Scalable Symmetric Block Ciphers Based on Group Bases

Dissertation

zur Erlangung des akademischen Grades
Doktor-Ingenieur
vorgelegt an der
Universität GH Essen
im Fachbereich Maschinenwesen

von

Valér Čanda

geboren in Rožňava,
Slowakische Republik

Tag der mündlichen Prüfung: 18. Juni 2001

Vorsitzender der Prüfungskommission:	Prof. Dr.-Ing. Rudolf Tracht
Gutachter:	Prof. Dr.-Ing. A. J. Han Vinck
	Prof. Dr. Tran van Trung

Kurzfassung

Die Sicherheit und die Effizienz einer Blockchiffre sind zweifellos die wichtigsten Kriterien für Beurteilung ihrer Qualität. Außer diesen elementaren Eigenschaften werden aber auch Skalierbarkeit und Einstellbarkeit als besonders wichtig und wünschenswert betrachtet, nicht nur weil sie es ermöglichen, daß die Chiffre auf verschiedensten Plattformen optimal betrieben wird, sonder auch, weil sie die nötige Sicherheitsreserve für ihre zukünftige Benutzung garantieren. Einer der möglichen Ansätze zur Konstruktion von skalierbaren und einstellbaren Blockchiffren basiert auf den Gruppenbasen.

Eine Gruppenbasis ist ein mathematisches Objekt das es ermöglicht, eine Permutation aus einer bestimmten Gruppe in einen eindeutigen Koordinatenvektor zu zerlegen. Das Verschlüsselungsprinzip der untersuchten Chiffren besteht in "Übersetzung" einer Permutation zwischen zwei zufällig gewählten Gruppenbasen β_1 und β_2. Solche Bijektion auf der Menge der Permutationen kann als eine Verschlüsselungsfunktion eingesetzt werden, weil sie durch β_1 und β_2 eindeutig bestimmt ist, und weil sie ohne diese Basen nicht wiederhergestellt werden kann. Das Gruppenbasenpaar (β_1, β_2) spielt also die Rolle eines geheimen Schlüssels. Dieser Ansatz ist aus mathematischer Sicht sehr direkt und einfach, und die resultierende Chiffren besitzen mehrere wünschenswerten Eigenschaften, wie z.B. einen außergewöhnlich großen Schlüsselraum, skalierbare Block- und Schlussellänge, usw. Einige bisher unbeantwortete Fragen bezüglich Sicherheit, Effizienz und Implementationstauglichkeit dieser Kryptosysteme - insbesondere des neuesten Repräsentanten TST - stellen ein interessantes Forschungsthema dar.

Diese Dissertation untersucht die verfügbaren auf den Gruppenbasen basierenden Kryptosysteme und präsentiert zwei neue verbesserten Designs. Nach einer allgemeinen Einführung in die Problematik und einem Überblick der wichtigen theoretischen Grundlagen folgt der Kern dieser Arbeit, in dem die neuen Ergebnisse in drei logischen Teilen präsentiert werden.

Im ersten Teil wird das neueste auf den Gruppenbasen basierende Kryptosystem TST im Detail analysiert. Dabei wird die Effizienz der zwei möglichen Permutationsdarstellungen - der sogenannten kompakten und der kartesischen Darstellung - verglichen, eine effiziente Implementierung der Schlüsselgenerierung diskutiert, und die substantiellen Charakteristiken wie Durchsatz, Speicherbedarf und Initialisierungsverzögerung gemessen. Außerdem wird eine Sicherheitsanalyse durchgeführt, bei der die statistischen Eigenschaften des Kryptosystems untersucht werden und ein kryptographischer Angriff konstruiert wird. Die Ergebnisse unserer

Analyse zeigen, dass eine Softwareimplementierung von TST in allen Aspekten über wesentlich weniger Leistungsfähigkeit verfügt als die anderen modernen Blockchiffren. Obendrein weist TST beträchtliche statistische Defekte auf, und die vereinfachten TST Versionen können sogar mit einer effizienten kryptanalytischen Attacke vollständig entschlüsselt werden.

Eine mögliche Lösung dieser bei TST auftretenden Probleme wird in dem zweiten Teil dieser Arbeit präsentiert. Mit Hilfe einer Erweiterung der Gruppenbasen kann eine starke Diffusion der Faktorisierungs- und Kompositionsoperationen garantiert werden. Hierbei wird vor jedem Faktorisierungsschritt eine einfache Transformation durchgeführt, die jede Permutationskoordinate von allen Bits der zu faktorisierenden Permutation abhängig macht. Unsere Tests haben bestätigt, daß die statistischen Eigenschaften dieser erweiterten TST Version beträchtlich besser sind. Dank den besseren Diffusionseigenschaften kann sogar eine einfachere Trägergruppe eingesetzt werden, mit der der Speicherbedarf reduziert und der Durchsatz erhöht werden kann. Das bedeutet, daß nicht nur die Sicherheit sondern auch die Effizienz des neuen Kryptosystems verbessert wurden.

In dem letzten Teil dieser Arbeit wird eine iterative Version von TST vorgestellt. Der elementare Baustein dieses Chiffrendesigns entspricht einem Faktorisierungsschritt in einer Gruppenbasis, statt einer echten Faktorisierung wird jedoch eine konstante Funktion mehrmals iterativ angewandt. Die wesentlichen Vorteile dieses Ansatzes gegenüber TST sind deutlich reduzierter Speicherbedarf, erhöhter Durchsatz, und verbesserte Flexibilität. Die Block- und Schlüssellänge sind, genau wie bei TST, frei wählbar, außerdem ermöglicht das neue Kryptosystem eine freie Justierung zwischen der Sicherheit, der Geschwindigkeit und dem Speicherbedarf. Mit der entsprechenden Anzahl von Runden kann die Chiffre hervorragende Sicherheit bieten, was sowohl unsere Kryptanalyse, als auch die statistischen Tests bestätigt haben.

Preface

This thesis describes the results of my research carried out at the Institute for Experimental Mathematics, University of Essen, Germany, in the period from October 1998 till June 2001. The research has been funded by the German Research foundation DFG as a part of the graduate program "Theoretical and Experimental Methods of Pure Mathematics". There are many people I would like to thank for their contribution to the final quality of this thesis.

First of all, I would like to thank my supervisor Prof. Tran van Trung for his confidence, advices, and the numerous discussions of my research results. My research work would have been much harder without his friendly support and guidance.

I would further like to thank Prof. Han Vinck for his all-round support and the pleasant atmosphere he was able to create in our working group. Special thank goes to Prof. Spyros Magliveras for his hospitality and support during my stay at the University of Nebraska in Lincoln as well as for his valuable comments on my research work.

Finally, I would like to thank my colleagues from the Digital Communications Group at IEM for their support, friendship, and their contribution to the pleasant working environment in our group. Special thanks in this respect go to Jürgen Häring, Peter Gober, Christoph Haslach, Lenka Fibíková, Chaichana Mitrpant, and Sosina Martirosyan.

Essen, June 2001

Valér Čanda

Contents

1 Introduction **1**

2 Preliminaries **5**
2.1 Group Theory 5
2.1.1 Basic Terminology 5
2.1.2 Group Bases 7
2.1.3 Binary Permutation Groups 10
2.2 Randomness 14
2.2.1 Evaluation of Randomness Quality 15
2.3 Block Ciphers 17
2.3.1 Evaluation of Encryption Quality 19

3 Current Symmetric Cryptosystems Based on Group Bases **23**
3.1 Encryption Principle 23
3.2 Cryptosystem PGM 26
3.2.1 Key Generation 26
3.2.2 Properties 28
3.3 Cryptosystem TST 29
3.3.1 Carrier Group 29
3.3.2 Key Generation 31
3.3.3 Properties 33
3.4 Summary and Open Problems 34

4 Analysis of Cryptosystem TST **37**
4.1 Efficiency 37
4.1.1 Key Generation 38
4.1.2 Throughput 44
4.1.3 Memory Requirements 44
4.2 Security 46
4.2.1 Generic Statistical Evaluation 46
4.2.2 White Box Analysis 50
4.3 Summary 59

5 A New Cryptosystem Based on Extended Group Bases 61
5.1 Extended Group Bases . . . 61
5.2 Cryptosystem TST' . . . 64
5.2.1 Carrier Group . . . 64
5.2.2 Family of Transformations $\mathcal{T}$ . . . 65
5.3 Efficiency . . . 68
5.3.1 Key Generation . . . 68
5.3.2 Throughput . . . 69
5.3.3 Memory Requirements . . . 70
5.4 Security . . . 71
5.4.1 Generic Statistical Evaluation . . . 71
5.4.2 White Box Analysis . . . 73
5.5 Summary . . . 75

6 A New Iterative Cryptosystem 77
6.1 Motivation . . . 77
6.2 Iterative TST Design . . . 79
6.2.1 Round Function . . . 79
6.2.2 Implementation Issues . . . 84
6.2.3 Iterative TST . . . 89
6.2.4 Secure Number of Rounds . . . 90
6.3 Efficiency . . . 92
6.3.1 Key Generation . . . 92
6.3.2 Throughput . . . 92
6.3.3 Memory Requirements . . . 94
6.4 Security . . . 94
6.4.1 Generic Statistical Evaluation . . . 95
6.4.2 White Box Analysis . . . 97
6.5 Summary . . . 101

7 Conclusions 103

A Basic Empirical Tests 107

B The DieHard Test Battery 109

List of Notations 115

Bibliography 119

Chapter 1

Introduction

People always tried to protect their secrets. Two parties that wanted to exchange private messages in the past centuries had either to meet in person or to *trust* a messenger. As long as the carrier behaved nicely, everything went fine. But once he got curious or corrupt, nobody could stop him breaking the seal and reading the message. People were aware of that threat and started very soon using simple forms of steganography and cryptography to ensure the confidentiality of their messages. *Steganography* is the art of communicating in a way which hides the existence of the communication. Typical example of such a communication is usage of an invisible ink, or creation of miniature dots under those letters of an innocent looking text which belong to the actual message. When steganography is used, an unauthorized person should not even recognize that there is a secret message being sent. On the other hand, *cryptography* is the art of protecting information by transforming it into an unreadable format. The presence of a secret is not hidden anymore, but without knowing the secret *key* nobody can understand the message. A simple cryptographic protection is, for example, usage of a secret alphabet (e.g. hieroglyphs), so that a person not knowing the alphabet does not understand the message. Such a transformation of the information is called *encryption* and the inverse transformation is said to be the *decryption*. Cryptography is principally stronger than steganography, because the complete security of steganography resides in the fact that an adversary does not know that (or how) a secret information is hidden. In cryptography the complete security resides in the key, so even if an adversary knows the exact method used, he is not able to recover the message without knowing the key.

The classical simple data protection methods became insufficient with the advancing technological progress. Nowadays, in the age of computers, networks, and wireless communication, the volume and the worth of the confidential information is higher than ever. The technical possibilities of an adversary increased dramatically as well. Large amounts of information can be reproduced, stored, transmitted (and therefore abused) in just few seconds, so the protection of sensitive data from unauthorized access becomes extremely important. To protect the kind of information whose abuse might cause immense financial losses to a company or whose confidentiality guarantees the national security of a state one needs cryptography

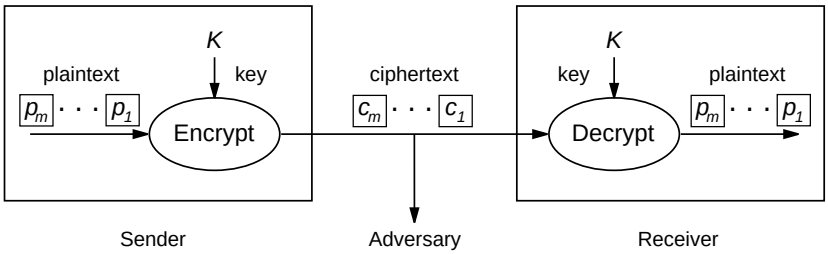


Figure 1.1: Symmetric encryption

which is founded on the most recent scientific knowledge. Beside the protection of top secret information, there are many everyday applications involving cryptography. When we use a mobile phone or a teller machine, when we surf the Internet, watch pay-tv, or open our car remotely, there is always a cryptographic application in the background. In addition to the *confidentiality*, cryptography provides for several other information security objectives, like *integrity*, *authenticity*, *non-repudiation*, etc. Modern cryptography as a public scientific discipline is still rather young. The complete cryptographic research was held secret for many years, and the first scientific papers dealing with information security appeared in the late forties. An intensive and systematic (public) cryptographic research started only in the early seventies. Cryptography is nowadays a very dynamic and rapidly developing scientific discipline.

According to the principle of work, cryptography can be divided into two major areas - *symmetric cryptography* (also called secret-key cryptography), and *public-key cryptography*. In symmetric cryptography both communicating parties know the same secret key which enables them to encrypt and decrypt the messages. When using public-key cryptography, all communicating parties possess two different keys - a so-called *public key*, and a *private key*. The first one can only be used for encrypting a message, and the second one only for decrypting it. Figure 1.1 displays a secure communication by means of symmetric cryptography. Both parties had agreed on a secret key K beforehand. The sender, who wants to deliver a confidential message to the receiver, uses K as an input parameter of a symmetric encryption algorithm which transforms the readable message (*plaintext*) into an unreadable format (*ciphertext*). The encrypted message is transmitted over an insecure channel (e.g. a telephone line or a computer network) and is received by the second party. The receiver transforms the ciphertext into the original message by using the decryption algorithm and providing K as the input parameter. An adversary who is listening on the channel is able to eavesdrop the ciphertext, but because he does not know the secret value K, he is not able to perform the decryption and, hence, can not understand the message. The sender and receiver can securely exchange an arbitrary number of messages in this way. The set consisting of an encryption algorithm, the corresponding decryption algorithm, and the rules regarding formats of plaintexts, ciphertexts, and keys is said to be a *symmetric cryptosystem* or a *symmetric cipher*. Depending on the way a symmetric cipher processes the message we distinguish *block ciphers*, which split a message into blocks of equal length and encrypt each block

separately, and *stream ciphers* which encrypt the complete plaintext in one piece. In this work we will focus our attention on symmetric block ciphers.

There are several desirable properties which a good block cipher should possess. The most important ones are:

- *security* - which means that without knowing the key it should be impossible (or extremely hard) to obtain any information about the plaintext,
- *efficiency* - meaning that both encryption and decryption procedures should be fast and should not consume much memory,
- *adjustability* - meaning that it should be possible to perform a tradeoff between the provided security level, the encryption speed, and the allocated memory,
- *scalability* - meaning that both block length and key length should be variable,
- *theoretical foundations* - meaning that the encryption (and decryption) procedure should base on some mathematical principles.

The first two properties are essential for every cipher. If a cipher is not secure, there is no reason for using it, and if it is not efficient it usually can not be used for practical reasons. The theoretical foundations play an important role in the task of security estimation, because they make it possible to understand and analyze (and therefore trust) the cipher. The adjustability and scalability enable us to adapt the cipher for optimal use in various environments and applications (e.g. a small smart card vs. a powerful supercomputer), and they give us the flexibility and a security margin which might be necessary in the future. For example, when a breakthrough in computer technology occurs, making available computers million times faster than before, we will not have to invent a completely new cipher, we just start using a longer key. According to the degree of their scalability, block ciphers can be classified into the following four categories:

- *strongly scalable* - ciphers supporting *any* combination of the block length n and the key length k by design,
- *fully scalable* - ciphers with some minor restriction regarding the format of n and k but without upper limits for them,
- *partially scalable* - ciphers supporting only a finite set of possible values for n and k,
- *not scalable* - ciphers where n and k are fixed by design.

Unfortunately, adjustability and scalability are still not common in present block cipher designs. The first modern symmetric block ciphers, e.g. DES [Uni77], IDEA [LM91], SAFER [Mas94b, Mas94a], or Blowfish [Sch94], were not scalable at all. The development effort for the recent cryptographic standard AES [Nat97] has confirmed that the future direction of block cipher design will be moving more towards full

scalability. In particular, the AES candidate ciphers were *required* to support three different key lengths. Even though some of the candidate ciphers supported even multiple block lengths, none of them provided full scalability. There is still a need for well founded block cipher designs that are fully scalable and adjustable. As the design of secure and efficient block ciphers is a very complex task involving extensive theory from several areas of mathematics and computer science, considerable work has to be done until new, well founded, scalable, and adjustable designs become widespread.

This thesis is concerned with scalable block cipher designs based on group bases. A *group basis*, whose notion was introduced in the late seventies, is a mathematical object which can be used for constructing both symmetric and public-key cryptosystems. The especial attractiveness of this structure resides in its full scalability and the strong mathematical foundations. In this work we summarize the state of the art in symmetric cryptography based on group bases, analyze a recent encryption scheme, and present some new, more efficient designs. Our analysis of the cryptosystems does not only involve the security aspects, we also discuss the practical implementation of the cryptosystems including the optimal data representation and suitable algorithms, and we measure the essential efficiency parameters like memory requirements, key setup delay, and throughput. The thesis is structured as follows:

- *Chapter 2* briefly introduces the theory which is utilized in the thesis. The fundamental concepts and terminology from group theory, randomness theory, and block cipher theory are summarized here.
- *Chapter 3* describes the current symmetric cryptosystems based on group bases. The encryption-decryption principles are explained and the properties of the ciphers are discussed.
- *Chapter 4* analyzes the efficiency and security of cryptosystem TST in detail. At the end of the chapter the advantages and disadvantages of the cryptosystem are discussed.
- *Chapter 5* introduces the notion of extended group bases and presents a new scalable cryptosystem based on this idea. The efficiency and security of the new cryptosystem is analyzed in detail and compared with cryptosystem TST.
- *Chapter 6* presents a simplification of the TST encryption idea, and describes the design of a new, fully scalable, iterative cryptosystem. Again, the efficiency and security of the new cryptosystem are analyzed and compared with the previous two designs.
- *Chapter 7* summarizes the achieved results and presents some open problems.
- *Appendix A* describes the basic empirical randomness tests for binary sequences.
- *Appendix B* describes the battery of statistical tests which have been used for our generic cipher evaluation.

Chapter 2

Preliminaries

This chapter summarizes the theoretical background which is utilized in the thesis. We introduce the necessary terminology and notations and describe some techniques used in the field.

The first section introduces basic definitions from group theory with impact on the notion of *group basis* which is substantial for our thesis. For more detailed introduction to the basic concepts of group theory and permutation groups the reader is referred to [Big89, Wie64, DM96, But91]. A deeper introduction to group bases and their properties can be found in [Mag86, Mem89, MM92].

The second section explains the idea of randomness and pseudorandomness and gives an overview of the available randomness tests. More theory on this topic can be found in [Knu97, Chap. 3], [Cal94] and [MVV97, Chap. 5].

The third section briefly formalizes the notion of a block cipher and describes the methods for evaluation of block cipher quality. We present the idea of *generic cipher analysis* as well as *white box cryptanalysis*. More discussion on cipher evaluation can be found in many scientific papers. A generic cipher analysis is given, for example, in [SB00] and [Hey97]. An introduction to the common attacks on block ciphers can be found in [Sti95, Sch96] and some deeper description of analysis techniques can be found, for instance, in [BS90, Mat94, LMM92, LH94, BBS99].

2.1 Group Theory

The idea of group basis is based on the theory of permutation groups. In what follows, we will briefly summarize the basic definitions.

2.1.1 Basic Terminology

Definition 2.1.1 Permutation
Let X be a non-empty finite set. A bijective mapping $p : X \longrightarrow X$ is called a permutation of X.

X is usually a contiguous set of integers, typically $\mathbb{N}_n = \{1, 2, \ldots, n\}$ or $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$. We will preferably use the set $\mathbb{Z}_n$ because it is more suitable for a computer representation. There exist exactly $n!$ different permutations for any set of n elements. The set of all permutations of $\mathbb{Z}_n$ is denoted by S_n.

A permutation can be written in several notations. The simplest one is the *cartesian notation*.

Definition 2.1.2 Cartesian Notation of a Permutation
Let p be a permutation of $\mathbb{Z}_n$. The cartesian notation of p is the vector $[p(0), p(1), \ldots, p(n-1)]$.

For example, the permutation p of $\mathbb{Z}_5$ given by values $p(0) = 2$, $p(1) = 4$, $p(2) = 3$, $p(3) = 0$ and $p(4) = 1$ can be written in cartesian notation as $[2, 4, 3, 0, 1]$. Any two permutations can be composed into a new permutation as follows.

Definition 2.1.3 Composition of Permutations
Let a and b be two permutations of $\mathbb{Z}_n$. The product of a and b denoted by $c = a * b$ is the permutation c of $\mathbb{Z}_n$ such that $c(i) = b(a(i))$ for $i = 0 \,..\, n-1$.

Notation $a * b$ is usually shortened to ab. The result of the operation $a * b$ is said to be the *product* of a and b. The set of all permutations of $\mathbb{Z}_n$ forms a *group* with respect to the composition operation $*$.

Definition 2.1.4 Group
A group $\mathcal{G}$ is an ordered pair $(G, *)$ where G is a set and $*$ is a binary operation on G which satisfies the following four axioms:

- *For any $x, y \in G$ it holds that $x * y \in G$. (Closure)*
- *For any $x, y, z \in G$ it holds that $(x * y) * z = x * (y * z)$. (Associativity)*
- *There is an element $e \in G$ such that $e * x = x * e = x$ for all $x \in G$. (Existence of identity)*
- *For all $x \in G$ there is an $x' \in G$ such that $x * x' = x' * x = e$. (Existence of inverse)*

The element e is called *identity* and is often denoted by id or 1. The element x' usually denoted by x^{-1} is said to be an *inverse* element for x. The number of elements in the set G is denoted by $|G|$ and is known as the *order* of $\mathcal{G}$. The group $\mathcal{S}_n = (S_n, *)$ of all permutations of n elements is known as the *symmetric group* and its order is $n!$.

Definition 2.1.5 Abelian Group (Commutative Group)
Let $\mathcal{G} = (G, *)$ be a group. $\mathcal{G}$ is called abelian when $xy = yx$ for every $x, y \in G$.

Definition 2.1.6 Subgroup
Let $\mathcal{G} = (G, *)$ be a group and let H be a non-empty subset of G. If $\mathcal{H} = (H, *)$ is a group under the operation $*$ of $\mathcal{G}$, then $\mathcal{H}$ is called a subgroup of $\mathcal{G}$ and we write $\mathcal{H} \leq \mathcal{G}$.

If a set G forms a group with respect to a particular operation whose identity is clear from the context (e.g. permutation composition when speaking about permutation groups), the notation $\mathcal{G} = (G, *)$ is often simplified to G. In this case one speaks of "group G" instead of using the accurate description "group $\mathcal{G}$". We will prefer the simplified notation as well.

The last few definitions of this section will be utilized in the definition of transversal group basis in the next section.

Definition 2.1.7 Right Coset
Let H be a subgroup of a group G. The right coset Hg of H with respect to an element $g \in G$ is defined to be the set obtained by multiplying each element of H by g. That is: $Hg = \{hg \mid h \in H\}$.

The element g is said to be a *coset representative* of Hg. Any two cosets Hg_1 and Hg_2 are either equal or disjoint, i.e. either $Hg_1 = Hg_2$ or $Hg_1 \cap Hg_2 = \emptyset$. Thus, there exists a unique partitioning of a group G into right cosets of any fixed subgroup $H \leq G$.

Definition 2.1.8 Right Transversal
Let H be a subgroup of a group G. A right transversal of H in G is a complete set of right coset representatives, i.e. a set $\{g_1, g_2, \ldots, g_k\}$, $g_i \in G$ such that $Hg_1 \cup Hg_2 \cup \cdots \cup Hg_k = G$ and $Hg_i \neq Hg_j$ for every $i \neq j$.

For more definitions and examples on group theory the reader is referred to [Big89, DM96].

2.1.2 Group Bases

A *group basis* (also called *logarithmic signature*) is a fundamental data structure for finite permutation groups. The notion was introduced by [Mag86].

Definition 2.1.9 Group Basis (Logarithmic Signature)
Let G be a finite group. A group basis for G is an ordered collection $\beta = (B_0, \ldots, B_{w-1})$ of ordered subsets $B_i = (b_{i,0}, \ldots, b_{i,r_i-1})$ of G, such that each element $p \in G$ can be expressed uniquely as a product of the form

$$p = b_{0,x_0} \cdot b_{1,x_1} \cdots b_{w-1,x_w-1}, \quad b_{i,x_i} \in B_i.$$

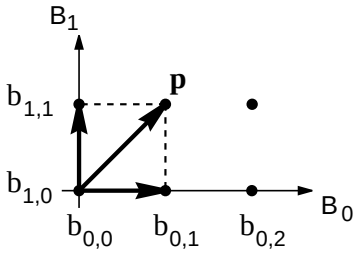


Figure 2.1: Basis as a coordinate system

The B_i are called the *blocks* of β, the vector of block lengths $r = (r_0, r_1, \ldots, r_{w-1})$ is called the *type* of β and the number w is the *dimension* of β. Each $p \in G$ corresponds to a unique *index vector* $x = (x_0, x_1, \ldots, x_{w-1})$, where $x_i \in \mathbb{Z}_{r_i}$. The space of all index vectors is $X = \mathbb{Z}_{r_0} \times \mathbb{Z}_{r_1} \times \cdots \times \mathbb{Z}_{r_{w-1}}$. The index set X has cardinality $|X| = r_0 \cdot r_1 \cdots r_{w-1} = |G|$.

A basis β describes a bijective mapping $\tilde{\beta} : X \longrightarrow G$ as follows:

$$\tilde{\beta}(x) = \tilde{\beta}(x_0, x_1, \ldots, x_{w-1}) = b_{0,x_0} \cdot b_{1,x_1} \cdots b_{w-1,x_{w-1}} = p.$$

When computing $p = \tilde{\beta}(x)$, we say that p is *composed* from factors b_{i,x_i}. Computing the inverse function $x = \tilde{\beta}^{-1}(p)$ is called *factorization* of p with respect to β.

We will demonstrate the idea of group basis in a simple example. Example 2.1.1 shows a 2-dimensional group basis for the permutation group S_3 with 6 permutations. The permutation $[1, 0, 2] \in S_3$ can be factorized into coordinates $(1, 1)$ with respect to β.

Example 2.1.1 A Group Basis for $G = S_3$

$G = \{[0,1,2],\ [0,2,1],\ [1,0,2],\ [1,2,0],\ [2,0,1],\ [2,1,0]\}$
$w = 2$,
$r = (3, 2)$,

$p = [1,0,2] = [2,0,1] * [0,2,1] = b_{0,1} * b_{1,1}$,
thus, the index vector is $x = (1,1)$, *and we write*
$\tilde{\beta}(1,1) = [1,0,2]$ *resp.* $\tilde{\beta}^{-1}([1,0,2]) = (1,1)$

β

B_1	$[0,2,1]$	$b_{1,1}$
	$[0,1,2]$	$b_{1,0}$
B_0	$[1,2,0]$	$b_{0,2}$
	$[2,0,1]$	$b_{0,1}$
	$[0,1,2]$	$b_{0,0}$

One can think of a group basis as a kind of w-dimensional discrete coordinate system as illustrated in Figure 2.1. The six permutations of S_3 might be seen as points in a 2-dimensional space. Any one of the six points can be expressed as a unique sum[1] of two points - one from each axis. The two axes, the first with three, and the second with two points, correspond to the two blocks of β.

The crucial property of group bases from the cryptographic point of view is the fact that *there is an enormous number of different group bases for a given group.*

[1] Addition of points in this discrete geometry is defined by means of vectors as:
$(x_1, y_1) + (x_2, y_2) = (x_1 + x_2 \ mod\ 3,\ y_1 + y_2 \ mod\ 2)$.

We denote the set of all bases that generate G by $\mathcal{B}_G$. For example, the tiny group S_3 from our example has 924 different bases. 6! of them are one-dimensional bases of type (6) and $2!.2^3.3! + 3!.3^2.2!$ of them are two-dimensional bases of types (2,3) and (3,2).

According to the complexity of factorization algorithm, the bases can be classified as *wild*, *tame* or *supertame*.

Definition 2.1.10 Tame, Supertame and Wild Group Basis
Let G be a group of order n and let β be a group basis of G.

- *β is called tame if the factorization with respect to β can be achieved in time polynomial in n.*
- *β is called supertame if the factorization with respect to β can be performed in time $O(n^2)$.*
- *β is called wild if it is not tame.*

The elementary operation (i.e. the unit of the measure of complexity) is considered to be a memory lookup.

Supertame bases are the most suitable for construction of symmetric cryptosystems because of their efficient factorization. On the other hand, the wild bases can be utilized in the design of one-way functions for public-key cryptosystems [MSvT00].

A special class of group bases - the so called *transversal* group bases - contains only bases which are guaranteed to be tame. (For proof see [Hor98, Sec. 2.2].)

Definition 2.1.11 Transversal Group Basis
Let $\Gamma : G_0 < G_1 < \cdots < G_{w-1} = G$ be a chain of subgroups and $\beta = (B_0, \ldots B_{w-1})$ a group basis of G such that $B_0 = G_0$ and block $B_i = (b_{i,0}, b_{i,1}, \ldots, b_{i,r_i-1})$ is a right transversal of G_{i-1} in G_i for all $1 \leq i \leq w-1$, i.e. $G_i = G_{i-1}B_i$. Basis β is then called transversal with respect to Γ.

We will denote the set of all transversal group bases for a group G by $\mathcal{B}^*_G$. A supertame transversal basis for S_n of a special form is called the *canonical basis* α. This basis is of particular interest for the generation of random transversal group bases for S_n.

Definition 2.1.12 Canonical Group Basis for S_n
The group basis $\alpha = (A_0, A_1, \ldots, A_{n-2})$ of type $r = (2, 3, \ldots, n)$ where every block A_i contains only permutations $a_{i,j}$, $j = 0$ .. $i+1$, such that $a_{i,j}(i+1) = j$, $a_{i,j}(j) = i+1$ and $a_{i,j}(x) = x$ for all $x \neq i+1, j$, is called the canonical basis for S_n. The one-element set $k_i = \{i+1\}$ is said to be the key-letter set in A_i.

2.1.3 Binary Permutation Groups

A permutation group is said to be a *binary group* (or a *2-group*) if its order is a power of 2. The S_n, for instance, is in general not a binary group because $|S_n| = n! \neq 2^m$ for any $n > 2$. A special class of 2-groups are the *elementary 2-groups*.

Definition 2.1.13 Elementary 2-Group
A 2-group G is called elementary when $x * x = id$ for all $x \in G$.

The simplest binary group available is the *Abelian* (Definition 2.1.5) *elementary 2-group* denoted by $\mathbb{Z}_2^n$. This group consists of permutations of $2n$ symbols in form $p = [a_0, a_1, \ldots, a_{2n-1}]$ such that for every pair of symbols a_{2k}, a_{2k+1}, $k \in \mathbb{Z}_n$ either $a_{2k} = 2k$ and $a_{2k+1} = 2k+1$, or else $a_{2k} = 2k+1$ and $a_{2k+1} = 2k$. Every permutation $p \in \mathbb{Z}_2^n$ can be represented very effectively with the so-called *compact representation*.

Definition 2.1.14 Compact Representation of Elements of $\mathbb{Z}_2^n$
Let $p = [a_0, a_1, \ldots, a_{2n-1}]$, be a permutation from $\mathbb{Z}_2^n$. The binary vector $x = (x_0, x_1, \ldots, x_{n-1})$, $x_i \in \{0, 1\}$ such that $x_i = 0$ if and only if $a_{2i} = 2i$, otherwise $x_i = 1$, is called the compact representation of p.

In other words, the i-th bit of the compact representation indicates, whether or not p affects a swap on positions a_{2i} and a_{2i+1}. The compact representation is *optimal* in terms of memory requirements because it uniquely represents any of the 2^n permutations of $\mathbb{Z}_2^n$ by exactly n bits.

Another benefit of the compact representation is the fact that it enables a direct and very efficient composition of permutations. Let $\oplus$ denote a binary XOR of two values. If a binary vector x_1 is the compact representation of $p_1 \in \mathbb{Z}_2^n$ and x_2 is the compact representation of $p_2 \in \mathbb{Z}_2^n$, then the vector $x_1 \oplus x_2$ is the compact representation of the product $p_1 p_2$.

Example 2.1.2 Group $\mathbb{Z}_2^3$ and Its Representations

$\mathbb{Z}_2^3 =$ *(cartesian representation)*	*(compact representation)*
$[0, 1, 2, 3, 4, 5]$	000
$[1, 0, 2, 3, 4, 5]$	100
$[0, 1, 3, 2, 4, 5]$	010
$[1, 0, 3, 2, 4, 5]$	110
$[0, 1, 2, 3, 5, 4]$	001
$[1, 0, 2, 3, 5, 4]$	101
$[0, 1, 3, 2, 5, 4]$	011
$[1, 0, 3, 2, 5, 4]$	111

Example of the group operation
in cartesian rep. $[1, 0, 2, 3, 5, 4] * [0, 1, 3, 2, 5, 4] = [1, 0, 3, 2, 4, 5]$
in compact rep. $101 \oplus 011 = 110$

A systematic n-dimensional group basis of $\mathbb{Z}_2^n$ - so called *canonical basis* - is of particular importance in the design of symmetric cryptosystems.

Definition 2.1.15 Canonical Basis for $\mathbb{Z}_2^n$
The canonical basis α for $\mathbb{Z}_2^n$ has the form $\alpha = (A_0, A_1, \ldots, A_{n-1})$ where for every $i = 0 \,..\, n-1$ the block A_i consists of the following two permutations (written in the compact representation) $a_{i,0} = (\underbrace{0 \ldots 0}_{n \text{ times}})$ and $a_{i,1} = (\underbrace{0 \ldots 0}_{i-1 \text{ times}} \; 1 \; \underbrace{0 \ldots 0}_{n-i \text{ times}})$.

The first element of every block A_i is the identity and the second element is the permutation performing a single swap on the positions $2i$ and $2i+1$. The one-element set $c_i = \{i\}$ is called the set of *key bit positions* for block A_i. It plays an important role during factorization with respect to α.

In contrast with $\mathbb{Z}_2^n$, the group $\mathcal{H}_s$ is the most complex 2-group. $\mathcal{H}_s$, known as the *Sylow 2-subgroup of S_n*, is the largest binary subgroup of the full symmetric group. When $n = 2^s$, the order of $\mathcal{H}_s$ is 2^{2^s-1}.

Definition 2.1.16 Sylow 2-subgroup $\mathcal{H}_s$ of the symmetric group S_n, $n = 2^s$.
The group $\mathcal{H}_s$ is defined recursively as follows:

- $\mathcal{H}_1 = \mathcal{T}_s$
- $\mathcal{H}_s = (\mathcal{H}_{s-1} \times \mathcal{H}_{s-1}) \cdot \mathcal{T}_s, \quad for \; s > 1.$

The group $\mathcal{T}_s = \{\iota, \tau_s\}$ consists of two permutations of 2^s elements. ι is the identity and τ_s is an involution which swaps the two halves $\{0, \ldots, 2^{s-1}-1\}$ with $\{2^{s-1}, \ldots, 2^s - 1\}$, each of length 2^{s-1}. For example $\mathcal{T}_1 = \{[0,1],[1,0]\}$, $\mathcal{T}_2 = \{[0,1,2,3],[2,3,0,1]\}$, etc. The product $(\mathcal{H}_{s-1} \times \mathcal{H}_{s-1}) \cdot \mathcal{T}_s$ is called the *wreath product* of $\mathcal{H}_{s-1}$ with $\mathcal{T}_s$. Example 2.1.3 presents the three smallest Sylow groups and their orders.

Example 2.1.3 $\mathcal{H}_s$ for $s = 1, 2, 3$

$\mathcal{H}_1 = \mathcal{T}_1 = \{[0,1],[1,0]\}$
$|\mathcal{H}_1| = 2^{2^1-1} = 2$

$\mathcal{H}_2 = (\mathcal{H}_1 \times \mathcal{H}_1) \cdot \mathcal{T}_2 = \{[0,1,2,3],[1,0,2,3],[0,1,3,2],[1,0,3,2],$
$[2,3,0,1],[2,3,1,0],[3,2,0,1],[3,2,1,0]\}$
$|\mathcal{H}_2| = 2^{2^2-1} = 8$

$\mathcal{H}_3 = (\mathcal{H}_2 \times \mathcal{H}_2) \cdot \mathcal{T}_3 = \{[0,1,2,3,4,5,6,7], \ldots, [7,6,5,4,3,2,1,0]\}$
$|\mathcal{H}_3| = 2^{2^3-1} = 128$

As for $\mathbb{Z}_2^n$, every $\mathcal{H}_s$ also has a unique canonical basis α_s. Each of the $2^s - 1$ blocks contains two permutations and has one key bit position $c_i = \{i\}$. α_s is constructed recursively as follows:

Example 2.1.4 Canonical Basis for $\mathcal{H}_s$, $s = 1, 2, 3, \ldots$

α_1 :

A_0	$[1,0]$ $[0,1]$

α_2 :

A_2	$[2,3,0,1]$ $[0,1,2,3]$
A_1	$[0,1,3,2]$ $[0,1,2,3]$
A_0	$[1,0,2,3]$ $[0,1,2,3]$

α_3 :

A_6	$[4,5,6,7,0,1,2,3]$ $[0,1,2,3,4,5,6,7]$
A_5	$[0,1,2,3,6,7,4,5]$ $[0,1,2,3,4,5,6,7]$
A_4	$[0,1,2,3,4,5,7,6]$ $[0,1,2,3,4,5,6,7]$
A_3	$[0,1,2,3,5,4,6,7]$ $[0,1,2,3,4,5,6,7]$
A_2	$[2,3,0,1,4,5,6,7]$ $[0,1,2,3,4,5,6,7]$
A_1	$[0,1,3,2,4,5,6,7]$ $[0,1,2,3,4,5,6,7]$
A_0	$[1,0,2,3,4,5,6,7]$ $[0,1,2,3,4,5,6,7]$

α_s :

$\hat{I}_{s-1}$	I_{s-1}
I_{s-1}	$\hat{I}_{s-1}$
J_{s-1}	$\hat{\alpha}_{s-1}$
α_{s-1}	$\hat{J}_{s-1}$

$I_s = \{0, 1, \ldots 2^s - 1\}$,
$\hat{I}_s = 2^s + I_s = \{2^s, 2^s + 1, \ldots 2^{s+1} - 1\}$
J_s denotes a $(2^{s+1} - 2) \times 2^s$ array each row of which is equal to I_s,
$\hat{J}_s = 2^s + J_s$.

The elements of $\mathcal{H}_s$ can also be written in compact representation which is optimal in terms of memory.

Definition 2.1.17 Compact Representation of Elements of $\mathcal{H}_s$
Let p be a permutation from $\mathcal{H}_s$. The binary vector $\tilde{\alpha}_s^{-1}(p) = (x_0, x_1, \ldots x_{w-1})$, $w = 2^s - 1$, is called the compact representation of p.

A conversion between the cartesian and compact representation of permutations in $\mathcal{H}_s$ is straightforward and can be performed very effectively [Hor98]. The permutation composition in $\mathcal{H}_s$ is a non-commutative and non-linear operation. It can be performed directly in the compact representation according to the following algorithm:

Algorithm 2.1.1 Product of Permutations of $\mathcal{H}_s$ in Compact Representation
Let the binary vectors $a = (a_0, \ldots, a_{n-1})$, $n = 2^s - 1$, consisting of bits $a_i \in \{0, 1\}$, and $b = (b_0, \ldots, b_{n-1})$, consisting of bits $b_i \in \{0, 1\}$, be two permutations of $\mathcal{H}_s$, both written in the compact representation. The product $c = a * b$ can be computed as follows:

input a, b

set $c = (0, 0, \ldots, 0)$

call MulPart(0, 0, $n - 1$)

output c

where the recursive procedure MulPart(i_a, i_b, l) *is defined as follows:*

set $t = a_{i_a+l}$
if $l > 1$ then
  set $l' = l/2$
  if $t \neq 0$ then
    call MulPart(i_a, $i_b + l'$, $l' - 1$)
    call MulPart($i_a + l'$, i_b, $l' - 1$)
  else
    call MulPart(i_a, i_b, $l' - 1$)
    call MulPart($i_a + l'$, $i_b + l'$, $l' - 1$)
  endif
endif
set $c_{i_a+l} = t \oplus b_{i_b+l}$

The inverse operation $c = a/b = a{*}b^{-1}$ can be performed directly in the compact representation as well.

Algorithm 2.1.2 "Division" of Permutations of $\mathcal{H}_s$ in Compact Representation
Let all the assumptions from Algorithm 2.1.1 be fulfilled. The operation $c = a/b = a * b^{-1}$ also known as the inverse product of a and b can be computed as follows:

input a, b
set $c = (0, 0, \ldots, 0)$
call DivPart(0, 0, $n - 1$)
output c

where the recursive procedure DivPart(i_a, i_b, l) *is defined as follows:*

set $t = a_{i_a+l} \oplus b_{i_b+l}$
if $l > 1$ then
  set $l' = l/2$
  if $t \neq 0$ then
    call DivPart(i_a, $i_b + l'$, $l' - 1$)
    call DivPart($i_a + l'$, i_b, $l' - 1$)
  else
    call DivPart(i_a, i_b, $l' - 1$)
    call DivPart($i_a + l'$, $i_b + l'$, $l' - 1$)
  endif
endif
set $c_{i_a+l} = t$

2.2 Randomness

Random values are used by many computer applications, e.g. games, scientific simulations, numerical Monte Carlo methods and, last but not least, in cryptographic primitives. Typical examples of cryptographic processes which need some source of randomness are generation of secret keys, prime number generation, stream ciphers based on the one-time pad principle, key-exchange protocols, secret sharing protocols, and many others. The quality of randomness in cryptography must be particularly high, because even smallest patterns or biases in a used "random" sequence might be exploited by an adversary and possibly completely disrupt the cryptosystem. In what follows we will introduce the notion of randomness and pseudorandomness and present some methods for testing of randomness quality.

Definition 2.2.1 Random Number Sequence (RNS)
Let $m \in \mathbb{Z}$, be a fixed integer, $m \geq 2$. A sequence of statistically independent and uniformly distributed integers $x_0, x_1, \ldots, x_l$, where every $x_i \in \mathbb{Z}_m$, is called a random number sequence. In the particular case when $m = 2$ we speak of a random bit sequence (RBS).

Value m is said to be the *range* of the sequence. Note that a truly random sequence of bits can be easily converted into a truly random sequence of numbers with any range m and vice versa, so there is neither a need to state the value m when speaking about an RNS, nor strictly differentiate between a RNS and a RBS.

True randomness can only be achieved by a device based on some non-deterministic *physical source of randomness*, e.g. thermal noise from a semiconductor diode, time between emissions of particles during radioactive decay, etc. Such a device is called a *random number generator* (RNG). The major advantage of a RNG is that its output is really random. However, the non-determinism of the generated sequence (i.e. the fact that the output cannot be reconstructed in the future) is inconvenient in some cases. Also the complexity and costs[2] of a RNG are to high for some applications. For these reasons a *pseudorandom number generator* (PRNG) is often used instead of a RNG.

Definition 2.2.2 Pseudorandom Number Generator (PRNG)
A pseudorandom number generator is a deterministic algorithm which, given a truly random binary sequence of length k, outputs a binary sequence of length $l \gg k$ which "appears" to be random. The input to the PRNG *is called the seed and the output is called a pseudorandom number sequence (*PRNS*).*

The output of a PRNG is not random. In fact, only a very small fraction 2^k of the total 2^l possible sequences can be generated. However, to sufficiently fulfill its role, it is usually enough for a PRNS if we cannot distinguish it from a RNS by means of statistics.

[2]Note that we really need a physical device. This task cannot be accomplished by a pure algorithm.

Again, there is an analogy between the notions of PRNG and PRBG (pseudorandom bit generator) so they can be used in place of each other. We will stick to PRNG.

A simple example of a PRNG is the *linear congruential generator*. Output of this generator is too weak for most cryptographic purposes.

Algorithm 2.2.1 Linear Congruential PRNG
Let m be a fixed range and let $x_0 \in \mathbb{Z}_m$ be a seed. The linear congruential PRNG *generates a sequence of numbers from $\mathbb{Z}_m$, $x_1, x_2, \ldots$ according to the recurrence*

$$x_n = a \cdot x_{n-1} + b \quad mod \quad m,$$

where a and b are the parameters which characterize the generator.

Many suitable combinations of parameters a, b and m (e.g. $a = 48271$, $b = 1$ and $m = 2^{31} - 1$) can be found in [Knu97].

2.2.1 Evaluation of Randomness Quality

Definition 2.2.2 defines a PRNS as a sequence which "appears" to be random. This formulation is rather vague and, in fact, it is not a trivial problem to estimate *how much random* a particular PRNS is. A human who would be presented 10000 bits, such that all of them would be zeros, would very probably think that the sequence is "definitely not random" and would reject the appropriate PRNG as "very weak". However, the sequence of 10000 zeros should be generated with the same probability as any other particular sequence of 10000 bits, and so it *should* appear even by a perfect RNG. Hence, when estimating the quality of a PRNG, we can not make definite conclusions, but rather *probabilistic*. We can say that the generator which produced a sequence of 10000 zero bits is with very high probability not good, but there is still some small probability that we are wrong.

The following two distributions of continuous random variables are widely used in statistical testing.

Definition 2.2.3 The Normal Distribution
A continuous random variable X has a normal distribution with mean μ and variance σ^2 if its probability density function is defined by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad -\infty < x < \infty.$$

X is said to follow $N(\mu, \sigma^2)$ distribution. The special case $N(0, 1)$ is called the *standard normal distribution.*

Definition 2.2.4 The χ^2 Distribution
Let $v > 1$ be an integer. A continuous random variable X has a χ^2 ("chi-square") distribution with v degrees of freedom if its probability density function is defined by

$$f(x) = \begin{cases} \frac{1}{\Gamma(v/2)\cdot 2^{v/2}} \cdot x^{(v/2)-1} \cdot e^{-x/2}, & 0 \leq x < \infty \\ 0, & x < 0 \end{cases}$$

where Γ is the gamma function defined as $\Gamma(t) = \int_0^\infty x^{t-1} \cdot e^{-x} dx$ for $t > 0$. The mean and variance of this distribution are $\mu = v$ and $\sigma^2 = 2v$.

Let us now briefly summarize the principles and notions of statistical testing. A *statistical hypothesis* H_0 is an assertion about a distribution of a random variable. A *test* of a statistical hypothesis is a procedure that is based on observation of the random variable and that leads to the (probabilistic) acceptance or rejection of H_0. The *significance level* α of the test of a hypothesis H_0 is the probability of rejecting H_0 when it is true. Such a wrong decision is called a *type I error*. The case when a wrong H_0 is accepted is said to be the *type II error*. In practice one usually chooses a significance level between 0.001 and 0.05 for the performed tests. A statistical test is implemented by specifying a *statistic* on the random sample, i.e. some function of the elements of the sample. Statistics are chosen so that they can be efficiently computed (e.g. number of zeros in a sequence of bits) and they follow an $N(0,1)$ or a χ^2 distribution. The value of statistic X for a sample sequence is computed and compared with the value expected for a truly random sequence as described below.

- Suppose that for a true RNS a statistic X follows a χ^2 distribution with v degrees of freedom and takes on larger values for non-random sequences. To achieve a significance level α a *threshold value* x_α is chosen so that $P(X > x_\alpha) = \alpha$. If the value X_s of the statistic for a sample sequence satisfies $X_s > x_\alpha$, then the sequence *fails* the test, otherwise it *passes* the test. Such a test is called a *one-sided* test.

- Suppose that for a true RNS a statistic X follows an $N(0,1)$ distribution and takes on both larger and smaller values for non-random sequences. To achieve a significance level α a threshold value x_α is chosen so that $P(X > x_\alpha) = P(X < -x_\alpha) = \frac{\alpha}{2}$. If the value X_s of the statistic for a sample sequence satisfies $X_s > x_\alpha$ or $X_s < -x_\alpha$, then the sequence fails the test, otherwise it passes the test. Such a test is called a *two-sided* test.

Tables of probabilities $P(X > x)$, so-called *percentiles*, for both $N(0,1)$ and χ^2 distributions can be found for instance in [MVV97, p. 177–178].

The randomness tests can be divided into two classes - the *empirical* and the *universal* tests. An empirical randomness test evaluates whether a sample sequence possesses *one specific property* that should be present by a RNS. We list some typical representatives of empirical tests in Appendix A. On the other hand, a universal randomness test is able to detect *any one of a very general class* of possible defects a generator may have. This includes all defects that are detectable by the basic

tests. A universal test is usually based on an entropy estimation, achieved e.g. by trying to compress the sequence using a universal source coding [Mau92b] or by measuring the Ziv-Lempel complexity of the sequence [Hey97]. A drawback of universal randomness tests over the simple tests is that the universal tests need a much longer sample sequence in order to be effective. For example, the Maurer test, introduced in [Mau92b], should theoretically be executed on more than 126 MB of data when used with parameter $L = 16$. For this reason *batteries of tests* are often used in practice. A battery usually consists of many basic tests and a generator is considered as satisfactory only if it passes *all* of them. This approach combines the advantages of both empirical and universal tests because it detects a very general class of defects even on a sample of a reasonable length. Some representatives of test batteries are for instance the *FIPS 140-1* [Nat94], the *DieHard* [Mar97] and the battery used in [SB00].

2.3 Block Ciphers

This section provides a short overview of basic cryptographic notions with impact on *block ciphers* and their evaluation.

Definition 2.3.1 Cryptosystem
A cryptosystem is a five-tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\mathcal{P}$ is a finite set of possible plaintexts, $\mathcal{C}$ is a finite set of possible ciphertexts, $\mathcal{K}$ is a finite set of possible keys, $\mathcal{E}$ is a set of mappings $\mathcal{P} \longrightarrow \mathcal{C}$ and $\mathcal{D}$ is a set of mappings $\mathcal{C} \longrightarrow \mathcal{P}$, such that for every $K \in \mathcal{K}$ there exists an encryption rule $e_K \in \mathcal{E}$ and a decryption rule $d_K \in \mathcal{D}$ having the property $d_K(e_K(x)) = x$ for every $x \in \mathcal{P}$.

The spaces $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{K}$ can be any mathematical spaces in general. However, subspaces of the space of *binary vectors* $\{0,1\}^n$, $n \in \mathbb{Z}$, are most suitable for practical use. The plaintext and ciphertext space are usually equal, i.e. $\mathcal{P} = \mathcal{C}$. In this case they are both called *message space* and are denoted by $\mathcal{M}$.

Note that the set $\mathcal{E}$ of functions e_K can be optionally considered as a single function with two input parameters, i.e. $e : \mathcal{K} \times \mathcal{P} \longrightarrow \mathcal{C}$. Analogously, $\mathcal{D}$ can be considered as a single function $d : \mathcal{K} \times \mathcal{C} \longrightarrow \mathcal{P}$. Nevertheless, in both cases the simplified notation $e_K(x)$ and $d_K(x)$ is typically preferred to the longer $e(K, x)$ and $d(K, x)$.

Definition 2.3.2 Symmetric Cryptosystem
A cryptosystem $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ is said to be symmetric if for every $K \in \mathcal{K}$ it is computationally easy to obtain the appropriate $d_K \in \mathcal{D}$ from an $e_K \in \mathcal{E}$ and vice-versa.

Definition 2.3.3 Block Cipher
A block cipher is a cryptosystem which breaks up the plaintext messages into blocks of fixed length n and encrypts one block at a time.

The two characteristic parameters of a symmetric block cipher are the *block length* n and the *key length* k (both in number of bits). The corresponding message space and key space are $\mathcal{M} = \mathbb{Z}_{2^n}$ and $\mathcal{K} = \mathbb{Z}_{2^k}$ respectively. If a cipher is *iterative*, the *number of rounds* r is another characteristic parameter.

Definition 2.3.4 Iterative Cipher
A block cipher using an encryption function of the form:

$$e_K(x) = f(k_r, f(k_{r-1}, f(\ldots f(k_1, x)\ldots)))$$

is called an r-round iterative cipher. The function $f : \mathcal{K}' \times \mathcal{M} \longrightarrow \mathcal{M}$ is said to be the round function, and the values $k_1, k_2, \ldots, k_r \in \mathcal{K}'$ are the round keys. The function $g : \mathcal{K} \longrightarrow \mathcal{K}'^{(r)}$ is called the key schedule mechanism.

Let $x \in \mathcal{P}$ be a plaintext and $y \in \mathcal{C}$, be the corresponding ciphertext obtained by an encryption $y = e_K(x)$ using a secret key $K \in \mathcal{K}$. The *Kerckhoff's principle* says that the whole security of a cipher should reside in the secrecy of the key, not in the secrecy of the algorithm. This means that knowing y but not knowing K it should not be possible to obtain any information about x - even if both encryption and decryption algorithms are publicly known.

The purpose of an *attack* on a block cipher in terms of *practical security* is constructing a function d' which is identical to the secret decryption rule d_K. If an adversary is able to accomplish this task, he will be able to decrypt all future ciphertexts until the active key is changed. As the encryption and decryption rules are publicly known, d' is usually constructed by recovering the secret key K. According to the information available, the attacks can be classified as *ciphertext-only* attack, *known-plaintext* attack, *chosen-plaintext* attack, etc. The simplest kind of known-plaintext attack is the so-called *brute force* attack based on the exhaustive key search. This attack, given a known pair (x, y), computes $x' = d_{K'}(y)$ for all possible $K' \in \mathcal{K}$ until the correct value K leading to $x' = x$ is found. The brute force attack finds the correct key on average after 2^{k-1} trial decryptions[3] and, hence, its complexity is $O(2^{k-1})$. This number of computations is infeasible for usual values of k (i.e. $k \geq 64$). A cipher is said to be practically secure when the complexity of the *best possible* attack against it is the same as the complexity of a brute force attack.

A more strict notion of security says that a cipher is secure when no oracle circuit can distinguish between the encryption function e_K and a truly random permutation of 2^n elements (see e.g. [LR88, ZMI90, Mau92a, NR97]). In this case the purpose of an attack is just *to distinguish* (not to recover) the encryption function. Obviously, if we cannot distinguish e_K from a random permutation, we can by no means attack the cipher. Such a cipher is *theoretically secure*. Hereby we say that the cipher provides *pseudorandomness* if it is resistant against a distinguishing oracle which is only allowed to perform chosen plaintext attack, and the cipher provides

[3]When $k > n$, it is principally possible to reconstruct d_K, without finding K. The complete d_K, as a table of 2^n rows, can be reconstructed by just $2^n - 1 < 2^{k-1}$ trial encryptions. This chosen-plaintext attack with complexity $O(2^n)$ is another type of a brute force attack.

super-pseudorandomness when the oracle is allowed to perform chosen plaintext and ciphertext attack.

A *perfect* block cipher providing the best possible security for a given block length n and key length k can be (theoretically) designed as follows. One generates a set of 2^k randomly chosen permutations $p_0, p_1, \ldots, p_{2^k-1}$ of 2^n elements and defines the encryption function $e_K : \{0,1\}^n \longrightarrow \{0,1\}^n$ as $e_K(x) = p_K(x)$. The corresponding decryption function $d_K : \{0,1\}^n \longrightarrow \{0,1\}^n$ will be defined as $d_K(x) = p_K^{-1}(x)$. This construction is perfect in terms of security, but can not be used in practice. The table of 2^k random permutations of 2^n elements occupies $2^k \cdot 2^n \cdot n$ bits of memory which is an astronomically large number for any of the nowadays typical configurations (n, k). For instance, when $(n = 128, k = 128)$, the required memory space is 2^{240} megabytes.

Practical block ciphers only *simulate* the optimal block cipher by combining the two basic *Shannon's techniques*.

- *Confusion* - obscures the relationship between the plaintext, the ciphertext and the key (e.g. by using a substitution component).
- *Diffusion* - spreads the information from the plaintext over the whole ciphertext (e.g. by using a transposition component).

A typical building block for a block cipher design is the so-called *S-box* (the "s" stays for substitution here). A u-on-v-bit S-box is a table consisting of 2^u rows, each containing an v-bit binary value. Such a table, embedded in the cipher design, is used to increase the confusion by transforming some u-bit input value x into a randomly looking v-bit output value y. More formally, an S-box implements a random mapping $S : \{0,1\}^u \longrightarrow \{0,1\}^v$ and we write $S(x) = y$. S-boxes are usually defined as fixed tables, but some ciphers employ key-dependent S-boxes as well. Moreover, S-boxes with special properties are sometimes used, for instance, a *P-box* is an u-on-u-bit S-box, such that $S(x) \neq S(x')$ for all $x \neq x'$. Such an S-box implements a permutation of 2^u elements.

Another typical building block of block ciphers is a *transposition* (also called a shuffle). A transposition can act at the bit level (i.e. a bit shuffle), or at the level of whole bytes or words. The main reason for such a transformation is increasing the diffusion of the cipher by permuting the parts of an input.

A typical block cipher combines various S-boxes, transpositions, and other transformations to achieve strong confusion and diffusion. In this way the behavior of the perfect block cipher, described above, is simulated.

2.3.1 Evaluation of Encryption Quality

The main task of a cipher is transforming a message in such a way that it cannot be recovered without the knowledge of the proper key. As for any practical problem, there are many proposals (i.e. encryption algorithms) suggesting how to accomplish

this task, and we need some methods which enable us to rate the quality of a particular cipher and to make comparisons among several ones. However, unlike most practical problems, in cryptography there is no simple metric to evaluate the quality of a particular encryption algorithm. To compare ciphers we have to utilize extensive statistical computations and the achieved results will be only probabilistic.

It is important to stress that, except for very trivial cases, we cannot *prove* that a particular cipher *is* secure. We only can prove that a cipher *is not* secure, for example, if we can find an attack which is faster than an exhaustive key search. If we expose a cipher to more and more attacks and are still not able to find any weakness, we get better and better *confidence* (not proof) that the cipher is strong.

Due to the confusion and diffusion performed by the encryption function the information contained in a plaintext is scrambled and spread and, consequently, all redundancy (i.e. patterns or multiplicity) in the message is destroyed. It follows that a good block cipher produces a very strong pseudorandom output when used properly. Methods for evaluation of block ciphers try to find some irregularities or biases in the encryption algorithm by means of statistics. According to utilized knowledge about the tested cipher the methods can be classified as *generic evaluation methods* (also called black box cryptanalysis) and *white box evaluation methods*. In both cases a successful result of an evaluation is only a *necessary* but not a *sufficient* condition for a cipher to be secure. If a cipher failed the evaluation, we *know* that it is not secure; if it passed we *believe* that it is secure. Only a combination of several evaluations and years of analysis can show a strong evidence that a particular cipher is strong.

2.3.1.1 Generic Evaluation Methods

When using a generic evaluation method, we do not need to know the exact design of the examined cipher. This approach is also called the *black box cryptanalysis*. We either treat the cipher as a binary transformation and explore the dependencies between the input and the corresponding output by means of statistics, or we treat the cipher as a set of permutations and examine the group theoretic properties of this set (e.g. closeness, distribution of element distances, etc.). In both cases we test, whether the produced data have the properties expected from an output of the ideal block cipher, using an extensive randomness testing.

The big advantage of the generic methods is their versatility which makes them immediately applicable to any block cipher. On the other hand, this approach does not provide us with an exact attack even if some defect has been found. In other words, using a generic analysis method, we can find out that "there is something wrong about that cipher" but we do not necessarily know how to exploit the weakness. Nevertheless, when observing such a defect, we can at least *detect* whether the particular cipher is being used or not, which should not be possible with a good cipher.

A generic evaluation method based on randomness testing, as described for example in [Hey97, SB00], usually consists of three independent stages shown in Fig-

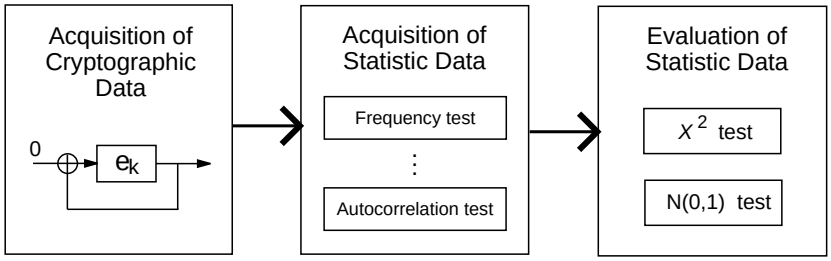


Figure 2.2: Example of a generic cipher evaluation

ure 2.2. In the first step one uses the tested cipher for producing a data sequence which should possess very good pseudorandomness properties. In our example the data are generated by encrypting a zero sequence in CBC mode. In the second step one applies some randomness tests on the generated sequence and obtains the appropriate X values (see also Section 2.2.1). In the final step the X values are either accepted or rejected depending on the used significance level, and possibly some graphical output is produced for better data visualization.

2.3.1.2 White Box Evaluation Methods

The white box cryptanalysis tries to find some weakness in the way the cipher is working, thus, the necessary condition for its applicability is the *exact* knowledge of design of the examined cipher. A common principle of these methods consists in observing some characteristic through the encryption rounds and searching for a statistically significant imbalance. Two typical approaches are *differential cryptanalysis*, which examines a sequence of differences between the input and an intermediate result after few rounds [BS90], and *linear analysis* which examines the sequence of so-called input-output sums [Mat94]. There are also some modifications and combinations of the two basic principles, like the *analysis of impossible differentials* [BBS99], the *differential-linear analysis* [LH94], etc. A typical target for this kind of analysis are the fixed S-boxes. Therefore, when S-boxes are used in a cipher design, they should have a sufficient size and should fulfill some non-trivial criteria (see e.g. [AT90]).

The disadvantage of a white box analysis is the fact that it requires a very detailed and sophisticated examination of the building blocks for a particular cipher. This is a time-consuming task, because it usually cannot be automated and has to be performed by an experienced person. Moreover, it has to be performed for every cipher separately. The advantage of this approach is, however, that once we have found a weakness, we usually can mount a chosen ciphertext attack[4] and estimate its complexity. Hence, white box evaluation methods provide an excellent measure for cipher strength.

[4]These attacks are highly theoretical and require trial encryptions of such a huge amount of data that they cannot be accomplished in a feasible period of time.

Chapter 3

Current Symmetric Cryptosystems Based on Group Bases

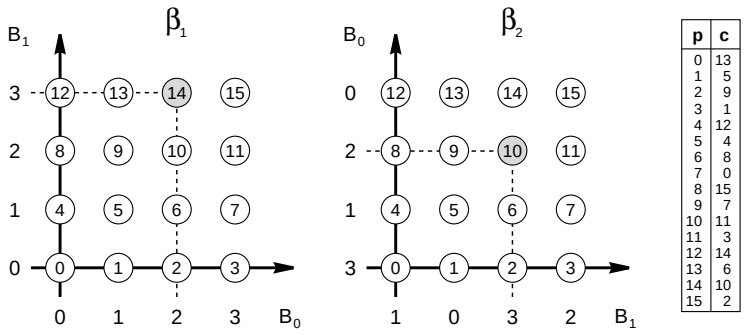
This chapter presents the state of the art in symmetric cryptography based on group bases. We describe the two major cryptosystems that utilize group bases, and discuss their properties and some practical implementation issues. Familiarity with the theory introduced in Section 2.1 is a prerequisite for reading this chapter.

In the first section we explain the idea of point mapping and coordinate mapping in general. The second section is concerned with the cryptosystem PGM which utilizes the full symmetric group S_n. A newer cipher TST which is based on the Sylow 2-subgroup of the symmetric group is introduced in the third section. In both cases we present the detailed encryption setup and the key generation procedure. Furthermore we list the advantages and disadvantages of the cryptosystems. The final fourth section of this chapter summarizes the properties of the introduced ciphers and lists some open questions that need to be answered.

For more information on PGM the reader is referred to [Mag86, MM92], and TST is described in detail in [Hor98].

3.1 Encryption Principle

When speaking about the cryptosystems based on group bases, the underlying groups are always finite permutation groups. The principle of symmetric encryption utilizing group bases can easily be explained when we think of a group basis as a coordinate system. This analogy has already been presented in Figure 2.1. A finite permutation group G can be imagined as an w-dimensional discrete geometric space. Every permutation in G corresponds to a point in the space. A group basis as coordinate system consists of axes (blocks), each containing several points (permutations). Every point in the space can be uniquely described by a vector of coordinates with respect to a particular basis.



p	c
0	13
1	5
2	9
3	1
4	12
5	4
6	8
7	0
8	15
9	7
10	11
11	3
12	14
13	6
14	10
15	2

Figure 3.1: A mapping of points between two coordinate systems

Because there exist extremely many different group bases for any particular group (see e.g. [MM92]), we can pick two tame bases β_1 and β_2 at random and perform a mapping of points from β_1 to β_2 or vice versa. This idea is illustrated in Figure 3.1. The 2-dimensional space in our example consists of 16 points. Both β_1 and β_2 are two dimensional bases of type $(4, 4)$. The point 14 is decomposed with respect to β_1 into coordinates (2,3), i.e. $\tilde{\beta_1}^{-1}(14) = (2, 3)$, and the composition of a point with the same coordinates in β_2 leads us to the point 10, i.e. $\tilde{\beta_2}(2, 3) = 10$. By mapping all other points in an analogous way we obtain a bijective mapping on $\mathbb{Z}_{16}$ as displayed on the right hand side of the figure. This mapping is defined by the pair of group bases (β_1, β_2) and the inverse mapping can be computed in a very similar way - one only has to use the bases in the opposite order (β_2, β_1). Both described transformations can only be performed with the knowledge of β_1 and β_2, and because both β_1 and β_2 are tame, these transformations can be performed efficiently. A symmetric block cipher based on this idea is defined as follows:

Definition 3.1.1 A Block Cipher Based on Point Mapping
Let G be a finite group, called the carrier group. Let $\lambda : \mathbb{Z}_{|G|} \longrightarrow G$ be any fixed bijective mapping. The plaintext and ciphertext spaces for the cipher are the same: $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{|G|}$. The key space is the set $\mathcal{K} = \mathcal{B}_G^* \times \mathcal{B}_G^*$.

Let $K = (\beta_1, \beta_2) \in \mathcal{K}$ be a secret key. Let $x \in \mathcal{P}$ be a plaintext and $y \in \mathcal{C}$ the corresponding ciphertext. The encryption function $e_K : \mathcal{P} \longrightarrow \mathcal{C}$ is defined by the rule

$$y = e_K(x) = \lambda^{-1}(\tilde{\beta_2}(\tilde{\beta_1}^{-1}(\lambda(x))))$$

and the decryption function $d_K : \mathcal{C} \longrightarrow \mathcal{P}$ is defined as

$$x = d_K(y) = \lambda^{-1}(\tilde{\beta_1}(\tilde{\beta_2}^{-1}(\lambda(y)))).$$

A slight modification of the *point mapping* described above is the principle of *coordinate mapping*. By this approach one does not factorize a point into a coordinate vector which is passed for composition to the second basis. Instead, one composes a point from a coordinate vector and passes the point to the second basis for factorization. Consequently, a unique numbering of coordinate vectors (instead of a numbering of points) is needed.

Definition 3.1.2 A Block Cipher Based on Coordinate Mapping
Let G be a finite group, called the carrier group. The plaintext and ciphertext spaces for the cipher are the same: $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{|G|}$. The key space is the set $\mathcal{K} = \mathcal{B}_G^* \times \mathcal{B}_G^*$.

Let $\beta_1 \in \mathcal{B}_G^*$ be a basis of type $r = (r_0, \ldots, r_{w_1-1})$ having the index space $X_1 = \mathbb{Z}_{r_0} \times \cdots \times \mathbb{Z}_{r_{w_1-1}}$, let $\beta_2 \in \mathcal{B}_G^*$ be a basis of type $r' = (r'_0, \ldots, r'_{w_2-1})$ having the index space $X_2 = \mathbb{Z}_{r'_0} \times \cdots \times \mathbb{Z}_{r'_{w_2-1}}$, and let $K = (\beta_1, \beta_2)$ be the secret key. Let $\lambda_1 : \mathbb{Z}_{|G|} \longrightarrow X_1$ and $\lambda_2 : \mathbb{Z}_{|G|} \longrightarrow X_2$ be two fixed bijective mappings. Let $x \in \mathcal{P}$ be a plaintext and $y \in \mathcal{C}$ the corresponding ciphertext. The encryption function $e_K : \mathcal{P} \longrightarrow \mathcal{C}$ is defined by the rule

$$y = e_K(x) = \lambda_2^{-1}(\tilde{\beta_2}^{-1}(\tilde{\beta_1}(\lambda_1(x))))$$

and the decryption function $d_K : \mathcal{C} \longrightarrow \mathcal{P}$ is defined as

$$x = d_K(y) = \lambda_1^{-1}(\tilde{\beta_1}^{-1}(\tilde{\beta_2}(\lambda_2(y)))).$$

In contrast with the point mapping, when performing coordinate mapping, we need two functions λ depending on the types of the used group bases. In fact, we need a whole class of functions providing a unique instance for every possible basis type r.

One should note that the number of dimensions in which we describe a group G is not an intrinsic property of G, it depends on the basis we use. For example, any of the 86400 seconds of a day can be noted uniquely in one-dimensional coordinates $\mathbb{Z}_{86400}$ as well as in three-dimensional coordinates $\mathbb{Z}_{24} \times \mathbb{Z}_{60} \times \mathbb{Z}_{60}$ (i.e. current hour, minute and second). It follows that the same group can be seen e.g. as one-dimensional or as three-dimensional, depending on the group basis used. Consequently, the two bases β_1 and β_2 used as the secret key in both presented encryption setups do neither need to be of the same type, nor of the same dimension.

Both presented encryption setups are general and therefore rather flexible. Each of them defines a whole *family of block ciphers*. Depending on the order of the used carrier group one can construct ciphers of different block lengths. (i.e. the ciphers are scalable.) The complexity of the group has impact on the security and efficiency of the ciphers. Dimensions of the bases affect the memory requirements and, through the size of the key space, also the security of the ciphers. (i.e. the ciphers are adjustable.) The numbering functions λ have no cryptographic importance, but their proper choice can improve the efficiency of a particular cipher. All in all, one can generate many ciphers with very different properties from the same specification, depending on the particular configuration used.

Beside the flexibility, the two presented encryption schemes have another specific property - a *huge key space*. The "classical" block ciphers (e.g. DES, IDEA or RC6) use a relatively short binary sequence as their key. Usual key lengths k lie between 64 and 192 bits. The corresponding key space $\mathbb{Z}_{2^k}$ is by several order of magnitudes smaller than the key space $\mathcal{B}_G^* \times \mathcal{B}_G^*$ of the ciphers based on group bases [MM92].

3.2 Cryptosystem PGM

The cryptosystem PGM was introduced in [Mag86]. It is based on the idea of coordinate mapping (Definition 3.1.2) and utilizes the full symmetric group S_n as a carrier group. The functions λ_1, λ_2 are based on the so-called *knapsack transformation.*

Definition 3.2.1 Knapsack Transformation
Let $x \in \mathbb{Z}_{|G|}$ be an integer. Let $X = \mathbb{Z}_{r_0} \times \cdots \times \mathbb{Z}_{r_{w-1}}$ be the index space of a group basis of type $r = (r_0, \ldots, r_{w-1})$. The knapsack transformation $\lambda : \mathbb{Z}_{|G|} \longrightarrow X$ decomposes x into components $x_0, \ldots, x_{w-1}$, such that $x_i \in \mathbb{Z}_{r_i}$, and $x = R_{w-1}x_{w-1} + \cdots + R_1x_1 + x_0$ where R_i are the so-called knapsack numbers defined by the recurrence $R_0 = 1$ and $R_i = R_{i-1} \cdot r_{i-1}$ when $i > 0$.

Several algorithms for computation of λ and λ^{-1} as well as their efficient hardware implementation are discussed in [Hor98, Chap. 4].

3.2.1 Key Generation

A key K of PGM consists of a random pair of transversal group bases, i.e. $K \in \mathcal{B}^*_{S_n} \times \mathcal{B}^*_{S_n}$. Most other conventional block ciphers use keys in the form of a k-bit binary sequence, i.e. $K \in \mathbb{Z}_{2^k}$. This simpler key format is more suitable for practical use, because it can be generated, memorized, handled and stored more easily than a pair of transversal group bases. Moreover, most cryptographic protocols and standards expect a key in the form of a binary sequence. Hence, to make PGM more convenient and practical, we need a deterministic algorithm for generating pseudorandom pairs of transversal group bases from a given binary key.

One efficient approach to generation of pseudorandom transversal group bases is based on the following transformations.

Definition 3.2.2 Transformations on Group Bases
Let β be a transversal group basis of a group G and let $r = (r_0, \ldots, r_{w-1})$ be the type of β. We define the following transformations on $\mathcal{B}^*_G$:

- T1: Commutative Block Shuffle *performs an arbitrary number of elementary block swaps. An elementary block swap exchanges two randomly chosen adjacent blocks $B_i = (b_{i,0}, \ldots, b_{i,r_i})$ and $B_{i+1} = (b_{i+1,0}, \ldots, b_{i+1,r_{i+1}})$ such that $b_{i,j} * b_{i+1,k} = b_{i+1,k} * b_{i,j}$ for every $j \in \mathbb{Z}_{r_i}$ and $k \in \mathbb{Z}_{r_{i+1}}$.*
- T2: Block Fusion *performs an arbitrary number of elementary fusions. An elementary fusion replaces two randomly chosen, adjacent blocks $B_i = (b_{i,0}, \ldots, b_{i,r_i-1})$ having the key-letter set c_i and $B_j = (b_{j,0}, \ldots, b_{j,r_j-1})$, $j = i + 1$, having the key-letter set c_j by a single longer block $B'_i = B_i \otimes B_j = (b_{i,m} * b_{j,n} : m \in \mathbb{Z}_{r_i}, n \in \mathbb{Z}_{r_j})$ having the key-letter set $c'_i = c_i \cup c_j$. The transformation changes the type of the basis from $r = (r_0, \ldots, r_i, r_{i+1}, r_{i+2}, \ldots, r_{w-1})$ to $r' = (r_0, \ldots, r_i \cdot r_{i+1}, r_{i+2}, \ldots, r_{w-1})$ and decreases the dimension of the basis from w to $w - 1$.*

- T3: Randomization *replaces each* $b_{i,j}$, $i = 1, \ldots, w-1$, $j \in \mathbb{Z}_{r_i}$ *by* $b'_{i,j} = b_{i,j} * \prod_{k=0}^{i-1} b_{k,l_k}$, *where* $l_k \in \mathbb{Z}_{r_k}$ *is chosen randomly for every combination of* i, j *and* k.

- T4: Element Shuffle *randomly changes the order of the elements within each block* B_i, $i \in \mathbb{Z}_w$.

- T5: Conjugation *replaces all elements* $b_{i,j}$, $i \in \mathbb{Z}_w$, $j \in \mathbb{Z}_{r_i}$ *by* $b'_{i,j} = g^{-1} * b_{i,j} * g$, *where* g *is a randomly chosen fixed permutation from* G.

All these transformations preserve the transversality of a processed group basis, i.e. if β is a transversal group basis for a group G, then $\beta' = Ti(\beta)$, $1 \leq i \leq 5$, is a transversal group basis for G as well. (For proof see [Hor98, Sec. 2.3].) $T1$ to $T5$ are not the only transformations with this property, but are the most straightforward and practical ones. For more examples of group basis transformations and a deeper theory on the topic see [MM92].

A single pseudorandom transversal group basis for a group G can be generated by the following algorithm.

Definition 3.2.3 Basis Generation Algorithm (BGA)
Let G *be a permutation group and let* β_0 *be a transversal group basis for* G. BGA *performs a sequence of transformations* $\beta_1 = T1(\beta_0)$, $\beta_2 = T2(\beta_1)$, $\ldots$, $\beta_5 = T5(\beta_4)$ *and outputs the randomized basis* $\beta = \beta_5$.

When used with PGM, BGA has to be performed twice in order to obtain a pair of pseudorandom bases. The initial permutation β_0 is set to the canonical basis α of S_n (Definition 2.1.12).

Because of the randomizing character of Ti the BGA needs a source of randomness. All randomizing steps executed during the transformations Ti should be performed according to an output of a pseudorandom number generator (PRNG) initialized by a seed K. The k-bit binary value $K \in \mathbb{Z}_{2^k}$ is an important input of the BGA and, because K uniquely determines the generated pair of bases (β_1, β_2), it can be used as the secret key of PGM. The definition of PGM itself does not specify any particular algorithm to be used as the PRNG. The algorithm can be chosen rather freely, it should, however, pass some non-trivial statistic tests.

In [Hor98] the author also defines a slightly simplified version of BGA utilized by the so-called *Standard-PGM* which is more suitable for an effective implementation. This version of BGA executes only the transformations T_3, T_4 and a simplified variant of T_5.

Because $|\mathcal{B}^*_{S_n} \times \mathcal{B}^*_{S_n}| \gg |\mathbb{Z}_{2^k}|$ for usual values n and k, the BGA can be made *scalable*, i.e. can support a variable key length k. To implement PGM with scalable keys one has to use a PRNG that supports a seed of variable length. This feature gives PGM more flexibility and preserves the advantage of the huge key space.

3.2.2 Properties

From the cryptographic point of view, PGM possesses a couple of desirable algebraic properties. For example, the set of all encryption functions is not closed under functional composition and hence it is not a group. As a consequence multiple encryption is possible. The group generated by the PGM encryption functions by taking the multiple encryption into account is the entire symmetric group $S_{|G|}$ - an astronomically large set of mappings. Other interesting algebraic and statistical properties of PGM are discussed in [MM89, Mem89, MM90, MM92] in more detail. Brought together, PGM is a very robust cryptosystem with an extremely large key space.

From a practical point of view, however, PGM is not that excellent. It has two major inherent drawbacks related to its implementation and usage.

- *Ciphertext Expansion.* The order of the carrier group S_n is not a power of two for any $n > 2$, i.e. $S_n = n! \neq 2^l$, $l \in \mathbb{Z}$. Hence, when establishing the block length of the cipher, one can only use $l = \lfloor \log_2(n!) \rfloor$ bits for a plaintext[1]. However, the appropriate ciphertext can take on all values up to $n! - 1$. For a binary representation of these values one needs $l' = \lceil \log_2(n!) \rceil$ bits. It follows that the encryption causes a data expansion by one bit, as $l' = l + 1$.

  This property is very undesirable, not only because it makes multiple encryption problematic, but especially because it disqualifies PGM from usage in most standardized cryptographic protocols expecting equal lengths of plaintext and ciphertext.

- *Complexity of the Knapsack Transformation.* The transformation λ_1 performed at the beginning and the inverse transformation λ_2^{-1} performed at the end of every encryption do not contribute to the cryptographic strength of the cipher, but are rather computationally intensive. This slows down the cipher unnecessarily. As a consequence, PGM is not competitive with most of modern block ciphers in terms of speed.

Both problems arise from the fact that the used carrier group S_n is not binary. If the order of G were a power of two, the cipher text expansion would not happen, because all plaintexts and ciphertexts would fit in exactly $\log_2(|G|)$ bits. The knapsack transformations would also not be necessary. Note that any index space $X = \mathbb{Z}_{r_0} \times \cdots \times \mathbb{Z}_{r_{w-1}}$ contains only subspaces whose orders r_i are divisors of $|G|$. Therefore, if $|G|$ is a power of 2, all r_i are powers of 2 as well. In this case the knapsack transformation of an index vector $x = (x_0, \ldots, x_{w-1}) \in X$ can simply be replaced by a concatenation of binary representations of coordinates x_i, i.e. $\lambda(x) = x_0||x_1|| \cdots ||x_{w-1}$.

Apparently, the problems of PGM could be solved simply by using a binary carrier group G instead of the full symmetric group S_n. However, in order to make PGM secure one has to use carrier groups of a sufficient order $|G| \approx 2^{128}$. Because even

[1] If $\lceil \log_2(n!) \rceil$ bits were used, the inputs $p = n!, \ldots, (2^{\lceil \log_2(n!) \rceil} - 1)$ could not be encrypted.

the biggest binary group $\mathcal{H}_s$ has a rather small order in comparison with S_n, one needs to work with much longer permutations when using a binary G. For instance, to achieve $|G| \geq 2^{127}$ one needs to use permutations of 34 elements when $G = S_n$ (because $|S_{34}| = 34! \geq 2^{127}$), but one needs permutations of 127 elements when $G = \mathcal{H}_s$ (because $|\mathcal{H}_7| = 2^{2^7-1} \geq 2^{127}$ and $\mathcal{H}_7 \leq S_{2^7}$). It follows that a simple use of binary carrier groups without any further improvements would significantly increase the memory requirements and, thus, reduce the efficiency of PGM.

The inconveniences of PGM have been avoided in the cryptosystem TST which was introduced in [Hor98]. The substantially new contribution of this design resides not only in the usage of special binary carrier groups, but especially in the definition of their compact representation.

3.3 Cryptosystem TST

The cryptosystem TST is based on the principle of point mapping (Definition 3.1.1). The encryption and decryption functions have been slightly extended to:

$$y = e_K(x) = \lambda^{-1}(\tilde{\beta}_2(R(\tilde{\beta}_1{}^{-1}(\lambda(x)))))$$

and

$$x = d_K(y) = \lambda^{-1}(\tilde{\beta}_1(R(\tilde{\beta}_2{}^{-1}(\lambda(y)))))$$

where the *bit reversing function* $R : \{0,1\}^n \longrightarrow \{0,1\}^n$ reverses the order of bits in a binary vector of length n, i.e. $R(b_0, b_1, \ldots, b_{n-1}) = (b_{n-1}, b_{n-2}, \ldots, b_0)$, $b_i \in \{0,1\}$. This extension of e_K and d_K was introduced in the interest of cryptographic strength. A deeper argumentation for this modification is given in [Hor98, Sec. 7.5].

3.3.1 Carrier Group

In general, any binary group can be used as a carrier group of TST . However, from both cryptographic and practical point of view the most suitable class of groups for TST is the $\mathcal{H}_s \times \mathcal{H}_1$. Let us discuss the reasons in more detail.

For the sake of cryptographic strength the carrier group should be as complex as possible. Simple commutative groups like $\mathbb{Z}_2^n$ are not very advisable. The most complex binary group available is the Sylow 2-group $\mathcal{H}_s$. However, $\mathcal{H}_s$ is not very suitable for another practical reason. It is desirable for any block cipher, that its block length is a multiple of 8, or possibly 32. Only these block lengths guarantee that the block representation in a computer will occupy a whole number of bytes, respectively words. The order of $\mathcal{H}_s$, however, is 2^{2^s-1} and, hence, the supported block length of the appropriate cipher would be $2^s - 1$ which is not equal to $8k$ for any $k \in \mathbb{Z}$.

Group $\mathcal{H}_s \times \mathcal{H}_1$ ensures the combination of both desirable properties. On one hand, its order has the optimal form $2^{2^s-1} \times 2$ which results into block length 2^s divisible by any 2^k, $k < s$. On the other hand, $\mathcal{H}_s \times \mathcal{H}_1$ is the group with the most complex structure among all such groups.

The compact representation of permutations in $\mathcal{H}_s \times \mathcal{H}_1$ bases on the compact representation in $\mathcal{H}_s$ (Definition 2.1.17).

Definition 3.3.1 Compact Representation of Elements of $\mathcal{H}_s \times \mathcal{H}_1$
Let p_1 be a permutation of $\mathcal{H}_s$ and p_2 a permutation of $\mathcal{H}_1$. The compact representation of the permutation $p \in \mathcal{H}_s \times \mathcal{H}_1$ given by $p(i) = p_1(i)$ for $i \in \{1, \ldots, 2^s - 1\}$ and $p(j) = 2^s + p_2(j - 2^s)$ for $j \in \{2^s, 2^s + 1\}$ is the binary vector $\alpha = \alpha_1 || \alpha_2$ where α_1 is the compact representation of p_1 in $\mathcal{H}_s$ and α_2 is the compact representation of p_2 in $\mathcal{H}_1$.

Note that a compact representation of a $p \in \mathcal{H}_s \times \mathcal{H}_1$ is an n-bit binary vector, where $n = 2^s$. When we observe this vector as a binary representation of an integer $x \in \mathbb{Z}_{2^n}$, we obtain a unique numbering of all elements of $\mathcal{H}_s \times \mathcal{H}_1$. This numbering is in fact used by TST as the function $\lambda : \mathbb{Z}_{|G|} \longrightarrow G$ according to the Definition 3.1.1.

A product of permutations $a * b$ in $\mathcal{H}_s \times \mathcal{H}_1$ can be performed directly in the compact representation. The utilized algorithm is almost the same as for $\mathcal{H}_s$ (Algorithm 2.1.1), only the highest bits of the operands must be processed as well.

Algorithm 3.3.1 Product of Permutations of $\mathcal{H}_s \times \mathcal{H}_1$ in Compact Representation
Let the binary vectors $a = (a_0, \ldots, a_{n-1})$ and $b = (b_0, \ldots, b_{n-1})$, $n = 2^s$, consisting of bits $a_i, b_i \in \{0, 1\}$, be two permutations of $\mathcal{H}_s \times \mathcal{H}_1$, both written in the compact representation. The product $c = a * b$ can be computed as follows:

input a, b
set $a' = (a_0, \ldots, a_{n-2})$, $(a' \in \mathcal{H}_s)$
set $b' = (b_0, \ldots, b_{n-2})$, $(b' \in \mathcal{H}_s)$
Obtain $c' = a' * b'$ using the Algorithm 2.1.1
set $c = (c'_0, c'_1, \ldots, c'_{n-2}, a_{n-1} \oplus b_{n-1})$
output c

The inverse product $c = a/b = a * b^{-1}$ can be computed in an analogous way with the only difference that the Algorithm 2.1.1 has to be replaced by Algorithm 2.1.2.

As both the product $a * b$ as well as the inverse product $a * b^{-1}$ of two permutations $a, b \in \mathcal{H}_s \times \mathcal{H}_1$ can be performed directly in the compact representation, the complete encryption or decryption procedure can be performed without any conversions. A plaintext $x \in \mathbb{Z}_{2^n}$ is simply *considered* (without doing any computations) as a compact representation of a permutation $p \in \mathcal{H}_s \times \mathcal{H}_1$. p is then directly factorized by β_1, the obtained index vector is reversed by R and passed for a composition into β_2. The resulting permutation p', obtained again directly in the compact representation, is considered as a number $y \in \mathbb{Z}_{2^n}$ which is the corresponding ciphertext. The advantage of this approach is the fact that the evaluation of λ and λ^{-1} does not require any computations.

3.3.2 Key Generation

The BGA of TST works in a very similar way as the one of PGM (Definition 3.2.3). The only difference is the initial basis β_0 which is in this case equal to the canonical basis for $\mathcal{H}_s \times \mathcal{H}_1$.

Definition 3.3.2 Canonical Basis for $\mathcal{H}_s \times \mathcal{H}_1$
The canonical basis α for $\mathcal{H}_s \times \mathcal{H}_1$ has the form $\alpha = (A_0, \ldots, A_{n-1})$ where $n = 2^s$ and for every $i \in \mathbb{Z}_n$ the block A_i consists of the following two permutations (written in compact representation) $a_{i,0} = (\underbrace{0 \ldots 0}_{n \text{ times}})$ and $a_{i,1} = (\underbrace{0 \ldots 0}_{i-1 \text{ times}} \; 1 \; \underbrace{0 \ldots 0}_{n-i \text{ times}})$.

We demonstrate the BGA on a simple carrier group $\mathcal{H}_3 \times \mathcal{H}_1$ in both the compact and cartesian representations in Example 3.3.1. The key-bits, printed with bold letters in the compact representation, play an important role during factorization which is performed according to Algorithm 3.3.2

Algorithm 3.3.2 Factorization in Compact Representation of $\mathcal{H}_s \times \mathcal{H}_1$
Let $\beta = (B_0, \ldots, B_{w-1})$ be a transversal group basis generated by the BGA. Let c_i be the set of key-bit positions of block B_i. Let the binary vector $p = (p^{(0)}, \ldots, p^{(n-1)})$, $n = 2^s$, consisting of bits $p^{(i)} \in \{0, 1\}$, be the compact representation of a permutation in $\mathcal{H}_s \times \mathcal{H}_1$. The vector of coordinates $x = (x_0, \ldots, x_{w-1})$ of p with respect to β can be computed as follows:

input p
for $i = w-1, w-2, \ldots, 0$ do
    Find $f_i = b_{i,j} \in B_i$, such that $b_{i,j}^{(k)} = p^{(k)}$ for all $k \in c_i$
    set $x_i = j$
    set $p = p * f_i^{-1}$
endfor
output $(x_0, \ldots, x_{w-1})$

This algorithm looks for the factors x_i of a permutation p sequentially, starting with the highest order $i = w - 1$ and going down to $i = 0$. The initial value p_{w-1} is equal to p. In every factorization step one has to find a factor f_i whose key bits are equal to those of p_i. The i-th coordinate of p is given by the position of f_i inside of the base block B_i. The intermediate result $p_{i-1} = p_i * f_i^{-1}$ is passed for further factorization to the next lower block B_{i-1} of β. Example 3.3.2 demonstrates the factorization algorithm on a simple 8-bit TST encryption. Note that the lengths of bit groups in the step 2 of Example 3.3.2 are established according to the types of group bases. For instance, the first coordinate with respect to β_1 is coded in 1 bit, because $\log_2(r_1) = 1$, the second in 2 bits, because $\log_2(r_2) = 2$, etc.

Example 3.3.1 Basis Generation Algorithm of TST

Canonical basis

$\beta_{\mathbf{0}}$	Cartesian	Compact
	[0,1,2,3,4,5,6,7,9,8]	0 0 0 0 0 0 0 **1**
B_7	[0,1,2,3,4,5,6,7,8,9]	0 0 0 0 0 0 0 **0**
	[4,5,6,7,0,1,2,3,8,9]	0 0 0 0 0 0 **1** 0
B_6	[0,1,2,3,4,5,6,7,8,9]	0 0 0 0 0 0 **0** 0
	[0,1,2,3,6,7,4,5,8,9]	0 0 0 0 0 **1** 0 0
B_5	[0,1,2,3,4,5,6,7,8,9]	0 0 0 0 0 **0** 0 0
	[0,1,2,3,4,5,7,6,8,9]	0 0 0 0 **1** 0 0 0
B_4	[0,1,2,3,4,5,6,7,8,9]	0 0 0 0 **0** 0 0 0
	[0,1,2,3,5,4,6,7,8,9]	0 0 0 **1** 0 0 0 0
B_3	[0,1,2,3,4,5,6,7,8,9]	0 0 0 **0** 0 0 0 0
	[2,3,0,1,4,5,6,7,8,9]	0 0 **1** 0 0 0 0 0
B_2	[0,1,2,3,4,5,6,7,8,9]	0 0 **0** 0 0 0 0 0
	[0,1,3,2,4,5,6,7,8,9]	0 **1** 0 0 0 0 0 0
B_1	[0,1,2,3,4,5,6,7,8,9]	0 **0** 0 0 0 0 0 0
	[1,0,2,3,4,5,6,7,8,9]	**1** 0 0 0 0 0 0 0
B_0	[0,1,2,3,4,5,6,7,8,9]	**0** 0 0 0 0 0 0 0

T_1 Commutative block shuffle

$\beta_{\mathbf{1}}$	Cartesian	Compact
	[4,5,6,7,0,1,2,3,8,9]	0 0 0 0 0 0 **1** 0
B_7	[0,1,2,3,4,5,6,7,8,9]	0 0 0 0 0 0 **0** 0
	[2,3,0,1,4,5,6,7,8,9]	0 0 **1** 0 0 0 0 0
B_6	[0,1,2,3,4,5,6,7,8,9]	0 0 **0** 0 0 0 0 0
	[0,1,2,3,6,7,4,5,8,9]	0 0 0 0 0 **1** 0 0
B_5	[0,1,2,3,4,5,6,7,8,9]	0 0 0 0 0 **0** 0 0
	[0,1,2,3,4,5,6,7,9,8]	0 0 0 0 0 0 0 **1**
B_4	[0,1,2,3,4,5,6,7,8,9]	0 0 0 0 0 0 0 **0**
	[1,0,2,3,4,5,6,7,8,9]	**1** 0 0 0 0 0 0 0
B_3	[0,1,2,3,4,5,6,7,8,9]	**0** 0 0 0 0 0 0 0
	[0,1,2,3,5,4,6,7,8,9]	0 0 0 **1** 0 0 0 0
B_2	[0,1,2,3,4,5,6,7,8,9]	0 0 0 **0** 0 0 0 0
	[0,1,2,3,4,5,7,6,8,9]	0 0 0 0 **1** 0 0 0
B_1	[0,1,2,3,4,5,6,7,8,9]	0 0 0 0 **0** 0 0 0
	[0,1,3,2,4,5,6,7,8,9]	0 **1** 0 0 0 0 0 0
B_0	[0,1,2,3,4,5,6,7,8,9]	0 **0** 0 0 0 0 0 0

T_2 Block Fusion

$\beta_{\mathbf{2}}$	Cartesian	Compact
	[6,7,4,5,0,1,2,3,8,9]	0 0 **1** 0 0 0 **1** 0
	[4,5,6,7,0,1,2,3,8,9]	0 0 **0** 0 0 0 **1** 0
B_3	[2,3,0,1,4,5,6,7,8,9]	0 0 **1** 0 0 0 **0** 0
	[0,1,2,3,4,5,6,7,8,9]	0 0 **0** 0 0 0 **0** 0
	[1,0,2,3,6,7,4,5,9,8]	**1** 0 0 0 0 **1** 0 **1**
	[0,1,2,3,6,7,4,5,9,8]	**0** 0 0 0 0 **1** 0 **1**
	[1,0,2,3,6,7,4,5,8,9]	**1** 0 0 0 0 **1** 0 **0**
	[0,1,2,3,6,7,4,5,8,9]	**0** 0 0 0 0 **1** 0 **0**
B_2	[1,0,2,3,4,5,6,7,9,8]	**1** 0 0 0 0 **0** 0 **1**
	[0,1,2,3,4,5,6,7,9,8]	**0** 0 0 0 0 **0** 0 **1**
	[1,0,2,3,4,5,6,7,8,9]	**1** 0 0 0 0 **0** 0 **0**
	[0,1,2,3,4,5,6,7,8,9]	**0** 0 0 0 0 **0** 0 **0**
	[0,1,2,3,5,4,7,6,8,9]	0 0 0 **1** **1** 0 0 0
	[0,1,2,3,5,4,6,7,8,9]	0 0 0 **1** **0** 0 0 0
B_1	[0,1,2,3,4,5,7,6,8,9]	0 0 0 **0** **1** 0 0 0
	[0,1,2,3,4,5,6,7,8,9]	0 0 0 **0** **0** 0 0 0
	[0,1,3,2,4,5,6,7,8,9]	0 **1** 0 0 0 0 0 0
B_0	[0,1,2,3,4,5,6,7,8,9]	0 **0** 0 0 0 0 0 0

T_3 Randomization

$\beta_{\mathbf{3}}$	Cartesian	Compact
	[6,7,4,5,1,0,2,3,8,9]	0 0 **1** 1 0 0 **1** 0
	[4,5,6,7,2,3,0,1,9,8]	0 0 **0** 0 0 1 **1** 1
B_3	[2,3,0,1,6,7,4,5,8,9]	0 0 **1** 0 0 1 **0** 0
	[0,1,2,3,7,6,5,4,8,9]	0 0 **0** 1 1 1 **0** 0
	[1,0,2,3,6,7,5,4,9,8]	**1** 0 0 0 1 **1** 0 **1**
	[0,1,3,2,6,7,5,4,9,8]	**0** 1 0 0 1 **1** 0 **1**
	[1,0,2,3,6,7,4,5,8,9]	**1** 0 0 0 0 **1** 0 **0**
	[0,1,2,3,6,7,4,5,8,9]	**0** 0 0 0 0 **1** 0 **0**
B_2	[1,0,3,2,4,5,7,6,9,8]	**1** 1 0 0 1 **0** 0 **1**
	[0,1,2,3,4,5,6,7,9,8]	**0** 0 0 0 0 **0** 0 **1**
	[1,0,2,3,4,5,7,6,8,9]	**1** 0 0 0 1 **0** 0 **0**
	[0,1,3,2,5,4,7,6,8,9]	**0** 1 0 1 1 **0** 0 **0**
	[0,1,3,2,5,4,7,6,8,9]	0 0 0 **1** **1** 0 0 0
	[0,1,3,2,5,4,6,7,8,9]	0 1 0 **1** **0** 0 0 0
B_1	[0,1,3,2,4,5,7,6,8,9]	0 1 0 **0** **1** 0 0 0
	[0,1,2,3,4,5,6,7,8,9]	0 0 0 **0** **0** 0 0 0
	[0,1,3,2,4,5,6,7,8,9]	0 **1** 0 0 0 0 0 0
B_0	[0,1,2,3,4,5,6,7,8,9]	0 **0** 0 0 0 0 0 0

T_4 Element shuffle

$\beta_{\mathbf{4}}$	Cartesian	Compact
	[4,5,6,7,2,3,0,1,9,8]	0 0 **0** 0 0 1 **1** 1
	[6,7,4,5,1,0,2,3,8,9]	0 0 **1** 1 0 0 **1** 0
B_3	[2,3,0,1,6,7,4,5,8,9]	0 0 **1** 0 0 1 **0** 0
	[0,1,2,3,7,6,5,4,8,9]	0 0 **0** 1 1 1 **0** 0
	[0,1,3,2,5,4,7,6,8,9]	**0** 1 0 1 1 **0** 0 **0**
	[0,1,2,3,4,5,6,7,9,8]	**0** 0 0 0 0 **0** 0 **1**
	[0,1,2,3,6,7,4,5,8,9]	**0** 0 0 0 0 **1** 0 **0**
	[1,0,2,3,4,5,7,6,8,9]	**1** 0 0 0 1 **0** 0 **0**
B_2	[1,0,2,3,6,7,4,5,8,9]	**1** 0 0 0 0 **1** 0 **0**
	[1,0,3,2,4,5,7,6,9,8]	**1** 1 0 0 1 **0** 0 **1**
	[0,1,3,2,6,7,5,4,9,8]	**0** 1 0 0 1 **1** 0 **1**
	[1,0,2,3,6,7,5,4,9,8]	**1** 0 0 0 1 **1** 0 **1**
	[0,1,3,2,5,4,6,7,8,9]	0 1 0 **1** **0** 0 0 0
	[0,1,2,3,4,5,6,7,8,9]	0 0 0 **0** **0** 0 0 0
B_1	[0,1,2,3,5,4,7,6,8,9]	0 0 0 **1** **1** 0 0 0
	[0,1,3,2,4,5,7,6,8,9]	0 1 0 **0** **1** 0 0 0
	[0,1,3,2,4,5,6,7,8,9]	0 **1** 0 0 0 0 0 0
B_0	[0,1,2,3,4,5,6,7,8,9]	0 **0** 0 0 0 0 0 0

T_5 Conjugation with [6,7,4,5,3,2,1,0,9,8]

$\beta_{\mathbf{5}}$	Cartesian	Compact
	[7,6,5,4,1,0,3,2,9,8]	1 1 1 1 1 **0** **1** 1
	[5,4,6,7,3,2,1,0,8,9]	1 0 0 1 1 **1** **1** 0
B_3	[2,3,0,1,6,7,4,5,8,9]	0 0 1 0 0 **1** **0** 0
	[3,2,1,0,4,5,6,7,8,9]	1 1 1 0 0 **0** **0** 0
	[1,0,3,2,5,4,6,7,8,9]	1 1 **0** 1 **0** 0 0 **0**
	[0,1,2,3,4,5,6,7,9,8]	0 0 **0** 0 **0** 0 0 **1**
	[2,3,0,1,4,5,6,7,8,9]	0 0 **1** 0 **0** 0 0 **0**
	[1,0,2,3,4,5,7,6,8,9]	1 0 **0** 0 **1** 0 0 **0**
B_2	[2,3,0,1,4,5,7,6,8,9]	0 0 **1** 0 **1** 0 0 **0**
	[1,0,2,3,5,4,7,6,9,8]	1 0 **0** 1 **1** 0 0 **1**
	[3,2,0,1,5,4,6,7,9,8]	1 0 **1** 1 **0** 0 0 **1**
	[3,2,0,1,4,5,7,6,9,8]	1 0 **1** 0 **1** 0 0 **1**
	[0,1,3,2,5,4,6,7,8,9]	**0** **1** 0 1 0 0 0 0
	[0,1,2,3,4,5,6,7,8,9]	**0** **0** 0 0 0 0 0 0
B_1	[1,0,3,2,4,5,6,7,8,9]	**1** **1** 0 0 0 0 0 0
	[1,0,2,3,5,4,6,7,8,9]	**1** **0** 0 1 0 0 0 0
	[0,1,2,3,4,5,6,7,8,9]	0 0 0 **0** 0 0 0 0
B_0	[0,1,2,3,5,4,6,7,8,9]	0 0 0 **1** 0 0 0 0

Note: The bold letters mark the key-bit positions.

Example 3.3.2 TST Encryption

Carrier group: $\mathcal{H}_3 \times \mathcal{H}_1$
Plaintext: $p = 10110110$
Key: $K = (\beta_1, \beta_2)$

1. Factorization $x = \tilde{\beta_1}^{-1}(p)$

$p =$	10110**11**0
$f_3 = b_{3,2} =$	10011**11**0
$x_3 = 2$	
$p' = p/f_3 =$	11**10100**0
$f_2 = b_{2,3} =$	00**10100**0
$x_2 = 3$	
$p'' = p'/f_2 =$	**11**000000
$f_1 = b_{1,1} =$	**11**000000
$x_1 = 1$	
$p''' = p''/f_1 =$	000**0**0000
$f_0 = b_{0,1} =$	000**0**0000
$x_0 = 1$	
$p'''' = p'''/f_0 =$	00000000
$x =$	(1,1,3,2)

β_1

$b_{3,3}$	1 1 1 1 1 **0 1** 1
$b_{3,2}$	1 0 0 1 1 **1 1** 0
$b_{3,1}$	0 0 1 0 0 **1 0** 0
$b_{3,0}$	1 1 1 0 0 **0 0** 0
$b_{2,7}$	1 1 **0** 1 **0** 0 0 **0**
$b_{2,6}$	0 0 **0** 0 **0** 0 0 **1**
$b_{2,5}$	0 0 **1** 0 **0** 0 0 **0**
$b_{2,4}$	1 0 **0** 0 **1** 0 0 **0**
$b_{2,3}$	0 0 **1** 0 **1** 0 0 **0**
$b_{2,2}$	1 0 **0** 1 **1** 0 0 **1**
$b_{2,1}$	1 0 **1** 1 **0** 0 0 **1**
$b_{2,0}$	1 0 **1** 0 **1** 0 0 **1**
$b_{1,3}$	**0 1** 0 1 0 0 0 0
$b_{1,2}$	**0 0** 0 0 0 0 0 0
$b_{1,1}$	**1 1** 0 0 0 0 0 0
$b_{1,0}$	**1 0** 0 1 0 0 0 0
$b_{0,1}$	0 0 0 **0** 0 0 0 0
$b_{0,0}$	0 0 0 **1** 0 0 0 0

β_2

$b'_{4,1}$	1 0 1 1 1 1 **1** 0
$b'_{4,0}$	0 0 0 1 1 1 **0** 1
$b'_{3,7}$	1 **1 1** 1 1 **0** 0 1
$b'_{3,6}$	1 **0 1** 1 0 **0** 0 0
$b'_{3,5}$	1 **0 1** 0 1 **1** 0 1
$b'_{3,4}$	0 **0 0** 1 1 **0** 0 0
$b'_{3,3}$	1 **0 0** 1 0 **1** 0 0
$b'_{3,2}$	0 **1 0** 1 1 **1** 0 0
$b'_{3,1}$	1 **1 1** 0 1 **1** 0 1
$b'_{3,0}$	0 **1 0** 0 0 **0** 0 0
$b'_{2,3}$	**0** 0 0 1 1 0 0 **1**
$b'_{2,2}$	**1** 0 0 0 1 0 0 **0**
$b'_{2,1}$	**0** 0 0 0 0 0 0 **0**
$b'_{2,0}$	**1** 0 0 1 1 0 0 **1**
$b'_{1,1}$	0 0 0 **0** 1 0 0 0
$b'_{1,0}$	0 0 0 **1** 0 0 0 0
$b'_{0,1}$	0 0 0 0 **1** 0 0 0
$b'_{0,0}$	0 0 0 0 **0** 0 0 0

$r = (2, 4, 8, 4), \quad 2 \times 4 \times 8 \times 4 = 256 = |\mathcal{H}_3 \times \mathcal{H}_1|$
$r' = (2, 2, 4, 8, 2), \quad 2 \times 2 \times 4 \times 8 \times 2 = 256 = |\mathcal{H}_3 \times \mathcal{H}_1|$

2. Bit Reversing $x' = R(x)$
$x = (1, 1, 3, 2) = 1||10||110||01 = 11011001$
$x' = R(11011001) = 10011011 = 1||0||01||101||1 = (1, 0, 2, 5, 1)$

3. Composition $c = \tilde{\beta_2}(x')$
$c = \tilde{\beta_2}(1, 0, 2, 5, 1) = b'_{0,1} * b'_{1,0} * b'_{2,2} * b'_{3,5} * b'_{4,1} = 01000011$

Thus $e_K(10110110) = 01000011$ *(in decimal numbers:* $e_K(109) = 194$*)*

3.3.3 Properties

Cryptosystem TST successfully avoids the two biggest disadvantages of PGM - the data expansion during encryption and the slow knapsack transformation. Both problems have been solved by using a binary carrier group based on the Sylow 2-group. The compact representation of permutations with the appropriate multiplication and division algorithms made it possible to perform the complete encryption and decryption procedures on binary vectors without any conversions.

Nevertheless, the TST proposed in 1998 is rather new, and still not much is known about its cryptographic properties and efficiency. Security considerations presented in [Hor98] are rather intuitive and are not supported by any practical experiments. The system has apparently been optimized for hardware, but the suitability for a software implementation is not discussed. In fact, until now there exists no fully functional version of TST which supports all its features. A simplified

hardware version, so-called *Chip-TST* supporting a single block length was proposed in [Hor98], but was not cryptographically analyzed.

All in all, TST seems to be a very promising cryptosystem based on novel ideas, but not much is known about its algebraic and cryptographic properties or about its practical efficiency. Before it can get widespread and commonly used, it has to be analyzed and attacked extensively.

3.4 Summary and Open Problems

Unlike most usual block ciphers, block ciphers based on group bases possess several desirable properties:

1. The encryption and decryption functions are very simple which leads to a better transparency and possibility of mathematical examinations. The security resides in the fact that one of extremely many mappings defined by pairs of group bases is chosen in random and kept secret.

2. There is no upper bound of achievable key space. Depending on the dimension of used group bases, one can construct ciphers with up to $2^n!$ different keys which is the theoretically possible maximum for a block cipher with block length n.

3. The block length and key length are fully scalable. Thus, one can construct many different ciphers from the same specification. That makes the design more flexible and adaptable for the future.

4. A memory vs. security tradeoff is possible by modification of dimension of used group spaces. That makes it possible to adjust the cipher properties for a particular platform (e.g. a simple smart card vs. a powerful workstation).

The first representative of this class of ciphers was cryptosystem PGM. In spite of its excellent cryptographic properties it was not very suitable for practical use, because of its two major drawbacks - message expansion, and the complexity of the used knapsack transformation. The recent cryptosystem TST, which does not suffer from these problems, is still too new. There are two major areas of open questions that have to be answered:

1. *Security.* What are the cryptographic properties of TST? Is the cryptosystem secure? Which is the minimal dimension of group bases that provides a sufficient level of security? Which PRNG are suitable for key generation?

2. *Efficiency.* How suitable is TST for implementation on different platforms? What is its throughput and the initialization time - especially in software? How high are the memory requirements when using the cartesian and compact representations of permutations?

A thorough examination is necessary to answer these questions. The answers will help us obtain a concise view of TST and its properties, and will make it possible to compare TST with other modern block ciphers.

Chapter 4

Analysis of Cryptosystem TST

This chapter analyzes the efficiency and security of the symmetric cryptosystem TST.

In the first section we analyze the efficiency of both initialization and encryption routines of TST. We propose a possible implementation of the key generation scheme together with a suitable pseudorandom number generator, and we present the resulting key generation delay. Furthermore, we measure the memory requirements and encryption speed. Each measurement was done separately for the cartesian and compact representation of permutations to compare their practical suitability.

The second section describes our security analysis of the TST cryptosystem. We introduce a generic evaluation method and present the results achieved by the analysis of TST. Furthermore we examine possible attacks on TST, and in the third section we summarize the determined properties of TST and compare it with some other modern symmetric cryptosystems.

4.1 Efficiency

In what follows, we analyze the efficiency of TST. All presented measurements have been performed with our C++ implementation of TST on a personal computer with Intel Pentium II processor running at 350 MHz. In our implementation we preferred flexibility to speed[1], so the achieved encryption speeds are only approximate. We provide the values for both the cartesian and compact representations of permutations to make clear which one is more suitable for a software implementation. As the block length of TST is scalable, we measure all values for both nowadays common block lengths 64 and 128 bits. For comparison purposes we use the 64-bit cipher IDEA [LM91, LMM92] and the 128-bit cipher AES [DR98] as representatives of efficient and strong block ciphers.

[1]For example, the block length of the cipher as well as all properties of the group bases could be configured without recompilation. The resulting dynamic memory allocation had certainly an influence on the efficiency. Nevertheless, an implementation with fixed parameters would not be suitable for testing, in spite of its higher speed.

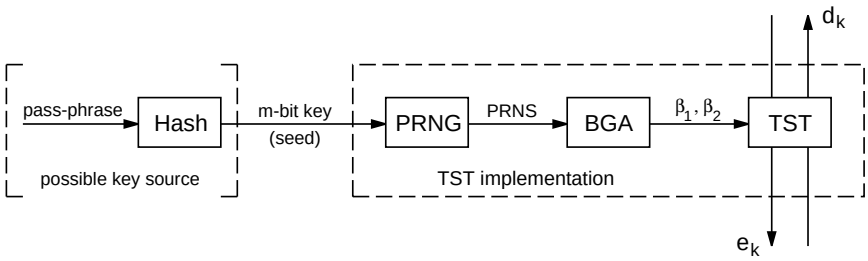


Figure 4.1: Key generation scheme for TST

Another factor which plays an important role in our measurements is the *average fusion extent* (AFE) of the used group bases. Let $\beta = (B_0, \ldots, B_{w-1})$ be a w-dimensional group basis of type $r = (r_0, \ldots, r_{w-1})$. The value

$$\frac{\sum_{i=0}^{w-1} \log_2(r_i)}{w}$$

is said to be the average fusion extent of β. AFE expresses how many blocks have been fused on average during a block fusion performed by BGA (see also Definition 3.2.2 and Section 3.3.2). For example, the average fusion extent of basis β_2 of type $r = (2, 4, 8, 4)$ presented in Example 3.3.1 is

$$\frac{\log_2(2) + \log_2(4) + \log_2(8) + \log_2(4)}{4} = 2.$$

A higher value of AFE contributes to better security, because the group bases become larger and the uncertainty for an attacker grows. On the other hand, the memory occupied by a group basis increases *exponentially* with a growing AFE. Hence, AFE is an interesting parameter, enabling us to perform memory vs. security trade-off.

Note that a variable AFE means a new degree of freedom in our experiments. For example, when determining memory requirements of an IDEA implementation, the result is a single number. When doing the same with TST, the result is rather a sequence of values for different AFE. Thus, the results for TST have usually the form of an exponential curve, while the appropriate values of IDEA and AES are constant.

4.1.1 Key Generation

In contrast to most "classical" cryptosystems, TST does not use a key in the form of an k-bit binary vector. Instead, the secret key consists of a pair of random group bases for a given carrier group $\mathcal{H}_s \times \mathcal{H}_1$. However, TST can be adapted to the standard k-bit key format as well, as shown in Figure 4.1. The two bases β_1, β_2 are generated by the basis generation algorithm (BGA) introduced in Section 3.3.2. A sequence of pseudorandom numbers (PRNS) generated by a pseudorandom number generator (PRNG) is used as a source of randomness within BGA. The PRNG is initialized with a k-bit binary seed. The value of the seed completely determines both generated bases and can therefore be used as a secret key K. The secret keys are either generated automatically (e.g. when used in communication protocols), or

else, can be entered directly by a user. In the second case the key can possibly be generated from a pass-phrase. This approach ensures a better distribution of the key values and better memorization by people.

4.1.1.1 Choice of a PRNG

The PRNG used in a TST implementation should possess at least the following three properties:

- *Sufficient number of possible seeds.* The seed length k should be at least 64 bits. Smaller values of k would not guarantee a sufficient size of the effective key space. For the sake of flexibility the PRNG should support a variable seed length k. The corresponding TST implementation will then have a *scalable key length*.
- *Good randomness properties of the output.* The generator should pass some non-trivial randomness tests, which will guarantee that the resulting group bases will be uniformly and randomly distributed over the complete $\mathcal{B}^*_{\mathcal{H}_s \times \mathcal{H}_1}$.
- *Efficiency.* In order to make the key initialization process as short as possible, the PRNG should be fast.

Among all possible PRNG candidates the first requirement immediately excludes all generators with 16 and 32-bit seeds, the second requirement excludes every simple PRNG (for instance the linear congruential generator, presented in Algorithm 2.2.1) and the third requirement disqualifies many cryptographically secure but slow generators (e.g. the Blum-Blum-Shub generator defined in [BBS87]).

In our implementation we decided to use the *lagged Fibonacci generator with Lütscher's approach*, described in [Knu97, pp.27–29,35,186–188]. The sequence generated by this PRNG is given by the recurrence

$$X_j = (X_{j-100} - X_{j-37}) \bmod 2^{32} \tag{4.1}$$

where the constants $(37, 100)$ - so-called *lags* - have been chosen very carefully. The period of the generated PRNS is guaranteed to be 2^{131} and the seed length can be scaled up to 3200 bits. These values can be further improved by changing the lags (see also [Knu97, p. 29]). The randomness of the generated Fibonacci sequence is strengthened by the Lütscher's approach which is motivated by chaos theory in dynamical systems. According to this approach, one should only use a few first numbers from a generated block of fixed length. For example, [Knu97, p. 188] suggests using the first 100 of every 1009 generated numbers. The resulting sequence $Y_1, Y_2, \ldots$ is then given by rule

$$Y_{100 \cdot k + j} = X_{1009 \cdot k + j}, \quad k = 0, 1, 2, \ldots \quad , j = 0, 1, \ldots, 99$$

where the values X_i are computed using the recurrence 4.1. The sequence $Y_0, Y_1, \ldots$ has very good statistical properties and can be generated at a rate of about 10 MB/s (i.e. 2.6×10^6 32-bit numbers per second).

4.1.1.2 Commutative Block Shuffle Algorithm

Another practical problem related to the TST key generation is a proper implementation of the commutative block shuffle (i.e. the transformation T1, defined in Definition 3.2.2). An important condition for preserving the transversality of a group basis is that only *commuting* adjacent block may be swapped during the shuffle. This can be ensured by a simple iterative algorithm.

Algorithm 4.1.1 Iterative Block Shuffling
Let $\beta = (B_0, B_1, \ldots, B_{w-1})$ be a group basis of a group G and let x be a fixed integer. The commutative block shuffle can be performed by the following algorithm:

```
input β
for i = 1,...,x do
    set a = Random(0, w-2)
    set b = a + 1
    if DoCommute(B_a, B_b) then
        call Swap(B_a, B_b)
    endif
endfor
output β
```

The function Random*(min, max) generates a random integer between min and max, the function* DoCommute*(B_a, B_b) returns true if the blocks B_a and B_b do commute and false otherwise, and the function* Swap*(B_a, B_b) swaps the blocks B_a and B_b in β.*

To ensure that every possible ordering of the blocks can appear on the output of the algorithm, x must be equal or higher than[2] $X_{min} = \frac{w}{2}(w-1)$. However, the swap operations tend to partially compensate each other and, hence, the number of repetitions that should be used to obtain all possible orderings *with the same probability* must be several times higher than X_{min}. Our simulations have shown that at least $x = 10 \times X_{min}$ should be used to ensure a good distribution of the outputs. For the purpose of these simulations we have defined the notion of a *distance* between two group bases.

Definition 4.1.1 Distance of Two Outputs of T1
Let $w = 2^s$ and let $\beta_1 = (B_0, \ldots, B_{w-1})$ and $\beta_2 = (B'_0, \ldots, B'_{w-1})$ be two group bases of type $r = (r_0, \ldots, r_{w-1})$ where every $r_i = 2$. Let $f^{(\beta_2)} : \mathbb{Z}_w \longrightarrow \mathbb{Z}_w$ be a function

[2] This is the maximum number of adjacent element swaps that are necessary to rearrange a fixed ordering of w elements to an arbitrary ordering.

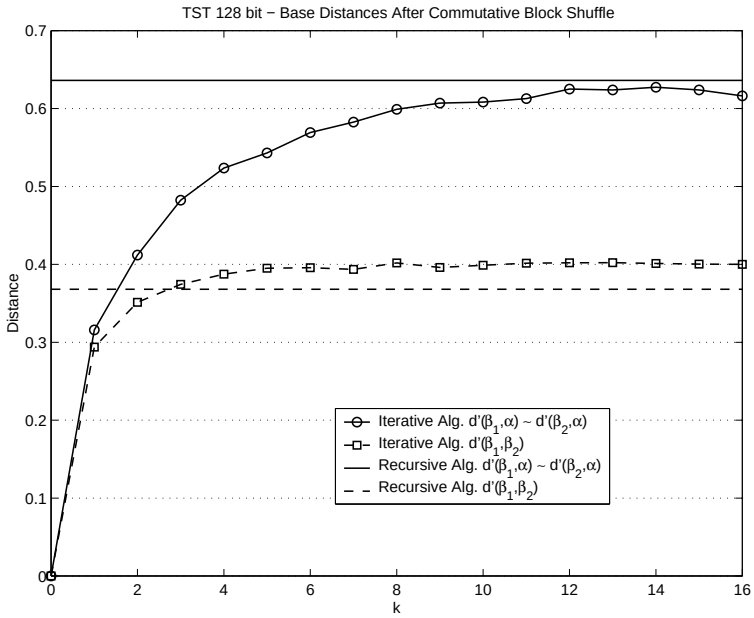


Figure 4.2: Iterative vs. recursive shuffling algorithm

which for every $x \in \mathbb{Z}_w$ *returns* $y \in \mathbb{Z}_w$ *such that the key-bit position of* B'_y *is* x*. We define the distance between* β_1 *and* β_2 *as*

$$d(\beta_1, \beta_2) = \sum_{i=0}^{w-1} |i - f^{(\beta_2)}(c_i)|$$

where c_i *is the key-bit position of block* B_i*.*

The maximum possible distance between two w-dimensional group bases of an arbitrary binary group of order 2^w is $d_{max} = \frac{w^2}{2}$. Thus, the function $d(\beta_1, \beta_2)$ can return only values between 0 and d_{max}. The distance 0 will occur when $\beta_1 = \beta_2$ and maximum distance d_{max} will be achieved when the blocks of β_1 and β_2 are in the reverse order. We also define a *normalized distance* $d'(\beta_1, \beta_2) = \frac{d(\beta_1,\beta_2)}{d_{max}}$ which will take on values between 0 and 1 for any dimension w.

Let β_1 and β_2 be two bases obtained from the canonical basis α by $x = k \times X_{min}$ cycles of the shuffling algorithm 4.1.1. The simulation results in Figure 4.2 show that $d'(\beta_1, \alpha)$, $d'(\beta_2, \alpha)$ as well as $d'(\beta_1, \beta_2)$ grow rapidly with an increasing k until $k \simeq 10$. For $k \gg 10$ the distances keep oscillating around some fixed values. Hence, if we want to obtain well distributed pairs of group bases using the iterative algorithm, we have to execute between $10 \times X_{min}$ and $20 \times X_{min}$ cycles. This approach is rather time-consuming.

The commutative block shuffle can be performed in a much more efficient way, using a recursive approach. Hereby we exploit the special structure of the carrier group $\mathcal{H}_s \times \mathcal{H}_1$. In this group the component $\mathcal{H}_s$ represented by the first $w-1$ blocks of the canonical group basis has a recursive structure (see also Definition 2.1.16).

The component $\mathcal{H}_1$ represented by the last block is guaranteed to commute with every other block.

Algorithm 4.1.2 Recursive Block Shuffling
Let $\alpha = (B_0, B_1, \ldots, B_{w-1})$, $w = 2^s$, be the canonical group basis of a group $\mathcal{H}_s \times \mathcal{H}_1$. The commutative block shuffle can be performed as follows:

set $\beta = \alpha$
call ShufflePart(0, $w - 2$)
set r = Random(0, $w - 1$)
call Insert(r, $w - 1$)
output β

Procedure Insert*(x, y) moves block B_y to the position x shifting the other blocks upwards by one position. The recursive procedure* ShufflePart*(i, l) is defined as follows:*

set $l' = (l - 1)/2$
if $l \leq 3$ then
    set r = Random(0, 1)
    if $r = 1$ then
        call Swap(B_i, B_{i+1})
    endif
else
    call ShufflePart(i, l')
    call ShufflePart($i + l'$, l')
    call Merge(i, $i + l'$, l')
endif

Procedure Merge*(a, b, l') randomly rearranges the blocks $B_a, \ldots, B_{a+l'-1}, B_b, \ldots, B_{b+l'-1}$ in such a way that neither the relative order of blocks in the lower half $\{B_a, \ldots, B_{a+l'-1}\}$ nor the relative order of blocks in the upper half $\{B_b, \ldots, B_{b+l'-1}\}$ is changed.*

The recursive Algorithm 4.1.2 generates bases with constantly good distribution, as can be seen in Figure 4.2. The distances $d'(\beta_1, \beta_2)$ and $d'(\beta_1, \alpha) \simeq d'(\beta_2, \alpha)$ of the generated group bases oscillate around values 0.636, and 0.368 respectively. Moreover, every possible basis is generated with the same probability and the generation time is 10 to 20 times shorter in comparison with the iterative Algorithm 4.1.1. Also the amount of consumed random numbers is significantly reduced. This contributes to the security of the key-generation process, because the probability that some weakness of the used PRNG might be exploited for an attack is significantly lower when using a PRNS of shorter length.

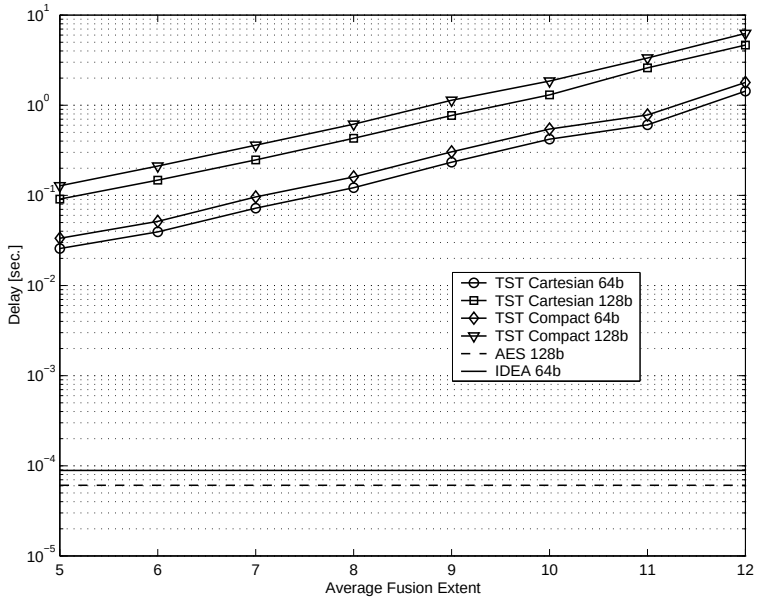


Figure 4.3: Key setup delay of TST

4.1.1.3 Key Setup Delay

The *key setup delay* is the time span that elapses between the point when a cipher receives a new key and the point when the cipher is ready to perform the first encryption. This delay is typically caused by an initialization of tables, precomputation of round keys, etc. The value is of minor importance when the cipher is only used for symmetric encryption, because the keys are usually not changed frequently. However, when using a block cipher as a cryptographic hash function (see for example [MVV97, Chap. 9]), a fast key setup significantly contributes to the achievable throughput.

Figure 4.3 displays the key setup delay for a 64-bit and a 128-bit version of TST in both the compact and cartesian representation of permutations. Apparently, the cartesian representation is only 1.3 to 1.4 times faster than the compact representation, but the impact of the block length on the delay is significantly stronger. The 64-bit version is almost four times faster than the corresponding 128-bit version.

We had expected that the key setup delay for TST would be noticeably longer than the delays of most other modern block ciphers, because the BGA operation during initialization is a complex task. Delays above 0.5 second are undesirable in most cases (even if the cipher is not used as a hash function) and, hence, AFE > 10 (respectively 8) is not recommendable with block length 64 b (respectively 128 b). As far as setup delays, when compared with modern block ciphers, TST is not competitive - not even for small AFE. The key setup of IDEA, for instance, is at least 300 times faster.

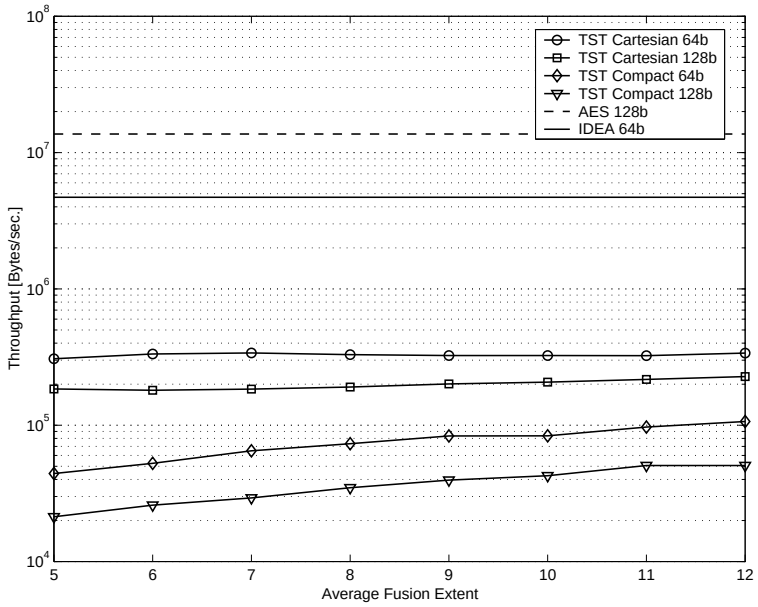


Figure 4.4: Encryption speed of TST

4.1.2 Throughput

Figure 4.4 displays the encryption speeds achieved by TST. The advantage of the cartesian over the compact representation becomes more significant than for the key setup. A speedup by factor up to 8 can be achieved when using the more efficient representation. Nevertheless, a comparison with other block ciphers is still disappointing. IDEA is on average 14.3 times faster a than a 64-bit TST and AES is even 70 (!) times faster than a 128-bit TST. While the key setup delay of TST has been at least partially suitable for practical use, the achieved throughput is completely insufficient. Encryption speed is, besides security, the second most important property of a cipher. A throughput of at least 2×10^6 B/s is desirable to make a block cipher interesting for practical use.

4.1.3 Memory Requirements

The memory requirements of TST grow exponentially with an increasing AFE. Figure 4.5 shows the minimal amount of memory that is needed for storing a TST key. Note that two bases of an n-bit TST occupy in compact representation at least

$$2 \cdot \frac{n}{AFE} \cdot 2^{AFE} \cdot \left\lceil \frac{n}{8} \right\rceil$$

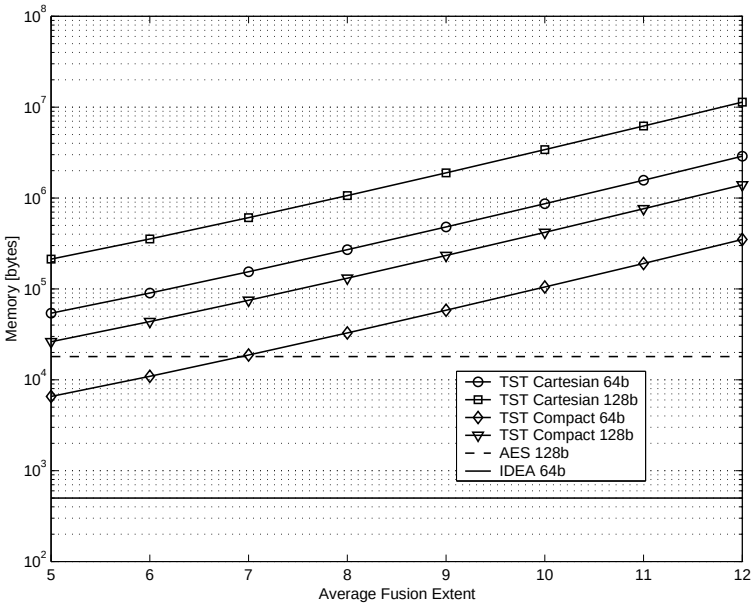


Figure 4.5: Memory requirements of TST

bytes of memory (i.e. 2 bases $\times \frac{n}{AFE}$ blocks $\times 2^{AFE}$ rows $\times \lceil \frac{n}{8} \rceil$ bytes per row)[3]. The same two bases will occupy at least

$$2 \cdot \frac{n}{AFE} \cdot 2^{AFE} \cdot \left\lceil \frac{(n+2) \cdot \lceil \log_2(n+2) \rceil}{8} \right\rceil$$

bytes in cartesian representation (i.e. roughly 8 times more space per row for $n \leq 256$). These values are rather optimistic, because a real implementation must in addition allocate some indexation arrays for fast factorization, a few temporary variables, etc. For example, the real memory requirements of our C++ implementation were about twice as big as the theoretically possible minimum.

It is desirable that a cipher requires an allocation of only a few tens of kilobytes of memory. Not only, because it will then fit completely into a fast cache memory of modern processors, but also because it is then implementable in restricted environments like smart-cards. We see that group bases of TST can fit into 100 KB only for maximum AFE values between 6 and 10, depending on the block length used and the representation. Unfortunately, the faster cartesian representation consumes about 8 times more memory than the slower compact representation. This property does not cause any problems on workstations, but makes TST less suitable for a smart-card implementation.

[3]The author in [Hor98] has shown that two bases created by a simplified version of BGA (used by the so-called Chip-TST) can be stored in only $\frac{n}{AFE} \cdot 2^{AFE} \cdot \frac{n-AFE}{8}$ bytes of memory, but here we want to analyze a full TST implementation.

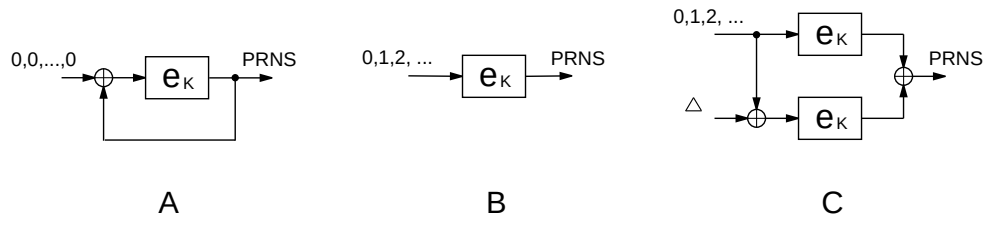


Figure 4.6: Setups for pseudorandom data generation

4.2 Security

In what follows, we provide a security analysis of TST. Hereby we apply both generic and white box evaluation methods. The basic idea of these approaches was already presented in Section 2.3.1.1.

4.2.1 Generic Statistical Evaluation

The purpose of this kind of statistical analysis is to determine, whether the cipher behaves as a strong pseudorandom number generator, i.e. whether its confusion and diffusion properties are strong. The evaluation process consists of three stages: acquisition of cryptographic data, acquisition of statistical data, and evaluation of statistical data (shown in Figure 2.2). In what follows, we describe the methodology which we have used in these three steps.

4.2.1.1 Acquisition of Cryptographic Data

The acquisition of cryptographic data can utilize any scheme which embeds the examined block cipher and produces a pseudorandom sequence. If the cipher is strong, the randomness of the produced sequence will be strong as well. Three examples of such schemes are presented in Figure 4.6. We denote the block length of the examined cipher by n. In setup A one encrypts a sequence of zeros (i.e. the plaintext blocks consisting of n zero bits) in the *cipher block chaining* mode (CBC) using a fixed key K. A good block cipher encrypts the zero block into a random block of n bits, $y_0 = e_K(0)$. This block is further encrypted into $y_1 = e_K(y_0)$ which is, supposing the cipher is strong, random again. By repeating this procedure l-times we obtain a sequence $y_0, y_1, \ldots, y_{l-1}$ which should be statistically indistinguishable from a sequence of l independent uniformly distributed numbers from $\mathbb{Z}_{2^n}$.

Setup B works without a feedback between the input and the output. When a non-periodic, highly redundant sequence of input blocks x_i is fed into the cipher, the output will be non periodic as well. And, because a good block cipher must destroy all redundancy of the input, the corresponding output will be indistinguishable from a sequence of pseudorandom numbers. The simplest (and most redundant) possible non-periodic input is a counter sequence $x_i = 0, 1, 2, \ldots, l$. The corresponding output sequence is $y_i = e_K(x_i)$. Many other input sequences are thinkable as well.

For example the authors in [SB00] use a *low density plaintext* sequence, which is the sequence where the first block consists of all zero bits, the next n different blocks have hamming weight one, the following $\binom{n}{2}$ different blocks have hamming weight 2, etc. The *high density plaintext* sequence, proposed in the same work, is created by a bitwise binary negation of all blocks of the low density sequence.

A slight modification of the setup B is a scheme producing the outputs $x_i \oplus e_K(x_i)$ for $i = 0, 1, \ldots, l$, where x_i is a highly redundant non-periodic sequence. However, when the input is really very redundant (e.g. the counter sequence above), the randomness properties of the output will not be very different from the ones of setup B.

Setup C utilizes the *avalanche property* of the examined cipher. The avalanche criterion says that for a good cipher a small change of an input should cause an unpredictable random change of the output, i.e. if we flip a single bit in a plaintext block x, every bit of the corresponding ciphertext block $y = e_K(x)$ should change with probability about one half. Setup C encrypts a highly redundant input sequence x_i, once without and once with a small change Δ. The hamming weight of Δ is usually chosen to be 1. The sequence of differences between the corresponding output blocks $y_i = e_K(x_i) \oplus e_K(x_i \oplus \Delta)$ should be pseudorandom if the avalanche effect of the tested cipher is strong.

There are many more possible data acquisition setups than just the three presented above. For instance, more examples can be found in [Hey97, Sec. 3.3]. It is important to say that different setups can have a very different sensitivity regarding detectable defects of the tested cipher. We show later that setup C is usually more sensitive than setup B which is furthermore significantly more sensitive than setup A. Because we want to test the quality of a cipher in its worst case, it is enough to consider the data obtained by the most sensitive of the tried setups.

4.2.1.2 Acquisition and Evaluation of Statistic Data

We have used the *DieHard* battery of statistical tests [Mar97] for our randomness testing. We decided to use DieHard, because it has proved itself during our experiments to be the most sensitive and most powerful tool among the freely available tests[4]. The test suite consists of 18 randomness tests, most of which are run with several different parameters. In total, 243 so-called p-values are produced for every tested sequence. These values should be uniformly distributed on $[0, 1)$ when the randomness of the tested sequence is strong. Both extremely high and extremely low p-values indicate a potential weakness of the sequence. The minimum length of an examined pseudorandom sequence is 8208 KB which is roughly 67×10^6 bits. For a more detailed description of the DieHard test suite see the Appendix B as well as the documentation provided in [Mar97].

In our tests we used 43 different random keys for every tested cipher. For each key a pseudorandom sequence of 8208 KB was generated, using setup B or C from

[4] DieHard detected more defects than, for example, the universal Maurer test and the FIPS 140-1 test battery.

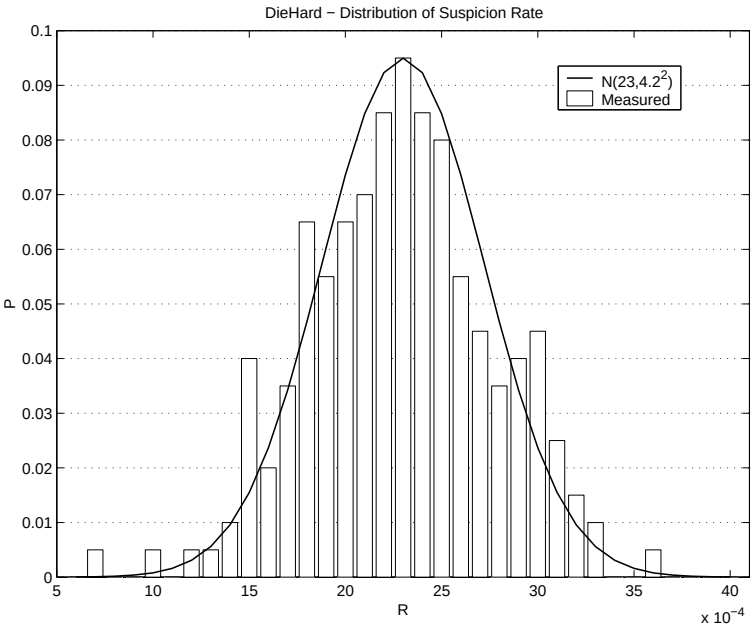


Figure 4.7: Expected distribution of suspicion rate

Figure 4.6. The sequence was tested with DieHard and the resulting p-values were classified as *passed* if $0.001 \leq p \leq 0.999$ and as *suspect* otherwise. Together $n_{total} = 43 \times 234$ results of type "suspect" or "passed" were obtained in this way. The *suspicion rate* $R = \frac{n_{susp}}{n_{total}}$, where n_{susp} is the number of suspect results, was measured for every cipher. Furthermore, to improve the sensitivity of method C, we measured the suspicion rate for several differences Δ and used the highest one found. The constant 43 as the number of tested sequences was chosen in order to obtain the suspicion rate with precision 0.0001 (i.e. $\frac{1}{n_{total}} \simeq 10^{-4}$).

The distribution of suspicion rates which can be expected for a strongly random input was obtained by extensive simulations. For generating a data sample that is *very close* to a truly random sequence we used setup A (Figure 4.6) together with the AES cipher. Figure 4.7 shows the distribution that was obtained by evaluating 200 suspicion rates, each measured on a group of 43 samples. One can see that the distribution is very similar to a normal distribution with mean $\mu = 0.0023$ and variance $\sigma^2 = 0.00042$. The randomness properties of a tested cipher can be considered as very good if the achieved suspicion rate is between 0.0006 and 0.0040. The probability that a good cipher will lead to a result lying outside of this interval is at most 0.001.

Let us demonstrate our cipher evaluation method on a simple example in Figure 4.8. The three thin dotted lines in the lower part of the graph mark the minimum (6×10^{-4}), the mean (2.3×10^{-3}) and the maximum (4×10^{-3}) for the interval of acceptable suspicion rates. All sequences leading to a suspicion rate in this interval can be considered as very strong. An example of a good result is the curve A which oscillates around the mean and never walks out of the interval. The other

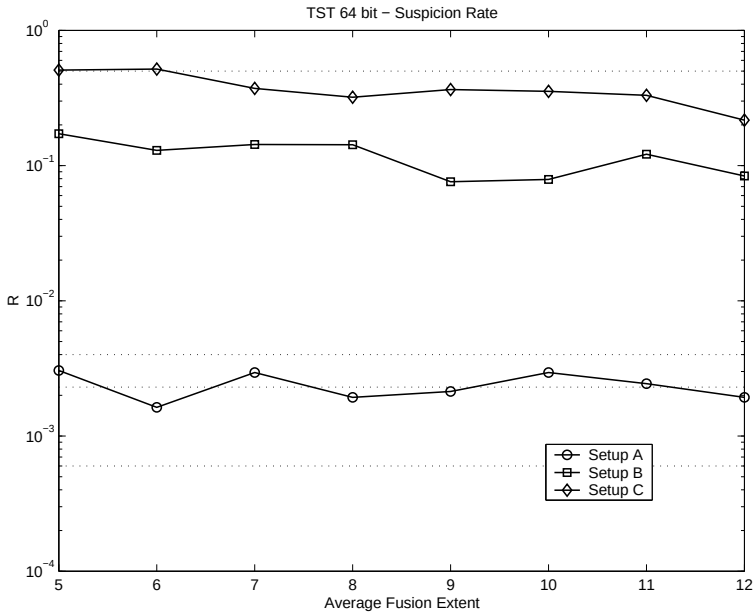


Figure 4.8: Example of suspicion rate profiles

two curves (B and C) are far above the good area and the corresponding sequences can therefore be classified as not random enough. The thin dotted line in the upper part of the graph ($R \simeq 0.5$) marks the level of randomness that can be achieved by a linear congruential generator (Algorithm 2.2.1). Such randomness is completely insufficient from the cryptographic point of view.

When we use the maximum of several suspicion rates achieved with different values of Δ, an optimal curve obtained by our experiments will not oscillate around the mean of the interval (as does the curve A), but rather slightly above it. Nevertheless, even in that setup a good PRNS will never lead to points lying above the interval of acceptable suspicion rates.

As already mentioned, the data acquisition setups in Figure 4.6 may result in different test sensitivities. Figure 4.8 demonstrates this fact in a very clear way. All three tests in the graph have been performed with the same cipher. It is obvious that setup A leading to a passed test has the lowest sensitivity because it did not detect any defects. Setup B is significantly more sensitive and leads to a failed test. The most sensitive setup C produces on average 3.3 times higher suspicion rates than setup B. Therefore we will preferably present the results achieved by setups B and C in our experiments.

Another typical behavior which can be observed in the example is the fact that the suspicion rate sinks when some security parameter is being increased. In our case an increasing AFE slightly improves the security properties of TST. An increasing *number of rounds* would have a similar effect if an iterative cipher were tested.

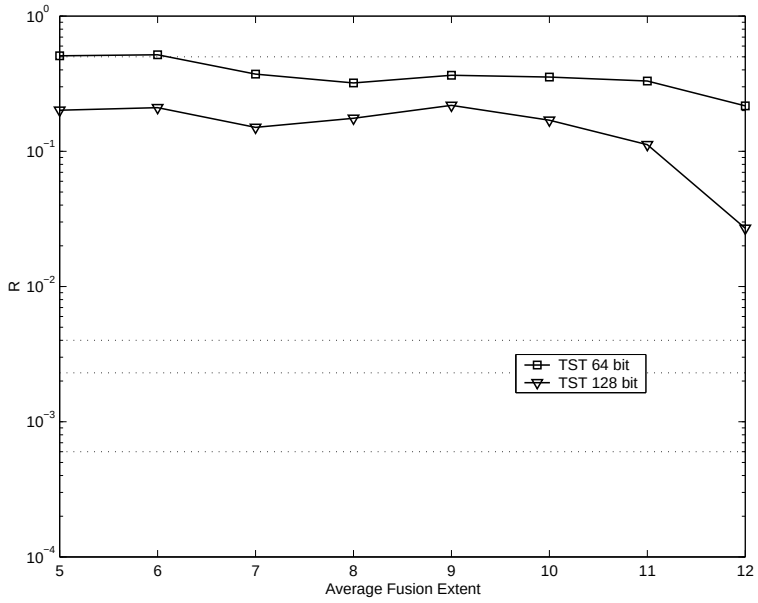


Figure 4.9: Suspicion rate profiles of TST

4.2.1.3 Results

Figure 4.9 shows the suspicion rates depending on AFE for both 64 and 128-bit TST. The statistical properties of TST are apparently *not good*. The 128-bit version is better than the 64-bit version, but still not good enough. The dropping of the curves is very weak. Even though the achieved results slightly improve with an increasing AFE, the safe interval can not be sufficiently reached in the examined range of AFE values.

4.2.2 White Box Analysis

In this section we will analyze the inner structure of TST. The exact knowledge of BGA enables us to estimate the potential size of the key space. Furthermore, we construct an attack by exploiting the redundancies in the structure of TST and its weak diffusion property.

4.2.2.1 Key Space

The effective key space of a TST implementation is given by the PRNG used. The theoretically possible key space, however, is limited only by the number of possible pairs of group bases that can be generated by the BGA. In what follows we will estimate this number.

The order of a carrier group $\mathcal{H}_s \times \mathcal{H}_1$ is equal to 2^{2^s}. The block size of the corresponding TST cipher is $n = 2^s$. The number of different outputs β_1 of a commutative

block shuffle (T1) executed by Algorithm 4.1.2 is

$$N'_s = 2^s \cdot N_s \tag{4.2}$$

where the values N_s are given by the recurrence

$$N_1 = 1, \quad N_s = \binom{2^s - 2}{2^{s-1} - 1} \cdot N_{s-1}^2. \tag{4.3}$$

This holds because the procedure Insert can modify the basis in 2^s different ways, and the procedure Merge in $\binom{2^s-2}{2^{s-1}-1}$ ways. The recurrent nature of the formula reflects the recursiveness of Algorithm 4.1.2.

Let us suppose that a block fusion (T2) performed on β_1 led to a basis β_2 of type $r = (r_0, \ldots, r_{w-1})$ where $w \leq 2^s$ and $r_i \geq 2$ for all i. The number of different ways in which a randomization (T3) on β_2 can be performed is given by

$$\prod_{i=1}^{w-1} \left[\prod_{j=0}^{i-1} r_j \right]^{r_i}. \tag{4.4}$$

A consequent element shuffle (T4) of β_3 can result into

$$\prod_{i=0}^{w-1} r_i! \tag{4.5}$$

different bases β_4.

As the randomization and the element shuffle are two independent operations, the total number of their combinations is the product of terms 4.4 and 4.5

$$N''_s = \prod_{i=0}^{w-1} r_i! \cdot \prod_{i=1}^{w-1} \left[\prod_{j=0}^{i-1} r_j \right]^{r_i} = r_0! \cdot \prod_{i=1}^{w-1} \left(r_i! \cdot \left[\prod_{j=0}^{i-1} r_j \right]^{r_i} \right). \tag{4.6}$$

The transformations T1,T2,T3,T4 altogether are not independent. Two different random sequences used during consecutive execution of T1 to T4 can lead to the same β_4. Because of these multiplicities, the total number N of possible bases β_4 can not be computed simply as a product of the results 4.2 and 4.6. Nevertheless, this product can be used as an upper bound for the value N.

$$N_{max} = N'_s \cdot N''_s \tag{4.7}$$

The appropriate lower bound for N is given by

$$N_{min} = \max\left\{ N'_s, \quad N''_s \right\}. \tag{4.8}$$

To give the reader an idea about the order of these number, we present an example.

Example 4.2.1 Estimation of the key space for a typical 64-bit TST

s	$n = 2^s$	$\lvert\mathcal{H}_s \times \mathcal{H}_1\rvert = 2^n$	N'_s
1	2	4	2
2	4	16	8
3	8	256	640
4	16	65536	3.51×10^8
5	32	4.29×10^9	2.39×10^{24}
6	64	1.84×10^{19}	1.67×10^{65}
7	128	3.40×10^{38}	5.25×10^{165}
8	256	1.16×10^{77}	6.23×10^{404}

Let, for instance, $s = 6$ *and* $r = (256, 256, 256, 256, 256, 256, 256, 256)$.

$$\begin{aligned} N''_6 &= r_0! \cdot \prod_{i=1}^{w-1} \left(r_i! \cdot \left[\prod_{j=0}^{i-1} r_j \right]^{r_i} \right) = 256! \cdot \prod_{i=1}^{7} \left(256! \cdot \left[\prod_{j=0}^{i-1} 256 \right]^{256} \right) \\ &= (256!)^8 \cdot 256^{256} \cdot 256^{2 \cdot 256} \cdot 256^{3 \cdot 256} \cdot 256^{4 \cdot 256} \cdot 256^{5 \cdot 256} \cdot 256^{6 \cdot 256} \cdot 256^{7 \cdot 256} \\ &= (256!)^8 \cdot 256^{256 \cdot (1+2+3+4+5+6+7)} = (256!)^8 \cdot 256^{7168} \\ &\simeq 5.38 \times 10^{21317} \end{aligned}$$

$$\begin{aligned} N_{min} &= \max\left\{ N'_6, \quad N''_6 \right\} \\ &= 5.38 \times 10^{21317} \\ &\approx 2^{70816} \end{aligned}$$

$$\begin{aligned} N_{max} &= N'_6 \cdot N''_6 = 1.67 \times 10^{65} \cdot 5.38 \times 10^{21317} \\ &\simeq 8.99 \times 10^{21382} \\ &\approx 2^{71032} \end{aligned}$$

It follows that there are between 2^{70816} *and* 2^{71032} *theoretically possible bases of type* r *which can be generated by the* BGA. *The corresponding key space consists of* two *such bases, i.e.* $|\mathcal{K}| = |\mathcal{B}^*_{\mathcal{H}_6 \times \mathcal{H}_1}|^2$. *Therefore,* 141632 *to* 142064 *bits would be necessary to represent a possible key.*

Note that conjugation (T5) has not been considered in the example. As there are 2^n different permutations in $\mathcal{H}_s \times \mathcal{H}_1$ the total number of bases after T5 could be as high as 2^n times the earlier estimate. The corresponding key length after T5 would be, hence, increased by up to $2n$ bits. Moreover, we have examined only one particular type r. The real number of bases of all possible types that can be generated by BGA is still higher.

4.2.2.2 An Attack on a Simplified TST

TST can not be easily attacked by a differential or linear cryptanalysis in the usual way, because these attacks are designed for iterative ciphers. We propose a new attack constructed especially for TST. We will present the basic idea of the attack on a simple TST instance with two 2-dimensional bases of type $r = (2^m, 2^m)$. To make the explanation simpler, we will suppose that no commutative block shuffle has been performed during BGA, i.e. the key bit positions are contiguous in every base block. Such a setup was, for example, used in *Chip-TST* proposed in [Hor98].

The encryption scheme A in Figure 4.10 displays the initial situation. The block length of the cipher is $n = 2m$, thus, a plaintext x can be considered as a concatenation of two m-bit halves a and b. Analogously, an n-bit ciphertext y consists of two m-bit halves a'' and b''. The two-dimensional group basis β_1, which is used for the factorization of x during an encryption, is displayed on the left hand side of the figure. It consists of two blocks - the lower one B_0 and the upper one B_1 - both of which can be stored in the compact representation as tables of $2^m \times 2m$ bits. The contents of B_0 can be divided into two continuous areas: the white area marked as 0 containing only zero bits, and the simple shaded area marked as P_3 containing 2^m different m-bit rows. Obviously, P_3 is a table representation of a permutation of 2^m elements. The columns of B_0 occupied by P_3 correspond to the key bit positions of B_0. The block B_1 of β_1 consists of two distinct areas as well. The simple shaded one (P_1) represents a permutation of 2^m elements and the double shaded one (S_1) contains $2^m \times m$ uniformly distributed random bits. Again, P_1 represents the key bits of B_1. The second basis β_2, used for the composition of y, is displayed on the right hand side of the figure. Because of the bit reversing operation R (described in Section 3.3) β_2 is turned upside-down. The structure of its two blocks, consisting of the parts 0, P_2, P_4, and S_2, is analogous to the one of β_1.

The two mappings represented by S_1 and S_2 are general $m \times m$-bit S-boxes (see e.g. Section 2.3) and the four bijective mappings (permutations) represented by P_1, P_2, P_3, and P_4 are $m \times m$-bit S-boxes of special structure. The binary operation $\otimes$ denotes the permutation product in the carrier group (performed by Algorithm 3.3.1) and the operation $\oslash$ denotes the inverse operation to $\otimes$, i.e. the "division" $u \oslash v = u \otimes v^{-1}$. All input, intermediate, and output values are binary vectors of $2m$ bits considered as compact representations of permutations in the carrier group. When an encryption is being performed, a plaintext $a||b$ is factorized with respect to the group basis β_1, and the corresponding ciphertext $a''||b''$ is composed using the basis β_2. The second division performed during factorization and the first multiplication performed during composition (both marked with dotted lines) do not need to be completed, because their outgoing resp. incoming value is guaranteed to be the identity permutation represented by a vector of $2m$ zero bits.

The described encryption scheme A can be translated into an equivalent, but more understandable, scheme B as follows. When looking for the first factor of $a||b$ with respect to β_1 according to Algorithm 3.3.2, we take that particular row of B_1 whose key bits (i.e. the right half corresponding to a row of P_1) are equal to b. This can be formally written as $x_1 = P_1^{-1}(b)$ and $(a'||0) = (a||b) \oslash (S_1(x_1)||b)$.

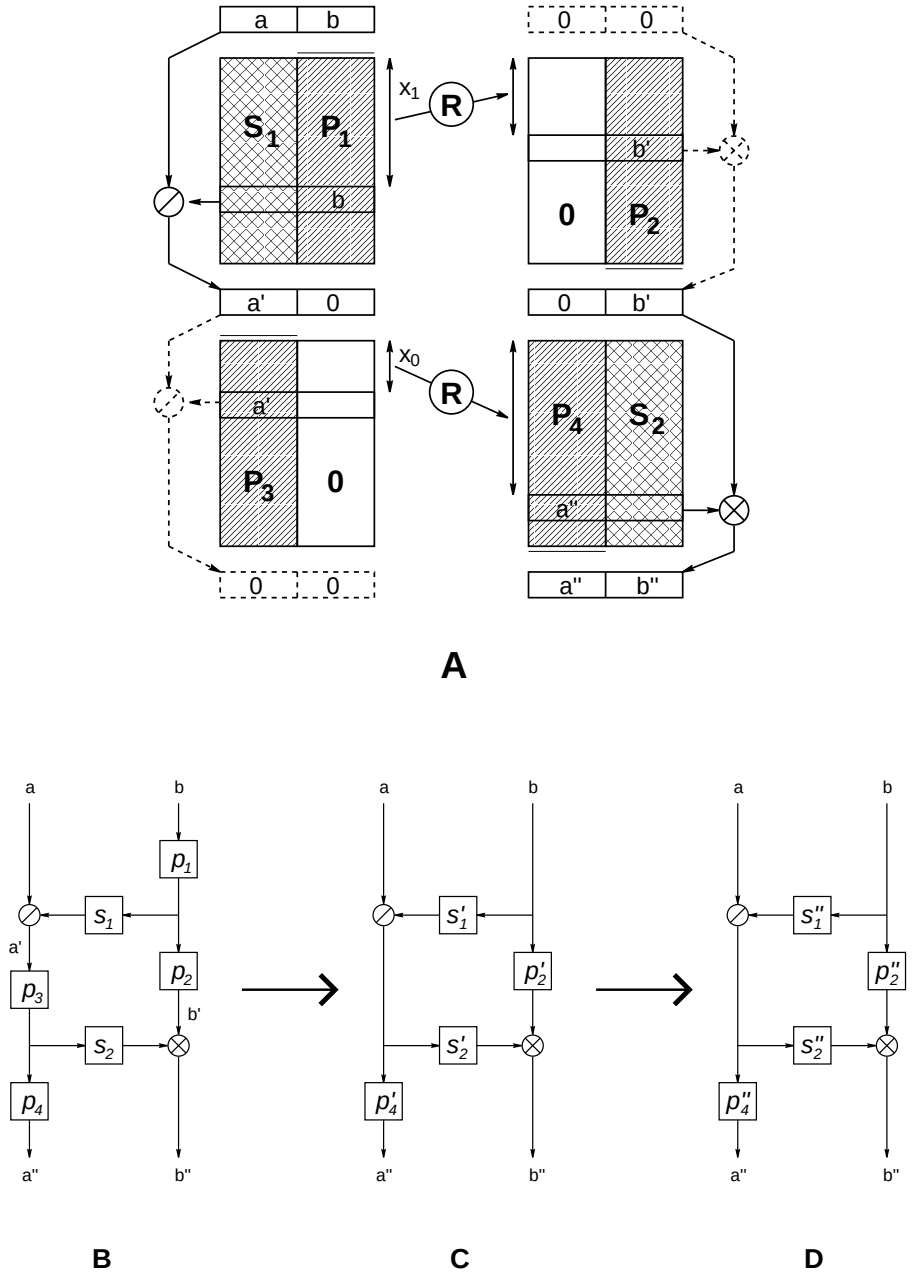


Figure 4.10: Analysis of TST

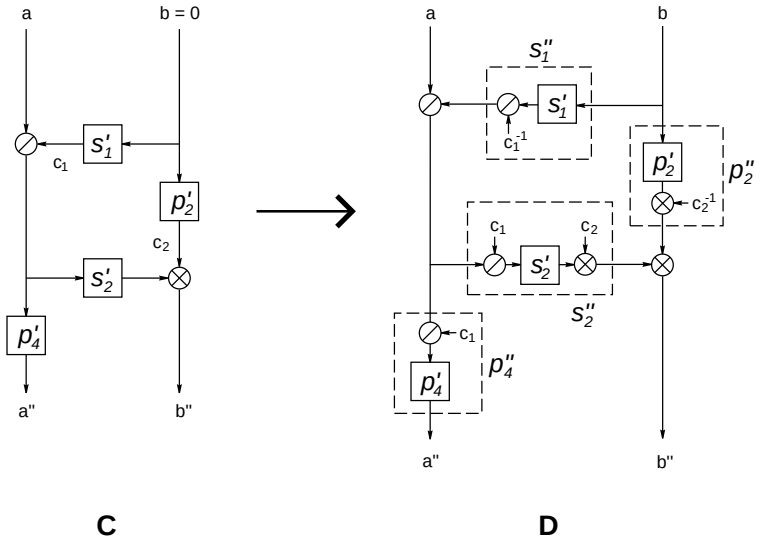


Figure 4.11: Equivalence of function quadruples

Furthermore, we reverse the coordinate x_1 and pass it to P_2 for a composition, i.e. $(b'||0) = (0||0) \otimes (P_2(R(x_1))||0)$. The functions P_1, S_1 and P_2 can be replaced by functions $p_1(x) = P_1^{-1}(x)$, $s_1(x) = S_1(x)$ and $p_2(x) = P_2(R(x))$ respectively, without changing the mapping $a||b \longrightarrow a'||b'$. An analogous transformation can be made with P_3, P_4 and S_2, without changing the mapping $a'||b' \longrightarrow a''||b''$.

The scheme B can be further simplified to scheme C by removing the permutations p_1 and p_3 and using $s'_1(x) = s_1(p_1(x))$, $p'_2(x) = p_2(p_1(x))$, $s'_2(x) = s_2(p_3(x))$, and $p'_4(x) = p_4(p_3(x))$, instead of s_1, p_2, s_2 and p_4 respectively. The mapping $a||b \longrightarrow a''||b''$ is still equivalent to the one defined by scheme A.

Now we can exploit the fact that scheme C is *redundant*, i.e. there are several possible quadruples of functions s'_1, s'_2, p'_2, p'_4 for every particular mapping $a||b \longrightarrow a''||b''$. For instance, the functions s'_1, s'_2, p'_2 and p'_4 in scheme C of Figure 4.10 can be replaced by functions $s''_1(x) = s'_1(x) \oslash c_1^{-1}$, $p''_2(x) = p'_2(x) \otimes c_2^{-1}$, $p''_4(x) = s'_4(x \oslash c_1)$, and $s''_2(x) = s'_2(x \oslash c_1) \otimes c_2$ respectively, where c_1 and c_2 are two arbitrarily chosen constants. This idea is demonstrated in Figure 4.11. One can see that because every group operation is associative, the operations $u \oslash c_1^{-1} \oslash c_1$, as well as $u \otimes c_2^{-1} \otimes c_2$ will compensate each other and the new scheme D will implement exactly the same mapping $a||b \longrightarrow a''||b''$ as the scheme C. Note that there is nothing special about the particular operations $\otimes$ and $\oslash$. For the purpose of our attack these can be basically *any* two group operations on $\{0,1\}^m$. Also note that there exists *no zero element* in a group (Definition 2.1.4), i.e. there is no $z \in G$, such that $z * x = z$ for all $x \in G$. Thus, the values c_1 and c_2 can be chosen to be truly arbitrarily, without any restrictions.

The equivalence of the schemes B, C, and D in Figure 4.10 implies that *we do not necessarily need to recover the original functions of scheme B, we rather find an equivalent quadruple of functions s''_1, s''_2, p''_2, p''_4 which implement the same map-*

ping. By reconstructing the tables representing s_1'', s_2'', p_2'' and p_4'' we can *completely recover the encryption function* (which can easily be inverted to the corresponding decryption function) and, hence, break the cipher. Hereby we do not need to know the exact values of c_1 and c_2, we just have to ensure that they are fixed.

The constants c_1 and c_2 used in the scheme D of Figure 4.11 can be chosen arbitrarily. Hence, we *decide* to use $c_1 = s_1'(0)$ and $c_2 = p_2'(0)$ which will ensure that $s_1''(0) = 0$ and $p''(0) = 0$. In other words, from the many possible equivalent quadruples of s_1'', s_2'', p_2'' and p_4'' we will look for that particular one, for which $s_1''(0) = 0$ and $p''(0) = 0$. (Such a quadruple of mappings does exist, as the values $c_1 = s_1'(0)$ and $c_2 = p_2'(0)$ do exist.) The scheme D can now be attacked in the following way.

First, let us fix $b = 0$ and encrypt all 2^m possible inputs. For every input $a_i||0$, where $a_i = 0, \ldots, (2^w - 1)$, we obtain an output $a_i''||b_i''$, such that $a_i'' = p_4''(a_i)$ and $b_i'' = s_2''(a_i)$. In other words, we are able to obtain all 2^m rows of the tables representing p_4'' and s_2'' with just 2^m trial encryptions. The appropriate s_1'' and p_2'' can be recovered in another 2^m steps. Hereby we fix $a = 0$ and perform a trial encryption for every possible input $0||b_i$. Using the corresponding outputs $a_i''||b_i''$, we can compute $s_1''(b_i) = p_4''^{-1}(a_i'')$ and $p_2''(b_i) = b_i'' \oslash s_2''(p_4''^{-1}(a_i''))$ for every possible $b_i = 0, \ldots, (2^w - 1)$, i.e. completely recover both s_2'' and p_4''.

All in all, we have shown that it is possible to completely reveal the encryption and decryption functions of the cipher with just 2^{m+1} trial encryptions. The memory space necessary for this attack is the same as the memory space required for a regular TST encryption.

4.2.2.3 Generalizations and Extensions of the Attack

The attack on TST with two 2-dimensional bases of type $r = (2^m, 2^m)$ can be extended to a more general class of TST representatives in a very straightforward way. The vulnerable class of TST ciphers includes all instances whose w-dimensional bases β_1 and β_2 of types $r = (r_0, r_1, \ldots, r_{w-1})$ and $r' = (r_{w-1}, r_{w-2}, \ldots, r_0)$ were constructed without a commutative block shuffle.

The extended attack works similarly as the basic version. First, a scheme with (unknown) S-boxes and permutations, equivalent to the original encryption function, is constructed in an analogous way as described above. Second, by consecutive parsing of all possible $\log_2(r_i)$-bit sub-blocks of the input, the corresponding S and P tables are reconstructed. The complexity of such attacks is only $\sum_{i=0}^{w-1} r_i$ trial encryptions. For instance, the simplified TST version, so-called Chip-TST, proposed in [Hor98] is vulnerable against these attacks and is, hence, not secure in spite of its extremely large key space.

Even if the bases of a TST implementation have been generated *with* a commutative block shuffle during BGA, a similar attack might still be possible. The only obstacle for mounting an attack in this case is the unknown ordering of the key

bit positions[5]. Once we know the key bit positions for particular blocks, we can partition the plaintext vector into w variables, each of length r_i. Even though these partitions are not continuous as, for instance, the parts a and b in the example above, we can parse them part by part, similarly as in the simple example above, and obtain the w^2 secret functions with just $\sum_{i=0}^{w-1} r_i$ trial encryptions.

The remaining question is, whether or not it is hard to obtain the key bit ordering. Unfortunately, it is relatively easy. Because of the special structure of the carrier group $\mathcal{H}_s \times \mathcal{H}_1$, some key bit positions are more probable to appear in the lower blocks, and some can not appear there at all. Suitable candidates for the r_{w-1} key bit positions of the last block can be found by means of statistics. In what follows, we describe one of the possible techniques.

When trying all n differences with weight 1 in setup C of Figure 4.6, the Δ values corresponding to the key bit positions of the last block of β_1 will produce the worst randomness. The reason for this behavior is the *weak avalanche effect* in TST described in Theorem 4.2.1.

Theorem 4.2.1 *Let G be a binary permutation group, let $\beta = (B_0, \ldots, B_{w-1})$ be a group basis for G generated by* BGA*, and let Δ be a fixed binary vector. Furthermore, let c_i denote the set of key bit positions of a block B_i of β. When two permutations p and $p' = p \oplus \Delta$ (both written in compact representation) have coordinates $(x_0, \ldots, x_{w-1}) = \widetilde{\beta}^{-1}(p)$ and $(x'_0, \ldots, x'_{w-1}) = \widetilde{\beta}^{-1}(p')$ respectively, then x_i is equal to x'_i for all $i > m$, where $m \in \mathbb{Z}_w$ is the maximal value for which Δ contains a non-zero bit on a position from c_m.*

Sketch of proof: The weak avalanche effect of a factorization is basically caused by the *recursive character* of the Sylow group $\mathcal{H}_s$ (Definition 2.1.16) which is the biggest component of the TST carrier group $\mathcal{H}_s \times \mathcal{H}_1$. The recursive structure of $\mathcal{H}_s$ has the following consequences:

1. Let us consider the operation $v \leftarrow v \otimes c$ in the compact representation of $\mathcal{H}_s \times \mathcal{H}_1$ (performed by Algorithm 2.1.1). Let both v and c be binary vectors (i.e. a compact representations of permutations from $\mathcal{H}_s \times \mathcal{H}_1$). When c is fixed and v varies through all possible values, the graph of dependencies between the bits v_i of v consists of one binary tree corresponding to the component $\mathcal{H}_s$ and one separated vertex corresponding to the component $\mathcal{H}_1$. Figure 4.12 A presents a sample graph of bit dependencies inside of v for $\mathcal{H}_3 \times \mathcal{H}_1$. Any change of a bit corresponding to a vertex v_i of the tree can only affect v_i itself and the bits which correspond to the descendant vertices of v_i. This follows from the way the Algorithm 2.1.1 is working.

   This dependency structure is rather sparse and consequently the diffusion of the group operation in $\mathcal{H}_s \times \mathcal{H}_1$ is weak. The desired "all on all" bit dependencies, which would provide the optimal diffusion, are displayed on the right

[5] More precisely, only the sets of key bits corresponding to the particular blocks need to be known - the exact order of key bits is not important.

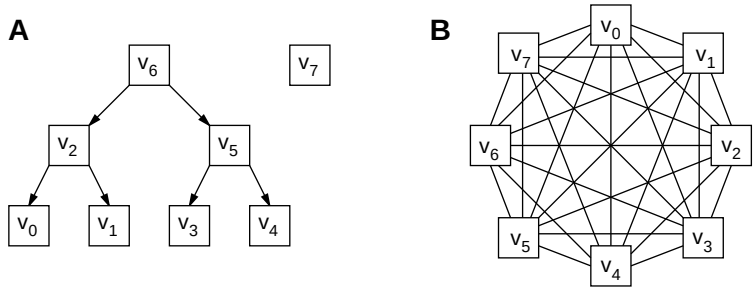


Figure 4.12: Bit dependencies during operation $v \leftarrow v \otimes c$ in $\mathcal{H}_3 \times \mathcal{H}_1$ and $\mathbb{Z}_{256}$

hand side of the figure. For example, the group operation '+ mod 256' on the set $\mathbb{Z}_{256}$ has the desired property. This means that a change of any bit of the input v can influence any bit of the output $v + c \bmod 256$ when v is variable and c is fixed.

2. When factorizing a permutation p with respect to a transversal group basis β, each factor x_i depends *only* on those bits of an intermediate result p_{i+1}, which correspond to the key bit positions of the block B_i of β. This follows from the Algorithm 3.3.2 (see also Example 3.3.2 for more details).

3. Commutative block shuffle is the only step of BGA which can affect the ordering of key bit positions in β. Commutative block shuffle is performed with the canonical basis α (Definition 3.3.2) which contains only blocks of the form $B_i = (id, p_i)$, where the compact representation of p_i contains a single non-zero bit on the i-th position. The key bit position of B_i is i. Only *commuting* blocks can be swapped during commutative block shuffle.

4. Two blocks (id, p_i) and (id, p_j) of α commute if and only if p_i and p_j commute. Furthermore, p_i and p_j commute if and only if the corresponding vertices v_i and v_j in the appropriate dependency graph (e.g. the one in Figure 4.12) are not a descendant of each other. For example, according to Figure 4.12, the permutations p_6 and p_3 in $\mathcal{H}_3 \times \mathcal{H}_1$ *do not* commute, because v_3 is a descendant of v_6. On the other hand, p_5 and p_1 *do* commute, because neither v_5 is a descendant of v_1, nor vice versa. This property follows from the particular structure of $\mathcal{H}_s$ which is discussed in [Hor98] in more detail.

5. It follows from observations 3 and 4 that no descendant can be moved above its ancestor during BGA. More precisely: Let c_i and c_j be the sets of key bit positions of blocks B_i and B_j of a basis β which was generated by BGA. If a vertex v_k corresponding to a key bit position $k \in c_i$ is an ancestor of a vertex v_l corresponding to $l \in c_j$, then $i \geq j$. (The equality is possible because of block fusion.)

6. The points 1, 2, and 5 together imply the Theorem 4.2.1.

□

According to Theorem 4.2.1, any change Δ of an input can only influence the coordinates produced by:

- the highest block whose key bits are included in Δ, and
- the blocks below.

When Δ contains non-zero bits only on the key bit positions of the lowest block B_0 of β_1, the factors x_i for all $i = (w-1), \ldots, 1$ will be fixed, and only x_0 will vary. Consequently, only the block B_0 of β_1 and the block B_{w-1} of β_2 will contribute to the randomness of the output, and all other blocks of both bases will be passed by. This simplification, enforceable by an adversary, makes it principally possible to reveal the key bit ordering and, consequently, to perform the attack on TST in spite of the commutative block shuffle used.

It has to be noted that even more sophisticated attacks on TST are thinkable. An attack might, for instance, exploit the fact that a particular PRNG used is not cryptographically secure. The exact internal status of the PRNG which we suggested in Section 4.1.1.1 can be reconstructed by knowing just 100 consecutive 32-bit numbers from the generated pseudorandom sequence. With this knowledge one can trace the complete sequence in both directions - the future as well as the previous values. This property of a PRNG would *not* cause any problems *if* TST itself were secure, but as soon as some weakness of TST enables an adversary to reveal a long enough consecutive part of any group basis, the other parts might be easily recomputed using the exact knowledge of the particular TST implementation together with the knowledge of the pseudorandom sequence. In this way the complexity of an attack (i.e. the number of needed trial encryptions) could further be reduced.

4.3 Summary

We have examined the efficiency and security properties of TST. Table 4.1 presents the efficiency comparison of TST with some of the fifteen AES candidates. The TST values (using AFE = 9) have been obtained with our C++ implementation, and the values of the AES candidates have been taken from [Gla99, ea99]. In both cases the examined block length was 128 bits. The times are stated in *clock cycles* on Intel Pentium II processor, which makes them comparable independently on the frequency of a particular processor used.

We have shown that the special structure of a TST key can be made transparent for a user, and the cipher can accept a key in the usual m-bit binary form. When using a proper PRNG, the key length of TST can be made fully scalable. The commutative block shuffle in BGA can be performed in several ways. The most efficient one is the recursive Algorithm 4.1.2 which is about 10 to 20 times faster than a simple iterative algorithm. In spite of this speedup, the complete basis generation procedure is still rather complex and the corresponding key setup delay of TST is significantly longer than with the other block ciphers. Nevertheless, when choosing AFE ≤ 10, the key setup delays of TST are acceptable.

The cartesian representation of permutations speeds up the encryption in comparison with the compact representation roughly by factor 6. But even then, the

Cipher	**Key Setup Time**†	**Encryption Time**†
TST Compact	398×10^6	141526
TST Cartesian	269×10^6	27882
Magenta	30	6539
Frog	1.4×10^6	2417
Loki 97	7430	2134
Safer+	4278	1722
Rijndael (AES)	305, 1389‡	374
RC6	1632	270

† Measured in clock cycles on Intel Pentium II processor

‡ Key setup delays for encryption and decryption are different

Table 4.1: Efficiency of TST in comparison with other ciphers

encryption speed of TST is still not competitive with the most modern block ciphers. The throughput of about 200 KB/s is completely insufficient. The winner of the AES competition *Rijndael* encrypts about 14 MB/s on the same platform.

The memory requirements of TST grow exponentially with an increasing AFE, but can be held in a sensible range when AFE ≤ 10. Memory requirements of up to 1 MB are feasible on workstations, but are unsuitable for smart cards.

The most serious concern about TST relates to its security. Even if the size of the potential key space of TST is astronomically large, the statistical randomness properties of the cipher are not good. The quality of encryption slightly improves with an increasing AFE, but can not reach a sufficient level of security in an acceptable range of AFE. Moreover, a chosen-plaintext attack with complexity only $\sum_{i=0}^{w-1} r_i$ can be mounted for simplified TST versions. Hereby the weak diffusion in the carrier group is exploited. The presented attack idea appears to be further extendable to a general TST, and some more sophisticated attacks based on the used PRNG might further be possible.

All in all, there is a need for improvements in the design of TST which would lead to higher security and, possibly, also to better efficiency.

Chapter 5

A New Cryptosystem Based on Extended Group Bases

This chapter introduces a novel idea of *extended group bases* and proposes a new TST-like symmetric cryptosystem utilizing this class of bases. The main aim by designing the new cryptosystem was to strengthen TST by improving its diffusion properties.

In the first section we introduce the notion of extended group basis as a generalization of the group basis for a permutation group. We explain the basic idea and the theoretical concepts, and discuss some possibilities of effective implementation.

In the second section we describe a new symmetric cryptosystem TST' based on the extended group bases of binary groups. Because of its improved diffusion properties, the new cryptosystem makes it possible to use a new carrier group $\mathbb{Z}_2^n$ in addition to the original $\mathcal{H}_s \times \mathcal{H}_1$. Owing to its commutativity and simpler structure, the group $\mathbb{Z}_2^n$ enables a faster and more flexible realization of the TST encryption idea.

We present an efficiency analysis of the new cryptosystem in the third section and a security analysis in the fourth section of this chapter. In both cases we use the same methodology which has already been used for the TST analysis in Chapter 4. The obtained results are therefore directly comparable, and the advantages and drawbacks of the two cryptosystems can be effectively evaluated. The resulting comparison of TST with TST' as well as a summary of the properties of the new cryptosystem is presented in the final fifth section.

5.1 Extended Group Bases

The notion of an *extended group basis* generalizes the idea of a group basis for permutation group.

Definition 5.1.1 Extended Group Basis
Let G be a permutation group and let $\mathcal{T}$ be a set of bijective transformations on G, $\mathcal{T} = \{T_i : G \longrightarrow G, 0 \leq i < w\}$. A $\mathcal{T}$-extended group basis for G of type $r = (r_0, \ldots, r_{w-1})$ is a pair $\beta^{(\mathcal{T})} = (\beta, \mathcal{T})$, where $\beta = (B_0, B_1, \ldots, B_{w-1})$ is an ordered collection of ordered subsets $B_i = (b_{i,0}, b_{i,1}, \ldots, b_{i,r_i-1})$ of G, such that each element $p \in G$ can be expressed uniquely as a product of the form

$$p = id \otimes^{(0)} b_{0,x_0} \otimes^{(1)} b_{1,x_1} \otimes^{(2)} \cdots \otimes^{(w-1)} b_{w-1,x_w-1}, \quad b_{i,x_i} \in B_i.$$

The operations $\otimes^{(i)}$ on G are defined as $a \otimes^{(i)} b = T_i(a * b)$, $a, b \in G$, where $*$ is the usual permutation composition on G.

In the special case when $T_i(g) = g$ for all $g \in G$, $i \in \mathbb{Z}_w$, we obtain the usual group basis as given in Definition 2.1.9.

The composition of a permutation $p \in G$ from a coordinate vector $x = (x_0, \ldots, x_{w-1})$ with respect to an extended group basis $\beta^{(\mathcal{T})}$ is denoted by $\tilde{\beta}^{(\mathcal{T})}(x) = p$. The inverse operation - factorization of p with respect to $\beta^{(\mathcal{T})}$ - is denoted by $\tilde{\beta}^{(\mathcal{T})-1}(p) = x$. The binary operation inverse to $\otimes^{(i)}$, which is necessary for performing a factorization, will be denoted by $\oslash^{(i)}$. This operation is defined as $a \oslash^{(i)} b = T_i^{-1}(a) * b^{-1}$ for all $a, b \in G$.

The transversal group bases are of special interest for the construction of symmetric cryptosystems, because they are guaranteed to be tame. When a group basis β of a group G is transversal with respect to a chain of subgroups $G_0 < G_1 < \cdots < G_{w-1} = G$ (see also Definition 2.1.11), and the set of transformations $\mathcal{T}$ of the corresponding extended group basis $\beta^{(\mathcal{T})}$, having the blocks identical with β, contains only transformations of the form $T_i : G_i \longrightarrow G_i$ for $0 \leq i < w$ with polynomial complexity, then $\beta^{(\mathcal{T})}$ is tame as well. Moreover, if β is supertame and the complexity of all T_i is $O(n)$, the resulting $\beta^{(\mathcal{T})}$ is supertame. This is ensured, because the equality of the blocks of β and $\beta^{(\mathcal{T})}$ makes the factor lookup operation equally fast in both bases. Efficient transformations T_i with linear complexity are discussed in more detail in Section 5.2.2.

TST-like cryptosystems employ only binary permutation groups to avoid a plaintext expansion which was typical for PGM. A permutation p of a binary permutation group G of order 2^n can be written in the compact representation as an n-bit binary vector, so the transformations $T_i : G_i \longrightarrow G_i$ can be in this case considered as transformations on a set of binary vectors, $T_i : \{0,1\}^{\log_2(|G_i|)} \longrightarrow \{0,1\}^{\log_2(|G_i|)}$.

The improved generality of extended group bases can be illustrated with a geometric coordinate system similarly as in Figures 2.1 and 3.1. Let P_i denote a group of i points. In Figure 5.1 we show a factorization with respect to a 2-dimensional extended group basis $\beta^{(\mathcal{T})}$. The chain of subgroups in our example is $P_4 < P_{16} = G$. When we factorize the point 5, we undo the transformation T_1 first. T_1 is nothing else than a permutation of 16 elements, and T_1^{-1} maps 5 on 14. Thus, we will look for the second coordinate of point 14 in P_{16}. We do it in the same way as in a "non-extended" 2-dimensional coordinate system, which will lead us to the coordinate 3. The intermediate element from the next subgroup P_4 is the point 2. We undo the

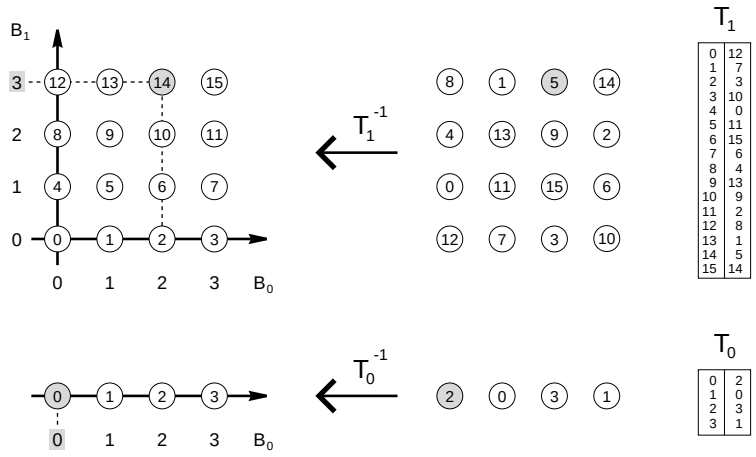


Figure 5.1: Factorization with respect to an extended group basis

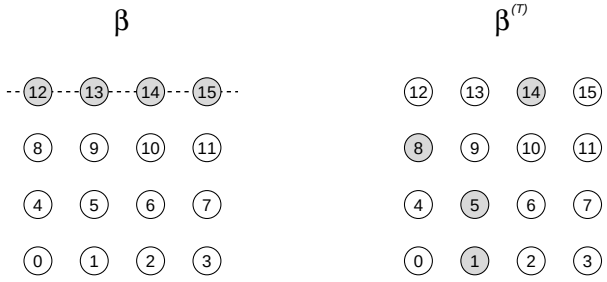


Figure 5.2: Points whose second coordinate is 3

transformation T_0 (a permutation of four elements) and obtain the point 0 which leads us to the first factor 0. It follows that the coordinate vector of point 5 with respect to $\beta^{(\mathcal{T})}$ is $(0, 3)$.

We see that the transformations T_i in fact do not increase the complexity of the factorization algorithm. However, they do increase the complexity of the dependencies between the points and their coordinates with respect to $\beta^{(\mathcal{T})}$. While the second coordinate in a usual group basis β depends *exclusively* on the vertical position of the point, the same coordinate in the extended group basis $\beta^{(\mathcal{T})}$ depends on *both* vertical and horizontal position of the point. Figure 5.2 demonstrates this property on the points whose second coordinate is 3. When using the usual group basis β, different inputs (i.e. points) belonging to the same coset (i.e. vertical position) have always the same coordinate on the particular position (i.e. the second coordinate in our example). With $\beta^{(\mathcal{T})}$ any small change of the input can influence all its coordinates. It follows that the *diffusion* properties of the function $\tilde{\beta}^{(\mathcal{T})-1}$ (as well as $\tilde{\beta}^{(\mathcal{T})}$) have been improved. Note that this effect can not be achieved by any of the BGA steps[1] and it brings us a new cryptographic quality. An extended group basis

[1]More precisely, an equivalent effect can be achieved by an element shuffle (transformation T4 in Definition 3.2.2), but *only for the lowest coordinate* of a point. Any higher coordinates cannot be affected in this way.

combines the fast factorization properties of a transversal group basis for a binary group with a high complexity[2] which is typical for the full symmetric group.

5.2 Cryptosystem TST'

We introduce a new cryptosystem TST' based on the principle of point mapping (Definition 3.1.1). The encryption setup is the same as the one of the cryptosystem TST (Section 3.3), but in contrast to TST, the new cryptosystem will use extended group bases. The corresponding encryption and decryption functions are:

$$y = e_K(x) = \lambda^{-1}(\tilde{\beta_2}^{(\mathcal{T})}(R(\tilde{\beta_1}^{(\mathcal{T})-1}(\lambda(x)))))$$

and

$$x = d_K(y) = \lambda^{-1}(\tilde{\beta_1}^{(\mathcal{T})}(R(\tilde{\beta_2}^{(\mathcal{T})-1}(\lambda(y))))).$$

Analogously to TST, the secret key is a random pair of extended group bases, $K = (\beta_1^{(\mathcal{T})}, \beta_2^{(\mathcal{T})})$. The decryption function is basically the same as the encryption function, only the roles of $\beta_1^{(\mathcal{T})}$ and $\beta_2^{(\mathcal{T})}$ are swapped. The functions λ and λ^{-1} do not require any computations, exactly as the ones of TST.

5.2.1 Carrier Group

The binary carrier group group $\mathcal{H}_s \times \mathcal{H}_1$ utilized by TST can be used with TST' as well. However, because of their improved diffusion properties, the extended group bases can also be combined with some simpler binary groups. As the diffusion of TST was ensured only by the group operation $*$ in the carrier group G, the complexity of G had to be maximized (see also Section 3.3.1). Unfortunately, this had a bad impact on the efficiency of TST, because the operation $*$ in G became (especially in software) rather time consuming. On the other hand, the major contribution to the diffusion of TST' is given by the transformations T_i, not by the carrier group. Thus, one can prefer the efficiency to complexity when choosing a carrier group for TST'. The simplest binary permutation group available is the elementary Abelian group $\mathbb{Z}_2^n$ introduced in Section 2.1.3. This carrier group has a couple of very attractive properties.

We have shown that a product of two permutations $a*b$ in $\mathbb{Z}_2^n$ can be computed as a single XOR of their compact representations. This operation is significantly faster than the permutation composition in $\mathcal{H}_s \times \mathcal{H}_1$, performed by Algorithm 2.1.1. Hence, we expect that the efficiency of TST' based on $\mathbb{Z}_2^n$ will be higher. Also a scaling of the block length n can be done with a better granularity when using $\mathbb{Z}_2^n$. In contrast to the group $\mathcal{H}_s \times \mathcal{H}_1$, which enabled only block lengths of the form $n = 2^s$, $s \in \mathbb{N}$, the group $\mathbb{Z}_2^n$ makes it possible to use any $n \in \mathbb{N}$. For instance, the block length 192 bits, whose supporting was originally intended as a condition for the AES candidates, can easily be achieved by $\mathbb{Z}_2^n$, but not by $\mathcal{H}_s \times \mathcal{H}_1$. Moreover, $\mathbb{Z}_2^n$ is a commutative

[2]Note that the complexity of the bit dependencies, as discussed in Figure 4.12, is "all on all" for the operations $\oslash^{(i)}$ and $\otimes^{(i)}$.

group, which means that the block swaps during a commutative block shuffle can be performed without any restrictions by a very straightforward algorithm. Not only the problems described in Section 4.1.1.2 are not relevant anymore, but also the number of possible bases which can be generated by the BGA increases. This means a larger key space and a higher uncertainty for an adversary. Last but not least, the dilemma between the compact and the cartesian representation of permutations is not present when using $\mathbb{Z}_2^n$. When implementing TST, we had to decide, whether to use the cartesian or the compact representation of $\mathcal{H}_s \times \mathcal{H}_1$. The first one was significantly faster but consumed significantly more memory, the second one was shorter but slow. This contrast does not appear with TST' based on $\mathbb{Z}_2^n$, because in this case the compact representation is both faster and shorter.

5.2.2 Family of Transformations $\mathcal{T}$

Definition 5.1.1 does not restrict the form and the properties of transformations T_i in any way. We have shown that it is advisable to use transformations of the form $T_i : \{0,1\}^{\log_2(|G_i|)} \longrightarrow \{0,1\}^{\log_2(|G_i|)}$ together with an extended transversal binary carrier group, in order to make the factorization efficient. Nevertheless, some more properties of T_i might be recommendable from the cryptographic point of view. In what follows, we will discuss the most suitable form of T_i in more detail.

Let G be a binary permutation group of order 2^n and let $\beta = (B_0, \ldots, B_{w-1})$ be a w-dimensional group basis for G of type $r = (r_0, \ldots, r_{w-1})$. The size of c_i (the set of key bit positions of block B_i) is $|c_i| = \log_2(r_i)$. We denote the i-th bit of a binary vector x as $x^{(i)}$, and we define a function $\gamma : \{0,1\}^n \times \mathbb{Z}_n^k \longrightarrow \{0,1\}^k$ as the function which extracts the specified bits from an n-bit binary vector. For example, $\gamma(001\mathbf{01}011, \{3,4,7\}) = 011$. Another function $\delta : \{0,1\}^n \times \mathbb{Z}_n^k \times \{0,1\}^k \longrightarrow \{0,1\}^n$ initializes the specified bits of a binary vector with specified values. For example, $\delta(00101011, \{3,4,7\}, 000) = 001\mathbf{00}01\mathbf{0}$.

When a plaintext p (considered as a permutation written in the compact form) is encrypted with TST, p is first factorized with respect to β_1, and the resulting coordinates are passed to β_2. During the i-th factorization step an intermediate result p_{i+1} is divided by a factor b_{i,x_i} from the block B_i of β_1. The coordinate x_i is passed to β_2, and the intermediate result $p_i = p_{i+1} \oslash b_{i,x_i}$ is further factorized in the next lower block B_{i-1} of β_1. This procedure is continued, until $p_0 = id$ is reached. In order to create an optimal diffusion, every coordinate x_i should depend on all variable bits of the intermediate result p_{i+1}. Because x_i depends *only* on those bits of $p_{i+1}^{(j)}$ of p_{i+1} for which $j \in c_i$ (see Algorithm 3.3.2 for more details), an optimal transformation T_i should make these bits dependent on *all* variable bits of p_i. T_i does not need to change any bits $p_{i+1}^{(j)}$ such that $j \notin c_i$.

More precisely, an optimal T_i has the form $T_i(p_{i+1}) = \delta(p_{i+1}, c_i, h(p_{i+1}))$, where the function $h : \{0,1\}^n \longrightarrow \{0,1\}^{\log_2(r_i)}$ ensures that every bit of an output $y = h(x) \in \{0,1\}^{\log_2(r_i)}$ depends non-idly on every bit of the input $x \in \{0,1\}^n$. Such a function is usually called an n-to-$\log_2(r_i)$-bit hash function. There exist many possible n-to-m-bit hash functions, for instance, a simple XOR-*checksum*, a *cyclic*

redundancy code (CRC), and many others. A hash function can implement a fixed or, if appropriate, possibly also a key-dependent mapping.

A hash function $h : \{0,1\}^n \longrightarrow \{0,1\}^m$, $m \leq n$, suitable for construction of T_i should possess at least the following five properties:

1. The composite function $\delta(p, c, h(p))$ must be invertible for all $p \in \{0,1\}^n$, $c \in \mathbb{Z}_n^*$ (where $\mathbb{Z}_n^*$ denotes all possible subsets of $\mathbb{Z}_n$).
2. The computation of $h(x)$ should not be time consuming.
3. Every bit of the output should depend on every bit of the input, i.e. for every $i \in \mathbb{Z}_n$, $j \in \mathbb{Z}_m$ there should exist two inputs $x_1, x_2 \in \{0,1\}^n$, such that $x_1^{(i)} \neq x_2^{(i)}$ and $h(x_1)^{(j)} \neq h(x_2)^{(j)}$
4. Every output bit should be balanced, i.e. the value $\frac{\sum_{x \in \{0,1\}^n} h(x)^{(i)}}{2^n}$ should be approximately equal to $\frac{1}{2}$ for every $i \in \mathbb{Z}_n$.
5. The function should not be affine (linear), i.e. it should not be true that for every $j \in \mathbb{Z}_m$ there exists a combination of coefficients $l_i \in \{0,1\}$, $i \in \mathbb{Z}_{n+1}$, such that for any $x \in \{0,1\}^n$ it holds $h(x)^{(j)} = l_0 + \sum_{i=1}^{n} l_i \cdot x^{(i-1)}$.

The need for properties 1 and 2 is obvious. If the composite function $\delta(p, c, h(p))$ were not invertible, we would not be able to decrypt, and if the computation of $h(x)$ were too complex, it would non-necessarily slow down the cipher. The third property ensures an optimal diffusion, because it guarantees that a change of any input bit can influence all output bits. The fourth property ensures that the function has no strong biasses which might be exploited for a cryptanalysis. Finally, the fifth property contributes to a better confusion, improving the non-linearity of the factorization in the carrier group. More theory on cryptographically strong functions and their construction can be found e.g. in [AT90].

The invertibility of the transformation $T_i(p_{i+1}) = \delta(p_{i+1}, c_i, h(p_{i+1}))$ can be ensured in a simple and efficient way by using a T_i of the form

$$p'_{i+1} = T_i(p_{i+1}) = \delta(p_{i+1}, c_i, h(\gamma(p_{i+1}, \mathbb{Z}_n \setminus c_i)) \oplus \gamma(p_{i+1}, c_i)) \tag{5.1}$$

as shown on the right hand side of Figure 5.3. This refinement ensures that the inverse transformation T_i^{-1} can be computed in the following way

$$p_{i+1} = T_i^{-1}(p'_{i+1}) = \delta(p'_{i+1}, c_i, h(\gamma(p'_{i+1}, \mathbb{Z}_n \setminus c_i)) \oplus \gamma(p'_{i+1}, c_i)) \tag{5.2}$$

independently of the properties of function h. The operation $\oplus$ can be basically any invertible binary operation on the set of binary vectors.

Because T_i in the form 5.1 is always invertible, we only have to ensure that our n-to-m-bit hash function h is efficient, strongly diffusive, balanced and non-linear. This can, for example, be achieved by chaining a simple non-linear m on m bit bijective function p, as shown in Figure 5.4. The operation $\otimes$ can be any efficiently computable group operation on $\{0,1\}^m$, e.g. bitwise XOR, addition modulo 2^m, etc.

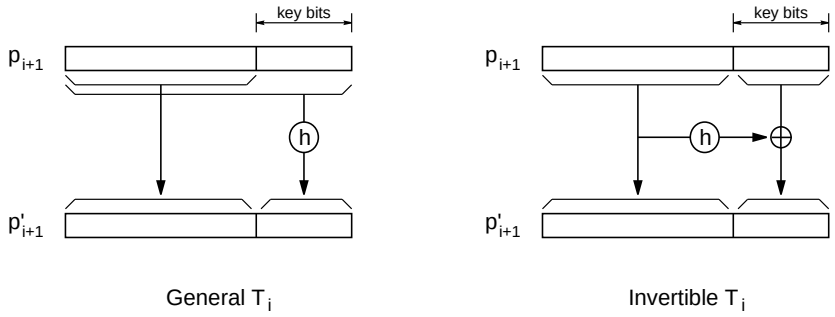


Figure 5.3: General vs. invertible T_i

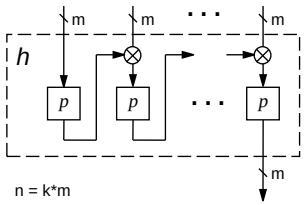


Figure 5.4: Possible implementation of h

When n is not a multiple of m, the last few input bits of h constructed according to Figure 5.4 can not be initialized from an input x. (Note that x has only n bits, while h expects $\lceil \frac{n}{m} \rceil \times m$ bits.) In such a case the remaining input bits of h should be constantly set to zeros to make the function deterministic. Among other possible, here are some examples of functions p, some of them key dependent, some not.

- $p(x) = rot_c(x)$, where rot_c denotes a rotation of the argument by c positions and c is relatively prime to m
- $p(x) = c{\cdot}x \pmod{2^m}$, where c is an m-bit prime containing roughly $\frac{m}{2}$ non-zero bits
- $p(x) = x(2x+1) \pmod{2^m}$, this function has, for instance, been used in RC6 [RRSY98]
- $p(x) = p_k(x)$, where p_k is a key-dependent permutation of 2^m elements (i.e. a pseudorandom table of $m \times 2^m$ bits, such that no two rows are equal)
- $p(x) = c^{x+k_i \pmod{2^m}} \pmod{2^m}$, where c is an m-bit prime and k_i are some key dependent values

Note that if a key-dependent function h is used, this function and its secret parameters are considered as *part of the extended group basis* according to Definition 5.1.1. Consequently, a key for TST' is completely determined by the pair of group bases used. Also note that chaining, as described here, is just one of the possible realizations of h. This scheme is particularly interesting, because of its simplicity and efficiency. Nevertheless, other realizations are very well possible. When

designing a family of transformations $\mathcal{T}$, one should just ensure that a particular implementation satisfies the five requirements listed in this section.

5.3 Efficiency

Our efficiency evaluation of TST' was performed in the same way as the evaluation of TST described in Section 4.1. We examine 64-bit and 128-bit versions of TST' for both Sylow and Abelian carrier groups, using a variable AFE. We compare the new cryptosystem with the original[3] TST as well as with the cryptosystems IDEA and AES.

5.3.1 Key Generation

A key for TST' based on group $\mathcal{H}_s \times \mathcal{H}_1$ consists of two randomly chosen extended transversal group bases for this group. Because a key of TST has exactly the same form, the key generation setup for TST' based on $\mathcal{H}_s \times \mathcal{H}_1$ can be the same. When using the commutative carrier group $\mathbb{Z}_2^n$, the BGA of TST' can be slightly simplified. The commutative block shuffle can be performed in a very straightforward way by Algorithm 5.3.1.

Algorithm 5.3.1 Trivial Block Shuffling
Let $\beta = (B_0, B_1, \ldots, B_{w-1})$ be a group basis of a commutative group G. The commutative block shuffle can be performed by the following algorithm:

input β
for $i = 0, \ldots, (w-1)$ do
    set $j = \mathsf{Random}(i,\ w-1)$
    call $\mathsf{Swap}(B_i,\ B_j)$
endfor
output β

The function Random*(min, max) generates a random integer between min and max, and the function* Swap*(B_i, B_j) swaps the blocks B_i and B_j in β (when $i = j$, the function does nothing).*

This algorithm generates every possible ordering of blocks with the same probability $\frac{1}{w!}$. Because of the guaranteed commutativity of $\mathbb{Z}_2^n$, the more complicated Algorithms 4.1.1 and 4.1.2 are no longer necessary.

The requirements for a PRNG used during the BGA are the same as with TST (see Section 4.1.1.1 for more details). Hence, in our TST' implementation we use the same lagged Fibonacci PRNG that we used with TST.

[3]Hereby we consider the TST version using the cartesian representation of permutations which has shown itself to be significantly more efficient than the version based on the compact representation.

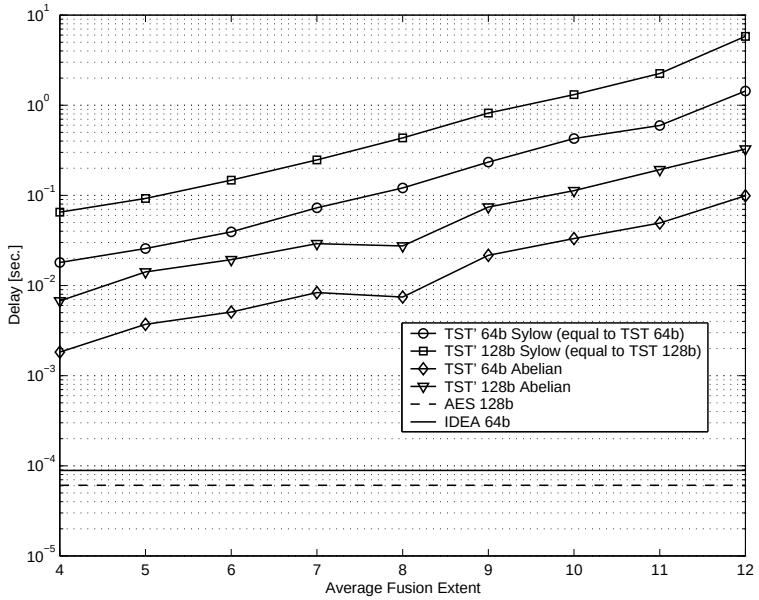


Figure 5.5: Key setup delay of TST'

5.3.1.1 Key Setup Delay

Figure 5.5 displays the key setup delays for TST' using both carrier groups. Note that the times achieved by TST' based on the Sylow group are the same as the ones of TST. This was expected, because the same key setup scheme has been used in both cases.

The irregularity on the curves for Abelian group when AFE $= 8$ is caused by the internal data representation in our implementation. 8 is the last number for which all 2^{AFE} values (e.g. indices of rows inside of a base block, coordinates, etc.) fit into one byte. For AFE > 8 already two bytes must be allocated for every such variable. This sudden change of word length manifests itself as a small peak on the curves.

One can see that the key setup delays of TST' based on the Abelian group are roughly 11 times shorter than the ones of TST. In spite of the fact that these values are still not competitive to IDEA and AES, the delays staying significantly under 0.5 seconds enable a comfortable use of TST' for symmetric encryption. The suitability of TST' for building cryptographically secure hash functions, however, is still not very good. Only delays substantially below 10^{-3}s would be appropriate for that purpose.

5.3.2 Throughput

The encryption speed of TST' depends substantially on the carrier group used. The middle pair of curves in Figure 5.6 corresponds to the original TST cryptosystem. As expected, TST' based on the Sylow group is slower than the original TST. The complexity of factorizations and compositions is the same for both cryptosystems,

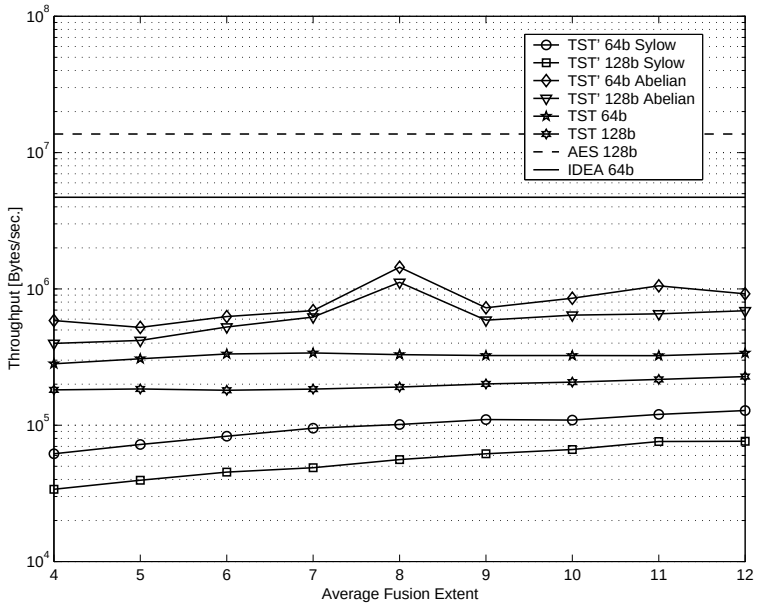


Figure 5.6: Encryption speed of TST'

but TST' has to perform the transformations T_i in addition. When using the Abelian group, this disadvantage is more than compensated. A 64-bit TST' is 2.5 times faster and a 128-bit TST' is even 3.2 times faster than the original TST. Unfortunately, the achievable speeds are still not competitive to the ones of IDEA and AES.

There is an interesting local maximum by AFE = 8 on the curves corresponding to $\mathbb{Z}_2^n$. This peak is not of pure implementational nature. The extraordinary increased speed is caused by a combination of two factors. First, the already mentioned optimal length of variables (8 bits = exactly 1 byte) plays a role and, second, the extraordinary regularity of group bases for AFE = 8, which will be discussed in Section 5.4 in more detail, contributes to this speedup. A similar peak can be expected in all points where the two mentioned factors appear together, e.g. AFE = 8, 16, 32, etc.

Although the locally improved speed for AFE = 8 seems to be an advantage, this fast configuration should preferably not be used, because the security properties of the cryptosystem have their local minimum exactly at this point.

5.3.3 Memory Requirements

TST' does not bring any new values regarding the minimal memory requirements. The memory requirements with the Sylow group are the same as the ones of TST. The amount of memory which is necessary when we use the Abelian group is the same as the one of TST with the compact representation, i.e. roughly eight times smaller for $n \leq 256$. These results are displayed in Figure 5.7.

However, the eight times smaller memory requirements of TST' are not counterbalanced by a decreased throughput, as it was the case with TST using the compact

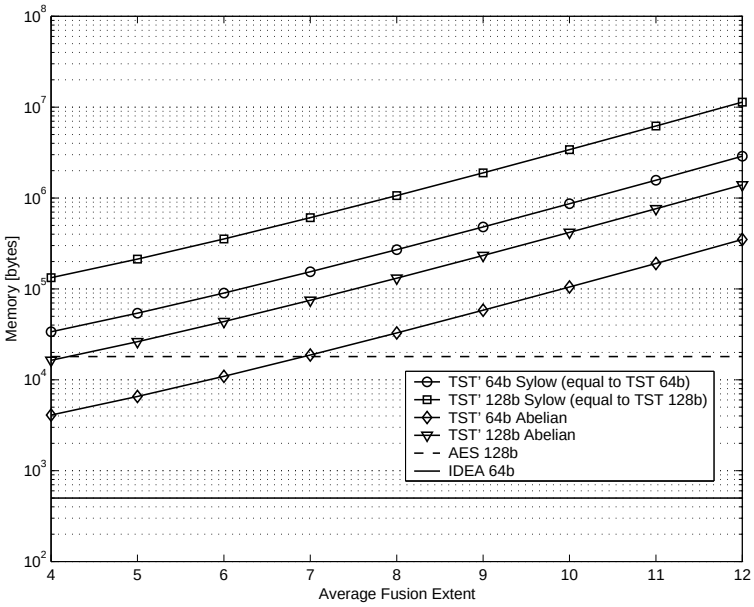


Figure 5.7: Memory requirements of TST'

representation. On the contrary, the speed of TST' is higher when using the smaller memory. This property of TST' can actually be considered as a decrease of memory requirements by a factor of 8 in comparison with TST.

5.4 Security

In what follows we analyze the security of TST' using both generic and white box evaluation methods. In both cases we compare the results of TST' with the ones of the original TST to make clear, whether or not the intended improvement of security properties was achieved.

5.4.1 Generic Statistical Evaluation

The methodology used for the statistical evaluation of TST', whose results are presented in this section, was exactly the same as the one which has been used for TST evaluation in Section 4.2.1.

The positive influence of the extended group bases on the statistical properties of TST' is demonstrated in Figure 5.8. In contrast to TST, whose suspicion rate is too high and sinks very slightly, the randomness properties of TST' based on the Sylow carrier group are very good. Even the version based on the simple Abelian group has a very rapidly sinking suspicion rate for a growing AFE. However, the curves corresponding to the Abelian group do not sink monotonously. Apparently, there is a strong local maximum (weakness) for AFE = 8. This deviation is caused by an

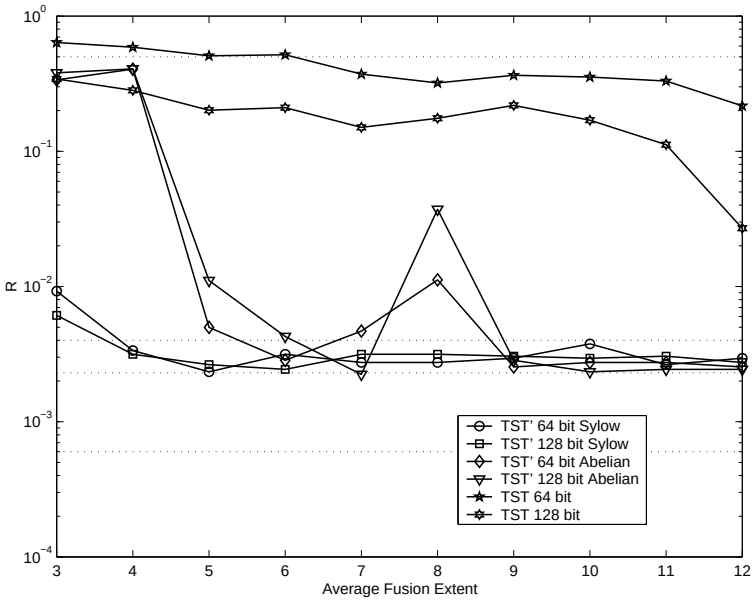


Figure 5.8: Suspicion rate profiles of TST'

extraordinary regularity of the group bases which can be explained as follows[4].

Because both examined plaintext lengths $n = 64$ and 128 bits are divisible by 8, the blocks of the two group bases β_1 and β_2 are mapped "one to one" when AFE $= 8$. In this special case no parts of blocks overlap between the bases, and every coordinate x_i with respect to β_1 completely determines the coordinate x'_{w-1-i} with respect to β_2. This is exactly the case which we exploited in the attack on TST described in Section 4.2.2.2.

An increasing AFE affects the statistical properties of TST' in two ways. First, there is an exponentially strong trend which improves the pseudorandomness properties until a certain level is reached which is not distinguishable from perfect randomness. Second, there are local peaks at those points where AFE divides the block length n (i.e. 64 or 128 bits in our case). The falling-off of the suspicion rate is the superior of the two trends and, therefore, after reaching a certain level, the local maxima become negligible. Our investigations have shown that the deviations become most obvious in the middle part of the curve. For small values of AFE the randomness is very poor anyway, so the even worse values for the divisors of n (e.g. AFE $= 4$) are not that flagrant. On the other hand, when AFE is large enough, the positive influence of large group bases is much stronger than the local deviations, and these deviations (e.g. for AFE $= 16$) become negligible again.

It is noticeable that we do not observe similar maxima on the TST curves. This is supposedly caused by the following two factors. First, the randomness properties of TST are rather poor, so the deviations are not obvious, and second, the more

[4]The following considerations belong, in fact, to the white box analysis because they are based on the inner structure of TST'. Nevertheless, we think that it is more appropriate to place them into this section, where the problem arises for the first time.

complex structure of $\mathcal{H}_s \times \mathcal{H}_1$ might suppress creation of such maxima. The fact that there are no similar peaks on the curves corresponding to TST' based on the Sylow group supports this hypothesis. Nevertheless, even in its worst maximum the randomness of TST' is significantly better than the one provided by TST. The randomness properties of TST' based on Abelian group, achievable with AFE ≥ 9, are very strong, and 9 is therefore the minimum AFE value advisable for practical use. With the slower Sylow group even AFE ≥ 4 seems to be sufficient.

5.4.2 White Box Analysis

The extended group bases of TST' improve its cryptographic properties in comparison with TST. On the other hand, the usage of a simpler carrier group $\mathbb{Z}_2^n$ might compromise the security of TST' in some way. We analyze the influence of these modifications, estimate the new size of the potential key space, and investigate the possibility of new attacks.

5.4.2.1 Key Space

The theoretical key space size of TST' depends on the carrier group used. When using the Sylow group, the key space is identical to the one of TST which has already been analyzed in Section 4.2.2.1. When using the commutative Abelian group, the key space becomes larger. The block swaps performed during a commutative block shuffle of a canonical basis for $\mathbb{Z}_2^n$ are not limited in any way, because all block do commute. It follows that the number of possible outputs of T1 is the same as the number of all possible permutations of n elements, i.e. $n!$. This value is roughly 2^n times higher than the corresponding value 4.2 for TST. This improvement corresponds to approximately n additional key bits. Nevertheless, when speaking about hundreds of thousands of key bits (see e.g. Example 4.2.1), such an improvement is marginal.

5.4.2.2 Attacks on TST'

From the cryptographic point of view, the fact that the key bits are spread uniformly among the blocks of group bases is much more relevant than the increase of the hypothetically achievable key length. The uniform distribution of key bits in $\mathbb{Z}_2^n$ disables an adversary to make good predictions regarding the probability of appearance of some key bit positions in the lower (respectively higher) blocks. Thus, even if a TST' based on $\mathbb{Z}_2^n$ had the weak avalanche effect of TST (which is not the case), the prediction possibilities described in Section 4.2.2.3 would be reduced.

The contribution of transformations T_i to the security of TST' is even more important. Let us consider a version of TST' which is simplified in a similar way as the one described in Section 4.2.2.2. Figure 5.9 shows a very simple TST' with block length $n = 2m$ whose two group bases have type $r = (2^m, 2^m)$. As already mentioned, the transformation T_0 affecting the last coordinate x_0 is the only one whose action is equivalent to an element shuffle of the corresponding base block.

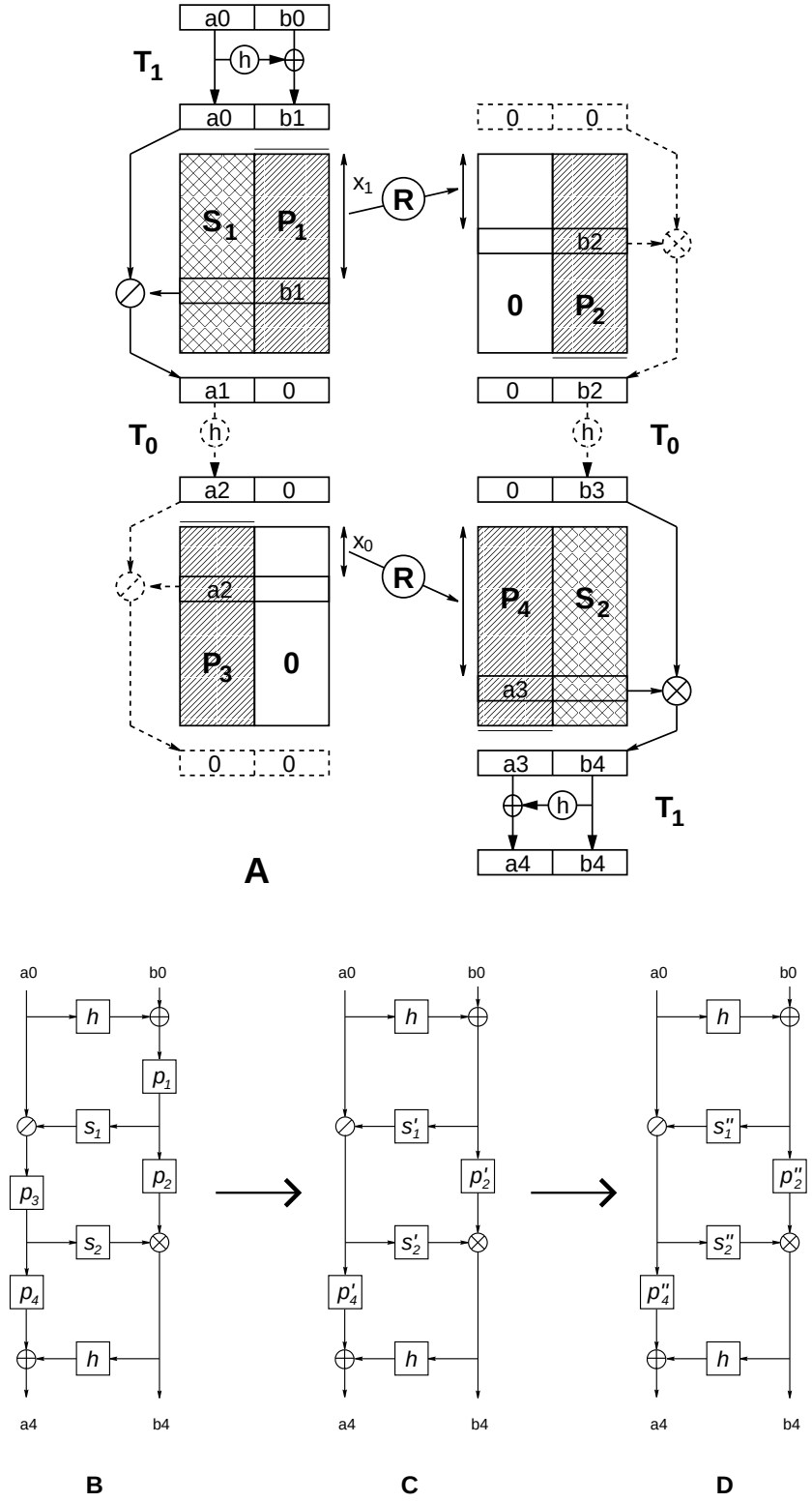


Figure 5.9: Analysis of TST'

Hence, the evaluation of T_0 can be omitted. The simplification from scheme A to scheme D is done in the same way as the one performed in Section 4.2.2.2. The operations $\oplus$, $\otimes$, and $\oslash$ are not particularly important for the attack, and can be therefore realized by any group operations on $\{0,1\}^m$.

A cryptanalysis of scheme D can not be done in the trivial way described in Section 4.2.2.2. When setting $b = 0$ and varying a through all 2^m possible values, the input of function s_1'' will not be constant, because of the transformation T_1. However, when the exact h is publicly known (i.e. h is not key-dependent), the attack can be performed in a slightly modified way. Instead of keeping $b = 0$, we can perform the trial encryptions of inputs $a_i||b_i$, where $a_i = 0, \ldots, (2^m - 1)$, and b_i is an inverse element to $h(a_i)$ with respect to the group operation $\oplus$. Using this approach we can keep the input of s_1'' and p_2'' constant and, hence, attack the scheme with the same complexity as the scheme D in Figure 4.10. It follows that T_i, $i > 0$ should *always* be made key dependent.

When the uncertainty of the output of T_i is at least m bits, an adversary can not predict the input of s_1'' and, consequently, can not construct the attack described above. To guarantee such an uncertainty a T_i can, for example, be implemented as chaining of a key-dependent bijection h which is incompatible with $\oplus$ and can implement (depending on the particular key used) at least 2^m different mappings. "Incompatible with $\oplus$" means that the output value of term $h(k,a) \oplus b$ should not be controllable without the knowledge of k. For example, $h(k,a) = k \oplus a$ is a bad choice for h, because by setting $b_i = a_i$ the output of $h(k,a) \oplus b$ can be held constant. On the other hand, a good choice for $h(k,a)$ is a random permutation of 2^m elements depending on k. For such an implementation of h the output of $h(k,a) \oplus b$ can be correctly predicted with probability at most $\frac{1}{2^m}$.

It follows that with a proper choice of the transformation family $\mathcal{T}$ for the extended group bases of TST' the cryptosystem can be made resistant against an efficient class of attacks which could be successfully mounted on TST.

5.5 Summary

The new cryptosystem TST' is based on *extended group bases* involving a class of transformations $\mathcal{T}$. This new design ensures a strong avalanche effect of the cipher, because every coordinate of a permutation factorized with respect to an extended group basis depends on all its bits. Owing to its improved diffusion properties the new cryptosystem can utilize a new simpler carrier group $\mathbb{Z}_2^n$ in addition to the original group $\mathcal{H}_s \times \mathcal{H}_1$.

The security of TST' based on both groups is improved. When using the commutative carrier group, the statistical properties of the cipher are steadily good for AFE ≥ 9. Moreover, the distribution of key bits among the blocks of group bases is uniform in $\mathbb{Z}_2^n$. Consequently, the prediction of the key bit positions becomes harder, and the potential key space grows slightly. The statistical properties of TST' based on the Sylow 2-group are even better. With only AFE ≥ 4 a sufficient level of randomness can be achieved.

Cipher	**Key Setup Time**†	**Encryption Time**†
TST' (Sylow)	269×10^6	90629
TST' (Abelian)	$\mathbf{25 \times 10^6}$	**9500**
TST (compact)	398×10^6	141526
TST (cartesian)	$\mathbf{269 \times 10^6}$	**27882**
Magenta	30	6539
Frog	1.4×10^6	2417
Loki 97	7430	2134
Safer+	4278	1722
Rijndael (AES)	305, 1389‡	374
RC6	1632	270

† Measured in clock cycles on Intel Pentium II processor

‡ Key setup delays for encryption and decryption are different

Table 5.1: Efficiency of TST' in comparison with other ciphers

The attack which was successful against TST can be excluded in TST' by using key-dependent transformations $T_i \in \mathcal{T}$. The invertibility of these transformations can be ensured in a straightforward and efficient way based on a simple hash function. To further improve the security properties of TST' it is advisable to avoid AFE values which are not relatively prime to the block length n.

The simplicity and commutativity of group $\mathbb{Z}_2^n$ manifests itself in the improved efficiency of TST'. When using this new carrier group, the key setup is 11 times faster, the throughput is on average 3 times faster, and the memory requirements are roughly 8 times smaller than with the original TST. The efficiency of the group $\mathcal{H}_s \times \mathcal{H}_1$ is the same as with TST, but the throughput of TST' based on $\mathcal{H}_s \times \mathcal{H}_1$ is lower, because of the additional transformations T_i that have to be performed. Hence, in spite of its good cryptographic properties the Sylow 2-group seems not to be very suitable for practical use.

In Table 5.1 we present a performance comparison of TST' with TST (in both cases AFE = 9) and with some of the AES candidates. Note that a TST implementation should preferably use the cartesian representation of permutations, and a TST' implementation should be preferably based on the Abelian group. TST in the compact representation and TST' based on the Sylow 2-group are not very efficient, and are listed only for comparison purposes. The values for the AES candidate ciphers are the same as the ones in Table 4.1. We see that, although the new cryptosystem TST' is still not as efficient as the AES candidates, it is significantly faster than the original TST. The improved efficiency properties of TST' combined with its full scalability might make it interesting for some applications.

Chapter 6

A New Iterative Cryptosystem

This chapter introduces a new iterative cryptosystem based on mathematical operations that are similar to the ones of TST'. The new cipher does not use real group bases anymore, but utilizes a key-dependent random table similar to one block of a group basis, and the structure of its round is very similar to one factorization-composition step known from TST'. This new design demonstrates a possible combination of two cipher design approaches - the "group basis oriented" and the "classical iterative" one.

In the first section we analyze the reasons for the relatively low efficiency of TST and TST' in comparison with the modern iterative block ciphers. Based on these considerations we suggest some modifications of the encryption scheme, which will lead us to an iterative structure relative to TST'. We show that there is a relationship between our new scheme and the unbalanced Feistel networks - a typical framework for iterative cipher design.

In the second section we present a concrete scalable block cipher family based on our new scheme. We describe the encryption setup, and analyze the influence of the variable parameters on the properties of the cryptosystem. Furthermore we discuss some implementation issues and present one possible realization of the cipher.

The following fourth and fifth section present an efficiency and security analysis of the new cryptosystem. This analysis is based on the same methodology which was used in the previous two chapters. The final sixth section summarizes the properties of the new iterative cipher and compares them with the ones of TST and TST'.

6.1 Motivation

A cryptosystem based on group bases encrypts a plaintext by translating it through two secret group bases. This approach is, from the mathematical point of view, very compact and straightforward. In fact, the complete encryption and decryption rules can be written in just two short formulae. Moreover, the block length and the key length of such ciphers are scalable by nature which is a very advantageous property. From the practical point of view, the two secret group bases can be seen as two big

key-dependent random tables with a special structure. These tables embody the encryption rule of the cipher, and a secret key consists, by definition, of both of them.

Secret tables as building blocks of a cipher contribute to its security, because they are a source of a bigger uncertainty for an adversary, and they suppress the possibility of precomputation attacks. However, when the size of the tables is the only factor contributing to the security of a cipher, one is forced to use tables that are *really big*. The resulting memory requirements (see e.g. Figure 5.7) make such a cipher unsuitable for use in restricted environments, e.g. smart cards. Even on powerful workstations with large RAM the efficiency of such a cipher suffers. Here, the memory requirements are not the primary problem, because the bases do not exhaust the big memory space available by far. However, the tables are too big to fit into the fast cache memory[1] and must therefore be stored in the slower main memory. Even worse, due to the nature of encryption, the table accesses happen on random offsets, so the success rate of cache prediction algorithms is completely decimated, and the main memory is accessed at its slowest speed. As a consequence, even the fastest secure version of TST' is roughly 25 times slower than the AES algorithm which only uses a few kilobytes of tables. Also the key setup time suffers from the large tables, because they must be initialized by a rather time-consuming non-trivial algorithm. The resulting key setup delays exclude this class of ciphers from every application which requires frequent key setups, e.g. cryptographic hash functions.

Another inherent disadvantage of cryptosystems like TST or TST', beside their inefficiency, is the strict relation between security and memory requirements. The security of these ciphers can only be increased through a bigger AFE, which will cause an exponential growth in the table size. Unfortunately, it is not possible to increase the security at the cost of encryption time by unchanged memory requirements. This property does not allow us to make a free tradeoff between memory, speed, and security.

Virtually all popular modern symmetric cryptosystems, including DES, IDEA, GOST, Skipjack, Safer, RC5, as well as most of the fifteen AES candidates, are iterative ciphers (Definition 2.3.4). This design, based on multiple repetitions of a fixed round function with different round keys, is so popular because of its simplicity, efficiency, and the well controllable security level. The round function is usually simple and fast enough to be implemented even on smart cards, and the desired security level can be achieved by repeating the function a sufficient number of times. Unfortunately, the design of most iterative cryptosystems is rather fixed, and their block and key lengths are not scalable. In what follows we suggest a combination of the table-based encryption approach, known from TST', with the efficient iterative approach. The resulting symmetric cryptosystem will be scalable by nature, but more efficient than the ciphers based on group bases.

[1] A typical size of the fastest first level cache of modern processors is 64 kB and the second level cache has usually 512 kB. These spaces are shared by all running processes, thus, an efficient encryption algorithm should not use tables longer than few tens of kilobytes in total.

6.2 Iterative TST Design

Designing an iterative cipher involves a round function design, a key schedule design, and an estimation of the secure number of rounds. In this section we demonstrate these three steps for a new symmetric cryptosystem called *Iterative TST*.

6.2.1 Round Function

Let us consider a TST' encryption for $n = 64$ and AFE $= \frac{n}{4} = 16$ displayed on the left hand side of Figure 6.1. The transversal group basis β_1 for the carrier group $G = \mathbb{Z}_2^{64}$ is based on the chain of subgroups $G_0 = \mathbb{Z}_2^{16} < G_1 = \mathbb{Z}_2^{32} < G_2 = \mathbb{Z}_2^{48} < G_3 = \mathbb{Z}_2^{64} = G$. For the sake of simplicity we display the key bits for each base block on adjacent positions. In a real group basis, created by the BGA, these bits would be randomly shuffled. Nevertheless, we can disregard the shuffling now, because once we know the exact key bit ordering, we can move them to adjacent positions by transposing the columns of the bases and redefining the operations $\oslash$ and $\otimes$.

The first factorization step, enclosed within the dashed rectangle, starts with a transformation T of the input vector $p \in G_3$. This transformation affects only those bits of p' which lie on the key bit positions for B_3. All other bits of p' are equal to the ones of p. In the next step the factor b_{3,x_3} in B_3 is found, so that the key bits of b_{3,x_3} are equal to the ones of p'. The intermediate result $p'' \in G_2$, $p'' = p' \oslash b_{3,x_3}$, whose bits on the key bit positions for B_3 are guaranteed to be zeros, is passed to the next factorization step. The coordinate x_3 is reversed and used for the composition in β_2.

Every pair of corresponding factorization and composition steps that are connected through a coordinate x_i has together 64 variable input- and 64 variable output-bits. A composite transformation P-R-P at every level can be replaced by one equivalent permutation p, as was shown in Section 4.2.2.2. When we consider an input vector of the original encryption scheme as a sequence of four 16-bit segments, we see that every segment plays the role of key bits for a certain factorization-composition level. The active key bit segment is moving from the right most position in the first level to the left most position in the last level.

A transformation which is very similar to this TST' encryption can be constructed by four repetitions of a slightly modified first factorization step displayed on the right hand side of Figure 6.1. Hereby, the transformation T is implemented with a key-dependent hash function h, as was proposed in Section 5.2.2, the highest coordinate x_3 is "reused" on the key bit positions of the output vector p'' (these bits would be otherwise zeros), and the right-to-left movement of the active key segment is ensured by a segment rotation ξ at the end of the "factorization step". The key-dependent random table consisting of parts S and P corresponds to a "block of a group basis". It can be considered as two separate S-boxes: the cross shaded S of the size $2^m \times (n-m)$ bits, and the simple shaded P of the size $2^m \times m$ bits. P, whose all rows are distinct, represents a permutation of 2^m elements and corresponds to the "key bits of the basis block".

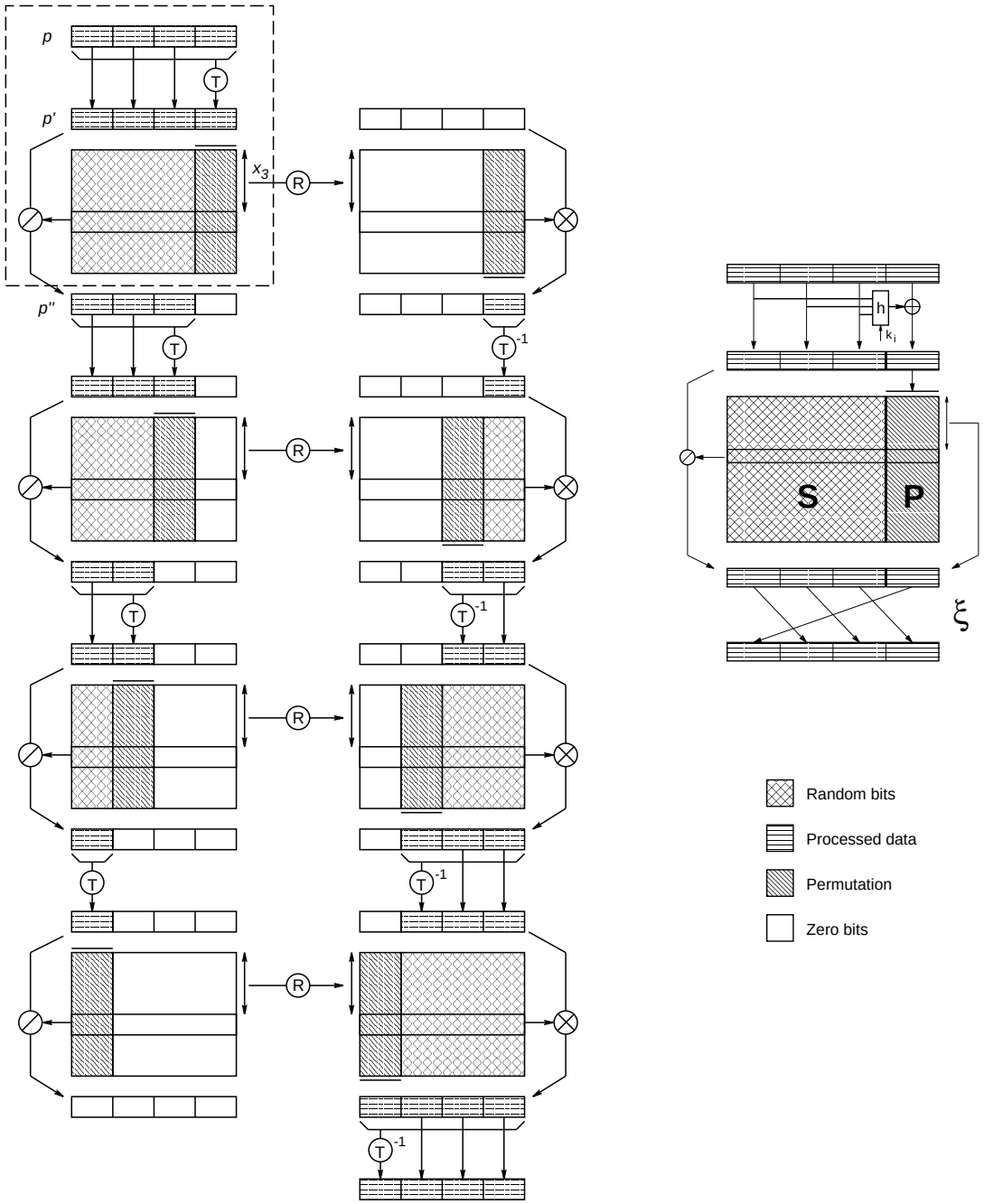


Figure 6.1: One factorization step of TST' as a round function

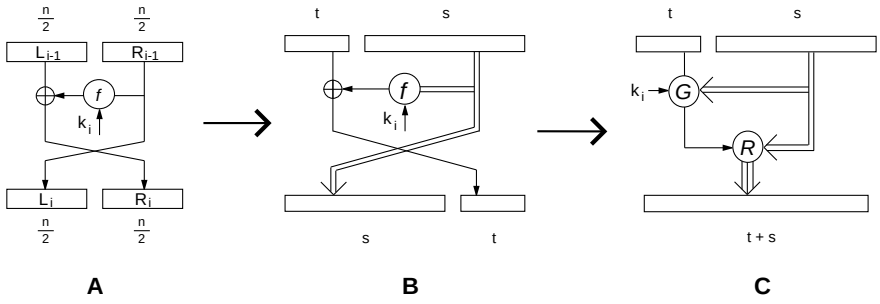


Figure 6.2: Unbalanced Feistel networks

The resulting 64-to-64-bit mapping will be called a *TST round*, and we will use it as a round function of a new symmetric cryptosystem *Iterative TST*. The value $m \leq \frac{n}{2}$, corresponding to the "number of key bits" in a TST round, will be called a *segment length*, and a consecutive sequence of m bits $x_{k \cdot m+0}, \ldots, x_{k \cdot m+(m-1)}$, $k \in \mathbb{Z}$, of a binary vector x will be called a *segment*. The left most segment of a binary vector x will be denoted by x_L, and the rest of the vector by x_R.

6.2.1.1 TST Round as an Unbalanced Feistel Network

The *Feistel structure*, displayed in Figure 6.2 A, is a scheme commonly used for designing round functions of block ciphers. A cipher based on the Feistel structure divides an n-bit plaintext into two $\frac{n}{2}$-bit halves L_0 and R_0, and processes them by r successive rounds. For $i = 1, \ldots, (r-1)$ the i-th round transforms an input $L_{i-1}||R_{i-1}$ into $L_i||R_i$, such that $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$, where f is a non-linear function, and k_i is the i-th round key. The last r-th round does not swap the halves and, hence, the resulting ciphertext is $L_r||R_r$, where $R_r = R_{r-1}$ and $L_r = L_{r-1} \oplus f(R_{r-1}, k_r)$. This encryption scheme is called *Feistel network*, and it is guaranteed to be invertible for any f (even if f is not invertible). When a plaintext x has been encrypted into ciphertext y by r Feistel rounds, using the round keys $k_1, \ldots, k_r$, the inverse transformation $x = d_K(y)$ can be performed by *the same scheme* using the round keys in the reverse order $k_r, \ldots, k_1$. This property enables an efficient hardware implementation of Feistel networks, because both encryption and decryption operations can be performed using the same circuit.

Unbalanced Feistel networks (UFN) were introduced and analyzed independently in [SK96] and [Nyb96]. A Feistel network is generalized to a UFN by introducing variable lengths of the two sub-blocks. Figure 6.2 B displays an s-on-t UFN. The sub-block L is called the *target* and its length is denoted by t. The sub-block R is said to be the *source* and its length is denoted by s. The conventional Feistel network is a special case of a UFN for $t = s = \frac{n}{2}$. A UFN is called *source heavy* when $s > t$ and *target heavy* when $t > s$. A UFN is said to be *homogeneous* when the f is identical in all rounds (except for the round keys k_i) and *heterogeneous* otherwise. A further generalization of a UFN leads to a *generalized unbalanced Feistel network* (GUFN) displayed in Figure 6.2 C. In this construction $\mathbf{R}$ must be an invertible

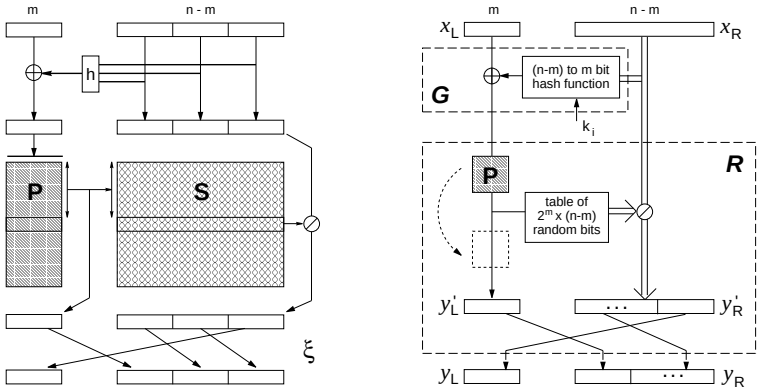


Figure 6.3: TST round as a GUFN

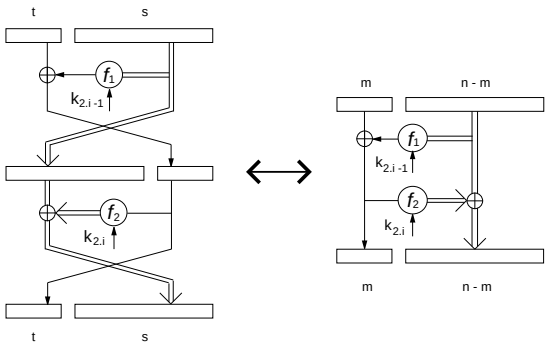


Figure 6.4: TST round as a heterogeneous UFN

function, and **G** must be reversible in the sense that there exists a function **H** such that for all k, x_s, x_t $\mathbf{H}(k, x_s, \mathbf{G}(k, x_s, x_t)) = x_t$.

When considering the TST round as a GUFN, the hash function h together with the operation $\oplus$ correspond to **G**, and the permutation P together with the operation $\oslash$ and segment rotation ξ are included in **R**. This equivalence is demonstrated in Figure 6.3[2]. Note that when we permute the rows of S according to permutation P, P can be placed *after* the S-box as well, without changing the round mapping. In fact, as both P and S are generated randomly, it does not matter whether P is placed before or after S.

A simplified version of Iterative TST, whose rounds do not contain P and ξ, can be considered as a heterogeneous UFN, consisting of source heavy $(n-m)$-on-m odd rounds, and target heavy m-on-$(n-m)$ even rounds. This equivalence is displayed in Figure 6.4. Such a UFN alternately utilizes two different key-dependent functions $f_1 : \{0,1\}^{n-m} \longrightarrow \{0,1\}^m$ in the odd rounds, and $f_2 : \{0,1\}^m \longrightarrow \{0,1\}^{n-m}$ in the even rounds. One simplified TST round corresponds in this case to two UFN rounds[3],

[2]To stress the similarity of the two structures, the key segment of the TST round is displayed on the left, in contrast to Figure 6.1 where it was displayed on the right.

[3]According to the terminology in [SK96], such a simplified TST round should be called a *cycle*, because its output bits are returned on their original positions.

h corresponds to f_1, and S to f_2. Even though we present this analogy, we do not intend to make use of such a simplified TST round, because both P and ξ contribute to the cryptographic quality of Iterative TST. Note that the segment rotation ξ at the end of a TST round ensures that all segments are regularly processed (as an input) by both f_1 and f_2. This is not the case in the presented UFN. Furthermore, permutation P ensures that all output segments of a TST round depend on all input segments in a non-linear way. Without P the dependence between x_L and y'_L would become linear.

6.2.1.2 Variable Components of a TST Round

The TST round, displayed in Figure 6.3, has been derived directly from the TST' encryption scheme. Similarly as in TST', there are several components in the TST round which can be implemented in different ways without changing the basic encryption principle. These variations, however, can have a substantial influence on the efficiency and security of the resulting cipher. The following components can be varied:

- Function h. This function corresponds to h_i used in T_i of TST' (see e.g. Section 5.2.2). The choice of h affects the efficiency and diffusion properties of the cipher.
- Random tables P and S, and the PRNG used for their generation. These components correspond to the random group bases of TST'. A weak PRNG used for initialization of P and S can substantially weaken the cipher. The number of different P's and S's used affects the security and memory requirements of the cipher. (One can use, for example, the same P and S in all rounds, use a fixed S and a variable P_i in every round, or vary both P_i and S_i in each round.)
- Operation $\oslash$. This operation corresponds to the group operation in the carrier group of TST'. Choices of different operations $\oslash$ have a similar effect as varying the carrier group of TST', i.e. can improve the efficiency and confusion of the cipher.
- Segment length m. This value corresponds to the AFE of TST'. Usage of a bigger m contributes significantly to the complexity of the implemented encryption mapping. On the other hand, it also exponentially increases the memory requirements.

In this context the TST round scheme should be understood as a framework for the construction of scalable block ciphers. Depending on the components used, the resulting cipher belonging to the Iterative TST family can have very different properties. In what follows we discuss some of the possible implementations of the variable components by taking the efficiency and security into account. The resulting version of Iterative TST, presented in this chapter, will be just one of many possible implementations.

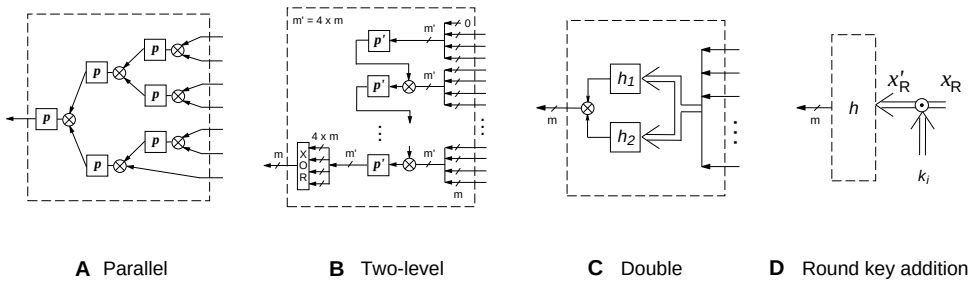


Figure 6.5: Efficient implementations of h

6.2.2 Implementation Issues

6.2.2.1 Hash function

Some possible implementations of h have been already discussed in Section 5.2.2. Furthermore, it was concluded in Section 5.4.2.2 that h should be made key-dependent to improve its attack resistance. The simplest implementation of h - a chaining of a non-linear bijection p with a binary operation $\otimes$ - was already shown in Figure 5.4. This segment-wise computation is simple and easily scalable, nevertheless, it requires too many computation steps, especially for a small m. There are some more efficient designs shown in Figure 6.5.

The tree variant A is most suitable for a hardware implementation because it employs the highest possible parallelization level. On the other hand, the two-level variant B is most suitable for an efficient software implementation. When m' is chosen to be the longest register length for a given implementation platform (e.g. 32 or 64 bits for nowadays usual processors), a speedup by factor up to $\frac{m'}{m}$ can be achieved in comparison with the scheme in Figure 5.4. The operation $\otimes$ should be in all cases an efficiently computable binary operation on $\{0,1\}^m$ (resp. on $\{0,1\}^{m'}$), like XOR, $+ \bmod 2^m$, etc. The non-linear bijection p (resp. p') can be implemented in many different ways (see e.g. the proposals in Section 5.2.2). To ensure that h is key-dependent, either p must be key-dependent, or else a round key k_i must be added to the input of h as shown in Figure 6.5 D. To prevent a contrived manipulation of the resulting hash value, the binary operation $\odot$ should not be distributive and associative with $\otimes$.

A more complex hashing scheme (Figure 6.5 C) can combine two hash values obtained by two different methods. The advantage of this approach is that, when h_1 and h_2 are independent, the resulting hash value can not be controlled by manipulations of the input. For example, when using the scheme from Figure 5.4 with a constant bijection p, an adversary can enforce all 2^m outputs of h by processing just 2^m input vectors. Hereby the first m bits of the input will be varied, and the other bits will be kept fixed. The adversary can not estimate the output values of h, but he knows that they are all different. Such a manipulation, which might be useful for an attack, is not possible with the scheme C. When h_1 and h_2 are independent and random, all changes in the input lead to a change in the output

with probability $\frac{2^m-1}{2^m}$. Even when h_1 and h_2 work segment-wise (word-wise), a contrived manipulation of the output can be obstructed significantly, e.g. by computing h_1 and h_2 on rotated versions of the input, i.e. $h = h_1(x_R) \otimes h_2(rot_k(x_R))$. This construction can be extended to an arbitrary number l of sub-hashes by computing $h = \bigotimes_{i=1}^{l} h_i(rot_{k \cdot i}(x_R))$. When $k \cdot l > (n-m)$, the shift value k should be relatively prime to $n-m$.

We presently propose two concrete realizations of h. The simpler implementation h_a is based on scheme B in Figure 6.5, and uses $p' : \{0,1\}^{m'} \longrightarrow \{0,1\}^{m'}$ of the form $p'(x) = c \cdot x$, where c is an m'-bit prime containing roughly $\frac{m'}{2}$ non-zero bits and having the highest bit equal to 1. The particular constants used are $m' = 32$, and $c = 3010192529$. This function is computed by Algorithm 6.2.1.

Algorithm 6.2.1 Function h_a
Let x be an $(n-m)$-bit binary vector, and let m' and c be two constants. The hash function $h_a : \{0,1\}^{n-m} \longrightarrow \{0,1\}^m$ is defined as follows:

input x
set $s = 0$
for $i = 0, \ldots, \lceil (n-m)/m' \rceil$ do
    set $s = c \cdot (s + \mathsf{GetSegm}(x, i, m'))$
endfor
set $y = 0$
for $i = 0, \ldots, \lceil m'/m \rceil$ do
    set $y = y \oplus \mathsf{GetSegm}(s, i, m)$
endfor
output y

The binary operation $+$ denotes an integer addition mod $2^{m'}$, and $\oplus$ denotes a binary XOR. *The function* $\mathsf{GetSegm}(v, i, l)$ *returns the i-th l-bit segment of a vector v, i.e. $w = \mathsf{GetSegm}(v, i, l) = (v_{i \cdot l}, \cdots, v_{i \cdot l + (l-1)})$, where v_j denotes the j-th bit of v. When the requested segment jets out of the vector v, the corresponding bits of w are zeros.*

The second (more complex) hash function h_b is based on scheme C shown in Figure 6.5. Both h_1 and h_2 are implemented according to scheme B, but the input of h_2 is rotated by k bits before processing. The bijection p' has the same form as in h_a. The particular constants used are $m' = 32$, $c_1 = 3010192529$, $c_2 = 3283113329$, and $k = 8$.

Algorithm 6.2.2 Function h_b
Let x be an $(n-m)$-bit binary vector, and let m', c_1, c_2, and k be constants. The hash function $h_b : \{0,1\}^{n-m} \longrightarrow \{0,1\}^m$ is defined as follows:

input x
Compute $y_1 = h_a(x)$ according to Algorithm 6.2.1 using $c = c_1$
Compute $y_2 = h_a(rot_k(x))$ according to Algorithm 6.2.1 using $c = c_2$
output $y_1 \otimes y_2$

The function $rot_k(x)$ rotates a binary vector x by k bits.

Because the proposed p' is not key-dependent, both h_a and h_b should be combined with a round key addition, as shown in Figure 6.5 D.

6.2.2.2 Random Tables

According to classical terminology the key-dependent table S, utilized by a TST round (Figure 6.3), is an m-on-$(n-m)$-bit *S-box*. Analogously, the key-dependent table P is an m-on-m-bit S-box of a special structure. (For a more detailed information on S-boxes see e.g. [AT90].) If we wish to keep the amount of secret table material of Iterative TST equal to that of TST', we would have to use a different random P and S in every round. However, we have seen in Section 6.1 that the extreme memory requirements of TST and TST' cause an undesirable slowdown of these ciphers. Therefore, we prefer using a fixed pair of tables in all rounds of our new cipher.

The composed operation consisting of the S-box S and the operation $\oslash$ (see Figure 6.3) can represent a variable mapping even if the same S-box is used in all rounds. This can be achieved in the following way. The round key addition proposed in Figure 6.5 D produces a key-dependent binary vector x'_R. When we use this vector (instead of x_R) as an input to $\oslash$, the composed operation $a \oslash' b = (a \odot k_i) \oslash b$ will become different for every round. Certainly, the new scheme using a single pair (P, S) has a smaller entropy (i.e. uncertainty for an adversary) than a version using variable tables. However, unlike the original TST', which can by design only perform *four* factorization-composition steps when AFE $= \frac{n}{4}$, the new scheme can be repeated *an arbitrary number* of times and, hence, implement a much more complex input-output mapping than the original TST'.

As a consequence of using single pair of S-boxes the memory requirements of Iterative TST will be significantly smaller than the ones of TST'. Figure 6.6 shows a comparison of the memory requirements for AFE $= \frac{n}{4}$. Version A, consuming $2 \cdot \frac{n}{AFE} \cdot 2^{AFE} \cdot n$ bits, corresponds to the standard TST' with two bases. Version B, requiring $\frac{n}{AFE} \cdot 2^{AFE} \cdot (n + AFE)$ bits, can be achieved without loss of generality by discarding the zero blocks of the group bases. Version C, which is achievable through simplifications of BGA , consumes $\frac{n}{AFE} \cdot 2^{AFE} \cdot (n - AFE)$ bits. (This configuration is not advisable because of its serious security flaw which we have

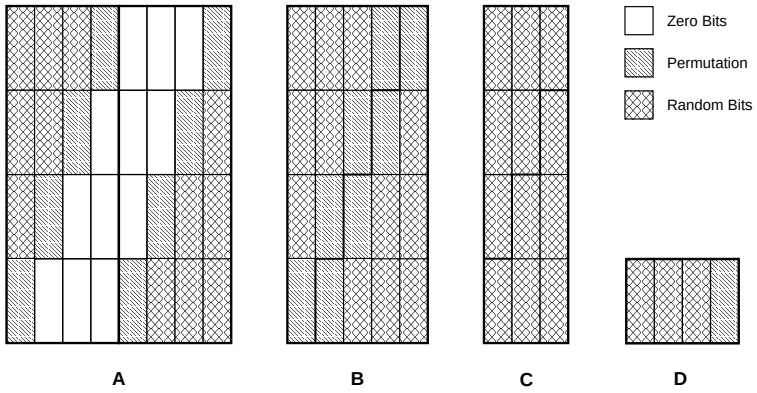


Figure 6.6: Memory requirements comparison

discussed in Section 5.4.2.2.) Finally, version D, corresponding to our iterative proposal, requires only $2^{AFE} \cdot n$ bits of memory.

Both P and S, as well as the round keys k_i are generated by a PRNG. The seed value used for initialization of the PRNG is equal to the secret key K. We suggest using the same PRNG as proposed for our TST implementation in Section 4.1.1.1. The key setup of Iterative TST is performed by Algorithm 6.2.3.

Algorithm 6.2.3 Key Setup of Iterative TST
Let $K \in \{0,1\}^k$ be a k-bit secret key. The iterative TST key setup generates a permutation table P, an S-box S, and round keys $k_1, \ldots, k_r$ as follows:

input K
call Randomize(K)
for $i = 0, \ldots, (2^m - 1)$ do
    set $P_i = i$
endfor
for $i = 0, \ldots, (2^m - 1)$ do
    set j = Random(i, $2^m - 1$)
    call Swap(P_i, P_j)
endfor
for $i = 0, \ldots, (2^m - 1)$ do
    for $j = 0, \ldots, (n - m - 1)$ do
        set $S_{i,j}$ = Random(0, 1)
    endfor
endfor

for $i = 1, \ldots, r$ do

set $k_i = \mathsf{Random}(0,\ 2^{n-m} - 1)$

endfor

output $P,\ S,\ k_1, \ldots, k_r$

The function $\mathsf{Randomize}(x)$ *initializes the pseudorandom number generator used with a seed value* x*, function* $\mathsf{Random}(min, max)$ *generates a random integer between min and max, and the function* $\mathsf{Swap}(P_i, P_j)$ *swaps the items* P_i *and* P_j *of an array* P*.*

6.2.2.3 Binary Operations

Until now there have been four binary operations involved in the TST round design:

- $\odot : 2^{n-m} \times 2^{n-m} \to 2^{n-m}$ used for the round-key addition,
- $\otimes : 2^{m'} \times 2^{m'} \to 2^{m'}$ used for the chaining in h,
- $\oplus : 2^{m} \times 2^{m} \to 2^{m}$ used for adding the output from h to x_L, and
- $\oslash : 2^{n-m} \times 2^{n-m} \to 2^{n-m}$ used for combining the output of the S-box with x_R.

All these operations form groups with underlying sets $\{0,1\}^{n-m}$, $\{0,1\}^{m}$, and $\{0,1\}^{m'}$ respectively. It has been shown in [LM91] that it is advisable to use incompatible (i.e. non-distributive and non-associative) operations o_1 and o_2, whenever an output of o_1 is used as an input in o_2, or vice versa. It follows that the pairs of operations $(\odot, \otimes)$, $(\otimes, \oplus)$, and $(\odot, \oslash)$, used in the TST round function, should not be compatible. Moreover, all four operations should be efficiently computable on modern processors.

According to the requirements above we suggest the following realization of the binary operations:

- $\odot$ - bit-wise XOR of two $(n-m)$-bit vectors,
- $\otimes$ - m'-bit integer addition mod $2^{m'}$,
- $\oplus$ - bit-wise XOR of two m-bit values, and
- $\oslash$ - word-wise addition mod $2^{m'}$ of two $(n-m)$-bit vectors (denoted by $\boxplus$).

This proposal is based on alternating a binary XOR with a regular integer addition. The operations $\odot$ and $\oslash$ process their operands as arrays of m'-bit vectors, where m' is the maximal register length for a particular platform ($m' > m$). Word-wise processing not only improves the speed in comparison with a segment-wise processing, but in case of $\oslash$ it also improves the cryptographic properties of the operation. When

$m < m'$, an m'-bit integer addition creates more complex dependencies between the input and output bits than an m-bit integer addition.

The operation $\oslash$ of TST' was implemented by an inverse permutation product in the particular carrier group. The investigations in Section 5.4 have shown that the cryptographic properties of TST' based on $\mathcal{H}_s \times \mathcal{H}_1$ are better than those of the version based on $\mathbb{Z}_2^n$. Unfortunately, the cryptographically stronger operation was rather slow. The operation $\oslash$ of Iterative TST is implemented by the word-wise addition $\boxplus$ whose bit dependence complexity (see e.g. Figure 4.12) is at least as good as the one of $*$ in $\mathcal{H}_s \times \mathcal{H}_1$. For instance, when $n = 128$ and $m' = 32$ a change of a single input bit during operation $v \leftarrow v * c$ affects on average 16.5 output bits in the group $(\mathbb{Z}_{32}^4, \boxplus)$, but only 6.02 bits in the group $(\mathcal{H}_7 \times \mathcal{H}_1, \oslash)$. Moreover, the operation $\boxplus$ is as fast as $*$ in $\mathbb{Z}_2^n$, because e.g. 32-bit additions take on most modern processors the same time as 32-bit XOR's. It follows that the operation $\boxplus$ used in Iterative TST combines the complexity of $\mathcal{H}_s \times \mathcal{H}_1$ with the efficiency of $\mathbb{Z}_2^n$.

6.2.3 Iterative TST

Upon the considerations above, we formalize the TST round function of our Iterative TST implementation.

Definition 6.2.1 TST Round Function
Let $h : \{0,1\}^{(n-m)} \longrightarrow \{0,1\}^m$ be a hash function, let $P \in \mathcal{S}_{2^m}$ be a permutation of 2^m elements, let S be an $2^m \times (n-m)$ binary array, and let $k_i \in \{0,1\}^{n-m}$ be a round key. A TST round function $f^{(h,P,S)} : \{0,1\}^n \times \{0,1\}^{n-m} \longrightarrow \{0,1\}^n$ is defined as follows:

$$f^{(h,P,S)}(x, k_i) = rot_m(P(x_L \oplus h(x_R \oplus k_i))||((x_R \oplus k_i) \boxplus S(x_L \oplus h(x_R \oplus k_i)))),$$

where $S(i)$ denotes the i-th vector (row) in S, $\boxplus$ denotes a word-wise addition modulo $2^{m'}$, $||$ denotes a concatenation, and rot_m denotes a left rotation by m bits.

The inverse function $f_{inv}^{(h,P,S)}$ is computed in the following way:

$$f_{inv}^{(h,P,S)}(y, k_i) = rot_{-m}((P^{-1}(y_L) \oplus h(y_R \boxminus S(P^{-1}(y_L))))||((y_R \boxminus S(P^{-1}(y_L))) \oplus k_i)),$$

where $\boxminus$ is a word-wise subtraction modulo $2^{m'}$, i.e. the inverse operation to $\boxplus$.

Figure 6.7 shows an example of a TST round for $m = \frac{n}{8}$ with the lowest-order bit placed on the left. The function h utilized in our implementation is computed according to Algorithm 6.2.2. The tables P and S as well as the round keys k_i are generated according to Algorithm 6.2.3.

An r-round cryptosystem *Iterative TST* encrypts a plaintext x by performing r TST rounds, i.e. $x_0 = x$, and $x_i = f^{(h,P,S)}(x_{i-1}, k_i)$ for $i = 1, \ldots, r$. The resulting ciphertext $y = e_K(x)$ is equal to $rot_{-m}(x_r)$. (i.e. the cryptographically insignificant segment rotation of the last round is undone). The corresponding decryption $x = d_K(y)$ is computed by performing r inverse TST rounds, i.e. $y_r = y$, $y_{i-1} = f_{inv}^{(h,P,S)}(y_i, k_i)$ for $i = r, \ldots, 1$, and $x = rot_m(y_0)$.

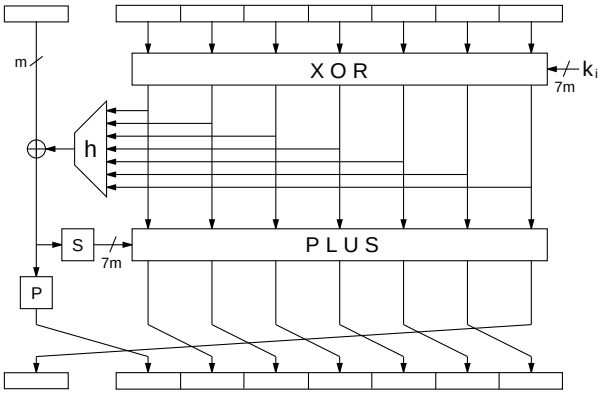


Figure 6.7: Iterative TST

6.2.3.1 Scalability

The block length and the key length of Iterative TST can be freely scaled by changing the parameters n and k respectively. The cryptosystem provides a unique instance (cipher) for any configuration (n, m, k, r), where $n, m, k, r \in \mathbb{N}$, and $m \leq \frac{n}{2}$. The security of the cryptosystem can be scaled in two ways:

1. By increasing the number of rounds. This approach does not increase the memory requirements. The security is improved at the cost of speed.

2. By increasing the segment length (corresponding to AFE of TST'). This approach does not slowdown the cipher. The security is improved at the cost of memory.

Consequently, one can perform a free tradeoff between security, memory, and speed, which was not possible in TST and TST'.

6.2.4 Secure Number of Rounds

The fundamental question regarding every iterative cipher is, how many rounds need to be performed to make the cipher secure. In this section we deal with this question for Iterative TST in terms of practical security (defined in Section 2.3).

The authors of [AB96] have shown that the cipher BEAR is provably secure in the sense that attacks which will find its key would yield attacks on the underlying components. BEAR is a 3-round UFN defined as follows.

Definition 6.2.2 Block Cipher BEAR
Let $H_K(M)$ be a keyed hash function which returns a result of a fixed size k bits for a message M of arbitrary length and a key K. Let furthermore $S(M)$ be a key stream generator, i.e. a function which given an input M of length k generates a pseudo random output of an arbitrary length.

A secret key K consists of two independent subkeys K_1 and K_2, both longer than k bits. An n-bit plaintext $x = L||R$, where $|L| = k$ and $|R| = n - k$ is encrypted as follows:

$$\begin{aligned} L' &= L \oplus H_{K_1}(R) \\ R' &= R \oplus S(L') \\ L'' &= L' \oplus H_{K_2}(R') \end{aligned}$$

and the resulting ciphertext is $y = L''||R'$.

The corresponding decryption of a ciphertext $y = L''||R'$ is performed by computing $L' = L'' \oplus H_{K_2}(R')$, $R = R' \oplus S(L')$, $L = L' \oplus H_{K_1}(R)$, and $x = L||R$. When the components H and S are cryptographically secure, BEAR is secure against almost all attacks. For special H and S BEAR might be vulnerable to a combined chosen plaintext and ciphertext attack, but this hypothetical weakness can be avoided by using an additional fourth round [AB96].

The encryption idea of BEAR is surprisingly similar to the one of Iterative TST. BEAR is in fact a 1.5-round version of the simplified Iterative TST, presented in Figure 6.4, which uses cryptographically secure h and S. It follows that the lower bound for the security of Iterative TST using ideal h and S is 2 rounds. The S-boxes S and P of a real-life Iterative TST can be made ideal (in terms of practical security) when a cryptographically secure PRNG is used for the key setup. The real-life implementations of h, however, will be usually far from ideal, because due to efficiency optimizations one will usually use simpler (cryptographically insecure) hash functions. Consequently, one must definitely execute more than just two rounds to make Iterative TST secure.

Referring to Figures 6.3 and 6.7, we will now analyze the properties of Iterative TST based on statistically (not cryptographically) strong components. Let us suppose that each of the 2^m possible outputs of f_1 is produced with the same probability. For instance, the f_1 defined by Algorithms 6.2.1 or 6.2.2 meets this requirement. Let us furthermore suppose that S produces a pseudorandom $(n-m)$-bit vector for each of its possible inputs, and P produces a unique pseudorandom m-bit vector for each of its possible inputs. S and P meet these requirements with very high probability when they have been generated by a statistically strong PRNG. Based on these assumptions, the sum $x_L \oplus f_1(x_R)$ takes on every possible value with probability $\frac{1}{2^m}$ and, consequently, both y_L and y_R will change in one of 2^m randomly looking possible ways, whenever either x_L or x_R have changed. The transformations performed by different rounds of Iterative TST can be considered as independent, because the round keys k_i are generated by random, and the operation $\odot$ is incompatible with both $\otimes$ and $\oslash$. Consequently, the number of different pseudorandom ways in which an input x can be modified into an output y by r TST rounds is $2^{r \cdot m}$. To reconstruct a mapping consisting of $2^{r \cdot m}$ point-wise independent randomly looking transformations one needs to encrypt $2^{r \cdot m}$ different inputs. This number is larger than the number of *all possible* inputs when $r \geq \frac{n}{m}$. It follows that $\frac{n}{m}$ is the minimal practically secure number of rounds. When $k < n$, the complexity $O(2^{r \cdot m})$ must only

be greater than the complexity of an exhaustive key search $O(2^k)$ and, hence, the minimal practically secure number of rounds is $\frac{k}{m}$ in that case. It follows that at least $min(\lceil \frac{n}{m} \rceil, \lceil \frac{k}{m} \rceil)$ rounds should be executed for a given configuration (n, m, k) of Iterative TST .

The analysis above is based on assumptions which might not always be fulfilled. For instance, even a strong PRNG can sometimes generate an S whose rows S_i and S_j are equal for some i and j. A TST round using such an S can not ensure 2^m possible differences between x_R and y'_R and should, hence, be executed more than just $min(\lceil \frac{n}{m} \rceil, \lceil \frac{k}{m} \rceil)$ times. Even though the probability of generating such an unfortunate S is very low[4] for usual n and m, we suggest performing one additional round to provide a certain security margin. The total number of rounds which we propose for a given configuration (n, m, k) is therefore

$$r_p = min\left(\left\lceil \frac{n}{m} \right\rceil, \left\lceil \frac{k}{m} \right\rceil\right) + 1.$$

6.3 Efficiency

The presented efficiency evaluation examines 64-bit and 128-bit versions of Iterative TST using variable segment length m. For every configuration (n, m) the number of executed rounds is $\lceil \frac{n}{m} \rceil + 1$ (i.e. we suppose a constant key length $k \geq 128$). The obtained results are compared with the ones of TST' based on the Abelian group as well as with AES and IDEA.

6.3.1 Key Generation

The key material of Iterative TST consists of a permutation P, an S-box S, and round keys $k_1, \ldots, k_r$. These objects are generated from a main key K by Algorithm 6.2.3. In comparison with the BGA of TST or TST', this algorithm is very simple and straightforward. The resulting key setup delays, presented in Figure 6.8, are noticeably shorter than the ones of TST' or TST. For smaller values m the performance of Iterative TST is competitive with AES and IDEA. The cryptosystem appears to be suitable even for the construction of cryptographic hash functions when $m \leq 8$. The key setup gets slower with growing m, but on average is 24 times (respectively 56 times) faster than the corresponding key setup of 64-bit (respectively 128-bit) TST'. A comparison with TST is even clearer, the average speedup factor is 225 for a 64-bit version and 582 for a 128-bit version.

6.3.2 Throughput

The encryption speeds of Iterative TST are presented in Figure 6.9. The achievable speedup factor is 3 to 4.5 in comparison with TST', and 10 to 14 in comparison with TST. The encryption speeds grow with an increasing m, as the number of rounds to

[4] See Theorem 6.4.1 for more details.

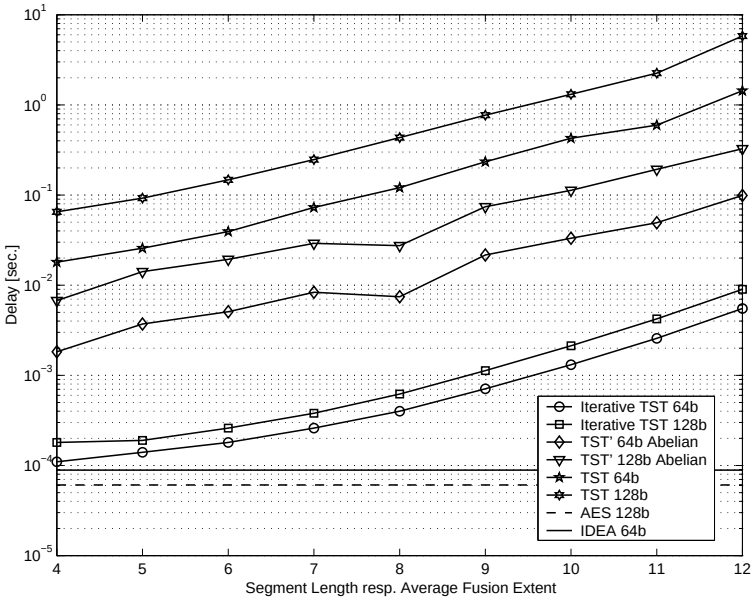


Figure 6.8: Key setup delay of Iterative TST

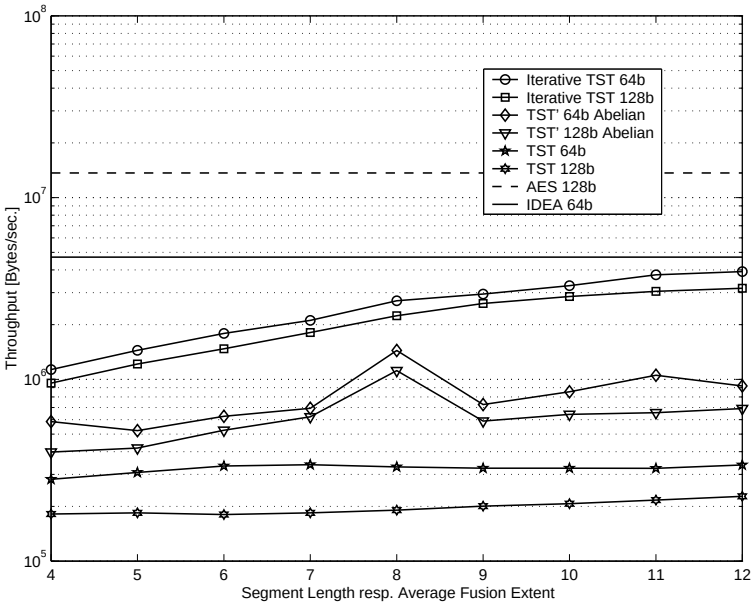


Figure 6.9: Encryption speed of Iterative TST

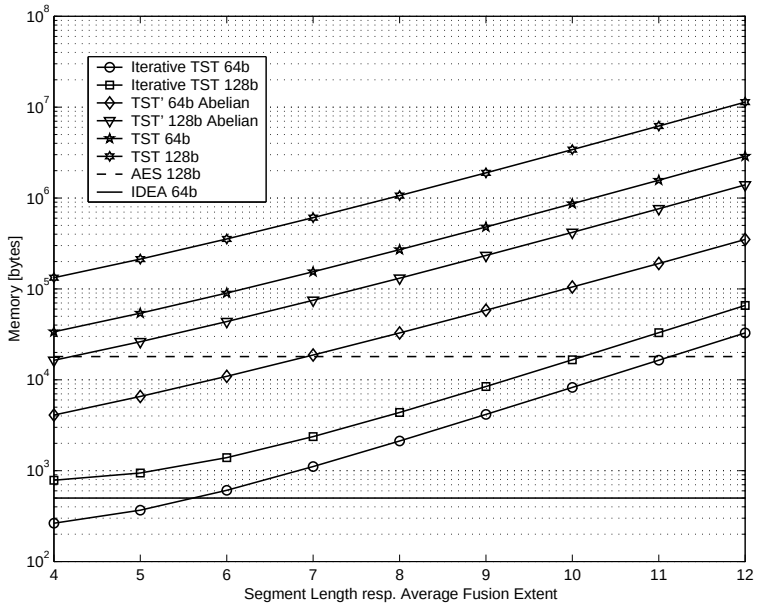


Figure 6.10: Memory requirements of Iterative TST

be executed decreases. Even though the excellent performance of AES can still not be achieved, the throughput of Iterative TST is competitive with IDEA for $m \geq 11$.

6.3.3 Memory Requirements

Reducing the high memory consumption of TST and TST' was one of the main design goals of our Iterative TST proposal. The minimal memory requirements for a configuration (m, n, r) are

$$(2^m + r) \cdot \left\lceil \frac{n-m}{8} \right\rceil$$

bytes, i.e. $2^m \cdot m$ bits for P plus $2^m \cdot (n-m)$ bits for S, and $r \cdot (n-m)$ bits for the round keys k_i. These values are presented in Figure 6.10. The resulting memory requirements are fully competitive with the ones of AES and IDEA. In comparison with TST' the average improvement factor 24 (resp. 13) is achieved when $n = 128$ (resp. 64). The average improvement factors over TST are 193, and 107 respectively. Iterative TST is suitable for a smart card implementation when $m \leq 11$.

6.4 Security

Some security properties of Iterative TST have been already discussed in Section 6.2.4. The number of rounds r_p, proposed there, is expected to ensure strong statistical properties and resistance against white box analysis. In what follows, we will further investigate security of Iterative TST. The methodology used for the presented evaluations is the same as the one used with TST (Section 4.2) and TST' (Section 5.4).

n	m	4	5	6	7	8	9	10	11	12	13	14	15	16
64	$\mathbf{r_p}$	17	14	12	11	9	9	8	7	7	6	6	6	5
	$\mathbf{r_s}$	7	6	5	4	4	3	3	3	3	3	2	2	2
	c	2.4	2.3	2.4	2.8	2.3	3.0	2.7	2.3	2.3	2.0	3.0	3.0	2.5
128	$\mathbf{r_p}$	33	27	23	20	17	16	14	13	12	11	11	10	9
	$\mathbf{r_s}$	8	6	5	5	4	4	3	3	3	3	2	2	2
	c	4.1	4.5	4.6	4.0	4.3	4.0	4.7	4.3	4.0	3.7	5.5	5.0	4.5

Table 6.1: Proposed vs. statistically secure r for Iterative TST

6.4.1 Generic Statistical Evaluation

Figure 6.11 displays suspicion rate profiles for a 64-bit Iterative TST. (The curves of a 128-bit version are almost identical.) The upper graph shows the statistical properties *depending on the number of executed rounds*. The 16 curves (going from right to the left) correspond to different segment lengths $m = 4, \ldots, 16$. The higher the segment length m, the less rounds are necessary to reach the secure interval. The output of Iterative TST using $m = 4$ (the right most curve) is statistically indistinguishable from a RNS when $r \geq 7$. The highest three segment lengths $m = 14, 15, 16$ provide a strong randomness already after two rounds. The lower graph presents the suspicion rates *depending on the segment length*. In this case every curve corresponds to a certain number of rounds. For example, the curve corresponding to one round sinks very slightly and can not reach the secure interval for $m \leq 16$. It follows that executing one round produces statistically weak results, unless we use an extremely high m. On the other hand, when executing, say, 4 rounds, all segment lengths above 6 produce a strong pseudorandomness.

The proposed number of rounds that should be executed for a given configuration (n, m, k) is $r_p = min(\lceil \frac{n}{m} \rceil, \lceil \frac{k}{m} \rceil) + 1$. This number is several times higher than the statistically strong number of rounds denoted by r_s. The corresponding security coefficient, based on randomness, is $c = \frac{r_p}{r_s}$. For instance, the statistical security coefficient of AES is 3.3 and the one of RC6 is 5 [SB00]. Table 6.1 lists the values r_p, r_s, and c of Iterative TST for different segment lengths m, supposing a constant key length of 128 bits. Obviously, when $n = 64$, the number of executed rounds is 2 to 3 times higher than the number of rounds providing a strong randomness. For $n = 128$ the coefficient takes on values between 3.7 and 5.5. It follows that, in contrast to TST' (see e.g. Figure 5.8), *any* segment length m in a suspicion rate profile of Iterative TST will provide (more than) strong randomness when executing the proposed r_p rounds.

Note that there exists no security margin in the sense of rounds, when speaking about the statistical properties of TST'. The only possibility how to incorporate a security margin into a TST'-like cryptosystem is increasing AFE, which has many undesirable side effects. For example, we concluded that AFE = 9 is the minimal value providing a strong pseudorandomness of the TST' output. To introduce a security margin we might use, say, AFE = 12 instead, but the memory requirements of such a TST' version are roughly ten times higher than for AFE = 9, so the price

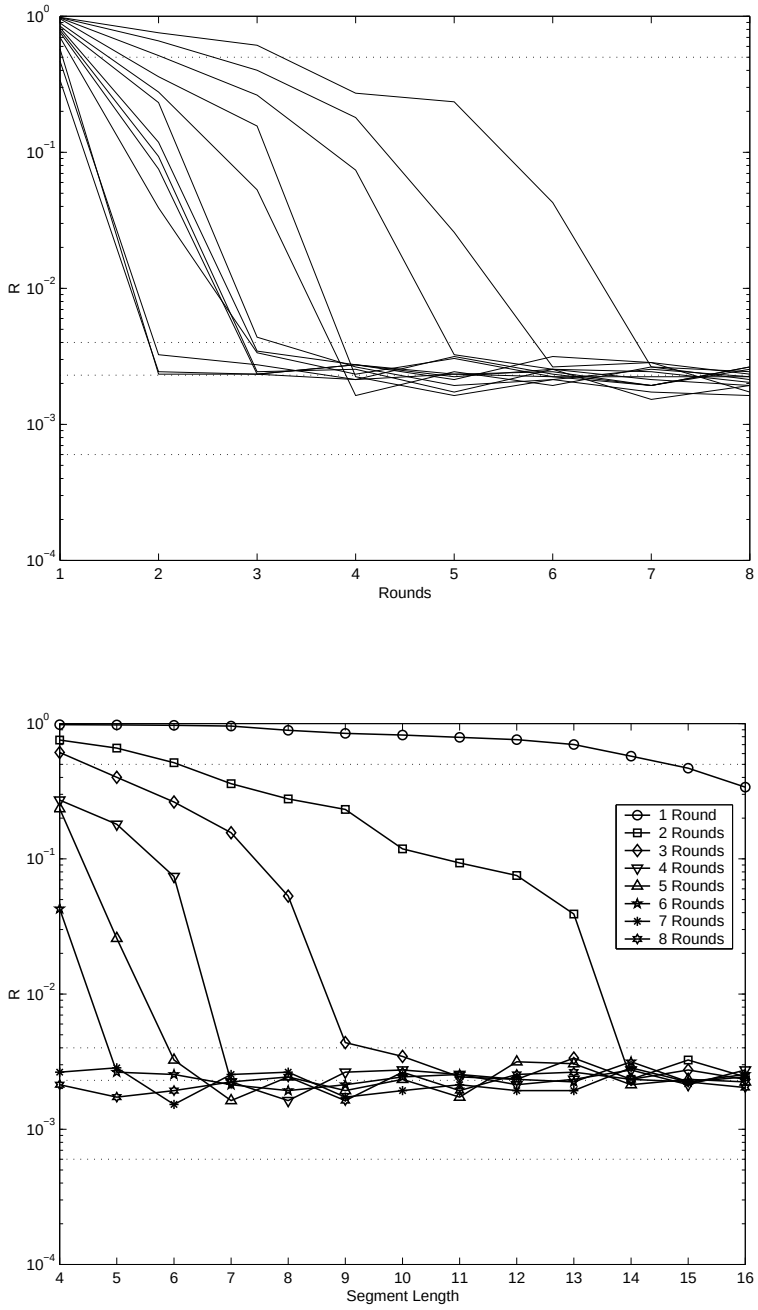


Figure 6.11: Suspicion rate profiles of Iterative TST

paid for ensuring the margin would be rather high.

6.4.2 White Box Analysis

It has been said in Section 6.2.4 that even a perfect random number generator can sometimes generate an S-box whose rows are not distinct. Theorem 6.4.1 estimates the probability of generating such an unfortunate S-box.

Theorem 6.4.1 *Let n be a block length and m a segment length of a particular instance of Iterative TST. The probability that at least two rows the S-box S are equal is approximately*

$$1 - \frac{1}{e^{2m}} \cdot \left(\frac{2^{n-m}}{2^{n-m} - 2^m} \right)^{2^{n-m} - 2^m + \frac{1}{2}}$$

when S was generated by a strong PRNG.

Proof: Let $r = n - m$, and let S be a table of $2^m \times r$ uniformly distributed random bits. The number of all possible tables S is

$$N_{all} = 2^{r \cdot 2^m} \tag{6.1}$$

The number of tables having all 2^m rows different is

$$N_{dif} = 2^r \cdot (2^r - 1) \cdot (2^r - 2) \cdots (2^r - (2^m - 1)) \tag{6.2}$$

because the first row of S can contain any of the 2^r possible r-bit vectors, the second row can contain any of the remaining $2^r - 1$ vectors, etc. Expression 6.2 can be simplified to

$$N_{dif} = \frac{2^r!}{(2^r - 2^m)!} \tag{6.3}$$

because e.g. $6 \cdot 5 \cdot 4 = \frac{6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{3 \cdot 2 \cdot 1} = \frac{6!}{3!}$. Now, using Stirling's formula

$$n! \approx \sqrt{2\pi n} \cdot \left(\frac{n}{e} \right)^n$$

we can approximate the expression 6.3 as follows

$$\begin{aligned}
N_{dif} &\approx \frac{\sqrt{2\pi 2^r} \cdot \left(\frac{2^r}{e}\right)^{2^r}}{\sqrt{2\pi(2^r - 2^m)} \cdot \left(\frac{2^r - 2^m}{e}\right)^{2^r - 2^m}} \\
&= \sqrt{\frac{2^r}{2^r - 2^m}} \cdot \frac{(2^r)^{2^r} \cdot e^{2^r - 2^m}}{(2^r - 2^m)^{2^r - 2^m} \cdot e^{2^r}} \\
&= \sqrt{\frac{2^r}{2^r - 2^m}} \cdot \frac{1}{e^{2^m}} \cdot \frac{(2^r)^{2^r - 2^m + 2^m}}{(2^r - 2^m)^{2^r - 2^m}} \\
&= \sqrt{\frac{2^r}{2^r - 2^m}} \cdot \frac{1}{e^{2^m}} \cdot \frac{(2^r)^{2^r - 2^m} \cdot (2^r)^{2^m}}{(2^r - 2^m)^{2^r - 2^m}} \\
&= \sqrt{\frac{2^r}{2^r - 2^m}} \cdot \frac{1}{e^{2^m}} \cdot \left(\frac{2^r}{2^r - 2^m} \right)^{2^r - 2^m} \cdot (2^r)^{2^m} \\
&= \frac{(2^r)^{2^m}}{e^{2^m}} \cdot \left(\frac{2^r}{2^r - 2^m} \right)^{2^r - 2^m + \frac{1}{2}}
\end{aligned}$$

The approximate probability that all rows of S are different is, hence,

$$P_{dif} = \frac{N_{dif}}{N_{all}} \approx \frac{1}{e^{2^m}} \cdot \left(\frac{2^r}{2^r - 2^m} \right)^{2^r - 2^m + \frac{1}{2}} \tag{6.4}$$

The complementary probability (that at least two rows of S are equal) is $P_{eq} = 1 - P_{dif}$. After substituting $r = n - m$ we obtain

$$P_{eq} \approx 1 - \frac{1}{e^{2^m}} \cdot \left(\frac{2^{n-m}}{2^{n-m} - 2^m} \right)^{2^{n-m} - 2^m + \frac{1}{2}} \tag{6.5}$$

which proves the Theorem 6.4.1. □

These values are really small, e.g. $P_{eq} \approx 0.76 \times 10^{-5}$ for $n = 64$, $m = 16$, $P_{eq} \approx 0.45 \times 10^{-12}$ for $n = 64$, $m = 8$, $P_{eq} \approx 0.41 \times 10^{-24}$ for $n = 128$, $m = 16$, and $P_{eq} \approx 0.25 \times 10^{-31}$ for $n = 128$, $m = 8$. Moreover, even if two or three rows of S happen to be equal, the cipher is not weakened seriously. To significantly reduce the number of possible input-output differences achievable by a TST round, at least, say, 20% of the rows of S should be equal. The probability of generating such a weak S-box is negligible when we use a statistically strong PRNG .

6.4.2.1 Key Space

The maximum number of theoretically possible different keys for an Iterative TST configuration (n, m, r) is

$$2^m! \times 2^{2^m \cdot (n-m)} \times 2^{(n-m) \cdot r} \tag{6.6}$$

i.e. $2^m!$ possible values of P, $2^{2^m \cdot (n-m)}$ possible values of S, and $2^{(n-m)}$ possible values of every k_i for $1 \leq i \leq r$. For instance, when $n = 64$, $m = 8$, and $r = 9$, the maximum number of possible keys is $256! \times 2^{14336} \times 2^{504} \simeq 2^{16524}$. Certainly, this number is much smaller than the corresponding value for TST, presented in Example 4.2.1. Nevertheless, the potentially possible 16524-bit key is still much longer than the keys of most others symmetric cryptosystems.

When the key space above is not sufficient for some application, one can use a different P and S in every round. The achievable number of different keys will grow to

$$\left(2^m! \times 2^{2^m \cdot (n-m)} \times 2^{(n-m)} \right)^r \tag{6.7}$$

which corresponds to roughly 2^{148716} different keys for the configuration above. However, in the interest of efficiency, one should keep with a single P and S if possible. An astronomically large (hypothetical) key space is in most cases less important than the memory requirements and the encryption speed of a cipher.

6.4.2.2 Possible Attacks on Simplified Versions of Iterative TST

When the PRNG used for generating P, S, and k_i is statistically strong, it is very hard to determine the highest probable input-output difference of a TST round, which

would be necessary for a differential cryptanalysis. Analogously, without knowing P, S, and k_i, it is very hard to find out the highest imbalance of an input-output sum, which is necessary for a linear cryptanalysis. Consequently, it is hard to cryptanalyze Iterative TST in general.

Some attacks might be possible on simplified versions of the cryptosystem. For example, Iterative TST can be weakened by performing one or more of the following modifications:

1. reducing the number of executed rounds, i.e. for a given n and m executing significantly less than r_p rounds,
2. using an extremely small m (e.g. $m = 2$) without compensating through a high number of rounds,
3. using an extremely weak PRNG,
4. using a trivial, possibly linear, f_1 (e.g. a simple m-bit XOR checksum)
5. using compatible, group operations $\odot$, $\otimes$, and $\oslash$ (e.g. using a simple XOR in place of all three operations),
6. disregarding the round key addition at the beginning and/or the segment rotation at the end of a round.

The effect of simplifications 1 and 2 is equivalent and (especially when combined with the simplification 6) it can have a disastrous effect on the security of the cipher. When executing just few rounds for a small m without the segment rotations, an adversary can recover S and P by fixing x_R and varying x_L through all possible values. When he repeats this for several different x_R, he can compute the rows of S based on the low number of possible input-output differences. Nevertheless, this does not seem to be possible against the full Iterative TST.

Simplification 4 (especially when combined with simplification 5) could make it possible to change x_R in such a way that the output of f_1 is held fixed. Consequently, the active row of S could be held fixed for very many (specially prepared) different inputs. An attacker who would be able to fix the active S-box rows in $r - 1$ rounds, could easily recover S and P by "parsing" just the last r-th round. The complexity of such an attack might possibly be as low as $O(2^m)$. Again, this does not seem to be possible against full Iterative TST with a non-trivial f_1.

It is generally possible to reconstruct P and S from a relatively small fragment if they have been generated by a cryptographically insecure PRNG. When the PRNG is extremely simple, it might be possible to exploit even a tiny piece of information, say, 2×32 consecutive bits of S. In this case, when an adversary were able to somehow reveal a genuine 64 bit sequence from any place of S, he would be able to break the complete cipher. However, such a weak PRNG is not very probable to be used for the key setup, because it would limit the effectively achievable key space of the particular Iterative TST implementation to 64 bits. The internal state of the lagged Fibonacci PRNG proposed in our implementation is representable by

3200 bits[5], so an adversary would have to reveal at least 3200 consecutive bits of S to recompute the secret information. Or, possibly, he might recover just x bits of the sequence, and find the remaining $3200 - x$ bits by brute force. All this does not seem to be possible for the full Iterative TST .

6.4.2.3 Possible Improvements of Iterative TST

Based on the considerations above, there are several possible improvements of Iterative TST which can be made for further strengthening the cryptosystem, when necessary. These are:

1. increasing the number of executed rounds,
2. using more complex f_1 (e.g. by making the internal scheme of f_1 truly key-dependent),
3. using some more complex group operations, especially in place of $\odot$ and $\oslash$,
4. using a cryptographically (not just statistically) strong PRNG,
5. performing a rotation by c instead of m bits at the end of a round, where c and n are relatively prime.

The proposals 1 to 3 are rather straightforward, but they certainly cause a slowdown of the encryption speed. Modification 4, which works against the recomputation attacks on P and S, will definitely slow down the key setup procedure. Nevertheless, when using some efficient cryptographically secure PRNG (e.g. [KSF99]), the slowdown might still be acceptable.

The last modification does not increase the computational complexity and, hence, should not slow down the encryption. The motive for such a modification is making the cipher a *prime network*, i.e. ensuring that the length of cipher's cycle[6] is equal to its block length. For example, the authors of [SK96] have shown an evidence that a prime UFN is very hard to cryptanalyze. We expect that the same is true for a prime version of Iterative TST. The constant c should be specified according to the following rules:

[5] In fact, it is slightly more because of the used Lütscher's approach. See e.g the references in Section 4.1.1.1 for more details.

[6] *Cycle* is the minimal number of rounds which is needed for an input bit to come on its original position.

- c has no common divisors with n to make the network prime.
- c has no common divisors with m' to ensure that the cycle related to the m'-bit words (not just to the block length) is maximal as well. This will create more complex dependencies for f_1 and $\oslash$ which both work word-wise. (Note that the fulfillment of this condition is, in most cases, implied by the first condition, because both n and m' are usually powers of 2.)
- $c \geq m$ to ensure that the content of the segments completely changes after every round.
- When $\oslash$ is defined as $+ \bmod 2^{m'}$, c should be close to $\frac{m'}{2}$ to ensure that all bits appear alternately on both higher and lower order bit position inside the m'-bit words. This will improve the diffusion properties of the series of $\oslash$ operations.

According to the points above, for instance, value 17 is a suitable candidate for c when n is a power of 2, $m' = 32$, and $m \leq 17$. Value 31 might be used when $m' = 64$.

6.5 Summary

The cryptosystem Iterative TST was designed to improve the efficiency of TST and TST'. The new design simplifies the group basis oriented approach and combines it with the conventional iterative approach. The resulting cipher provides a full scalability, similar to TST and TST', but is more flexible and efficient than these ciphers. The structure of the new cryptosystem is similar to a heterogeneous UFN. The round design incorporates variable components and thus enables several possible realizations. Taking efficiency and security into account, we have introduced and analyzed one possible implementation of the cryptosystem.

From the statistical point of view the proposed number of rounds ensures excellent randomness properties of the cipher, and even provides a substantial security margin. In contrast to TST and TST' the statistical properties of Iterative TST are very good for *any* segment length m.

We were only able to construct attacks against weakened versions of Iterative TST . Constructing an attack on the full version does not seem to be feasible. The pseudorandom components used in our round design make a differential or linear cryptanalysis hard. Moreover, because of its flexibility, the cryptosystem can be easily strengthened in case that some weakness should be discovered in the basic version.

The efficiency of Iterative TST has been significantly improved over TST and TST'. The key setup of a 128-bit version is on average 56 times faster when compared with TST', and 582 times faster when compared with TST. The encryption speed is up to 4.5 times higher in comparison with TST' and up to 14 times higher in comparison with TST. The memory requirements have been reduced by a factor of 24, and 193 respectively. As a consequence, Iterative TST is suitable for both smart card

Cipher	**Key Setup Time**†	**Encryption Time**†
Iterative TST (m=12)	$\mathbf{1.2 \times 10^6}$	**1900**
Iterative TST (m=6)	**82000**	**3700**
TST'	25×10^6	9500
TST	269×10^6	27882
Magenta	30	6539
Frog	1.4×10^6	2417
Loki 97	7430	2134
Safer+	4278	1722
Rijndael (AES)	305, 1389‡	374
RC6	1632	270

† Measured in clock cycles on Intel Pentium II processor

‡ Key setup delays for encryption and decryption are different

Table 6.2: Efficiency of Iterative TST in comparison with other ciphers

implementation and usage in cryptographically secure hash functions when $m \leq 8$. Table 6.2 compares two representative variants of Iterative TST with cryptosystems based on group bases, as well as with some of the AES candidates.

Chapter 7

Conclusions

Adjustability and scalability are two very desirable properties of block ciphers. They make cryptosystems adaptable for various environments and provide a sufficient security margin for the future. Unfortunately, in spite their excellent efficiency and strong security properties, most of present day block ciphers are not fully scalable and adjustable by design.

One of possible approaches to constructing adjustable and scalable block ciphers is based on group bases. PGM and TST are two representatives of this class of ciphers. PGM, based on the full symmetric group, provides strong cryptographic properties, but has several inherent drawbacks that make it less suitable for practical use. The recent cryptosystem TST based on the Sylow 2-subgroup of the symmetric group has solved the problems of PGM in an impressing way, but has not been deeply cryptanalyzed yet. A hardware implementation of a simplified TST has proven itself to be efficient, but because the design is still rather new, there have been no other implementation efforts yet. Our thesis presents the first deep analysis of TST and introduces two new improved block cipher designs based on group bases.

The results of the thesis can be summarized as follows:

- *Chapter 4*

  We discuss an efficient implementation of the TST key generation procedure and presented its possible extension to the usual binary vector format as well as to the pass-phrase format.

  We implemented the first software version of TST. This was the first implementation ever which provided full functionality of TST.

  Our measurements confirm that the cartesian representation of permutations is more suitable for an efficient software implementation of TST than the compact representation. Unfortunately, the efficiency of TST in software is not competitive with modern block ciphers.

  Our generic security evaluation of TST based on randomness studies has shown that the statistical properties of TST are weak. We have shown, furthermore,

that efficient attacks can be mounted against TST - especially against its simplified chip version. The security problems are caused mainly by weak diffusion of the involved operations.

- *Chapter 5*

  We introduce the notion of an extended group basis whose factorization and composition operations ensure strong diffusion.

  We designed and implemented a modified variant of TST based on extended group bases. The new design called TST' makes it possible to use another, more efficient, commutative carrier group.

  TST' based on the commutative group is noticeably faster than the original TST. The key setup became simpler and faster, and the memory requirements of the most efficient version are reduced in comparison with TST.

  The statistical properties of TST' are noticeably better than those of TST, and the attack that was possible against TST can be avoided by using a proper class of extended group bases.

- *Chapter 6*

  By simplifying the TST' encryption scheme we have designed a round function that can be used for constructing an iterative version of TST. The resulting cryptosystem is fully scalable and adjustable.

  Our measurements have shown that the new cipher is much more efficient than the ciphers based on full group bases. The encryption speed, the key setup speed, as well as the memory efficiency have been improved significantly.

  The statistical properties of the new cipher are excellent in all configurations, and we were not able to construct any efficient attacks against the full version of the cryptosystem.

We conclude that there is an "intersection" between the iterative approach and the group basis oriented approach to the construction of block ciphers. A combination of these two designs makes it possible to construct block ciphers that are fully scalable and adjustable on the one hand, and reasonably efficient and easily implementable on the other hand. This seems to be a suitable way for designing well founded scalable block ciphers that might be used in the future.

There are still several open questions regarding the new iterative cryptosystem:

- The structure based on key dependent random tables seems not to be easily attackable by differential or linear cryptanalysis. Is it possible to mount a differential or linear attack on Iterative TST or, at least, on its simplified versions? What is the complexity of such attacks?

- Our implementation of the hashing function h is just one of many possible solutions. Are there some more efficient (or even optimal) designs for h?

- Block cipher BEAR (Definition 6.2.2) based on cryptographically secure components provides a kind of provable security. Is it possible to achieve provable security using Iterative TST based on some special (i.e. cryptographically or statistically strong) components? What is the necessary number of rounds for such a construction?

These and related problems might be of interest for some future research.

Appendix A

Basic Empirical Tests

In what follows we list the five basic empirical randomness tests for binary sequences [MVV97, p.181]:

Frequency test (monobit test). The purpose of this test is to determine whether the number of 0's (n_0) and number of 1's (n_1) in a sequence are approximately the same (as would be expected for a RNS). The statistic used is

$$X = \frac{(n_0 - n_1)^2}{n}$$

which approximately follows a χ^2 distribution with 1 degree of freedom.

Serial test (m-bit test). When $m = 2$, the purpose of this test is to determine, whether the numbers of occurrences of *overlapping* subsequences 00, 01, 10 and 11 in an n-bit sample sequence are approximately the same (as would be expected for a RNS). Note that $n_{00} + n_{01} + n_{10} + n_{11} = (n-1)$ since the subsequences are allowed to overlap. The statistic used is

$$X = \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n}(n_0^2 + n_1^2) + 1$$

which approximately follows a χ^2 distributions with 2 degrees of freedom. Certainly, the test can be generalized to higher values of m, and we can examine the triples, quadruples or longer subsequences of bits. However, because of practical problems one usually uses less exact tests (e.g. the poker test) for larger m's.

Poker test. Let m be a positive integer such that $\lfloor \frac{n}{m} \rfloor \geq 5 \cdot 2^m$ and let $k = \lfloor \frac{n}{m} \rfloor$. The poker test divides a sequence into k non-overlapping parts of m bits and counts the number of occurrences n_i of every possible m-bit block, $1 \leq i \leq 2^m$. The test determines whether each m-bit pattern appears approximately the same number of times (as would be expected for a RNS). The statistic used is

$$X = \frac{2^m}{k} \cdot \left(\sum_{i=1}^{2^m} n_i^2 \right) - k$$

which approximately follows a χ^2 distribution with $2^m - 1$ degrees of freedom. Note that the poker test is a generalization of the frequency tests (which is a special case for $m = 1$).

Runs test. A *run* is a subsequence consisting of consecutive 0's or consecutive 1's which is neither preceded nor succeeded by the same symbol. The purpose of the runs test is to determine whether the number of runs of various lengths in the sequence is as expected for a RNS. The expected number of runs of length i in a RNS of length n is $e_i = (n - 1 + 3)/2^{i+2}$. Let k be equal to the largest integer i for which $e_i \geq 5$. Let B_i be the number of runs of 1's and G_i the number of runs of 0's of length i for each $1 \leq i \leq k$. The statistic used is

$$X = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i}$$

which approximately follows a χ^2 distribution with $2k - 2$ degrees of freedom.

Autocorrelation test. The purpose of this test is to check for correlations between an n-bit sequence s and the shifted versions of it. Let d be a fixed integer such that $1 \leq d \leq \lfloor n/2 \rfloor$. The number of bits in s not equal to their d-shifts is $A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$. The statistic used is

$$X = 2 \cdot \frac{A(d) - \frac{n-d}{2}}{\sqrt{n-d}}$$

which approximately follows an $N(0, 1)$ distribution. Since small values of $A(d)$ are as unexpected as large values, a two-sided test should be used.

Note: Descriptions of more empirical tests can be found, for example, in [Knu97] and [SB00].

Appendix B

The DieHard Test Battery

The DieHard randomness test suite [Mar97] consists of the following tests:

The birthday spacings test. Choose m birthdays in a year of n days. List the spacings between the birthdays. If j is the number of values that occur more than once in that list, then j is asymptotically Poisson distributed with mean $\frac{m^3}{4n}$. Experience shows n must be quite large, say $n \leq 2^{18}$, for comparing the results to the Poisson distribution with that mean. This test uses $n = 2^{24}$ and $m = 2^9$, so that the underlying distribution for j is taken to be Poisson with $\lambda = \frac{2^{27}}{2^{26}} = 2$. A sample of 500 j's is taken, and a χ^2 goodness of fit test provides a p value. The first test uses bits 1 to 24 (counting from the left) from integers in the specified file. Then the file is closed and reopened. Next, bits 2 to 25 are used to provide birthdays, then 3 to 26 and so on to bits 9 to 32. Each set of bits provides a p-value, and the nine p-values provide a sample for a Kolmogorov-Smirnov test.

The overlapping 5-permutation test. The test looks at a sequence of one million 32-bit random integers. Each set of five consecutive integers can be in one of 120 states, for the 5! possible orderings of five numbers. Thus the 5th, 6th, 7th, ... numbers each provide a state. As many thousands of state transitions are observed, cumulative counts are made of the number of occurrences of each state. Then the quadratic form in the weak inverse of the 120×120 covariance matrix yields a test equivalent to the likelihood ratio test that the 120 cell counts came from the specified (asymptotically) normal distribution with the specified 120×120 covariance matrix (with rank 99). This version uses 1,000,000 integers, twice.

The binary rank test for 31×31 matrices. The leftmost 31 bits of 31 random integers from the test sequence are used to form a 31×31 binary matrix over the field $\{0, 1\}$. The rank is determined. That rank can be from 0 to 31, but ranks < 28 are rare, and their counts are pooled with those for rank 28. Ranks are found for 40,000 such random matrices and a χ^2 test is performed on counts for ranks 31, 30, 29, and ≤ 28.

The binary rank test for 32×32 matrices. A random 32×32 binary matrix is formed, each row a 32-bit random integer. The rank is determined. That rank can be from 0 to 32, ranks less than 29 are rare, and their counts are pooled with

those for rank 29. Ranks are found for 40,000 such random matrices and a χ^2 test is performed on counts for ranks 32, 31, 30, and ≤ 29.

The binary rank test for 6×8 matrices. From each of six random 32-bit integers from the generator under test, a specified byte is chosen, and the resulting six bytes form a 6×8 binary matrix whose rank is determined. That rank can be from 0 to 6, but ranks 0, 1, 2, and 3 are rare; their counts are pooled with those for rank 4. Ranks are found for 100,000 random matrices, and a χ^2 test is performed on counts for ranks 6,5 and ≤ 4.

The bitstream test. The file under test is viewed as a stream of bits. Call them $b_1, b_2, \ldots$. Consider an alphabet with two "letters", 0 and 1 and think of the stream of bits as a succession of 20-letter "words", overlapping. Thus the first word is $b_1 b_2 \ldots b_{20}$, the second is $b_2 b_3 \ldots b_{21}$, and so on. The bitstream test counts the number of missing 20-letter (20-bit) words in a string of 2^{21} overlapping 20-letter words. There are 2^{20} possible 20 letter words. For a truly random string of $2^{21}+19$ bits, the number of missing words j should be (very close to) normally distributed with mean 141,909 and sigma 428. Thus $\frac{j-141909}{428}$ should be a standard normal variate (z score) that leads to a uniform $[0,1)$ p value. The test is repeated twenty times.

The overlapping-pairs-sparse-occupancy test. The OPSO test considers 2-letter words from an alphabet of 1024 letters. Each letter is determined by a specified ten bits from a 32-bit integer in the sequence to be tested. OPSO generates 2^{21} (overlapping) 2-letter words (from $2^{21}+1$ "keystrokes") and counts the number M of missing words - that is 2-letter words which do not appear in the entire sequence. That count should be very close to normally distributed with mean 141,909, sigma 290. Thus $\frac{M-141909}{290}$ should be a standard normal variable. The OPSO test takes 32 bits at a time from the test file and uses a designated set of ten consecutive bits. It then restarts the file for the next designated 10 bits, and so on.

The overlapping-quadruples-sparse-occupancy test. The test OQSO is similar, except that it considers 4-letter words from an alphabet of 32 letters, each letter determined by a designated string of 5 consecutive bits from the test file, elements of which are assumed 32-bit random integers. The mean number of missing words in a sequence of 2^{21} four-letter words, ($2^{21}+3$ "keystrokes"), is again 141909, with sigma = 295. The mean is based on theory; sigma comes from extensive simulation.

The DNA test. This test considers an alphabet of 4 letters C,G,A,T, determined by two designated bits in the sequence of random integers being tested. It considers 10-letter words, so that as in OPSO and OQSO, there are 2^{20} possible words, and the mean number of missing words from a string of 2^{21} (overlapping) 10-letter words ($2^{21}+9$ "keystrokes") is 141909. The standard deviation sigma=339 was determined as for OQSO by simulation. (Sigma for OPSO, 290, is the true value (to three places), not determined by simulation.

The count-the-1's test on a stream of bytes. Consider the file under test as a stream of bytes (four per 32 bit integer). Each byte can contain from 0 to 8 ones, with probabilities 1, 8, 28, 56, 70, 56, 28, 8, 1 over 256. Now let the stream of bytes provide a string of overlapping 5-letter words, each "letter" taking values A, B, C, D, E. The letters are determined by the number of 1's in a byte 0, 1, or 2 yield A, 3 yields B, 4 yields C, 5 yields D and 6, 7, or 8 yield E. Thus we have a monkey at a typewriter hitting five keys with various probabilities (37, 56, 70, 56, 37 over 256). There are 5^5 possible 5-letter words, and from a string of 256,000 (overlapping) 5-letter words, counts are made on the frequencies for each word. The quadratic form in the weak inverse of the covariance matrix of the cell counts provides a χ^2 test Q5-Q4, the difference of the naive Pearson sums of $\frac{(OBS-EXP)^2}{EXP}$ on counts for 5- and 4-letter cell counts.

The count-the-1's test for specific bytes. Consider the file under test as a stream of 32-bit integers. From each integer, a specific byte is chosen, say the leftmost bits 1 to 8. Each byte can contain from 0 to 8 ones, with probabilities 1, 8, 28, 56, 70, 56, 28, 8, 1 over 256. Now let the specified bytes from successive integers provide a string of (overlapping) 5-letter words, each "letter" taking values A, B, C, D, E. The letters are determined by the number of 1's, in that byte 0, 1, or 2 $\longrightarrow$ A, 3 $\longrightarrow$ B, 4 $\longrightarrow$ C, 5 $\longrightarrow$ D, and 6, 7, or 8 $\longrightarrow$ E. Thus we have a monkey at a typewriter hitting five keys with various probabilities 37, 56, 70, 56, 37 over 256. There are 5^5 possible 5-letter words, and from a string of 256,000 (overlapping) 5-letter words, counts are made on the frequencies for each word. The quadratic form in the weak inverse of the covariance matrix of the cell counts provides a χ^2 test Q5-Q4, the difference of the naive Pearson sums of $\frac{(OBS-EXP)^2}{EXP}$ on counts for 5- and 4-letter cell counts.

The parking lot test. In a square of side 100, randomly "park" a car - a circle of radius 1. Then try to park a 2nd, a 3rd, and so on, each time parking "by ear". That is, if an attempt to park a car causes a crash with one already parked, try again at a new random location. (To avoid path problems, consider parking helicopters rather than cars.) Each attempt leads to either a crash or a success, the latter followed by an increment to the list of cars already parked. If we plot n: the number of attempts, versus k the number successfully parked, we get a curve that should be similar to those provided by a perfect random number generator. Theory for the behavior of such a random curve seems beyond reach, and as graphics displays are not available for this battery of tests, a simple characterization of the random experiment is used: k, the number of cars successfully parked after $n = 12{,}000$ attempts. Simulation shows that k should average 3523 with sigma 21.9 and is very close to normally distributed. Thus $\frac{k-3523}{21.9}$ should be a standard normal variable, which, converted to a uniform variable, provides input to a Kolmogorov-Smirnov test based on a sample of 10.

The minimum distance test. It does this 100 times choose $n = 8000$ random points in a square of side 10000. Find d, the minimum distance between the $\frac{n^2-n}{2}$ pairs of points. If the points are truly independent uniform, then d^2, the square of the minimum distance should be (very close to) exponentially distributed with

mean 0.995 . Thus $1-e^{-\frac{d^2}{0.995}}$ should be uniform on $[0,1)$ and a Kolmogorov-Smirnov on the resulting 100 values serves as a test of uniformity for random points in the square. Test numbers = 0 mod 5 are printed but the Kolmogorov-Smirnov test is based on the full set of 100 random choices of 8000 points in the 10000×10000 square.

The 3D spheres test. Choose 4000 random points in a cube of edge 1000. At each point, center a sphere large enough to reach the next closest point. Then the volume of the smallest such sphere is (very close to) exponentially distributed with mean $\frac{120\pi}{3}$. Thus the radius cubed is exponential with mean 30. (The mean is obtained by extensive simulation). The 3D spheres test generates 4000 such spheres 20 times. Each min radius cubed leads to a uniform variable by means of $1-e^{-\frac{r^3}{30}}$, then a Kolmogorov-Smirnov test is done on the 20 p-values.

The squeeze test. Random integers are floated to get uniforms on $[0,1)$. Starting with $k = 2^{31} = 2147483647$, the test finds j, the number of iterations necessary to reduce k to 1, using the reduction $k = \lceil k * U \rceil$, with U provided by floating integers from the file being tested. Such j's are found 100,000 times, then counts for the number of times j was $\leq 6, 7, \ldots, 47, \leq 48$ are used to provide a χ^2 test for cell frequencies.

The overlapping sums test. Integers are floated to get a sequence $U_1, U_2, \ldots$ of uniform $[0,1)$ variables. Then overlapping sums, $S_1 = U_1 + \ldots + U_{100}$, $S_2 = U_2 + \ldots + U_{101}$, ... are formed. The S's are virtually normal with a certain covariance matrix. A linear transformation of the S's converts them to a sequence of independent standard normals, which are converted to uniform variables for a Kolmogorov-Smirnov test. The p-values from ten Kolmogorov-Smirnov tests are given still another Kolmogorov-Smirnov test.

The runs test. It counts runs up, and runs down, in a sequence of uniform $[0,1)$ variables, obtained by floating the 32-bit integers in the specified file. This example shows how runs are counted: 0.123, 0.357, 0.789, 0.425, 0.224, 0.416, 0.95 contains an up-run of length 3, a down-run of length 2 and an up-run of (at least) 2, depending on the next values. The covariance matrices for the runs-up and runs-down are well known, leading to χ^2 tests for quadratic forms in the weak inverses of the covariance matrices. Runs are counted for sequences of length 10,000. This is done ten times. Then repeated.

The craps test. It plays 200,000 games of craps, finds the number of wins and the number of throws necessary to end each game. The number of wins should be (very close to) a normal with mean $200000p$ and variance $200000 \cdot p \cdot (1-p)$, with $p = \frac{244}{495}$. Throws necessary to complete the game can vary from 1 to infinity, but counts for all > 21 are lumped with 21. A χ^2 test is made on the number-of-throws cell counts. Each 32-bit integer from the test file provides the value for the throw of a die, by floating to $[0,1)$, multiplying by 6 and taking 1 plus the integer part of the result.

Note: Most of the tests in DieHard return a p-value, which should be uniform on $[0, 1)$ if the input file contains truly independent random bits. Those p-values are obtained by $p = F(X)$, where F is the assumed distribution of the sample random variable X - often normal. But that assumed F is just an asymptotic approximation, for which the fit will be worst in the tails. Thus one should not be surprised with occasional p-values near 0 or 1, such as 0.0012 or 0.9983. When a bit stream really *fails big*, the resulting p's will contain 0 or 1 on six or more decimal places [Mar97].

List of Notations

Units

s	Second
b	Bit
B	Byte (8 bits)
KB	Kilobyte (2^{10} bytes)
MB	Megabyte (2^{20} bytes)
MHz	Megahertz (10^6 cycles per second)

Abbreviations

AES	Advanced Encryption Standard	[Nat97]
AFE	Average Fusion Extent	Section 4.1
BEAR	Name of a symmetric cryptosystem	[AB96]
BGA	Basis Generation Algorithm	Sections 3.2.1, 3.3.2
CBC	Cipher Block Chaining	[Sch96, Sec. 9.3]
CRC	Cyclic Redundancy Code	[PFTV88, Sec. 20.3]
DES	Data Encryption Standard	[Uni77]
GUFN	Generalized Unbalanced Feistel Network	Section 6.2.1.1
IDEA	Name of a symmetric cryptosystem	[LM91]
PGM	Name of a symmetric cryptosystem	Section 3.2
PRNG	Pseudorandom Number Generator	Section 2.2
PRNS	Pseudorandom Number Sequence	Section 2.2
RC6	Name of a symmetric cryptosystem	[RRSY98]
RNG	(True) Random Number Generator	Section 2.2
RNS	(True) Random Number Sequence	Section 2.2
TST	Name of a symmetric cryptosystem	Section 3.3
TST'	Name of a symmetric cryptosystem	Section 5.2
UFN	Unbalanced Feistel Network	Section 6.2.1.1

Symbols

x', x'', x''', ...	Modified or alternative versions of x
x_i	i-th element of x
$x^{(i)}$	i-th bit of x (in cases when x_i has a different meaning)
$\|x\|$	Size or length of x
$x\|\|y$	Concatenation of x and y
$\lfloor x \rfloor$	Floor of x (i.e. $\max_{k \leq x}(k)$)
$\lceil x \rceil$	Ceiling of x (i.e. $\min_{k \geq x}(k)$)
$\simeq$	Approximately equal
$\approx$	(Very) roughly equal
$\gg$	Much bigger than
c, c', c_i	Constants
k	Coefficient or factor
Δ	Difference
$d(.,.)$	Distance between two objects
$d'(.,.)$	Normalized distance between two objects ($0 \leq d' \leq 1$)
ProcName	Name of a procedure used during an algorithm
$O(.)$	Computational complexity
$N(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2
χ^2	Chi-square distribution
$\square$	End of proof
$\mathbb{N}$	Infinite set $\{1, 2, 3, \ldots\}$
$\mathbb{N}_n$	Finite set $\{1, 2, \ldots, n\}$
$\mathbb{Z}$	Infinite set $\{0, 1, 2, \ldots\}$
$\mathbb{Z}_n$	Finite set $\{0, 1, \ldots, n-1\}$
$\mathbb{Z}_n^*$	Set of all possible subsets of $\mathbb{Z}_n$
x, y, z	Elements of a set or of a group
i, j, k, l	Indices
p, p', p_i	Permutations
id	Identity permutation
G, G_i	Groups
$\|G\|$	Order of G
$*$	The group operation for G
$/$	The inverse group operation for G
$\oplus, \odot, \otimes, \oslash, \boxplus, \boxminus, \otimes^{(i)}$	Diverse group operations
S_n	Full symmetric group
$\mathbb{Z}_2^n$	Elementary Abelian group (a subgroup of S_{2n})
$\mathcal{H}_s$	Sylow 2-subgroup of S_{2^s}
$\mathcal{H}_s \times \mathcal{H}_1$	Wreath product of two Sylow subgroups
$\mathcal{B}_G$	Set of all group bases for G
$\mathcal{B}_G^*$	Set of all transversal group bases for G
β, β', β_i	Various group bases
α	Canonical group basis

w	Dimension of a group basis
X	Space of coordinate vectors
B_i	The i-th block of a group basis
$b_{i,j}$	The j-th element of B_i
c_i	The set of key bit positions for B_i
r_i	Number of elements of B_i
$\tilde{\beta}(.)$	Composition with respect to β
$\tilde{\beta}^{-1}(.)$	Factorization with respect to β
$\mathcal{T}$	Set of transformations
T, T', T_i	Various transformations
$\beta^{(\mathcal{T})}$	$\mathcal{T}$-extended group basis
$\tilde{\beta}^{(\mathcal{T})}(.)$	Composition with respect to $\beta^{(\mathcal{T})}$
$\tilde{\beta}^{(\mathcal{T})-1}(.)$	Factorization with respect to $\beta^{(\mathcal{T})}$
$\mathcal{P}$	Plaintext space
$\mathcal{C}$	Ciphertext space
$\mathcal{M}$	Message space (notion used when $\mathcal{P} = \mathcal{C}$)
$\mathcal{K}$	Key space
$e_K(.)$	Encryption function
$d_K(.)$	Decryption function
x, x'	Plaintexts
y, y'	Ciphertexts
K, K'	Keys
k_i	Round key for the i-th round
n	Block length
k	Key length
k'	Round key length
m	Segment length
r	Number of rounds
r_p	Proposed r
r_s	Statistically strong r
P	Permutation box
S	Substitution box
$\{0,1\}^n$	Set of all n-bit binary vectors
$\oplus$	Binary XOR operation
$\gamma(.,.)$	Bit extraction function (Section 5.2.2)
$\delta(.,.,.)$	Bit initialization function (Section 5.2.2)
$rot_x(.)$	Rotation by x bits
$h(.)$	Hash function
x_L	First m bits of x
x_R	All but first m bits of x
ξ	Segment rotation (i.e. $rot_m(.)$)

Bibliography

[AB96] Ross Anderson and Eli Biham. Two practical and provably secure block ciphers: BEAR and LION. In Dieter Gollman, editor, *Fast Software Encryption, 3rd International Workshop Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 114–120, Berlin, Germany, 1996. Springer-Verlag.

[AT90] Carlisle Adams and Stafford Tavares. Structured design of cryptographically good S-boxes. *Journal of Cryptology*, 3(1):27–41, 1990.

[BBS87] Leonore Blum, Manuel Blum, and Mike Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing, vol. 15 (1986):2, 364-383*, 1987.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. *Lecture Notes in Computer Science*, 1592:12–23, 1999.

[Big89] Norman L. Biggs. *Discrete Mathematics*. Oxford Science Publications. Clarendon Press, revised edition, 1989.

[BS90] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1990.

[But91] Gregory Butler. *Fundamental Algorithmus for Permutation Groups*. Springer-Verlag, Berlin, 1991.

[Cal94] Cristian S. Calude. *Information and Randomness*. Springer, Berlin, 1994.

[DM96] John D. Dixon and Brian Mortimer. *Permutation groups*. Springer-Verlag, New York, 1996.

[DR98] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. NIST AES Proposal, June 1998.

[ea99] James R. Nechvatal et. al. Status report on the first round of the development of the advanced encryption standard. NIST Report, 1999.

[Gla99] Brian Gladman. Implementation experience with AES candidate algorithms. Second AES Conference, 1999.

[Hey97] Burkhard Heyber. *Zur Güte von Zufallsprozessen in der Kryptologischen Praxis*. PhD thesis, Department of Electrical Engineering, FernUniversität Hagen, Hagen, Germany, 1997.

[Hor98] Tamás Horváth. *The Symmetric Cryptosystem TST*. PhD thesis, Department of Engineering, University of Essen, Essen, Germany, 1998.

[Knu97] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, third edition, 1997.

[KSF99] John Kelsey, Bruce Schneier, and Niels Ferguson. Notes on the design and analysis of the Yarrow cryptographic pseudorandom number generator. In *Sixth Annual Workshop on Selected Areas in Cryptography '99: proceedings*, Berlin, Germany, 1999. Springer-Verlag.

[LH94] Susan K. Langford and Martin E. Hellman. Differential-linear cryptanalysis. *Lecture Notes in Computer Science*, 839:17–25, 1994.

[LM91] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In I. B. Damgård, editor, *Advances in cryptology EUROCRYPT '90: proceedings*, volume 473 of *Lecture Notes in Computer Science*, pages 55–70, Berlin, Germany, 1991. Springer-Verlag.

[LMM92] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In Donald Watts Davies, editor, *Advances in cryptology, EUROCRYPT '91: proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38, Berlin, Germany, 1992. Springer-Verlag.

[LR88] Michael G. Luby and Charles W. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, April 1988.

[Mag86] Spyros S. Magliveras. A cryptosystem from logarithmic signatures of finite groups. *Proceedings of the 29-th Midwest Symposium on Circuits and Systems*, pages 972–975, 1986.

[Mar97] George Marsaglia. Diehard – battery of randomnes tests. http://stat.fsu.edu/∼geo/diehard.html, http://www.helsbreth.org/random/diehard.html, 1997.

[Mas94a] James L. Massey. SAFER K–64: One year later. In Bart Preneel, editor, *Fast Software Encryption: Second International Workshop*, volume 1008 of *Lecture Notes in Computer Science*, pages 212–241, Berlin, 1994. Springer-Verlag.

[Mas94b] James L. Massey. SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm. In R. Anderson, editor, *Fast Software Encryption*, volume 809 of *Lecture Notes in Computer Science*, pages 1–17, Berlin, 1994. Springer Verlag.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *Advances in cryptology, EUROCRYPT '93: proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397, Berlin, Germany, 1994. Springer-Verlag.

[Mau92a] Ueli M. Maurer. A simplified and generalized treatment of Luby-Rackoff pseudorandom permutation generators. In R. A. Rueppel, editor, *Advances in Cryptology—EUROCRYPT 92*, volume 658 of *Lecture Notes in Computer Science*, pages 239–255. Springer-Verlag, 1992.

[Mau92b] Ueli M. Maurer. A universal statistical test for random bit generators. *Journal of Cryptology*, 5(2):89–105, 1992.

[Mem89] Nasir D. Memon. On logarithmic signatures and applications, 1989.

[MM89] Spyros S. Magliveras and Nasir D. Memon. Linear complexity profile analysis of the PGM cryptosystem. *Congressus Numerantium*, 72:51–60, 1989.

[MM90] Spyros S. Magliveras and Nasir D. Memon. Complexity tests for cryptosystem PGM. *Congressus Numerantium*, 79:61–68, 1990.

[MM92] Spyros S. Magliveras and Nasir D. Memon. Algebraic properties of cryptosystem PGM. *Journal of Cryptology*, 5(3):167–183, 1992.

[MSvT00] Spyros S. Magliveras, Douglas R. Stinson, and Trung van Tran. New approaches to designing public key cryptosystems using one-way functions and trap-doors in finite groups. *Preprint, Institute for Experimental Mathematics, University of Essen, Germany*, (8), 2000.

[MVV97] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA, 1997.

[Nat94] National Institute of Standards and Technology (NIST). *FIPS Publication 140-1: Security requirements for cryptographic modules*, 1994. N.I.S.T., National Technical Information Service, Springfield, Virginia.

[Nat97] National Institute of Standards and Technology. *Advanced Encryption Standard (AES) Development Effort*. http://csrc.nist.gov/encryption/aes/, National Institute for Standards and Technology, Gaithersburg, MD, USA, 1997.

[NR97] Moni Naor and Omer Reingold. On the construction of pseudo-random permutations: Luby-Rackoff revisited (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 189–199, El Paso, Texas, 4–6 May 1997.

[Nyb96] Kaisa Nyberg. Generalized feistel networks. In *Advances in Cryptology – ASIACRYPT '96*, pages 91–104, 1996.

[PFTV88] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipies in C: The art of Scientific Programming.* Cambridge University Press, http://www.nr.com/, Cambridge, England, 1988.

[RRSY98] Ronald L. Rivest, Matt J. B. Robshaw, R. Sidney, and Yiqun L. Yin. The RC6 block cipher. NIST AES Proposal, June 1998.

[SB00] Juan Soto and Lawrence Bassham. Randomness testing of the advanced encryption standard finalist candidates. *Third AES Candidate Conference*, 2000.

[Sch94] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). *Lecture Notes in Computer Science*, 809:191–204, 1994.

[Sch96] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley and Sons, Inc., New York, NY, USA, second edition, 1996.

[SK96] Bruce Schneier and John Kelsey. Unbalanced Feistel networks and block cipher design. In Dieter Gollman, editor, *Fast Software Encryption, 3rd International Workshop Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 121–144, Berlin, Germany, 1996. Springer-Verlag.

[Sti95] Douglas R. Stinson. *Cryptography Theory and Practice.* CRC Press, Boca Raton, 1995.

[Uni77] United States. National Bureau of Standards. *Data Encryption Standard.* U.S. National Bureau of Standards, Gaithersburg, MD, USA, January 1977.

[Wie64] Helmut Wielandt. *Finite permutation groups.* Academic Press, New York, 1964.

[ZMI90] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imal. On the construction of block ciphers provably secure and not relying on any unproved hypotheses. In *Advances in Cryptology: CRYPTO '89*, pages 461–480, Berlin, Germany, 1990. Springer-Verlag.