begin van encryptie wordt de initiële inwendige toestand verkregen door de *initialisatie-afbeelding* toe te passen op de combinatie van de geheime sleutel K en de publieke parameter Q. De cryptogramsymbolen worden verzonden via een zogenaamd *kanaal* en na ontvangst terug gedecrypteerd. Decryptie verloopt hetzelfde als encryptie met dit verschil dat de encryptietransformatie vervangen wordt door zijn inverse. In moderne schema's bestaan de symbolen uit bits of bitblokken met een constante lengte en hebben de cryptogramsymbolen dezelfde lengte als de berichtsymbolen.

De aanwezigheid van transmissiefouten in het kanaal kan de decryptie sterk bemoeilijken. De juiste ontvangst van een bepaald berichtsymbool veronderstelt de foutloze transmissie van het overeenkomstige cryptogramsymbool en de gelijkheid van inwendige toestand bij encryptie en decryptie. Dit laatste impliceert dat encryptor en decryptor synchroon lopen en dat de m laatste cryptogramsymbolen correct ontvangen zijn. Door bepaalde randvoorwaarden in het ontwerp in te bouwen kan men de nood aan synchronisme elimineren of ervoor zorgen dat het ingangsgeheugen gelijk is aan 0 zodat de inwendige toestand onafhankelijk is van de cryptogramsymbolen. Met betrekking tot foutengevoeligheid zijn de drie relevante eigenschappen dus de symbool- of bloklengte, de nood aan synchronisme en de waarde van m. Deze eigenschappen bepalen ook grotendeels de indeling van de encryptieschema's in klassen.

Men spreekt van stroomencryptie als men uit cryptogramsymbolen en de overeenkomstige berichtsymbolen gemakkelijk kan achterhalen wat de encryptietransformatie is. De veiligheid is gebaseerd op het feit dat deze transformatie steeds verandert onder invloed van de inwendige toestand. In onze analyse gaan we er van uit dat in stroomencryptie het bericht wordt omgezet naar een cryptogram door de bitsgewijze toevoeging van een *sleutelstroom*. Men spreekt van blokencryptie als het moeilijk is de encryptietransformatie te bepalen uit een verzameling koppels bestaande uit een cryptogramsymbool en het overeenkomstige berichtsymbool.

In synchrone stroomencryptie is de sleutelstroom afkomstig van een autonome eindige toestandsmachine. Deze autonome eindige toestandsmachine noemt men een synchroon *stroomcijfer*. Door middel van de publieke parameter Q kan men met eenzelfde sleutel een groot aantal verschillende sleutelstromen genereren. Door het eindige aantal mogelijke toestanden komt het synchrone stroomcijfer vroeg of laat in een periodische cyclus terecht. Voor de lengte van deze cycli is het voordelig dat de toestandsovergang omkeerbaar is. In vele toepassingen is het handig als men uit de initiële toestand rechtstreeks de toestand na een willekeurig aantal iteraties kan berekenen. Dit vereist een speciale structuur van de overgangstransformatie. Men spreekt dan van een *cryptografische gefilterde teller*.

In zelf-synchroniserende stroomencryptie is de sleutelstroom afkomstig van een zelf-synchroniserend stroomcijfer bestaande uit een eindige toestandsmachine met een eindig ingangsgeheugen. Het encrypterende symbool is een functie van de inwendige toestand en de *cijfersleutel*. Decryptie is mogelijk van zodra de laatste m cryptogramsymbolen correct ontvangen zijn. Het gevaar voor informatielekken door herhalingen in cryptogrammen wordt bepaald door het produkt van de symboollengte en het ingangsgeheugen. De terugkoppeling van de uitgang van een zelf-synchroniserend stroomcijfer naar zijn ingang resulteert in een autonome eindige toestandsmachine. Als die gebruikt wordt voor synchrone stroomencryptie, spreekt men van de uitgangsteruggekoppelde gebruikswijze. We hebben aangetoond dat het cyclisch gedrag sterk verbeterd kan worden door op een intelligente wijze terug te koppelen.

In eenvoudige blokencryptie wordt een bericht blok per blok omgezet in een cryptogram door middel van een omkeerbare transformatie die afhangt van een cijfersleutel. De sleutelafhankelijke transformatie wordt een *blokcijfer* genoemd. Eenvoudige blokencryptie heeft het nadeel dat de herhaling van blokken in een bericht resulteert in een herhaling in het cryptogram. Dit is niet geval bij de kettingblokencryptie, waarbij het blokcijfer wordt voorafgegaan door een bitsgewijze optelling van het vorige cryptogramsymbool. In deze gebruikswijze wordt het gevaar voor informatielekken door herhalingen in cryptogrammen bepaald door de bloklengte.

Als men de uitgang van een blokcijfer terugkoppelt naar zijn ingang verkrijgt men een autonome eindige toestandsmachine die men kan gebruiken voor synchrone stroomencryptie. Een blokcijfer kan ook gebruikt worden als filter in een cryptografische gefilterde teller. Tenslotte kan men ook zelf-synchroniserende stroomencryptie doen met een blokcijfer. De laatste m cryptogramsymbolen, opgeslagen in een doorschuifregister, worden afgebeeld op het encrypterende symbool door tussenkomst van het blokcijfer en een bitselectie.

Alle praktische cryptografische hashfuncties werken op een sequentiele manier zodat "voorbijkomende" berichten kunnen verwerkt worden zonder een beroep te moeten doen op uitwendige opslag. Eerst wordt het bericht verdeeld en zo nodig verlengd tot een ingangsreeks bestaande uit een geheel aantal blokken. Daarna wordt er aan de zogenaamde *kettingtoestand* een bepaalde beginwaarde toegekend. Deze toestand wordt iteratief getransformeerd door voor elk ingangsblok een *kettingtransformatie* toe te passen. Het hashresultaat wordt berekend uit de kettingtoestand nadat alle ingangsblokken "aangebracht" zijn. Met betrekking tot de weerstand tegen botsingen moeten zowel het hashresultaat als de kettingtoestand een bepaalde lengte hebben.

3 Cryptografische veiligheid

Het principe van Kerckhoffs (1835–1903) is één van de meest elementaire beginselen van de cryptografie. Het zegt dat een ontwerper moet veronderstellen dat de vijand op de hoogte is van het volledige encryptie- of hashmechanisme, met uitzondering van de geheime sleutel. In de definitie, de modellering en de analyse van de veiligheid gaat men er bijna altijd van uit dat de vijand geen a priori informatie heeft over de sleutel. Dit is in vele praktische omstandigheden echter niet het geval. Daarom houden wij in onze definities van veiligheid expliciet rekening met niet-uniforme sleutelselectie.

De toegang die een cryptanalyst heeft tot een cryptografisch schema wordt bepaald door de omstandigheden waarin het gebruikt wordt. Bijvoorbeeld, de toegang tot een encryptieschema varieert van de observatie van cryptogrammen tot de mogelijkheid tot manipulatie van de ingang en de publieke parameter Q. Een belangrijke theoretische aanval is exhaustief sleutel zoeken. Als de aanwezige a priori informatie over een bericht groter is dan de ontbrekende informatie over de geheime sleutel kan de juiste sleutel doorgaans gevonden worden door voor alle andere sleutels vast te stellen dat ze het cryptogram naar een inconsistent bericht decrypteren.

Ontwerpers van beveiligingssystemen zouden erg blij zijn met praktische cryptografische schema's waarvan men zwart op wit kan bewijzen dat ze veilig zijn. In de praktijk stuiten pogingen tot het ontwerp van bewijsbaar veilige schema's echter op grote problemen. Deze pogingen zijn vooral ingegeven door twee verschillende theorieën: informatietheorie en complexiteitstheorie.

Informatietheorie behandelt de mogelijkheid van aanvallen, zonder rekening te houden met de grootte van de cryptanalytische inspanning. Ze geeft scherpe bovengrenzen aan de haalbare onvoorwaardelijke beveiliging. Samenvattend kan men zeggen dat schema's die bewijsbaar veilig zijn in informatietheoretische zin een zeer grote hoeveelheid geheime sleutelbits vereisen, zowel in de realisatie van geheimhouding als van authentisering.

Complexiteitstheorie is een discipline binnen de computerwetenschappen die het gebruik van tijd en geheugen van algoritmen bestudeert. Binnen de cryptografie verwacht(te) men door de toepassing van complexiteitstheorie cryptografische schema's te kunnen ontwerpen die bewijsbaar veilig zijn tegen een vijand met beperkte middelen. Complexiteitstheorie, zoals ze toegepast wordt binnen cryptografie, laat echter alleen maar *asymptotische* uitspraken toe over de complexiteit van aanvallen in het slechtste geval (met betrekking tot de aanvaller). In praktijk zijn deze volkomen waardeloos. We hebben dit geïllustreerd met onze aanval op de zogenaamde Even-Mansour blokcijferconstructie [34]. Onze aanval is vernietigend voor alle blokcijfers van dit type met een realistische bloklengte. Toch is het bestaan van onze aanval niet in strijd met het complexiteitstheoretisch bewijs van veiligheid van de ontwerpers.

Velen geloven dat de toepassing van complexiteitstheorie aanleiding geeft tot gezonde ontwerpprincipes. Complexiteitstheorie heeft alleszins bijgedragen tot een manier van ontwerpen die wij aanduiden met de term "N-reductionistisch". Hierin ligt de nadruk op het herleiden van de veiligheid van één cryptografische component tot die van een andere waarvan verondersteld wordt dat die gemakkelijker te ontwerpen is. In praktijk is het echter onmogelijk om deze veronderstelling te verifiëren en legt de reductie vrij ernstige beperkingen op. Bovendien leidt deze aanpak de aandacht af van het echte ontwerp. De reductie wordt beschouwd als het "fundamentele" en "wetenschappelijke" gedeelte van het ontwerp, terwijl het ontwerp van de onderliggende component noodzakelijkerwijs "ad hoc" is, en bijgevolg minder interessant. De negatieve gevolgen van deze houding worden geïllustreerd door onze vaststelling van ontoelaatbare zwakheden in de cryptografische hashfunctie gebaseerd op cellulaire automaten van Ivan Damgård [29].

In praktijk is cryptografische veiligheid nauw verwant aan vertrouwen en engagement. Hierin is een belangrijke dubbele rol weggelegd voor de cryptografische "claim", een heldere formulering van de cryptografische veiligheid die elk nieuw ontwerp zou moeten vergezellen. Voor de cryptanalytici moet het een uitdaging zijn om een zwakheid te vinden die de claim weerlegt. Voor gebruikers en systeemontwerpers dient de claim als een specificatie van de externe cryptografische eigenschappen van het ontwerp. Het gebruik van een ontwerp impliceert vertrouwen in de geldigheid van de begeleidende claim. De enige beschikbare fundering voor dit vertrouwen bestaat in de vaststelling dat er in de publieke evaluatie van het ontwerp nog geen zwakheden zijn gevonden die de claim weerleggen, ondanks grote inspanningen.

Een *super cryptografische schema* heeft binnen zijn klasse en voor zijn dimensies de kleinst mogelijke voorspelbaarheid. We noemen een cryptografisch schema *K-veilig* als voor elke a priori sleutelverdeling een aanval die werkt voor het schema ook zou werken voor de meerderheid van super schema's van hetzelfde type en met dezelfde dimensies. We noemen een cryptografisch schema *hermetisch* als elke aangetoonde zwakheid van dat schema ook aanwezig is bij de meerderheid van de super schema's met dezelfde dimensies. K-veilig en hermetisch zijn zeer strenge definities van veiligheid en moeten het eenvoudig maken om bondige en heldere cryptografische claims te formuleren.

In veel toepassingen wordt er stilzwijgend van uitgegaan dat encryptie een garantie biedt voor de detectie van ongeoorloofde manipulaties tussen encryptor en decryptor. De geboden bescherming hangt echter sterk af van de relatieve redundantie in de boodschappen en het gebruikte encryptiemechanisme. We hebben de bovengrenzen van de geboden bescherming onderzocht met betrekking tot de verschillende soorten encryptie, twee soorten verdeelde redundantie en de toepassing van een cryptografische hashfunctie. Hieruit blijkt dat in deze context de toepassing van een cryptografische hashfunctie in combinatie met blokencryptie de voorkeur geniet.

4 Ontwerpstrategie

Onze ontwerpstrategie is er op gericht te komen tot encryptieschema's, cijfers en cryptografische hashfuncties met een aantal specifieke kenmerken. Een eerste wenselijk kenmerk is *eenvoud* met betrekking tot beschrijving en implementatie. Een ander kenmerk is structurele helderheid: de ontwerpen moeten gemakkelijk te analyseren zijn, zodat men zich kan verzekeren van weerstand tegen bepaalde aanvallen. Bovendien moeten de ontwerpen zich lenen tot efficiënte implementaties in hardware en software.

Een nuttig nieuw concept in deze context is de cryptografische eindige toestandsmachine. Hiermee kunnen we blokcijfers, synchrone stroomcijfers en cryptografische hashfuncties bouwen. De cryptografische eindige toestandsmachine bestaat functioneel uit vier blokken: het toestandsregister, de buffer, de overgangstransformatie en de controlelogica. Door middel van de overgangstransformatie wordt een nieuwe toestand berekend uit de vorige toestand en de inhoud van de buffer. Dit noemen we een *iteratie*. We gaan er van uit dat de overgangstransformatie omkeerbaar is, een grote uniformiteit heeft en in hardware aanleiding geeft tot een korte cyclustijd.

Door middel van de controlelogica kan men aangeven wat er in een cyclus met de toestand en de inhoud van de buffer gebeurt. De toestand kan worden geladen, geïtereerd, op 0 gezet of behouden. De buffer bestaat meestal uit een doorschuifregister verbonden met een ingang. Dit register kan worden geladen, behouden of op 0 gezet. De cryptografische schema's worden gespecifieerd in functie van operaties van de cryptografische eindige toestandsmachine.

In bijna alle ontwerpen bestaan blokcijfers uit het iteratief toepassen van een bepaalde omkeerbare sleutelafhankelijke ronde-transformatie. In onze aanpak is dat niet anders. In onze cryptografische eindige toestandsmachine bestaat de overgangstransformatie uit deze ronde-transformatie. De encryptie van één blok bestaat uit het laden van het berichtblok in het toestandsregister, het uitvoeren van een aantal iteraties en het uitlezen van het cryptogramblok uit het toestandsregister. De buffer bevat de cijfersleutel.

Om historische redenen maken de meeste ontwerpen voor synchrone stroomcijfers veelvuldig gebruik van lineair teruggekoppelde doorschuifregisters. De meerderheid van deze ontwerpen biedt echter zeer weinig bescherming als er regelmatig hersynchronisatie gebeurt zonder de introductie van nieuw sleutelmateriaal. Wij hebben aangetoond dat in het geval van een lineaire initialisatie-afbeelding en een lineaire overgangstransformatie, regelmatige hersynchronisatie leidt tot het volledig verdwijnen van de cryptografische veiligheid.

Wij stellen voor om synchrone stroomcijfers te realiseren met een cryptografische eindige toestandsmachine. De buffer en het toestandsregister worden geïnitialiseerd door het laden van enkele initialisatieblokken en het uitvoeren van een aantal blanco (zonder uitgang) iteraties. Daarna verschijnen er per iteratie een vast aantal toestandsbits aan de uitgang.

Bijna alle ontwerpen van cryptografische hashfuncties zijn gebaseerd op de iteratieve toepassing van een compressiefunctie met een ingang van constante lengte. Bij elke toepassing van de compressiefunctie wordt de tussentijdse kettingtoestand samen met een ingangsblok omgezet naar de volgende tussentijdse kettingtoestand. Het hashresultaat bestaat uit de waarde van de kettingtoestand nadat alle ingangsblokken aan de beurt zijn geweest. Dit beperkt de lengte van de kettingtoestand tot de lengte van het hashresultaat.

Wij stellen voor om een cryptografische hashfunctie te realiseren met een cryptografische eindige toestandsmachine. Nadat de buffer en het toestandsregister op 0 gezet zijn, worden de ingangsblokken in het register geladen. Voor elk ingangsblok dat geladen wordt, gebeurt er een iteratie. Nadat alle ingangsblokken geladen zijn, gebeuren er eventueel nog een aantal blanco (zonder ingang) iteraties. Daarna wordt het hashresultaat afgeleid uit de toestand. Het iteratieproces kan men beschouwen als een encryptie van de initiële toestand 0 naar de uiteindelijke toestand, met de ingangsblokken als ronde-sleutels. Hierin bestaat de ronde-transformatie uit de overgangstranformatie en het sleutelschema, dat de ronde-sleutels afleidt uit de cijfersleutel, uit de buffer. Voor hashfuncties met sleutel worden de ingangsblokken voorafgegaan *en* gevolgd door één of meer sleutelblokken. Een belangrijk voordeel van onze aanpak is de afwezigheid van een bovengrens voor de lengte van de kettingtoestand.

We geven een korte bespreking van de verschillende basiscomponenten die beschikbaar zijn voor de beschrijving van cryptografische eindige toestandsmachines. *Bitpermutaties* zijn over het algemeen goed geschikt voor hardware. In software moeten we ons beperken tot speciale gevallen zoals cyclische bitrotaties en blokpermutaties. *Bitsgewijze Booleaanse operaties* zijn uitstekend geschikt voor hardware *en* software. *Substitutietabellen* zijn geschikt voor hardware als het aantal ingangen niet te groot is, voor software als het aantal ingangen niet te klein is. *Modulaire rekenkundige bewerkingen* lenen zich maar matig tot hardware-implementaties. In software is het belangrijk dat de woordlengte aangepast is aan die van de processor. In onze ontwerpen maken we vooral gebruik van bitpermutaties en bitsgewijze Booleaanse operaties. We combineren deze operaties meestal in transformaties gekenmerkt door symmetrie en parallellisme, zogenaamde *translatie-invariante transformaties.*

5 Propagatie en correlatie

De aspecten van Booleaanse afbeeldingen die de keuze en schikking van de componenten in onze ontwerpen bepalen, zijn verschilpropagatie en correlatie. De klasse van aanvallen die gebruik maken van een voorspelbare verschilpropagatie wordt differentië cryptanalyse (DC) [6] genoemd. De klasse van aanvallen die gebruik maken van ongewenste correlaties noemt men lineaire cryptanalyse (LC) [71].

De originele formuleringen van LC en DC laten qua helderheid veel te wensen over. Daarom stellen we zowel voor de beschrijving van correlaties als die van verschilpropagatie een nieuw formalisme voor. De toepassing van deze nieuwe formalismen leidt tot een beter inzicht in de beschreven fenomenen en vergemakkelijkt theoretische afleidingen en bewijzen.

De correlatie tussen twee Booleaanse functies wordt aangegeven met een *correlatiecoëfficiënt* tussen -1 en $+1$. De lijst van de correlatiecoëfficiënten van een Booleaanse functie met alle lineaire Booleaanse functies noemt men de Walsh-Hadamard transformatie van die functie. Deze transformatie is een orthogonale basistransformatie in reële vectorruimten waarby Booleaanse functies de coördinaten vormen van vectoren met een speciale vorm. In deze context geeft de Walsh-Hadamard transformatie een uitdrukking voor de Booleaanse functie in de vorm van een lineaire combinatie van lineaire Booleaanse functies.

Door de correlatiecoëfficiënten tussen elke lineaire combinatie van uitgangsbits en elke lineaire combinatie van ingangsbits van een Booleaanse afbeelding in een matrix te schikken verkrijgen we zijn *correlatiematrix*. Uit de algebraïsche interpretatie van de Walsh-Hadamard transformatie volgt onmiddellijk een isomorfisme tussen de samenstelling van Booleaanse afbeeldingen en de vermenigvuldiging van de overeenkomstige correlatiematrices. Twee soorten componenten die veel voorkomen in cryptografische ontwerpen zijn affiene afbeeldingen en afbeeldingen bestaande uit de parallelle toepassing van een aantal substitutietabellen, zogenaamde *blok-parallelle afbeeldingen*. De elementen van correlatiematrices van beide soorten afbeeldingen zijn gemakkelijk te berekenen. Bovendien kunnen we op relatief eenvoudige wijze een aantal eigenschappen van correlatiematrices bewijzen.

Bij verschilpropagatie bestudeert men het bitsgewijze verschil tussen paren van uitgangen van een Booleaanse afbeelding die overeenkomen met paren van ingangen met een bepaald bitsgewijs verschil a. Het aantal paren met een bepaald verschil aan de uitgang b gedeeld door het totaal aantal paren noemen we de *prop ratio* van de verschilpropagatie van a naar b. Het negatieve binaire logaritme van de prop ratio van een verschilpropagatie noemen we zijn *restrictiegewicht*. Prop ratio's van affiene en blok-parallelle afbeeldingen zijn gemakkelijk te berekenen.

Binnen ons formalisme is het gemakkelijk te bewijzen dat voor Booleaanse afbeeldingen de tabel van de prop ratio's en de tabel met de gekwadrateerde elementen van de correlatiematrix verbonden zijn door een Walsh-Hadamard transformatie.

Onze formalismen zijn eenvoudig toe te passen op geïtereerde transformaties. Bij de beschrijving van correlatie spreken we van een *lineair spoor* met bijbehorende *correlatiebijdrage coëfficiënt* tussen -1 en 1. De correlatie tussen een lineaire combinatie van ingangsbits en een lineaire combinatie van uitgangsbits bestaat dan uit de som van de correlatiebijdrage coëfficiënten van de lineaire sporen die de twee lineaire combinaties verbinden. Bij de beschrijving van verschilpropagatie geeft dit aanleiding to het concept van *differentieel spoor* met bijbehorende *prop ratio*. De prop ratio van de verschilpropagatie van de ingang naar de uitgang is gelijk aan de som van alle prop ratio's van de differentiële sporen die ingangsverschil en uitgangsverschil verbinden. De toepassing van onze formalismen op geïtereerde transformaties maakt de veelal stilzwijgende veronderstellingen en benaderingen in praktische LC en DC aanvallen expliciet.

Vertrekkende van de correlatie- en verschilpropagatie-eigenschappen van lineaire transformaties enerzijds en blok-parallelle transformaties anderzijds komen we tot de formulering van de *strategie van het brede spoor*. Voor blok-parallelle transformaties is het restrictiegewicht van een verschilpropagatie groter dan of gelijk aan het produkt van het aantal betrokken substitutietabellen met het minimum restrictiegewicht van de tabellen. Het restrictiegewicht van een differentieel spoor is dus groter dan of gelijk aan het produkt van het totaal aantal betrokken tabellen met het minimum restrictiegewicht van de tabellen. De strategie van het brede spoor bestaat er in om een ronde-transformatie of een overgangstransformatie samen te stellen uit (onder andere) een blok-parallelle transformatie en een lineaire transformatie. De (liefst kleine) substitutietabellen van de blok-parallelle transformatie worden zo gekozen dat hun minimum restrictiegewicht zo groot mogelijk is. De lineaire transformatie moet zo ontworpen worden dat er geen differentiële sporen zijn met een klein aantal betrokken tabellen. Voor lineaire sporen geldt een analoge redenering. De strategie van het brede spoor is niet gebonden aan blok-parallelle transformaties maar kan even goed toegepast worden met bijvoorbeeld een translatie-invariante transformatie als niet-lineaire component.

6 Translatie-invariante transformaties

Translatie-invariante transformaties zijn transformaties die commuteren met translatie. Dit impliceert dat elke component van het beeld op dezelfde manier afhangt van de componenten in zijn *buurt*. De beschrijving van deze *lokale afbeelding* is dus voldoende voor de specificatie van de volledige transformatie. Door hun parallellisme en symmetrie lenen translatie-invariante transformaties zich zeer goed voor implementaties in hard- en software. Wij bestuderen alleen translatie-invariante transformaties op binaire 1-dimensionale eindige vectoren. Translatie komt hier overeen met een cyclische doorschuifoperatie.

Wat betreft omkeerbaarheid maken we een onderscheid tussen *lokale* en *globale* omkeerbaarheid. De globale omkeerbaarheid van een translatie-invariante transformatie kan worden gespecifieerd met een karakteristieke verzameling van gehele getallen. Een globaal inverteerbare transformatie is dan omkeerbaar voor vectoren met een dimensie die geen veelvoud is van enig element van deze karakteristieke verzameling. Lokale omkeerbaarheid komt overeen met een lege karakteristieke verzameling.

Lineaire translatie-invariante transformaties kunnen gemodelleerd worden door vermenigvuldiging met een binaire veelterm modulo $1 + x^n$. Gebruik makende van hun algebraïsche eigenschappen kunnen we gemakkelijk de karakteristieke verzamelingen van deze veeltermen bepalen. Voor niet-lineaire translatie-invariante transformaties is een totaal andere aanpak vereist. Tot nu toe waren de enige in de literatuur vermelde omkeerbare niet-lineaire translatie-invariante transformaties van het *behouden-landschap*-type. Onze nieuwe *zaad-en-sprong*-bewijsmethode brengt een groot aantal nieuwe omkeerbare transformaties en hun overeenkomstige karakteristieke verzamelingen aan het licht.

Diffusie is de term die gebruikt wordt om de uitspreiding van informatie aan te duiden. Wat betreft diffusie zijn de lineaire omkeerbare translatie-invariante transformaties veruit superieur aan hun niet-lineaire tegenhangers. In deze context definiëren we de *Hamminggewicht-distributietabel* en het *vertakkingsgetal* van een lineaire transformatie. Beiden geven een goed beeld van de minimale diffusie die een lineaire afbeelding realiseert met betrekking tot zowel verschilpropagatie als correlatie.

Van alle omkeerbare niet-lineaire translatie-invariante transformaties is eigenlijk alleen de meest eenvoudige, aangeduid met het symbool χ, nuttig voor ons. Door zijn kleine implementatiecomplexiteit en zijn eenvoudige niet-lineaire eigenschappen is het namelijk een ideale kandidaat voor de rol van niet-lineaire transformatie in de strategie van het brede spoor. Het restrictiegewicht van een verschilpropagatie hangt alleen af van het ingangsverschil. Met betrekking tot correlatie definiëren we analoog hiermee een correlatiegewicht dat alleen afhangt van de lineaire combinatie van uitgangsbits.

7 Blokcijfers

Het is een voordeel voor een blokcijfer dat de transformatie en zijn inverse door dezelfde hardwaremodule kunnen uitgevoerd worden. Dit wordt in de meeste blokcijfers gerealiseerd door toepassing van de zogenaamde Feistelstructuur. In een Feistel-ronde-transformatie ondergaat echter maar de helft van het tussentijdse encryptieresultaat een niet-lineaire transformatie. Van dit feit wordt dankbaar gebruik gemaakt in LC en DC. Daarom stellen wij een nieuwe, meer uniforme structuur voor. Hierin bestaat de ronde-transformatie uit de opeenvolging van een aantal eenvoudige transformaties. Het opleggen van de zelf-inverse eigenschap leidt dan tot een aantal algebraïsche vergelijkingen waaraan de transformaties moeten voldoen. Het enige verschil in encryptie en decryptie bestaat in de cijfersleutel en de zogenaamde *ronde-constanten*.

De ronde-transformatie bestaat uit de opeenvolging van 4 verschillende transformaties, elk met hun eigen specifieke bijdrage:

- een niet-lineaire transformatie,
- een lineaire transformatie voor de diffusie,
- de bitsgewijse optelling van de ronde-sleutel,
- bitpermutaties voor “verspreiding.”

Als niet-lineaire component stellen we een variant van χ voor, als lineaire component een modulaire vermenigvuldiging met een veelterm en als bitpermutatie de cyclische rotatie op deelblokken. In de resulterende structuur worden lineaire en differentiële sporen bepaald door dezelfde vergelijkingen en kan men dus aan optimalisatie doen met betrekking tot LC en DC in één enkele inspanning. Een ander aspect waar veel aandacht aan besteed is, is het elimineren van zwakheden door symmetrie.

We stellen twee specifieke ontwerpen voor: 3-Way met een bloklengte van 96 bits en BaseKing met een bloklengte van 192 bits. Beiden zijn eenvoudig te implementeren in hard- en software.

8 Stroom/hash modules

Synchrone stroomcijfers en cryptografische hashfuncties kunnen allebei gerealiseerd worden door middel van een cryptografische eindige toestandsmachine. De voorwaarden opgelegd door de twee toepassingen zijn verenigbaar en zelfs in hoge mate gelijklopend. Dit heeft ons op het idee gebracht om cryptografische modules te ontwerpen die kunnen gebruikt worden zowel voor hashen als voor synchrone stroomencryptie. De belangrijkste component van de eindige toestandsmachine is de overgangstransformatie. Deze bestaat zoals in het geval van blokcijfers uit de opeenvolging van een aantal eenvoudige transformaties: een niet-lineaire, een lineaire voor diffusie, een bitpermutatie voor verspreiding, de bitsgewijze optelling van buffer bits en de bitsgewijze optelling van een constante voor asymmetrie.

Met betrekking tot de hashfunctie is het ontwerp van de eindige toestandsmachine vooral gericht op de weerstand tegen botsingen. Een botsing komt overeen met een differentieel spoor dat aanleiding geeft tot een verschil in het hashresultaat gelijk aan 0. In het ontwerp streven we ernaar om controle van differentiële sporen onmogelijk te maken. Dit doen we door ervoor te zorgen dat er geen differentiële sporen kunnen voorkomen waarbij het restrictiegewicht kleiner is dan het aantal vrijheidsgraden in de ingangsblokken. Met betrekking tot het synchrone stroomcijfer moeten we ervoor zorgen dat de uitgangsbits niet toelaten de inwendige toestand te reconstrueren. Belangrijk in deze context zijn de lineaire sporen, die aangeven met welke lineaire combinaties van toestandsbits de uitgangsbits gecorreleerd zijn. De weerstand met betrekking tot aanvallen waarbij de cryptanalyst de publieke parameter Q manipuleert, moet gerealiseerd worden in de blanco iteraties van de initialisatie en kan bestudeerd worden met verschilpropagatie.

In dit kader hebben we SUBTERRANEAN voorgesteld, een stroom/hash module speciaal ontworpen voor hardware-implementaties. SUBTERRANEAN heeft een inwendige toestand van 257 bits, een buffer bestaande uit een doorschuifregister van 8 trappen van elk 32 bits en een uitgang van 16 bits. De hashfunctie heet SUBHASH en kan 32 bits hashen per klokcyclus. Het synchrone stroomcijfer heet SUBSTREAM en levert 16 encrypterende bits per klokcyclus. Onder leiding van Prof. Luc Claesen van IMEC zijn we tot een chip-implementatie van SUBTERRANEAN gekomen in de MIETEC 2.4μm CMOS standaardcellen technologie. Voor deze chips werden hashsnelheden gemeten tot 572 Mbit/s en encryptiesnelheden tot 286 Mbit/s. Hun toepasbaarheid wordt geïllustreerd in een demonstratie-opstelling waarin een rechtstreekse videocommunicatieverbinding wordt beschermd met encryptie.

Het ontwerp van SUBTERRANEAN is in grote mate bepaald door twee vroegere voorstellen: synchrone stroomencryptie door middel van *cellulaire automaten* en de hashfunctie CELLHASH. Cellulaire automaten zijn automaten met een translatie-invariante overgangstransformatie. In [104] stelde Stephen Wolfram voor om deze automaten te gebruiken als synchrone stroomcijfers. Na experimenten met de door hem voorgestelde cellulaire automaten kwamen we tot enkele bevindingen in verband met omkeerbaarheid en diffusie. Dit heeft geleid tot de formulering van een eigen concreet voorstel. Daarna hebben we onze verworven kennis van translatie-invariante transformaties gebruikt in het ontwerp van CELLHASH, een cryptografische hashfunctie. SUBHASH is in feite een verbetering van CELLHASH wat betreft implementeerbaarheid en de correlatie- en propagatie-eigenschappen van de overdrachtstransformatie. Als varianten van SUBTERRANEAN vermelden we ons synchroon stroomcijfer JAM en de hashfunctie BOOGNISH. JAM kan beschouwd worden als een miniatuurversie van SUBSTREAM en werd ontworpen ter vervanging van gekraakt stroomcijfer in een betaal-TV-systeem. BOOGNISH was een poging om een variant van SUBHASH te ontwerpen die ook geschikt is voor software-implementaties. Achteraf zijn we tot de conclusie gekomen dat de inwendige toestand te kort is voor de geambieerde cryptografische eigenschappen.

STEPRIGHTUP is een stroom/hash module die zeer goed geschikt is voor software-implementaties. Alle operaties gebeuren op 32-bit woorden. De inwendige toestand bestaat uit 17 woorden, de buffer uit een lineair teruggekoppeld doorschuifregister met 33 trappen van elk 8 woorden. Het aantal bewerkingen per byte bedraagt zowel voor de hashfunctie als voor het stroomcijfer 3,7 bitsgewijze Booleaanse operaties en 0,5 cyclische doorschuifoperaties op 32-bit woorden. De belangrijkste nieuwe elementen van StepRightUp ten opzichte van Subterranean zijn de terugkoppeling in de buffer en de grootte van buffer en toestand.

9 Zelf-synchroniserende stroomcijfers

Bitsgewijze zelf-synchroniserende stroomcijfers hebben het voordeel dat ze aan een bestaand communicatiesysteem kunnen toegevoegd worden zonder de noodzaak voor bijkomende synchronisatie. Er is echter weinig literatuur over het ontwerp en de analyse van deze componenten. In de meest verspreide manier om zelf-synchroniserende stroomencryptie te realiseren wordt gewoon gebruik gemaakt van een blokcijfer en een doorschuifregister. Dit schema vereist een volledige uitvoering van het blokcijfer voor de encryptie van elke bit.

In dit domein hebben we ons toegelegd op het ontwerp van bitsgewijze zelf-synchroniserende stroomcijfers geschikt voor implementatie in hardware. Hierbij hebben wij ons grotendeels afgezet tegen de ontwerpstrategie voorgesteld door Ueli Maurer in [74].

Eén van de centrale ideeën in [74] bestaat erin het zelf-synchroniserende stroomcijfer op te bouwen uit een eindige toestandsmachine (met eindig ingangsgeheugen) die cryptografisch veilig is en een uitgangsfunctie die cryptografisch veilig is. Op de keper beschouwd blijkt dit een onuitvoerbare constructie die overigens niet zinvol is. Een andere reductionistische constructie voorgesteld in [74] is die van een stroomcijfer bestaande uit de parallelle constructie van meerdere stroomcijfers. Volgens elke betekenisvolle definitie van veiligheid zou de resulterende constructie even veilig zijn als de sterkste van de component-stroomcijfers. Dit is echter niet het geval voor onze definities K-veilig en hermetisch.

In [74] wordt voorgesteld om een eindige toestandsmachine met eindig ingangsgeheugen te bouwen door de toepassing van seriële en parallelle samenstelling van zelf-synchroniserende stroomcijfers. De toepasbaarheid van deze constructiewijze wordt geïllustreerd met een concrete structuur. Na onderzoek blijkt deze structuur aanleiding te geven tot ontoelaatbare zwakheden in het stroomcijfer. De belangrijkste redenen hiervoor zijn de slechte propagatie- en correlatie-eigenschappen van serieel samengestelde zelf-synchroniserende stroomcijfers.

Hiertegenover plaatsen wij een eigen structuur. Zoals in [74] stellen we een zelf-synchroniserend stroomcijfer samen uit twee bouwblokken. In ons geval is dat een *voorwaardelijk complementerend doorschuifregister* (CCSR) en een *pijplijnstructuur*. Er wordt echter van deze componenten niet verwacht dat ze op zichzelf cryptografische veiligheid hebben. Voor een CCSR is de afbeelding van de laatste m ingangsbits op de inwendige toestand een injectie. Ons concreet voorstel voor een CCSR heeft nog de bijkomende eigenschappen van *verbreding* voor de grootste geheugens en parallelle injectie. De pijplijnstructuur kan men vergelijken met een aantal blokcijferronden met daartussen geheugencellen. De trappen van deze structuur zijn echter niet gebonden aan dezelfde voorwaarden als de ronden in een blokcijfer. De ingang van de CCSR bestaat uit de laatste cryptogrambit. De pijplijnstructuur heeft als ingang de inwendige toestand van het CCSR en als uitgang de encrypterende bit.

Het gebruik van een CCSR en een pijplijnstructuur heeft gevolgen voor het uitwendige gedrag van het zelf-synchroniserende stroomcijfer en de toepassing van terugkoppeling. Beiden vormen echter geen probleem voor de veiligheid en de bruikbaarheid.

10 Rekenkundige schema's

Rekenkundige bewerkingen zijn veelgebruikte operaties in cryptografische schema's. Een onvoorzichtig gebruik kan echter aanleiding geven tot onaanvaardbare zwakheden. Dit illustreren we met de botsingen die we gevonden hebben voor de cryptografische hashfunctie FFT-Hash [96] en de klassen van zwakke sleutels voor het blokcijfer IDEA [63].

FFT-Hash is een cryptografische hashfunctie met een kettingtransformatie die de huidige 128-bit kettingtoestand samen met een 128-bit ingangsblok omzet naar de volgende 128-bit kettingtoestand. Deze kettingtransformatie bestaat uit de iteratieve toepassing van een lineaire modulaire Fouriertransformatie en een niet-lineaire recursieve transformatie. Door gericht gebruik te maken van de slechte propagatie van modulaire vermenigvuldiging in het geval dat één van de argumenten 0 is, zijn we er in geslaagd botsingen te genereren.

IDEA is een blokcijfer met een bloklengte van 64 bits en een sleutellengte van 128 bits. De basisblokken bestaan uit bitsgewijze optelling, optelling modulo 2^{16} en vermenigvuldiging modulo $2^{16}+1$ met 0000_{hex} geïnterpreteerd als 2^{16}. Optelling modulo 2^{16} heeft lineaire eigenschappen wat betreft de propagatie van de meest significante bits en de correlatie van de minst significante bits. Vermenigvuldiging met een subsleutel gelijk aan 1 komt overeen met de identitieke afbeelding, vermenigvuldiging met -1 komt overeen met een complementatie van alle bits en vervolgens de optelling van 2. In beide gevallen erft de vermenigvuldiging de lineaire eigenschappen van de modulaire optelling. De voorwaarde dat een bepaalde multiplicatieve sleutel gelijk is aan ± 1 komt overeen met de voorwaarde dat 15 bepaalde bits van de cijfersleutel gelijk zijn aan 0. Door deze voorwaarde op te leggen voor een bepaalde deelverzameling van de multiplicatieve subsleutels kan men ervoor zorgen dat er een bepaald lineair en gemakkelijk vaststelbaar verband bestaat tussen de ingang en de uitgang in een IDEA-encryptie. De voorwaarden dat de multiplicatieve subsleutels gelijk zijn aan ± 1 herleiden zich tot de voorwaarde dat een bepaald gedeelte van de sleutelbits gelijk is aan 0. In de praktische gevallen zijn er echter nog een aantal sleutelbits vrij en is er dus een groot aantal sleutels waarvoor IDEA een vaststelbaar lineair gedrag vertoont.

Vermenigvuldiging met een constante factor modulo $2^n - 1$, ofwel *cyclische vermenigvuldiging*, is een aantrekkelijke component voor cryptografische schema's. Deze operatie is translatie-invariant en combineert over het algemeen een hoge diffusie met goede niet-lineaire eigenschappen. Bovendien kan ze compact beschreven worden en heeft ze een relatief grote flexibiliteit in implementaties. Als de vermenigvuldigingsfactor copriem is met $2^n - 1$ is de operatie omkeerbaar en bestaat de inverse operatie uit vermenigvuldiging met de multiplicatieve inverse van de factor. De selectie van een factor uit alle mogelijke kandidaten vereist de bepaling van een selectiecriterium.

Zowel in FFT-Hash als in IDEA hebben we gebruik gemaakt van het propagatiegedrag van modulaire vermenigvuldiging in de meest ongunstige omstandigheden. Daarom hebben we als selectiecriterium gekozen voor de grootste nontriviale prop ratio voor een factor. Dit komt overeen met het meest ongunstige verschilpropagatiegedrag. Voor een woordlengte van 32 bits zijn er meer dan vier miljard kandidaten. Voor grote woordlengten is er dus nood aan een efficiënte procedure om de grootste prop ratio te bepalen voor een factor.

We hebben de verschilpropagatie voor cyclische vermenigvuldiging grondig onderzocht. Door gebruik te maken van de translatie-invariantie en de wisselwerking tussen verschil modulo $2^n - 1$ en bitsgewijs verschil zijn we gekomen tot de formulering van een aantal bruikbare eigenschappen. De grootste prop ratio blijkt overeen te stemmen met het getal dat een bepaald zogenaamd *ternair gewicht* minimaliseert. We realiseren ons dat de meest ongunstige correlatie-eigenschappen in feite even belangrijk, zoniet belangrijker zijn. De efficiënte bepaling van de grootste nontriviale correlatie voor grote woordlengten is echter nog een open probleem.

Cyclische vermenigvuldiging op 32-bit woorden is de belangrijkste bewerking in ons blokcijfer MMB. Dit blokcijfer heeft zowel een bloklengte als een sleutellengte van 128 bits en leent zich goed tot software-implementaties. Door onze ontwerpkeuzen te baseren op worst-case gedrag vertoont MMB niet het type zwakheden dat aanwezig is in FFT-Hash en IDEA.

11 Besluiten

De relevantie van dit werk situeert zich op twee vlakken. Enerzijds is er de kritiek op bestaande ontwerppraktijken en zienswijzen en een alternatieve aanpak die zich hier tegen afzet. Anderzijds zijn er de eigen bijdragen in de vorm van nieuwe aanvallen, concrete ontwerpen en nieuwe formalismen.

Het ideaal van bewijsbare veiligheid wordt verworpen en vervangen door de erkenning van de rol van de cryptografische *claim*. Deze claim heeft een dubbele functie: een uitdaging voor cryptanalysten en een specificatie voor systeemontwerpers en gebruikers. De reductionistische aanpak wordt verworpen en vervangen door een aanpak waarbij het ontwerp hoofdzakelijk gestuurd wordt door de studie van de correlatie en verschilpropagatie. “Fundamenteel onderzoek” naar eigenschappen van Booleaanse functies en substitutietabellen wordt verworpen ten koste van een structurele aanpak waarbij de criteria voor deze componenten voortvloeien uit een concrete ontwerpstrategie ingegeven door LC en DC. Deze strategie is gebaseerd op helderheid en maakt veelvuldig gebruik van symmetrie. In dit kader stellen we translatie-invariante transformaties voor als een nieuwe belangrijke klasse van basiscomponenten voor cryptografische schema's. In blokcijferontwerp is de klassieke Feistelstructuur vervangen door een nieuwe structuur, en synchrone stroomcijfers en hashfuncties zijn samengebracht in één module.

Onze belangrijkste concrete bijdragen zijn:

- een éénduidige classificatie van encryptieschema's en cijfers,
- de nieuwe concepten K-veilig en hermetisch en hun implicaties,
- de cryptografische eindige toestandsmachine in het ontwerp van blokcijfers en stroom/hash modules,
- een nieuw formalisme voor de beschrijving van verschilpropagatie en correlatie, met het introduceren van correlatiematrices, prop ratio's en het aantonen van diverse onderlinge verbanden,
- de ontwerpstrategie van het brede spoor,
- de ontdekking van nieuwe klassen omkeerbare translatie-invariante transformaties,
- de Hamminggewicht-distributietabel en het vertakkingsgetal van lineaire afbeeldingen,
- de beschrijving van verschilpropagatie en correlatie in de transformatie χ,
- een nieuwe blokcijferstructuur waarin differentiële en lineaire sporen aan dezelfde voorwaarden onderworpen zijn en de concrete ontwerpen 3-WAY en BASEKING,
- het ontwerp en de chip-implementatie van de stroom/hash module SUBTERRANEAN,
- het ontwerp van de efficiënte stroom/hash module STEPRIGHTUP,
- de weerlegging van een aantal misvattingen met betrekking tot het ontwerp van zelf-synchroniserende stroomcijfers,
- onze ontwerpstrategie voor zelf-synchroniserende stroomcijfers, inclusief de introductie van voorwaardelijk complementerende doorschuifregisters en een concreet voorbeeldontwerp,
- het concept van hersynchronisatie-aanvallen op synchrone stroomcijfers, inclusief de beschrijving van vier concrete aanvallen,
- de constructie van botsingen voor FFT-Hash en de CA hashfunctie van Ivan Damgård,
- het aantonen van de beperkingen van de Even-Mansour blokcijfers,
- het aantonen van de klassen zwakke sleutels voor het blokcijfer IDEA,
- het onderzoek naar de verschilpropagatie van cyclische vermenigvuldiging,

- het ontwerp van het blokcijfer MMB.

We besluiten met enkele aanwijzingen voor verder onderzoek:

- cryptanalyse van de specifieke ontwerpen beschreven in deze thesis,
- ontwerp en analyse van efficiënte stroom/hash modules voor software zoals StepRightUp,
- verbeteren van de efficiëntie van de algoritmen om de kritische differentiële en lineaire sporen te vinden voor blokcijfers zoals 3-Way en BaseKing,
- onderzoek van de correlatie-eigenschappen van cyclische vermenigvuldiging,
- ontwerp van n-bit zelf-synchroniserende stroomcijfers met n groter dan 1,
- toepassen en uitbreiden van de theorie van correlatiematrices.