PROPAGATION CHARACTERISTICS OF RC5, RC6 AND TWOFISH CIPHERS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

SAVAŞ ARIKAN

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2003

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan ÖZGEN

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Mübeccel DEMİREKLER

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Melek D. YÜCEL

Supervisor

Examining Committee Members

Prof. Dr. Yalçın TANIK (Chairman) _____

Prof. Dr. Murat AŞKAR _____

Prof. Dr. Kemal LEBLEBİCİOĞLU _____

Assoc. Prof. Dr. Melek D. YÜCEL _____

MSc. Özgür İNCE _____

ABSTRACT

PROPAGATION CHARACTERISTICS OF RC5, RC6 AND TWOFISH CIPHERS

Arıkan, Savaş

M.S., Department of Electrical and Electronics Engineering

Supervisor: Assoc. Prof. Dr. Melek D. Yücel

December 2003, 117 pages

In this thesis, two finalists of the AES (Advanced Encryption Standard) contest, RC6 developed by Rivest et al, Twofish proposed by Schneier et al, and preceding algorithm of RC6 cipher, RC5, are studied. The strength of ciphers to cryptanalytic attacks is measured according to different criteria. The studied evaluation criteria are the avalanche criterion and its derivations. After the

implementation of the algorithms and the test procedures, they are compared with each other.

Different test criteria, including avalanche criterion, avalanche weight distribution (AWD) for randomness of RC5, RC6 and Twofish algorithms are applied; and the *S*-boxes of the Twofish algorithm are analyzed according to nonlinearity criterion. The avalanche criteria results of RC6 and Twofish are compared with NIST (National Institute of Standards and Technology) Statistical Test Suite results.

Keywords: Block Ciphers, RC5, RC6, Twofish, Avalanche Criteria, Nonlinearity Measure.

ÖZ

RC5, RC6 VE TWOFISH ŞİFRELERİNİN YAYILIM ÖZELLİKLERİ

Arıkan, Savaş

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Melek D. Yücel

Aralık 2003, 117 sayfa

Bu tezde, AES (Gelişmiş Şifreleme Standardı) yarışmasının finalistlerinden ikisi, RC6 Rivest ve arkadaşlarının geliştirdiği şifre, Twofish Schneier ve arkadaşlarının önerdiği şifre, ve ayrıca RC6 şifresinin öncül algoritması olan RC5 şifresi çalışıldı. Bu şifrelerin kriptografik ataklara karşı dayanıklılığı farklı ölçütler ile ölçüldü. Çalışılan ölçütler "Çığ Kriteri" ve türevleridir. Algoritmalar ve test prosedürleri gerçeklendikten sonra, aralarında karşılaştırıldılar.

RC5, RC6, ve Twofish algoritmalarının rastlantısallıkları için farklı test kriterleri; çığ kriteri, "Çığ Ağırlık Dağılımı" (ÇAD) uygulandı ve Twofish algoritmasının yerleşim kutuları doğrusal olmama ölçütlerine göre analiz edildi. RC6 ve Twofish şifrelerinin çığ kriteri sonuçları ile NIST (Ulusal Standartlar ve Teknoloji Enstitüsü) İstatiksel Test Süit sonuçları ile karşılaştırıldı.

Anahtar Kelimeler: Blok Şifreler, RC5, RC6, Twofish, Çığ Kriteri, Doğrusal Olmama Ölçütü.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor Assoc. Prof. Dr. Melek D. YÜCEL for her patient supervision, guidance and helpful suggestions.

I wish to thank ASELSAN Inc. for facilities provided for the completion of this thesis.

I would like to thank my family and my friends who have put up with me during these years for their continuous support and understanding.

Finally, I offer my special thanks to my fiancée, Menekşe, for her unshakable faith in me and her endurance with me of this long study.

TABLE OF CONTENTS

ABSTRACT ..... **iii**

ÖZ ..... **v**

ACKNOWLEDGEMENTS ..... **vii**

TABLE OF CONTENTS ..... **viii**

LIST OF TABLES ..... **xi**

LIST OF FIGURES ..... **xii**

CHAPTER:

CHAPTER 1 ..... **1**

1.1 BASICS OF CRYPTOGRAPHY 1

1.2 BLOCK CIPHERS 2

1.3 AIM AND OUTLINE OF THESIS 7

CHAPTER 2 ..... **9**

2.1 AES FINALIST ALGORITHMS 11

2.1.1 Overview of the Finalists *12*

2.1.2 Evaluation Criteria and Results *15*

2.2 RC5 CIPHER 17

2.2.1 Key Expansion of RC5 ..... *19*

2.2.2 Encryption of RC5 ..... *22*

2.2.3 Decryption of RC5 ..... *23*

2.2.4 Overview of Cryptanalytic Results for RC5 ..... *23*

2.3 RC6 ALGORITHM 25

2.3.1 Key Schedule *27*

2.3.2 Encryption of RC6 ..... *29*

2.3.3 Decryption of RC6 ..... *32*

2.3.4 Status of RC6 ..... *33*

2.4 TWOFISH ALGORITHM 33

2.4.1 Main Functions of Twofish Algorithm *40*

2.4.2 Sub-functions of Twofish Algorithm *46*

2.4.3 Cryptanalysis of Twofish *50*

CHAPTER 3 52

3.1 CRYPTANALYSIS TECHNIQUES IN BRIEF 52

3.2 STRENGTH AND CRYPTANALYSIS 54

3.3 LINEAR CRYPTANALYSIS 55

3.4 DIFFERENTIAL CRYPTANALYSIS 56

CHAPTER 4 59

4.1 AVALANCHE CRITERIA 59

4.1.1 Avalanche Weight Distribution *60*

4.1.2 Avalanche Criteria Analysis Procedures *63*

4.2 NONLINEARITY MEASURE 66

4.2.1 Basic Definitions of Nonlinearity Criteria *66*

4.2.2 Nonlinearity of S-boxes *71*

4.2.3 Nonlinearity Criterion *72*

CHAPTER 5 ..... **74**

5.1 AVALANCHE CHARACTERISTICS OF RC5 CIPHER 74

5.1.1 Avalanche Curves of RC5 Cipher *74*

5.1.2 Avalanche Wight Distribution (AWD) Curves of RC5 Cipher *77*

5.1.3 Resemblance Parameters for RC5 Cipher *80*

5.2 AVALANCHE CHARACTERISTICS OF RC6 CIPHER 82

5.2.1 Avalanche Criterion for RC6 Cipher *82*

5.2.2 Avalanche Wight Distribution (AWD) Curves of RC6 Cipher *84*

5.2.3 Resemblance Parameter for RC6 Cipher *88*

5.3 AVALANCHE CRITERIA AND DERIVATIONS APPLIED TO TWOFISH CIPHER 90

5.3.1 Avalanche Criterion for Twofish Cipher *90*

5.3.2 AWD Test for Twofish Cipher *93*

5.3.3 Resemblance Parameter Analysis Applied to Twofish Cipher *96*

5.4 NONLINEARITY MEASURE OF TWOFISH CIPHER 99

5.5 COMPARISON OF AVALANCHE CRITERIA WITH NIST STATISTICAL TEST SUITE 103

5.5.1 Description of the Statistical Tests *104*

5.5.2 Statistical Test Results and Comparison with Avalanche Criteria *106*

CHAPTER 6 ..... **110**

REFERENCES **113**

LIST OF TABLES

TABLE:

2.1. AES Round 1 Candidate Algorithms ...…….….. 9

2.2. Evaluation Results of AES Finalist Algorithms ……..…15

5.1. Behavior of RC5 for different error vector bits (*i*) …… 98

5.2. Behavior of RC6 for different error vector bits (*i*) …… 99

5.3. Behavior of Twofish for different error vector bits (*i*) …….. 99

5.4 Breakdown of the 189 statistical tests applied during randomness test applied by J. Soto [36]…….…..… 106

LIST OF FIGURES

FIGURES:

2.1. One Round of RC6 Encryption Algorithm ……..……. 30

2.2. Feistel Network…….. 35

2.3. Twofish Encryption Algorithm Block ……..……. 38

2.4. A view of a single round *F* function (128-bit key)……..……43

2.5. 49 *S*-box formulation of Twofish algorithm.……... 46

4.1. Binomial Distribution Curve for *n* = 128 bits …….…….. 61

5.1. Avalanche curves of RC5 for the first and second round (*r*=1, *r*=2) and chosen error bit positions (*i*), which represents different cases ……..... 76

5.2. Avalanche curves of RC5 for the third round (*r*=3) and chosen error bit positions (*i*), which represents different cases ……..……. 77

5.3. Avalanche weight distribution curves of RC5 for the first and second round (*r*=1, *r*=2) and chosen error bit positions (*i*), which represents different cases ……. 79

5.4. Avalanche weight distribution curves of RC5 for the third round (*r*=3) and chosen error bit positions (*i*), which represents different cases …….……..…. 80

5.5. Resemblance parameter curves of RC5 for different rounds (*r*): a) *r*=1 b) *r*=2 c) *r*=3 d) *r*=4 …….…….…….. 81

5.6. Avalanche curves of RC6 for the first and second round (*r*=1, *r*=2) and chosen error bit positions (*i*), which represents different cases ……..... 83

5.7. Avalanche curves of RC6 for the third round (r=3) and chosen error bit positions (i), which represents different cases ……..……..... 84

5.8. Avalanche weight distribution curves of RC6 for the first and second round (r=1, r=2) and chosen error bit positions (i), which represents different cases ……..86

5.9. Avalanche weight distribution curves of RC6 for the third round (r=3) and chosen error bit positions (i), which represents different cases …….……... 87

5.10. Resemblance parameter curves of RC6 for different rounds (r): a) r=1 b) r=2 c) r=3 d) r=4 ……..……..…….. 89

5.11. Avalanche curves of Twofish for the second (r=2) and chosen error bit positions (i), which represents different cases ……...……...……..... 92

5.12. Avalanche curves of Twofish for the third round (r=3) and chosen error bit positions (i), which represents different cases ……..……... 93

5.13. Avalanche weight distribution curves of Twofish for the second round (r=2) and chosen error bit positions (i), which represents different cases…….……....95

5.14. Avalanche weight distribution curves of Twofish for the third round (r=3) and chosen error bit positions (i), which represents different cases…….……....96

5.15. Resemblance parameter curves of Twofish for different rounds (r): a) r=1 b) r=2 c) r=3 ……..……. 97

5.16. Nonlinearity of S-boxes of Twofish a) S-box0 b) S-box1 c) S-box2 d) S-box3 ……....…….....……. 101

5.17. Nonlinearity values of S-boxes of Twofish (Average: 81,215) ……..... 102

5.18. Nonlinearity values of S-boxes of Twofish (Average: 81,02625) …….……..... 102

5.19. Nonlinearity values of S-boxes of Twofish (Average: 81, 0625) ……...…103

5.20. Results of 189 statistical tests applied to Twofish…….……..... 108

5.21. Results of 189 statistical tests applied to RC6…….…….. 109

CHAPTER 1

INTRODUCTION

1.1 Basics of Cryptography

Cryptography is Greek word for "hidden writing". It is the art and science of transforming information into an intermediate form which secures that information while in storage or in transit.

Cryptography includes; *secrecy* (confidentiality, or privacy, or information security), *message authentication* (integrity), *no repudiation* (the inability to deny sending a message), *access control* (user or source authentication), *availability* (keeping security services available).

Modern cryptography generally depends upon translating a message into one of an astronomical number of different intermediate representations, or ciphertexts, as selected by a key. If all possible intermediate representations have similar appearance, it may be necessary to try all possible keys to find the one which

deciphers the message. By creating mechanisms, cipher algorithms, with an astronomical number of keys, this approach can be made impractical.

A cipher algorithm includes an encryption scheme, which has five ingredients: plaintext and ciphertexts, encryption and decryption algorithms and secret keys. The data that is encrypted is called the *plaintext*, or sometimes *cleartext*, and it is encrypted to give the *ciphertext*. The *key* is some secret information chosen by those wishing to communicate. For symmetric ciphers the key is same for both sender and receiver. Anyone possessing the key can decrypt the encrypted messages and the fact that both participants have to agree on a secret key before secure transmission can take place introduces problems. These problems are addressed by the fields of key management and key distribution.

An encryption scheme is said to be computationally secure if the cost of breaking the cipher exceeds the value of the encrypted information and the time required to break the cipher exceeds the useful life-time of the information. Also it should be noted that security depends on the secrecy of the key, and not on the secrecy of the algorithm.

1.2 Block Ciphers

Symmetric key algorithms use a single key for encryption and decryption, which should be shared by two parties who want to communicate secretly. Symmetric ciphers are divided into two main classes: block ciphers and stream ciphers. Block ciphers process fixed segments of the input (called the plaintext), and generate output

(called the ciphertext) segments of the same size. The segment size is called the block length of the cipher. Stream ciphers do not divide the plaintext into segments, they rather process each input bit continuously. Although several modes of use of a block cipher allow it to be used as a stream cipher the concise distinction may be indicated as follows [33]:

Block ciphers operate with a fixed transformation on large blocks of plaintext data; *stream ciphers* operate with a time-varying transformation on individual plaintext digits.

Two important attributes of a block cipher are the size of the key and the size of the block on which cipher operates, which are chosen at least as 128 bits in recent algorithms. It should be noted that some modes of use of a block cipher require the use of an *initialization value, IV*. The value of IV is often publicly known (since the security of the cryptosystem does not depend on this value being kept secret) and it is not considered to be part of the key.

A block cipher which operates on plaintext blocks of size n will be called n-bit block cipher, and the encryption of plaintext m using the chosen cipher under key k will be written as $E_k(m)$. Similarly, decryption of the ciphertext c will be denoted by $D_k(c)$. The decryption function D_k should be chosen as the inverse of the encryption function E_k ; hence, $D_k(E_k(m))=m$.

In an iterated block cipher, a complex (but perhaps weak) round function is used repeatedly, each time taking as input the output from the previous round. The most familiar example of such a cipher is Data Encryption Standard [23] in 1977, and

the iterated structure in DES has its origins in the Feistel Cipher [8]. Lucifer is designed by Feistel in early 1970s but firstly implemented and documented by Sorkin [35]. Lucifer is often mentioned as the starting point for the development of DES.

Some noteworthy block ciphers are given below in brief:

FEAL: The Fast Data Encryption Algorithm (FEAL) was proposed by Shimizu and Miyaguchi at Eurocrypt '87 [32]. It was intended to be very efficient when implemented in software, and was claimed to offer at least as much security as DES. Unfortunately, the security was soon found to be lacking.

LOKI: LOKI [5] was initially proposed in 1989 by Lawrie Brown, Josef Pieprzyk, Jennifer Seberry and is a DES-like iterative cipher that operates on 64-bit blocks and uses a 64-bit key. Its security is based on the use of a large *S*-box, taking 12 bits and outputting eight, which in turn is based in the use of irreducible polynomials. Also, developed version, LOKI97 was submitted in AES contest.

CAST: Designed by Adams and Tavares in 1990, CAST [1] is a 64-bit Feistel cipher. Instead of employing eight fixed *S*-boxes which map six bits to four, as it is found in DES, CAST uses four *S*-boxes map eight bits to 32, and the output of all four *S*-boxes is XORed together to produce the output from the round function.

IDEA: The International Data Encryption Algorithm (IDEA) first appeared as the Proposed Encryption Standard and was designed by Lai and Massey in 1992 [18]. It is an iterative cipher that operates on 64-bit blocks and uses a 128-bit key. The aim was to design a block cipher that could be efficiently implemented in both hardware and software, unlike DES which is primarily suitable for hardware encryption. The

operations used in IDEA are bitwise XOR, addition modulo 2^{16} and multiplication modulo $2^{16}+1$, with the value 0 corresponding to 2^{16}.

RC2: RC2 [26] was designed by Rivest for RSA Data Security, Inc in March of 1992. It is a confidential and proprietary cipher and so there are few details that can be readily disclosed. Like DES it is a 64-bit block cipher but it has a variable key size. One advantage is that the process of granting export approval for RC2 is greatly simplified if the key length is restricted to 40 bits, or 56 bits.

SAFER K-64: SAFER K-64 (Secure and Fast Encryption Routine with a Key of length 64 bits) was first proposed at Cambridge Algorithms Workshop in December of 1993 [19], by Massey. It is a byte-oriented iterated block cipher designed for efficient implementation in both software and hardware. It was initially proposed that six rounds can be used for greater security. Each round consists of a set of non-linear operations, including two different *S*-box permutations, which operate in parallel on each digit of the eight bytes in a block. Two different subkeys of 64 bits are used in each round. They are derived using the key schedule and introduced during this non-linear stage. The second part of each round is a series of linear mixing operations which is termed as *Pseudo-Hadamard Transform*. At the end of the last round, the final iteration of the linear transformation is followed by one further partial round of non-linear transformation using key material.

Skipjack: The first mention of Skipjack [34] came in April of 1994 when the White House announced a cryptographic initiative. Despite the fact that Skipjack is a classified algorithm and full details of the algorithm remain secret, the few details that

have been emerged suggest that Skipjack is an iterative block cipher, using 32 rounds and a key of length 80 bits.

RC5: RC5 is a block cipher designed by Rivest for RSA Data Security, Inc. Presented at the Leuven Algorithms Workshop in December of 1994 [27]. The cipher is fully parameterized in that the block size, the key size and the number of rounds can all vary. A likely version of RC5 is perhaps RC5-32/16/10 where the block size is 64 bits, there are 16 rounds and the key length is 10 bytes. The algorithm begins by expanding a variable-length key into a set of look-up tables. Then two very simple operations are used repeatedly to mix in the key and transform data which is called as data-dependent rotation.

In 1997, National Institute of Standards and Technology (NIST) announced a contest to an Advanced Encryption Standard (AES) to replace Data Encryption Standard (DES). 15 algorithms were submitted and within these 15 algorithms, five finalist algorithms were selected as AES candidates. And finally, Rijndael algorithm was selected as the new encryption standard October 2, 2000. RC6 and Twofish ciphers were among the finalist algorithms and these algorithms are analyzed according to different criteria in this thesis. Besides, RC5 cipher is implemented because it is the former algorithm of the AES contest finalist, RC6 cipher. The algorithms of studied ciphers and cryptanalysis of them are given in the following chapters.

1.3 Aim and Outline of Thesis

In this thesis the main point is to investigate the avalanche characteristics of Twofish and RC6 ciphers, which are AES contest finalists. To do so, the ciphers are implemented and then analyzed according to avalanche criterion and avalanche weight distribution criterion. The results of the analysis are compared with NIST's results. Besides, *S*-boxes of the Twofish cipher are tested according to nonlinearity criterion.

This thesis is organized as follows. Chapter 2 gives the encryption and decryption algorithms of the studied block ciphers RC5, RC6, and Twofish. In Chapter 3, the most known cryptanalysis techniques, linear cryptanalysis [17, 11] and differential cryptanalysis [4, 11], that have been applied to block ciphers are briefly reviewed. In Chapter 4, description and methodology of some test criteria that are used to measure the strength of the ciphers against cryptanalytic attacks are given. The studied test criteria are avalanche [12], nonlinearity measure [30] and their derivations.

In Chapter 5 we give the results of avalanche criterion and its derivative, avalanche weight distribution (AWD) analysis. The resemblance of AWD curves to ideal binomial distribution is measured by the resemblance parameter, for RC5, RC6 and Twofish ciphers. The comparison of three ciphers is made according to avalanche criteria. Also in Chapter 5, the nonlinearity of the *S*-boxes of the Twofish cipher and the effects of keywords on the nonlinearity measure are investigated. Avalanche criteria results of RC6 and Twofish algorithms are compared with NIST Statistical

Test Suite in this chapter. Finally, Chapter 6 summarizes the work of the thesis, giving directions for future research.

CHAPTER 2

MODERN BLOCK CIPHERS

On January 2, 1997, the US National Institute for Security Technologies (NIST) announced a contest to an Advanced Encryption Standard (AES) to replace the previous Data Encryption Standard (DES). NIST called for public submissions for new block ciphers as candidates of the AES algorithm. NIST intended that AES would be an unclassified, publicly disclosed encryption algorithm, available royalty-free, worldwide. At a minimum, the algorithm would have to implement symmetric key cryptography as a block cipher and support a block size of 128 bits and key sizes of 128, 192, and 256 bits. 15 algorithms were submitted for consideration by August 1998. After expert analysis of the candidates, five finalist algorithms were selected in 1999. The selected algorithms were MARS, RC6, Rijndael, Serpent and Twofish. In Table 2.1 the semi-finalist 15 algorithms, their submitters, type and cryptanalysis results of some eliminated ciphers are given.

Table 2.1: AES Round 1 Candidate Algorithms

Country of Origin	Candidate Algorithm	Submitter(s)	Type	Cyrptanalysis
Australia	LOKI97	Lawrie Brown, Josef Pieprzyk, Jennifer Seberry	Feistel Network	– Rijmen and Knudsen – Differential: 2^{56} chosen plaintexts – Linear: 2^{56} known plaintexts
Belgium	RIJNDAEL	Joan Daemen, Vincent Rijmen	Substitution-Permutation Network	
Canada	CAST-256	Entrust Technologies, Inc.	Modified Feistel Network	
Canada	DEAL	Outerbridge, Knudsen	Feistel Network	– 2^{70} chosen ciphertexts, 2^{121} steps, (Lucks, 128) – 2^{70}, chosen plaintexts, 2^{121} steps, (Knudsen, 192) – 2^{56} chosen ciphertexts, 2^{145} steps, (Lucks, 192) – Meet in middle, 2^{224} steps, (Knudsen, 256)
Costa Rica	FROG	TecApro Internacional S.A.		– Wagner, Ferguson, and Schneier – Differential: 2^{58} chosen plaintext – Linear: 2^{56} known plaintexts
France	DFC	Centre National pour la Recherche Scientifique (CNRS)	Feistel Network	– Weak keys, reduce to 6 round cipher, prob. 2^{-64}, (Coppersmith) – Weak keys, pt=ct, prob. 2^{-128}, (Coppersmith)
Germany	MAGENTA	Deutsche Telekom AG	Feistel Network	– Biham, Biryukov, Ferguson, Knudsen, Schneier, Shamir – 2^{64} chosen plaintexts, 2^{64} steps – 2^{33} known plaintexts, 2^{97} steps
Japan	E2	Nippon Telegraph and Telephone Corporation	Feistel Network	

Table 2.1 (cont'd) AES Round 1 Candidate Algorithms

Country of Origin	Candidate Algorithm	Submitter(s)	Type	Cyrptanalysis
USA	HPC	Rich Schroeppel		
USA	MARS	IBM	Modified Feistel Network	
USA	RC6	RSA Laboratories	Modified Feistel Network	
USA	SAFER+	Cylink Corporation	Substitution-Permutation Network	– 2 known plaintexts, 2^{37} memory, 2^{241} steps, (2^{56}, Kelsey) – 2^{56} chosen plaintext encrypted with 2 keys, 2^{216} steps, (256, Kelsey)
USA	TWOFISH	Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson	Feistel Network	
UK, Israel, Norway	SERPENT	Ross Anderson, Eli Biham, Lars Knudsen	Substitution-Permutation Network	

2.1 AES Finalist Algorithms

As mentioned, MARS, RC6, Rijndael, Serpent and Twofish algorithms were selected as finalists of the contest to AES. In the following section the algorithms are described briefly and the evaluation criteria and results are given.

2.1.1 Overview of the Finalists

The five finalists shared a number of features. All are iterated block ciphers: they specify a transformation that is repeated ("iterated") a number of times on the data block to be encrypted or decrypted. Each iteration is called a round, and the transformation is called the round function. Each finalist also specifies a method for generating a series of working keys, also known as subkeys, from the original user key. The round functions take distinct subkeys as input along with the data block.

For each finalist, the very first and last cryptographic operations mix subkeys with the data block to prevent an adversary who does not know the keys from even beginning to encrypt the plaintext or decrypt the ciphertext. Whenever this subkey mixing does not naturally occur as the initial step of the first round or the final step of the last round, the finalists specify the subkey mixing as an extra step called pre- or post-whitening.

Four of the finalists (Rijndael, Serpent, MARS, Twofish) specify substitution tables, called *S*-boxes: and three of the finalists (MARS, RC6, Twofish) specify variations on the Feistel structure. In the classic Feistel structure, half of the data block is used to modify the other half of the data block, and then the halves are swapped.

The two finalists that do not use a Feistel structure (Rijndael, Serpent) process the entire data block in parallel during each round using substitutions and linear

transformations; thus, these two finalists are examples of substitution-linear transformation networks.

MARS [6] has several layers: key addition as pre-whitening, 8 rounds of un-keyed forward mixing, eight rounds of keyed forward transformation, 8 rounds of keyed backward transformation, eight rounds of un-keyed backward mixing, and key subtraction as post whitening. The 16 keyed transformations are called the cryptographic core. The un-keyed rounds use two 8x32 bit *S*-boxes, addition, and the XOR (exclusive-or) operation. In addition to those elements, the keyed rounds use 32-bit key multiplication, data-dependent rotations, and key addition. Both the mixing and the core rounds are modified Feistel rounds in which one quarter of the data block is used to alter the other three quarters. MARS was submitted by IBM.

RC6 [28] is a parameterized family of encryption ciphers that essentially use the Feistel structure; 20 rounds were specified for the AES submission. The round function of RC6 uses variable rotations that are regulated by a quadratic function of the data. Each round also includes 32-bit modular multiplication, addition, XOR, and key addition. Key addition is also used for pre- and post-whitening. RC6 was submitted to the AES development effort by RSA Laboratories.

Rijndael [7] is a substitution-linear transformation network with 10, 12 or 14 rounds, depending on the key size. A data block to be encrypted by Rijndael is split into an array of bytes, and each encryption operation is byte-oriented. Rijndael's round function consists of four layers. In the first layer, an 8x8 *S*-box is applied to each byte. The second and third layers are linear mixing layers, in which the rows of

the array are shifted, and the columns are mixed. In the fourth layer, subkey bytes are XORed into each byte of the array. In the last round, the column mixing is omitted. Rijndael was submitted by Joan Daemen (Proton World International) and Vincent Rijmen (Katholieke Universiteit Leuven).

Serpent [2] is a substitution-linear transformation network consisting of 32 rounds. Serpent also specifies non-cryptographic initial and final permutations that facilitate an alternative mode of implementation called the bit slice mode. The round function consists of three layers: the key XOR operation, 32 parallel applications of one of the eight specified 4x4 *S*-boxes, and a linear transformation. In the last round, a second layer of key XOR replaces the linear transformation. Serpent was submitted by Ross Anderson (University of Cambridge), Eli Biham (Technion), and Lars Knudsen (University of California San Diego).

Twofish [29] is a Feistel network with 16 rounds. The Feistel structure is slightly modified using 1-bit rotations. The round function acts on 32-bit words with four key-dependent 8x8 *S*-boxes, followed by a fixed 4x4 maximum distance separable matrix over GF(2^8), a pseudo-Hadamard transform, and key addition. Twofish was submitted by Bruce Schneier, John Kelsey, and Niels Ferguson (Counterpane Internet Security, Inc.), Doug Whiting (Hi/fn, Inc.), David Wagner (University of California Berkeley), and Chris Hall (Princeton University).

After the finalists were announced, NIST sought further public review and comment on the algorithms. The comment period ended on May 15, 2000, and NIST conducted its final review of comments and analyses.

2.1.2 Evaluation Criteria and Results

Evaluation criteria for the new AES algorithm were declared when NIST first called for submissions in September 1997. The evaluation criteria were divided into three major categories: security, cost, and algorithm and implementation characteristics.

Security was the most important factor in the evaluation and encompassed features such as resistance of the algorithm to cryptanalysis, soundness of its mathematical basis, randomness of the algorithm output, and relative security compared with other candidates.

Cost was a second important area of evaluation that encompassed licensing requirements, computational speed and efficiency on various platforms, and memory requirements. One of NIST's goals was for the AES algorithm to be available worldwide on a royalty-free basis, so public comments were specifically sought on intellectual property claims and any potential conflicts. The speed of the algorithm on a variety of platforms needed to be considered, and assessments were made of speed based on 128, 192 and 256 bit keys. Memory requirements and software implementations were also important.

The third area of evaluation was algorithm and implementation characteristics such as flexibility, suitability to hardware and software, and the simplicity (or complexity) of the algorithm. Flexibility includes the ability of an algorithm to handle key and block sizes beyond the minimum that must be supported, to be implemented

securely and efficiently in many different types of environments, to be implemented as a stream cipher or hashing algorithm, and to provide additional cryptographic services. It must be feasible to implement an algorithm in both hardware and software, and efficient firmware implementations were an evaluation advantage.

Because analysis and discussion often involved issues in more than one of the three main criteria, NIST gave most importance to security, and cost and algorithm characteristics were considered together as secondary criteria.

In October 2000, NIST released its report on the development of an Advanced Encryption Standard which compared the five Round 2 algorithms in a number of categories. The table below summarizes the relative scores of the five candidates (1=low, 3=high):

Table 2.2 Evaluation Results of AES Finalist Algorithms

	MARS	RC6	Rijndael	Serpent	Twofish
General security	3	2	2	3	3
Implementation of security	1	1	3	3	2
Software performance	2	2	3	1	1
Smart card performance	1	1	3	3	2
Hardware performance	1	2	3	3	2
Design features	2	1	2	1	3

NIST recommended adoption of the Rijndael algorithm, and released a draft Federal Information Processing Standard (FIPS) AES Specification for public review and comment in February 2000. Final selection of Rijndael was announced in October 2, 2000. And finally in November 26, 2001 NIST published FIPS 197 as the announcement of AES. The main reason for this selection can be summarized as follows [9].

"When considered together, Rijndael's combination of security, performance, efficiency, implementability, and flexibility make it an appropriate selection for the AES."

In the following part of this chapter detailed descriptions of the algorithms RC5, RC6 and Twofish are given.

2.2 RC5 Cipher

RC5 [27] is designed by Ronald Rivest for RSA Data Security (now RSA Security) in December of 1994. It is a parameterized algorithm with a variable block size, a variable key size, and a variable number of rounds. Allowable choices for the block size are 32 bits (for experimentation and evaluation purposes only), 64 bits (for use a drop-in replacement for DES), and 128 bits. The number of rounds can range from 0 to 255, while the key can range from 0 bits to 2048 bits in size. Such built-in variability provides flexibility at all levels of security and efficiency.

The heavy use of data-dependent rotations and the mixture of different operations provide the security of RC5. Two of the most distinguished features of

RC5 are the heavy use of data-dependent rotations and the exceptionally simple encryption routine. The former feature has been shown to be useful in preventing certain advanced types of attack, while the latter feature makes RC5 both easy to implement, and very importantly, more amenable to analysis than many other block ciphers. In particular, the use of data-dependent rotations helps defeat differential and linear cryptanalysis.

There are three routines in RC5: key expansion, encryption, and decryption. In the key-expansion routine, the user-provided secret key is expanded to fill a key table whose size depends on the number of rounds. The key table is then used in both encryption and decryption.

RC5 has three important parameters: w (the word size), r (the number of rounds), b (number of bytes in secret key K). In the encryption algorithm of RC5 the $2w$ plaintext is divided to two w-bit register using standard little-endian convention: the first byte occupies the low-order bit positions of register A, and so on, so that the fourth byte occupies the high-order bit positions of register A, the fifth byte occupies the low-order bit positions in B, and the eighth (last) byte occupies the high-order bit positions in B. Then these registers are cyclic-shifted and XOR-ed by them then with other registers. The key-bits are mixed by cyclic-shifting and XOR processes by the expanded key array S, which is simply constituted by the magic numbers provided by Rivest and cyclic-shifting process of the keyword. Within these operations the registers are always updated according to input data so that the idea of data dependent cryptography is achieved.

The key expansion, encryption, and decryption routines of RC5 use the following three primitive operations (and their inverses).

- Addition of words modulo 2^w, denoted by "+".
- Bit-wise XOR of words, denoted by $\oplus$.
- Rotation: the rotation of x to the left by y bits is denoted by $x <<< y$

Note that only the $\log_2(w)$ low-order bits of y affect this rotation. The algorithm explained above can be simply modeled as below steps.

2.2.1 Key Expansion of RC5

The key-expansion routine expands the user's secret key K to fill the expanded key array S, so that S resembles an array of $t = 2$ x $(r+1)$ random binary words determined by K. The key expansion algorithm uses two "magic constants", and consists of three simple algorithmic parts.

a Definition of the Magic Constants:

The key-expansion algorithm uses two word-sized binary constants P_w and Q_w. They are defined for arbitrary w as follows:

P_w = Odd $((e-2)2^w)$

Q_w = Odd $((\emptyset-2)2^w)$

where

e = 2,718281828459… (base of natural algorithms)

$\emptyset$ = 1,618033988749… (golden ratio),

And where Odd(x) is the odd integer nearest to x (rounded up if x is an even integer, although this won't happen here). For w = 16, 32 these constants are given below in binary and in hexadecimal.

P_{16} = 1011011111100001 = b7e1

Q_{16} = 1001111000110111 = 9e37

P_{32} = 10110111111000010101000101100011 = b7e15163

Q_{32} = 10011110001101110111100110111001 = 9e3779b9

b Converting the Secret Key from Bytes to Words:

The first algorithmic step of key expansion is to copy the b-byte secret key $K[0..b–1]$ into an array $L[0..c–1]$ of $c = [b/u]$ words, where $u = w/8$ is the number of bytes/word. This operation is done in natural manner, using u consecutive key bytes of K to fill up each successive word in L, low order byte to high order byte. Any unfilled byte positions of L are zeroed. In the case that $b = c = 0$, c is set to 1 and $L[0]$ is set to zero. In the following code sample the procedure is given.

c = [max (b, 1)/u]

for $i = b - 1$ **downto** 0 **do**

L [i/u] = (L [i/u] <<< 8) + K[i];

c Initializing the Array *S*:

The second algorithmic step of key expansion is to initialize array S to a particular fixed (key-independent) pseudo-random bit pattern, using an arithmetic

progression modulo 2^w determined by the "magic constants" P_w and Q_w. Since Q_w is odd, the arithmetic progression has period 2^w. In the following code sample, array S is initialized by P_w and then all of the entries of the array are found by adding Q_w to the previous entry. Since $t = 2x(r+1)$ binary words are required for r rounds of the encryption algorithm S is initialized as follows.

$S[0] = P_w$;

for $i = 1$ **to** $t - 1$ **do**

$S[i] = S[i-1] + Q_w$;

d Mixing in the Secret Key

The third algorithmic step of the key expansion is to mix in the user's secret key in three passes over the arrays S (of length t words) and L (of length c words). More precisely, due to the potentially different sizes of S and L, the larger array will be processed three times, and the other may be handled more times than three. In the following code sample temporary registers *reg*1 contains previous value of $S[i]$ and *reg*2 contains previous value of $L[j]$ the values of *reg*1, *reg*2 and $S[i]$ are added by modulo 2^w and rotated to left three and then this value is assigned to *reg*1. Later $L[j]$ is updated by adding *reg*1, *reg*2 and $L[j]$ and rotated to left by the value of *reg*1 plus *reg*2. Then the value of $L[j]$ is assigned to *reg*2. And this operation is repeated three times maximum of t or c is reached. The initial values of *reg*1 and *reg*2 are $S[0]$ and $L[0]$ respectively.

$i = j = 0$;

do *3****max** (t, c) **times:**

$S[i] = (S[i] + reg1 + reg2) <<<3$;

$reg1 = S[i]$;

$L[j] = (L[j] + reg1 + reg2) <<< (reg1 + reg2)$;

$reg2 = L[j]$;

$i = (i + 1) \bmod t$;

$j = (j + 1) \ mod \ c;$

2.2.2 Encryption of RC5

Firstly the input block is divided into two w-bit registers A and B and $S[0]$ and S[1] are added respectively to A and B. The registers A and B are XORed and rotated to left by the value if B and summed with the even entries of $S[i]$ and this value is assigned to A. Later, the XORed value of B and A is rotated to left by A and then summed by odd entries of $S[i]$. And this value is assigned to B. This routine is repeated for r rounds. The following code sample gives the explained procedure.

$A = A + S[0]$;

$B = B + S[1]$;

for $i = 1$ **to** r **do**

$A = ((A \oplus B) <<<B) + S[2*i]$;

$B = ((B \oplus A) <<< A) + S[2*i + 1]$;

The output is in the registers A and B.

2.2.3 Decryption of RC5

The decryption routine is easily derived from the encryption routine. In this routine the inverse formulation of encryption routine is processed.

for $i = r$ **downto** 1 **do**

$B = ((B - S[2*i + 1]) >>> A) \oplus A;$

$A = ((A - S[2*i]) >>> B) \oplus B;$

$B = B - S[1];$

$A = A - S[0];$

2.2.4 Overview of Cryptanalytic Results for RC5

Several techniques [14] have been developed for analyzing the security of block ciphers, including exhaustive key search attack, statistical tests, differential cryptanalysis and linear cryptanalysis. The last two types of attack, both considered substantial advances in recent years, are more sophisticated techniques for block cipher analysis. For differential cryptanalysis which is explained in Chapter 3, the basic idea is to choose two plaintexts with a certain difference between them so that the resulting ciphertexts have a difference with a specific value with a probability that is better than we might expect. Such a pair of differences (which lead to the concept of a "*characteristic*") is useful in deriving certain bits of the key. For linear cryptanalysis which is also explained in Chapter 3, the basic idea is to find a linear relation among bits of plaintext, ciphertext, and key which hold with a probability

that is not equal to 1/2. Such a "*linear approximation*" can potentially be used to obtain information about the key.

The first cryptanalytic results on RC5 were given by Kaliski and Yin [13] at Crypto'95. By analyzing the basic structure of the encryption routine as well as the properties of data-dependent rotations, it is possible to construct differential characteristics and linear approximations of RC5 that are useful for mounting differential and linear attacks. Both attacks are quite effective on RC5 with a very small number of rounds, but the plaintext requirements increase quickly as the number of rounds grows. The use of data-dependent rotations and the incompatibility between the different arithmetic operations used in encryption help prevent both differential and linear cryptanalysis.

At Crypto'96, Knudsen and Meier [16] presented nice improvements over Kaliski and Yin's differential attack by a careful analysis of the relations between input, output, and the sub-keys used in the first two rounds of encryption. They were able to improve the plaintext requirements by a factor of up to 512 by exploiting the characteristics in an innovative and sophisticated way. They also considered the existence of certain weaker keys for RC5 with respect to which their attack can be further enhanced.

Moriai, Aoki, and Ohta [22] have investigated the strength of RC5 against linear cryptanalysis by focusing on the bias of linear approximations for *fixed* keys, rather than the average bias over *all* possible keys which is the customary model for linear cryptanalysis. They also considered a mini-version of RC5 with much reduced

word size and computed the percentage of keys that yield ciphers less resistant to linear cryptanalysis than the average case analysis.

In late 1995, Kocher [17] developed what are called *timing attacks* that are generally applicable to many cryptosystems. In such an attack, an opponent tries to obtain information about the secret key (or private key) by recording and analyzing the time used for cryptographic operations that involve the key. Kocher observed that RC5 may be subject to timing attacks if RC5 is implemented on platforms for which the time for computing a single rotation is proportional to the rotation amount.

With regards to the less sophisticated brute-force attack of trying each key in turn, the security of RC5 is obviously dependent on the length of the encryption key that is used (as is the case with all ciphers). RC5 has the attractive feature that the length of the key can be varied (unlike the situation with DES for instance) and so the level of security against these attacks can be tuned to suit the application. It is hoped that the resistance of ciphers to exhaustive key search attacks can be more accurately gauged in the future. Some of the posted challenges, such as RC5 encryption with a 40- and 48-bit key were solved very quickly, as was expected. But some of the longer key lengths are likely to remain an unsolved challenge for some considerable time to come.

2.3 RC6 Algorithm

RC6 is a block-cipher submitted to NIST for consideration as the new Advanced Encryption Standard (AES). The design of RC6 began with a consideration

of RC5, and modifications were then made to meet the AES requirements, to increase security, and to improve performance. The inner loop, however, is based around the same "half-round" found in RC5. The algorithm can be seen as two Feistel-networks which are combined through data-dependent rotations over the blocks together with a 32-bit multiplication function.

RC5 is improved to obtain RC6 for the following considerations:

- The requirements of AES are the 128 bit input/output blocks. To do so RC5 has two 64 bit blocks but this can not be implemented by the very well known compilers and RC6 solves this problem by having 4 blocks so that to have a 128 bit plaintext there are four 32-bit blocks in RC6.

- In the encryption section in RC6 there are two rotations per round where in RC5 there is one rotation per round. This improves the immunity to differential and linear cryptanalysis attacks.

- In RC6 integer multiplication is involved. This improves the diffusion property and rotation amounts are dependent on all bits of another register where RC5 has just low order bits' contribution.

Like RC5, RC6 is a fully parameterized family of encryption algorithms. A version of RC6 is more accurately specified as RC6-*w/r/b* where the word size is *w* bits, encryption consists of non-negative number of rounds *r*, and *b* denotes the length of encryption key in bytes. RC6-*w/r/b* operates on units of four 2-bit words using the following six basic operations.

- $a + b$ integer addition modulo 2^w

- $a - b$ integer subtraction modulo 2^w

- $a \oplus b$ bitwise XOR of w-bit words

- a x b integer multiplication modulo 2^w

- a <<< b rotate the w-bit word a to the left by the amount given by the least significant $\log_2 w$ bits of b.

- a >>> b rotate the w-bit word a to the right by the amount given by the least significant $\log_2 w$ bits of b.

2.3.1 Key Schedule

The key schedule of RC6-*w/r/b* is practically identical to the key schedule of RC5-*w/r/b*. Indeed, the only difference is that for RC6-*w/r/b*, more words are derived from the user-supplied key for use during encryption and decryption. Sufficient zero bytes are appended to give a key length equal to a non-zero integral number of words; these key bytes are then loaded in little-endian fashion into an array of *c w-bit* words $L[0],...,L[c-1]$. Thus the first byte of key is stored as the low-order byte of $L[0]$, etc., and $L[c-1]$ is padded with high-order zero bytes if necessary. (Note that if $b = 0$ then $c = 1$ and $L[0] = 0$.) The number of w-bit words that will be generated for the additive round keys is $2r + 4$ and these are stored in the array $S[0,...,2r + 3]$.

The constants P_{32} = B7E15163 and Q_{32} = 9E3779B9 (hexadecimal) are the same "magic constants" as used in the RC5 key schedule. The value of P_{32} is derived from the binary expansion of $e-2$, where e is the base of the natural logarithm

function. The value of Q_{32} is derived from the binary expansion of $\emptyset$ -1, where $\emptyset$ is the Golden Ratio. Similar definitions from RC5 for P_{64} etc. can be used for versions of RC6 with other word sizes. These values are somewhat arbitrary, and other values could be chosen to give "custom" or proprietary versions of RC6.

The user supplies a key of b bytes, where $0 < b < 255$. *From* this key, $2r + 4$ words *(w*-bits each) are derived and stored in the array $S[0,..., 2r + 3]$. This array is used in both encryption and decryption the key schedule for RC6-*w/r/b* is as follows: array S is initialized by P_w and then all of the entries of the array are found by adding Q_w to the previous entry. After finding array S, this array will be mixed with the key register $L[j]$; temporary registers *reg*1 contains previous value of $S[i]$ and *reg*2 contains previous value of $L[j]$ the values of *reg*1, *reg*2 and $S[i]$ are added by modulo 2^w and rotated left by three and then this value is assigned to *reg*1. Later $L[j]$ is updated by adding *reg*1, *reg*2 and $L[j]$ and rotated to left by the value of *reg*1 plus *reg*2. Then the value of $L[j]$ is assigned to *reg*2. And this operation is continued until three times maximum of t or c is reached. The procedure is given in the following code sample.

Input: User supplied b byte key preloaded into c-word array $L[0,\ldots,c\text{-}1]$ Number r of rounds

Output: w-bit round keys $S[0,\ldots,2r + 3]$

Procedure:

$S[0] = P_w$

for $i = 1$ **to** $2r + 3$ **do**

$S[i] = S[i - 1] + Q_w$

$reg1 = reg2 = i = j = 0$

$v = 3$ x max $\{c, 2r+4\}$

do 3***max** (t, c) **times:**

$reg1 = S[i] = (S[i] + reg1 + reg2) <<< 3$

$reg2 = L[j] = (L[j] + reg1 + reg2) <<< (reg1 + reg2)$

$i = (i + 1) \bmod (2r + 4)$

$j = (j + 1) \bmod c$

2.3.2 Encryption of RC6

RC6 works with four w-bit registers A,B,C,D which contain the initial input plaintext as well as the output ciphertext at the end of encryption. Register B is initialized with $S[0]$ and register D is initialized with $S[1]$. After initialization $t = B$ x $(2B + 1)$ and $u = D$ x $(2D + 1)$ values are found and rotated left by $\log_2 w$. The registers A and t are XORed and rotated to left by the value of u and summed with the even entries of $S[i]$ and this value is assigned to A. Later the XORed value of C and u is rotated to left by t and then summed by odd entries of $S[i]$. This value is assigned to C. The first byte of plaintext or ciphertext is placed in the least-significant byte of A; the last byte of plaintext or ciphertext is placed into the most-significant byte of D. $(A,B,C,D) = (B,C,D,A)$ is used to mean the parallel assignment of values on the right to registers on the left. And this routine is repeated for r rounds. Then, finally $S[2r+2]$

and $S[2r+3]$ are added to A and C respectively. The sample code of this procedure is as follows.

Input: Plaintext stored in four w-bit input registers A, B, C, D Number r of rounds w-bit round keys $S[0,\ldots,2r + 3]$

Output: Ciphertext stored in A, B, C, D

Procedure:

$B = B + S[0]$

$D = D + S[1]$

for $i = 1$ **to** r **do**

{

$t = (B \text{ x } (2B + 1)) <<< \log_2 w$

$u = (D \text{ x } (2D + 1)) <<< \log_2 w$

$A = ((A \oplus t) <<< u) + S[2i]$

$C = ((C \oplus u) <<< t) + S[2i + 1]$

$(A, B, C, D) = (B, C, D, A)$

}

$A = A + S[2r + 2]$

$C = C + S[2r + 3]$

The block diagram of encryption procedure for one round is given in Fig. 2.1.

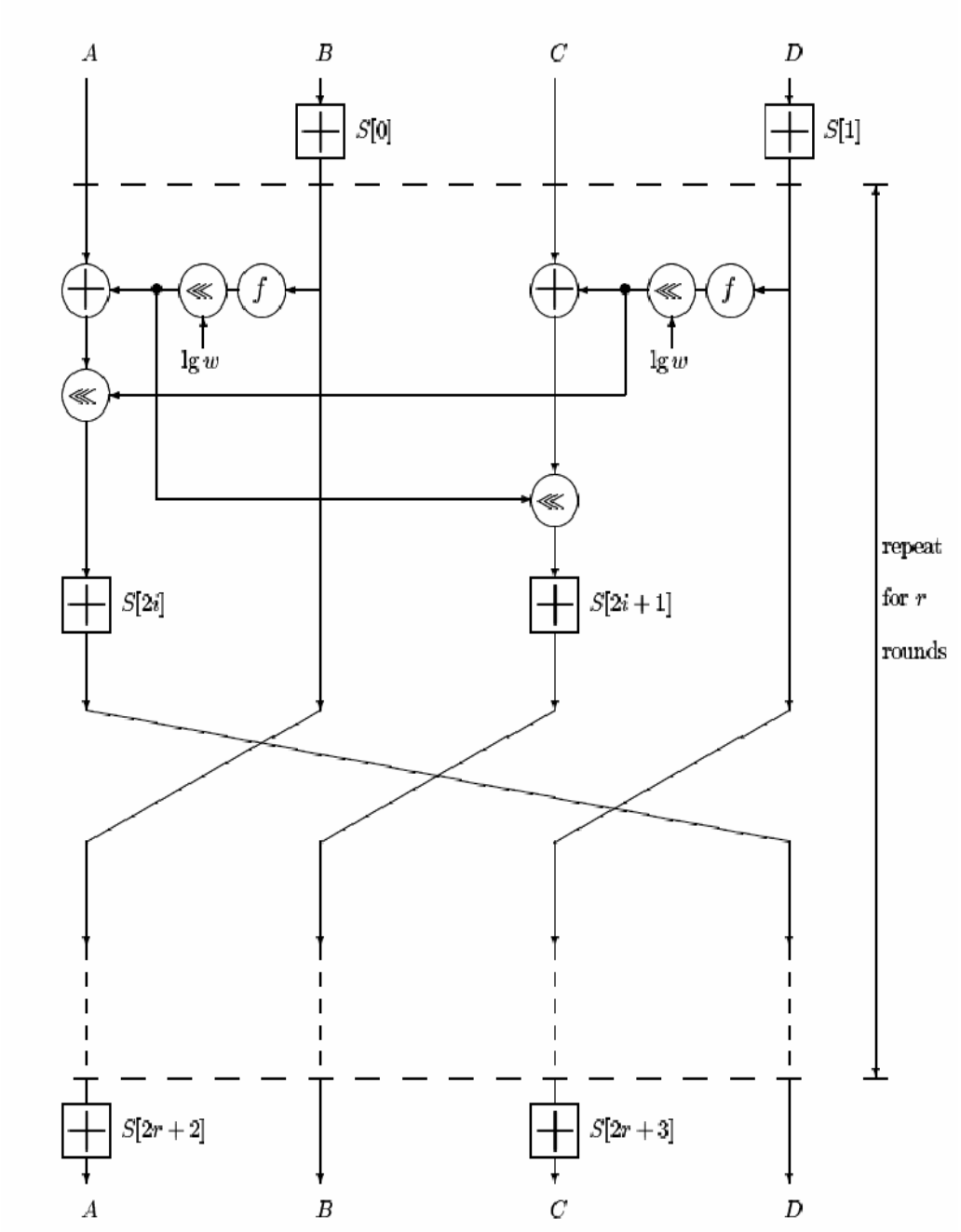

Figure 2.1 One Round of RC6 Encryption Algorithm

2.3.3 Decryption of RC6

The decryption routine is easily derived from the encryption routine. In this routine the inverse formulation of encryption routine is processed.

Input: Ciphertext stored in four w-bit input registers *A, B, C, D* Number *r* of rounds *w*-bit round keys $S[0,\ldots,2r + 3]$

Output: Plaintext stored in *A*, *B*, *C*, *D*

Procedure:

$C = C - S[2r + 3]$

$A = A - S[2r + 2]$

for $i = r$ **downto** 1 **do**

{

$(A, B, C, D) = (D, A, B, C)$

$u = (D \times (2D + 1)) <<< \log_2 w$

$t = (B \times (2B + 1)) <<< \log_2 w$

$C = ((C - S[2i + 1]) >>> t) \oplus u$

$A = ((A - S[2i]) >>> u) \oplus t$

}

$D = D - S[1]$

$B = B - S[0]$

2.3.4 Status of RC6

Most existing cryptanalytic results on RC5 [13,16,17,22] depend on slow avalanche of change between rounds. The integer addition helps to provide a reasonable amount of change due to the effect of carry, but the most dramatic changes take place when two different rotation amounts are used at a similar point during the encryption of two related plaintexts. The incremental changes in arriving at RC6 from RC5: Two significant changes are the introduction of the quadratic function $B \times (2B + 1)$ and the fixed rotation by five bits.

The quadratic function is aimed at providing a faster rate of diffusion thereby improving the chances that simple differentials will spoil rotation amounts much sooner than is accomplished with RC5. The quadratically transformed values of *B* and *D* are used in place of *B* and *D* to modify the registers *A* and *C,* increasing the nonlinearity of the scheme while not losing any entropy (since the transformation is a permutation). The fixed rotation by five bits plays a simple yet important role in complicating both Linear and Differential cryptanalysis.

2.4 Twofish Algorithm

Twofish is one of the submissions to the AES selection process. It meets all the required NIST criteria; 128- bit block; 128-, 192-, and 256-bit key lengths; efficient on various platforms, etc. Twofish can be seen as two parallel Feistel-networks, where the outputs of each round function are combined. In each round, half

the block is input to the confusion stage, and the *S*-boxes are 8-bit *S*-boxes. Twofish was designed to meet NIST's design criteria for AES.

Twofish algorithm has been implemented by using six blocks. Below these blocks and brief explanation are given:

Feistel Network: A *Feistel network* is a general method of transforming any function (usually called the *F* function) into a permutation. The fundamental building block of a Feistel network is the *F* function: a key-dependent mapping of an input string onto an output string. An *F* function is always non-linear and possibly non-surjective (in which not all outputs in the output space can occur):

$$F : f\{0,1\}^{n/2} \mathrm{x}\, f\{0,1\}^{N} \rightarrow \{0,1\}^{n/2}$$

where *n* is the block size of the Feistel Network, and *F* is a function taking *n*/2 bits of the block and *N* bits of a key as input, and producing an output of length *n*/2 bits. In each round, the "source block" is the input to *F*, and the output of *F* is XORed with the "target block," after which these two blocks swap places for the next round. The idea here is to take an *F* function, which may be a weak encryption algorithm when taken by itself, and repeatedly iterate it to create a strong encryption algorithm. Two rounds of a Feistel network is called a "cycle". In one cycle, every bit of the text block has been modified once. Twofish is a 16-round Feistel network with a bijective *F* function. Fig 2.2. shows block diagram of Feistel Network.

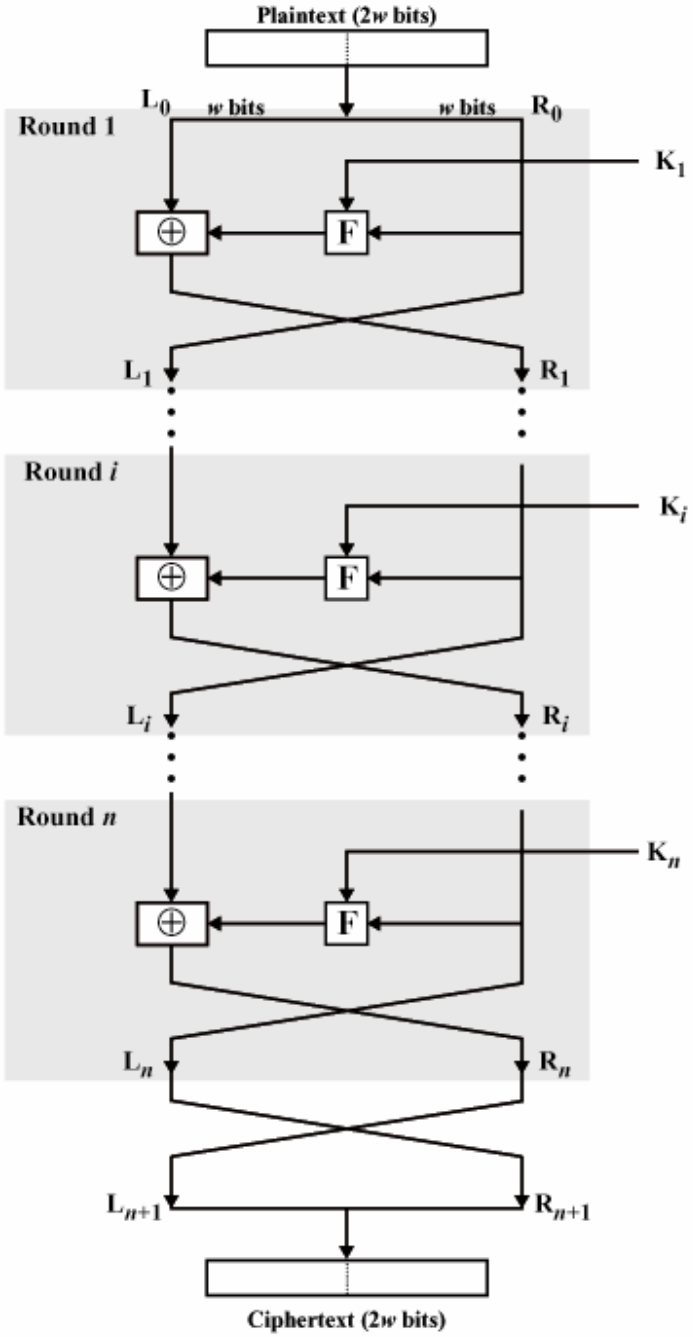

Figure 2.2 Feistel Network

S-boxes: An *S*-box is a non-linear substitution operation used in most block ciphers. *S*-boxes vary in both input size and output size, and can be created either randomly or algorithmically. Twofish uses four different, bijective, key-dependent, 8-by-8- bit *S*-boxes. These *S*-boxes are built using two fixed 8-by-8-bit permutations and key material.

MDS Matrices: A maximum distance separable (MDS) code over a field is a linear mapping from *a* field elements to *b* field elements, producing a composite vector of *a*+*b* elements, with the property that the minimum number of non-zero elements in any non-zero vector is at least *b*+1. MDS mappings can be represented by an MDS matrix consisting of *a* x *b* elements. Reed-Solomon (RS) error-correcting codes are known to be MDS. A necessary and sufficient condition for an *a* x *b* matrix to be MDS is that all possible square submatrices, obtained by discarding rows or columns, are non-singular. Twofish uses a single 4-by-4 MDS matrix over GF (2^8).

Pseudo-Hadamard Transforms: A Pseudo-Hadamard transform (PHT) is a simple mixing operation that runs quickly in software. Given two inputs, *a* and *b*, the 32-bit PHT is defined as:

$a' = a + b \bmod 2^{32}$

$b' = a + 2b \bmod 2^{32}$

Whitening: Whitening, the technique of XORing key material before the first round and after the last round, difficulty of key-search attacks against the remainder of the cipher. Whitening substantially increased the difficulty of attacking the cipher, by hiding from an attacker the specific inputs to the first and last rounds' *F* functions.

Twofish XORs 128 bits of subkey before the first Feistel round and another 128 bits after the last Feistel round. These subkeys are calculated in the same manner as the round subkeys, but are not used anywhere else in the cipher.

Key Schedule: The key schedule is the means by which the key bits are turned into round keys that the cipher can use. Twofish needs a lot of key material, and has a complicated key schedule. To facilitate analysis, the key schedule uses the same primitives as the round function.

Fig. 2.3 shows an overview of the Twofish block cipher. Twofish uses a 16-round Feistel-like structure with additional whitening of the input and output. The only non-Feistel elements are the 1-bit rotates. The plaintext is split into four 32-bit words. In the input whitening step, these are XORed with four key words. This is followed by sixteen rounds. In each round, the two words on the left are input to the *g* functions. (One of them is rotated by 8 bits first.) The *g* function consists of four byte-wide key-dependent *S*-boxes, followed by a linear mixing step based on an MDS matrix. The results of the two *g* functions are combined using a Pseudo- Hadamard Transform (PHT), and two keywords are added. These two results (called the outputs of the *F* function) are then XORed into the words on the right (one of which is rotated left by 1 bit first, the other is rotated right afterwards). The left and right halves are then swapped for the next round. After all the rounds, the swap of the last round is reversed, and the four words are XORed with four more key words to produce the ciphertext. So, the key schedule preparesa total of fourty 32-bit subkeys.

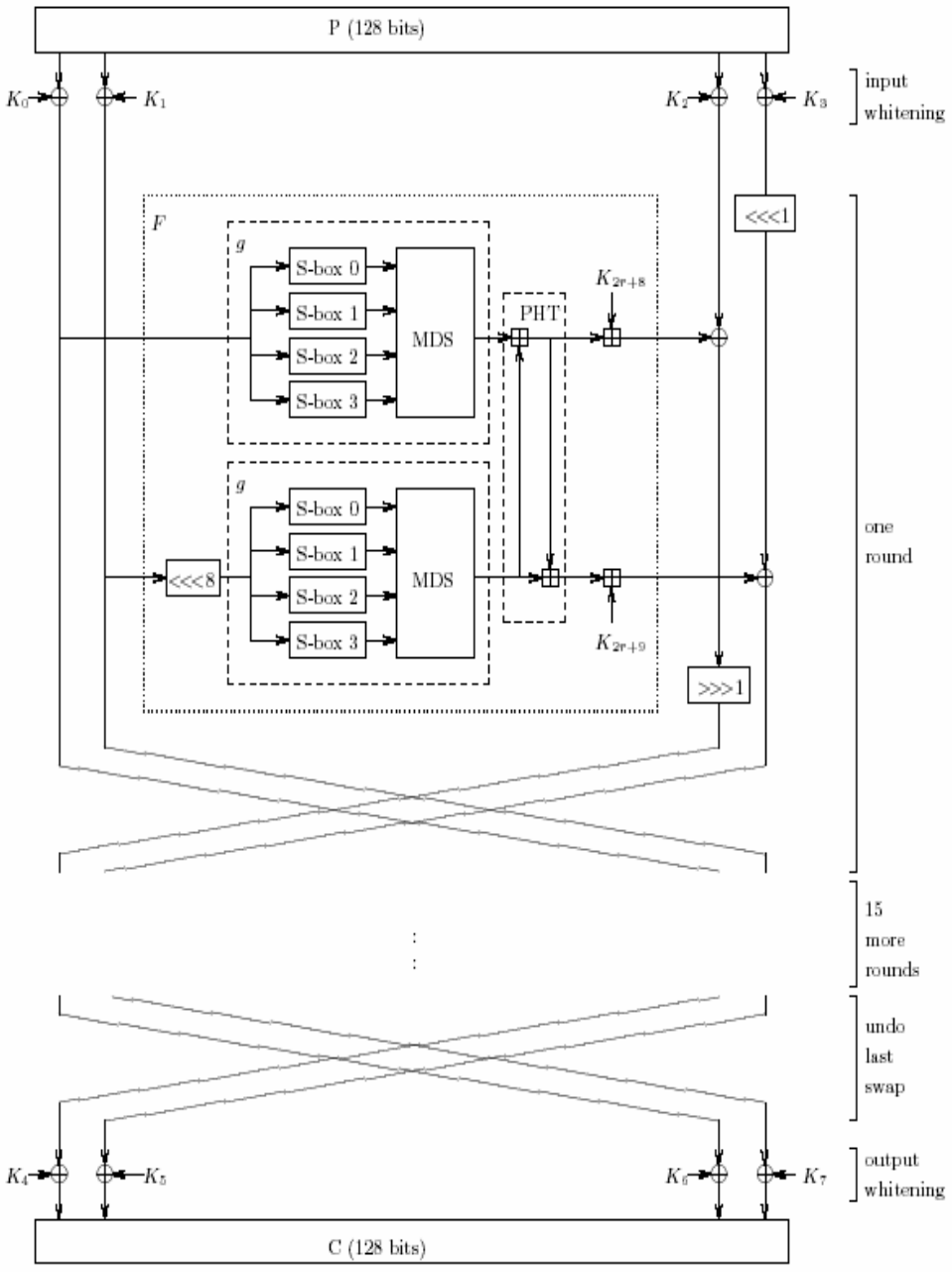

Figure 2.3 Twofish Encryption Algorithm Block

More formally, the 16 bytes of plaintext p_0,.., p_{15} (p_0 is the most significant byte of the plaintext, and p_{15} is the least significant bit of the plaintext) are first split into 4 words P_0,…, P_3 of 32 bits each using the little-endian convention.

$$P_i = \sum_{j=0}^{3} p_{(4i+j)} \cdot 2^{8j} \qquad i = 0,\ldots,3 \qquad (2.1)$$

In the input whitening step, these words are XORed with 4 words of the expanded key.

$$R_{0,i} = P_i \oplus K_i \qquad i = 0,..,3 \qquad (2.2)$$

In each of the 16 rounds, the first two words are used as input to the function F, which also takes the round number as input to select the appropriate subkeys. The third word is XORed with the first output of F and then rotated right by one bit. The fourth word is rotated left by one bit and then XORed with the second output word of F. Finally, the two halves are exchanged. Thus, outputs $F_{r,0}$ and $F_{r,1}$ of the F function and 4 input words R_{r+1} of the successive round are found as:

$$\begin{aligned}
&(F_{r,0}, F_{r,1}) = F(R_{r,0}, R_{r,1}, r)\\
&R_{r+1,0} = \mathrm{ROR}(R_{r,2} \oplus F_{r,0}, 1)\\
&R_{r+1,1} = \mathrm{ROL}(R_{r,3}, 1) \oplus F_{r,1}\\
&R_{r+1,2} = R_{r,0}\\
&R_{r+1,3} = R_{r,1}
\end{aligned} \qquad (2.3)$$

for $r = 0,\ldots,15$ and ROR and ROL are functions that rotate their first argument (a 32-bit word) left or right by the number of bits indicated by their second argument.

The output whitening step undoes the "swap" of the last round, and XORs the data words with 4 words of the expanded key. The output block is then

$$C_i = R_{16,(i+2) \bmod 4} \oplus K_{i+4} \qquad i = 0,\ldots,3$$

The four words of ciphertext are then written as 16 bytes $c_0,\ldots, c_{15}$ using the same little-endian conversion used for the plaintext. The output block is obtained as c_i.

$$c_i = \left\lfloor \frac{C_{\lfloor i/4 \rfloor}}{2^{8(i \bmod 4)}} \right\rfloor \bmod 2^8 \qquad i = 0,\ldots,15$$

2.4.1 Main Functions of Twofish Algorithm

a The Function *g*

The function *g* forms the heart of Twofish algorithm; it is the main component of the *F* function. It uses an 32-bit vectors *X* and an 64-bit vector *L* to produce the 32-bit output $Z=g(X,L)$. Its 32-bit input word *X* (*X* is either $R_{r,0}$ or $\mathrm{ROL}(R_{r,1},8)$) is split into four bytes. Each byte x_i is run through its own key-dependent *S*-box, s_i. The four *S*-box outputs y_i are interpreted as a vector of length 4 over $GF(2^8)$, and multiplied by the 4x4 MDS matrix (using the field $GF(2^8)$ for the computations). The resulting vector *Z* is a 32-bit word.

$$x_i = \lfloor X/2^{8i} \rfloor \bmod 2^8 \qquad i = 0,\ldots,3$$

$$y_i = s_i[x_i] \qquad i = 0,\ldots,3 \qquad (2.4)$$

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \bullet \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \tag{2.5}$$

$$Z = \sum_{i=0}^{3} z_i \cdot 2^{8i}$$

In (2.4) s_i are the key-dependent *S*-boxes (*S*-box0 to *S*-box3) and the elements of the second 64-bit input $L=(l_{0,0}\ l_{0,1}\ l_{0,2}\ l_{0,3}\ l_{1,0}\ l_{1,1}\ l_{1,2}\ l_{1,3})$ are used as the *S*-box constants, which are indicated in (2.7). The vector *L* is obtained from the keys. For MDS matrix multiplication (2.5) to be well-defined, the correspondence between byte values and the field elements of GF(2^8) are needed to be specified. GF(2^8) is represented as GF(2)[*x*]/*v*(*x*) where $v(x) = x^8+x^6+x^5+x^3+1$ is a primitive polynomial of degree 8 over GF(2). The field element $a = \sum_{i=0}^{7} a_i \cdot x^i$ with $a_i \in$ GF(2) is identified with the byte value $\sum_{i=0}^{7} a_i \cdot 2^i$. Note that, addition in GF(2^8) corresponds to an XOR of the bytes.

b The Function *F*

The function *F* mentioned in (2.3) is a key-dependent permutation on 64-bit values. It takes three arguments, two input words $R_{r,0}$ and $R_{r,1}$, and the round number *r* used to select the appropriate subkeys. $R_{r,0}$ is passed through the *g* function, which yields $T_{r,0}$. $R_{r,1}$ is rotated left by 8 bits and then passed through the *g* function to yield $T_{r,1}$. The 64-bit vector *L* which adjusts the *S*-box constants is prepared from the

original key as in (2.12), so, L=S=(S_1 S_0) The results $T_{r,0}$ and $T_{r,1}$ are then combined in a pseudo hadamard transformer and two words of the expanded key are also added modulo 2^{32} which is different from the XOR operation. The following set of equations describe the details of F function,

$$T_{r,0} = g(R_{r,0}, S)$$

$$T_{r,1} = g(\mathrm{ROL}(R_{r,1}, 8), S) \qquad (2.6)$$

$$F_{r,0} = (T_{r,0} + T_{r,1} + K_{2r+8}) \bmod 2^{32}$$

$$F_{r,1} = (T_{r,0} + 2\mathrm{T}_{r,0} + K_{2r+9}) \bmod 2^{32}$$

where ($F_{r,0}$,$F_{r,1}$) is the result of F. Fig. 2.4. shows the F function in detail, where (2.6) can be observed in the lower part of the figure that uses g functions. The upper part of the figure, which uses h functions, is related to the key schedule to be described by (2.13). The round keys K_{2r+8} and K_{2r+9} used in (2.6) are produced in the upper part of Fig. 2.4., as explained in (2.13). The h function also has key dependent S-boxes, where the S-box constants are prepared from the original key M, by dividing it into 32-bit pieces, M_0, M_1, M_2, M_3, and choosing either the even or the odd indexed segments, so respectively, M_e=(M_0 M_2) and M_o=(M_1 M_3) as shown in (2.11)

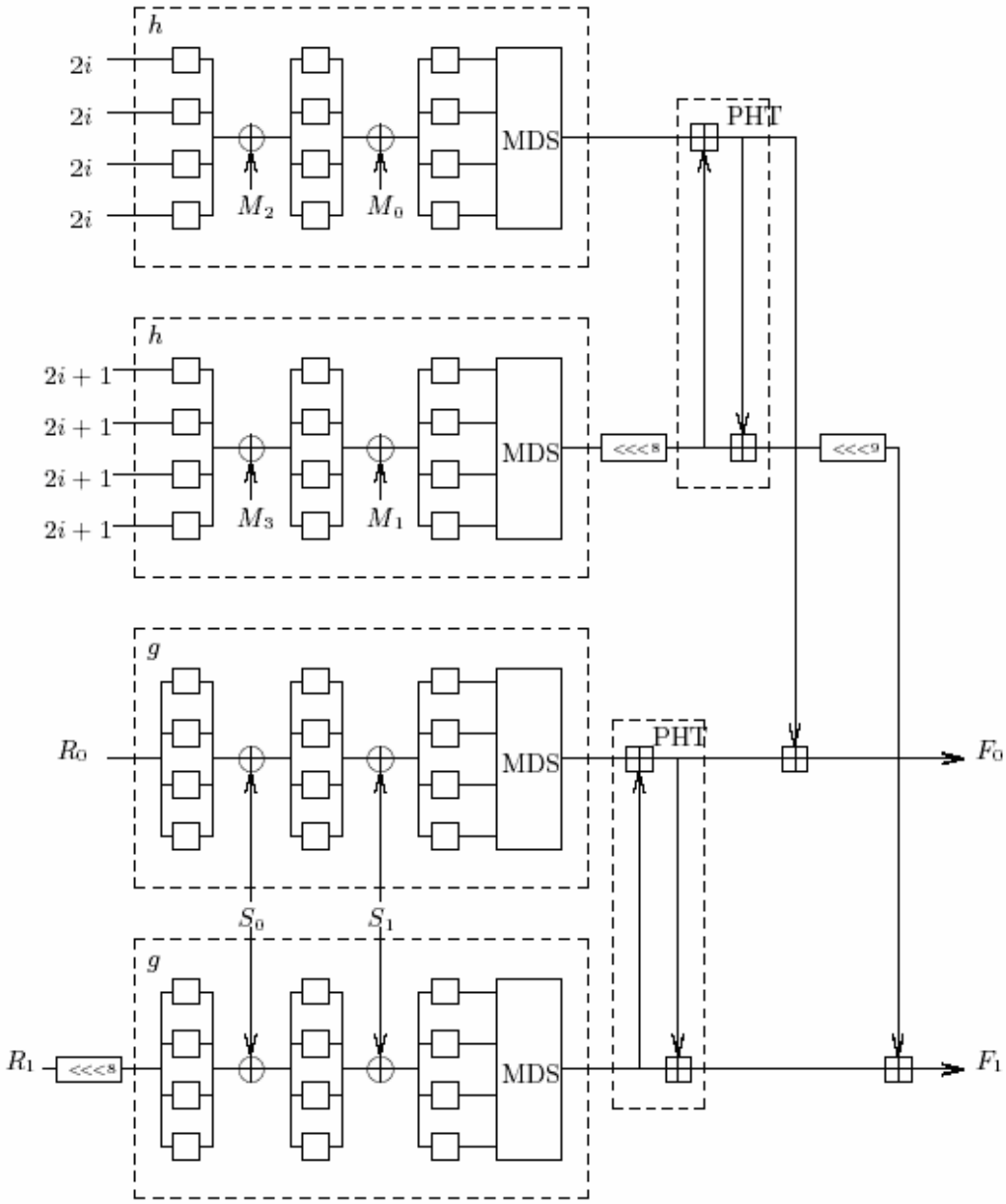

Figure 2.4 A view of a single round *F* function (128-bit key)

c The Function *h*

The function $h(X,L)$ is used to obtain expanded keywords of Twofish algorithm. *h* function is very similar to the function *g*, therefore equations (2.4) and (2.5) describe it completely. Its 32-bit input word X is split into four bytes. Each byte is run through its own key-dependent *S*-box. The four results are interpreted as a vector of length 4 over GF(2^8), and multiplied by the 4x4 MDS matrix (using the field GF(2^8) for the computations). The resulting vector is a 32-bit word.

Note that the of *h* and *g* functions are exactly same as each other but their inputs are different. X is obtained from $R_{r,0}$ or $R_{r,1}$ for the function *g*, whereas for the function *h*, it is chosen as the 32-bit vector $\rho=(i\ i\ i\ i)$ where i is the 8-bit vector corresponding to $i=0,\ldots,39$. Also the *S*-box constant vector L is different for *h* and *g* functions. In *h* function, L is either M_e or M_o, whereas in *g* function $L=S$. The method of obtaining the vectors S, M_e, and M_o from the original key is described in section 2.4.2.

d The Key-dependent *S*-boxes

Twofish algorithm uses a single 32x32 *S*-box which can be considered as four 8x8 *S*-boxes with different combinations of permutation boxes, q_0 and q_1, which are explained in section 2.4.2. As can be seen from Fig. 2.4, the *S*-boxes are used both in *h* and *g* functions. The combination of permutation boxes is the same for the *S*-boxes of *h* and *g* functions, but their input parameters are different. For $h(X,L)$ function the

input parameters are $\rho=X$ and $L=M_e$ or $L=M_o$. For $g(X,L)$ function the input parameters are $X=R_{r,0}$ or $X=\mathrm{ROL}(R_{r,1},8)$ and $S=L$.

32x32 *S*-box takes two inputs a 32-bit word X and a list $L = (L_0,\ldots,L_{k-1})$ of 32-bit words of length k, where k is the number of 64-bit segments in the original key. In this thesis Twofish algorithm is implemented for 128-bit keywords so $k=N/64=2$. The vectors X and L are split into bytes.

$$l_{i,j} = \lfloor L_i / 2^{8j} \rfloor \bmod 2^8$$

$$x_j = \lfloor X / 2^{8j} \rfloor \bmod 2^8$$

for $i = 0,\ldots, k - 1$ and $j = 0,\ldots,3$. Then the sequence of substitutions and XORs is applied.

$$y_0 = s_0[x_0] = q_1[q_0[q_0[x_0] \oplus l_{1,0}] \oplus l_{0,0}], \text{ (}S\text{-box0 formulation)}$$

$$y_1 = s_1[x_1] = q_0[q_0[q_1[x_1] \oplus l_{1,1}] \oplus l_{0,1}], \text{ (}S\text{-box1 formulation)}$$

$$y_2 = s_2[x_2] = q_1[q_1[q_0[x_2] \oplus l_{1,2}] \oplus l_{0,2}], \text{ (}S\text{-box2 formulation)}$$

$$y_3 = s_3[x_3] = q_0[q_1[q_1[x_3] \oplus l_{1,3}] \oplus l_{0,3}], \text{ (}S\text{-box3 formulation)} \qquad (2.7)$$

The output of the *S*-boxes is the 32-bit word Y in the form of $y_3y_2y_1y_0$. Fig. 2.5. shows the *S*-box formulation of 128 bit Twofish cipher.

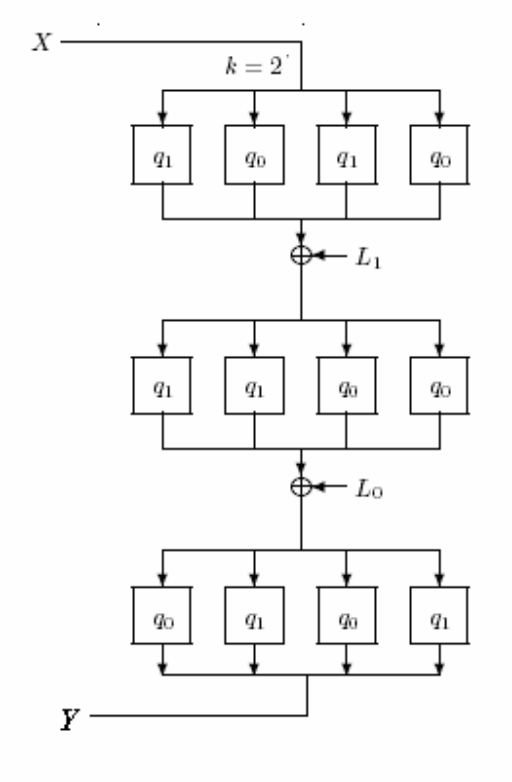

Figure 2.5 *S*-box formulation of Twofish algorithm

2.4.2 Sub-functions of Twofish Algorithm

a The Permutations q_0 and q_1

The permutations q_0 and q_1 are fixed permutations on 8-bit values. These permutation functions are the main components of the *S*-boxes. They are constructed from four different 4-bit permutations each. For the 8-bit input value x, the corresponding output value y is found by the following steps:

$a_0 = \lfloor x/16 \rfloor$ and $b_0 = x \bmod 16$

i.e., the byte is first split into two 4-bit nibbles, a_0 and b_0

$$a_1 = a_0 \oplus b_0$$

$$b_1 = a_0 \oplus \mathrm{ROR}(b_0, 1) \oplus (8a_0 \bmod 16)$$

$$a_2 = t_0[a_1]$$

$$b_2 = t_1[b_1]$$

$$a_3 = a_2 \oplus b_2$$

$$b_3 = a_2 \oplus \mathrm{ROR}(b_2, 1) \oplus 8a_2 \bmod 16$$

$$a_4 = t_2[a_3]$$

$$b_4 = t_3[b_3]$$

$$y = 16\, b_4 + a_4 \qquad (2.8)$$

As in (2.8), these nibbles are combined in a bijective mixing step. Each nibble is then passed through its own 4-bit table look-up. This is followed by another mixing step and table lookup. Finally, the two nibbles are recombined into a byte.

The equation set (2.8) describes both of the permutations q_0 and q_1, but the lookup tables $t_0,\ldots,t_3$ are different for q_0 and q_1.

For the permutation q_0, lookup tables are given by

$$t_0 = [\ 8\ 1\ 7\ D\ 6\ F\ 3\ 2\ 0\ B\ 5\ 9\ E\ C\ A\ 4\ ] \qquad (2.9)$$

$$t_1 = [\ E\ C\ B\ 8\ 1\ 2\ 3\ 5\ F\ 4\ A\ 6\ 7\ 0\ 9\ D\ ]$$

$$t_2 = [\ B\ A\ 5\ E\ 6\ D\ 9\ 0\ C\ 8\ F\ 3\ 2\ 4\ 7\ 1\ ]$$

$$t_3 = [\ D\ 7\ F\ 4\ 1\ 2\ 6\ E\ 9\ B\ 3\ 0\ 8\ 5\ C\ A\ ]$$

where each lookup table is represented by a list of the entries using hexadecimal notation. (The entries for the inputs 0,1,…,15 are listed in order.) Similarly, for q_1 the lookup tables are given by

$$t_0 = [\ 2\ 8\ B\ D\ F\ 7\ 6\ E\ 3\ 1\ 9\ 4\ 0\ A\ C\ 5\ ]$$

$$t_1 = [\ 1\ E\ 2\ B\ 4\ C\ 3\ 7\ 6\ D\ A\ 5\ F\ 9\ 0\ 8\ ]$$

$$t_2 = [\ 4\ C\ 7\ 5\ 1\ 6\ 9\ A\ 0\ E\ D\ 8\ 2\ B\ 3\ F\ ]$$

$$t_3 = [\ B\ 9\ 5\ 1\ C\ 3\ D\ E\ 6\ 4\ 7\ F\ 2\ 0\ 8\ A\ ] \qquad (2.10)$$

b The Key Schedule

The key schedule has to provide 40 words of the expanded key $K_0,\ldots,K_{39}$, and the constant vectors for the key-dependent *S*-boxes used in the *g* and *h* functions. Twofish is defined for keys of length $N = 128$, $N = 192$, and $N = 256$. Keys of any length shorter than 256 bits can be used by padding them with zeroes until the next larger defined key length. The parameter k is defined as $k = N/64$. The original key M consists of $8k$ bytes $m_0,\ldots,m_{8k-1}$. To obtain the constant vectors for key dependent *S*-boxes, the bytes are first converted into $2k$ words of 32 bits each

$$M_i = \sum_{j=0}^{3} m_{(4i+j)} \cdot 2^{8j} \qquad i = 0,\ldots,2k\text{-}1$$

and then into two word vectors of length k.

$$M_e = (M_0, M_2, \ldots, M_{2k-2})$$

$$M_o = (M_1, M_3, \ldots, M_{2k-1}) \qquad (2.11)$$

M_e and M_o are the constant vectors of the key dependent S-boxes employed in the h function, to obtain the expanded keys $K_0,\ldots,K_{39}$. For the 128-bit key length is used in this study, k=2, hence M_e=(M_0 M_2) and M_o=(M_1 M_3).

A third vector S of length k 32-bit words is also derived from the key, as the constant vector for the key dependent S-boxes of the function g. This is done by taking the key bytes in groups of 8, interpreting them as a vector over GF(2^8), and multiplying them by a 4x8 matrix derived from an RS code. Each result S_i of 4 bytes is then interpreted as a 32-bit word.

$$\begin{pmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,3} \end{pmatrix} = \begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 85 & 5A & 58 & DB & 9E & 03 \end{pmatrix} \bullet \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix} \qquad (2.12)$$

Using $S_i = \sum_{j=0}^{3} s_{i,j} \cdot 2^{8j}$ for i = 0,…,k - 1, one obtains the third vector S = (S_{k-1}, S_{k-2},…, S_0) Note that S lists the words in “reverse" order. For the RS matrix multiplication in (2.12), GF(2^8) is represented by GF(2)[x]/w(x), where $w(x) = x^8 + x^6 + x^3 + x^2 + 1$ is another primitive polynomial of degree 8 over GF(2).

For 128-bit keys, three vectors M_e, M_o, S are all 64-bit vectors, which forms the S-box constants. M_e and M_o are used in h function which produces the expanded key; whereas S is used in g function which encrypts the plaintext.

c The Expanded Key Words *Kj*

The words of the expanded key are defined using the h function. The input vector X of the $h(X,L)$ function is derived from the initial vector $\rho = 2^{24} + 2^{16} + 2^8 + 2^0$. To evaluate 40 keywords, one computes for all values of i=0,…,19.

$$A_i = h(2i\,\rho, M_e)$$

$$B_i = \mathrm{ROL}(h((2i + 1)\,\rho, M_o),\ 8)$$

$$K_{2i} = (A_i + B_i) \bmod 2^{32}$$

$$K_{2i+1} = \mathrm{ROL}((A_i + 2B_i) \bmod 2^{32},\ 9) \qquad (2.13)$$

Notice that for producing A_i the first argument of h function has all byte values equal to $2i$, and the second argument of h is M_e. B_i is computed similarly using $2i + 1$ as the byte value and M_o as the second argument, with an extra rotate over 8 bits. The values A_i and B_i are then combined in a PHT. One of the results is further rotated by 9 bits. The two results K_{2i} and K_{2i+1} form the two 32-bit words of the expanded key.

2.4.3 Cryptanalysis of Twofish

A summary of successful attacks performed by the designers of the cipher [29] is as follows:

5-round Twofish (without the post-whitening) with 2^{225} chosen plaintext pairs and 2^{51} computations of the function g. 10-round Twofish (without the pre- and post-whitening) with a chosen-key attack, requiring 2^{32} chosen plaintexts and about 2^{11} adaptive chosen plaintexts, and about 2^{32} work.

The fact that Twofish seems to resist related-key attacks well is arguably the most interesting result, because related-key attacks give the attacker the most control over the cipher's inputs. Based on analysis, it is conjectured that there exists no more efficient attack on Twofish than brute force. The most efficient attack against Twofish with a 128 bit key has a complexity of 2^{128}, the most efficient attack against Twofish with a 192-bit key has a complexity of 2^{192}, and the most efficient attack against Twofish with a 256-bit key has a complexity of 2^{256}.

CHAPTER 3

CRYPTANALYSIS ATTACKS

3.1 Cryptanalysis Techniques in Brief

Cryptanalysis is the aspect of cryptology which concerns the strength analysis of a cryptographic system, and the penetration or breaking of a cryptographic system.

The goal of an attack is to reveal some unknown plaintext, or the key, which will reveal the plaintext. Some of well-known cryptanalysis techniques are explained below.

Brute Force (also Exhaustive Key Search): Try to decipher ciphertext under every possible key until readable messages are produced.

Codebook (the classic "code-breaking" approach): Collect a codebook of transformations between plaintext and ciphertext.

Differential Cryptanalysis: Find a statistical correlation between key values and cipher transformations (typically the XOR of text pairs), then use sufficient defined plaintext to develop the key.

Linear Cryptanalysis: Find a linear approximation to the keyed *S*-boxes in cipher, and use that to reveal the key.

Meet-in-the-Middle: Given a two-level multiple encryption, search for the key by collecting every possible result for enciphering a known plaintext under the first cipher, and deciphering the known ciphertext under the second cipher; then find the match.

Key Schedule: Choose keys which produce known effects in different rounds.

Birthday (usually a hash attack): Use the birthday paradox, the idea that it is much easier to find two values which match than it is to find a match to some particular value.

Formal Coding (also Algebraic): From the cipher design, develop equations for the key in terms of known plaintext, then solve those equations.

Correlation: In a stream cipher, distinguish between data and confusion, or between different confusion streams, from a statistical imbalance in a combiner.

Dictionary: Form a list of the most-likely keys, then try those keys one-by-one (a way to improve brute force).

Replay: Record and save some ciphertext blocks or messages (especially if the content is known), then re-send those blocks when useful.

Many attacks try to isolate unknown small components or aspects so they can be solved separately, a process known as divide and conquer.

3.2 Strength and Cryptanalysis

Because there are no tools for the discussion of strength under all possible attacks, cipher "strength" is normally discussed in the context of particular attacks. Each known attack approach can be elaborated for a particular cipher, and a value calculated for the effort required breaking the cipher in that way; this may set an "upper bound" on the unknown strength of the cipher. And while this is certainly better than not knowing the strength with respect to known attacks, such attacks may not represent the actual threat to the cipher in the field. In general, "lower bound" or "true" strength of a cipher is not known. So, unless a cipher is shown to be weaker than can be accepted, cryptanalysis provides no useful information about cipher strength.

Two most powerful cryptanalysis techniques applied to symmetric-key block ciphers are the linear cryptanalysis and the differential cryptanalysis. Linear cryptanalysis was introduced by Matsui [20] at EUROCRYPT ’93 as a theoretical attack on the Data Encryption Standard (DES) and later successfully used in the practical cryptanalysis of DES; differential cryptanalysis was first presented by Bilham and Shamir [4] at CRYPTO ’90 to attack DES and eventually the details of the attack were packaged as a book. Although the early target of both attacks was DES, the wide applicability of these attacks to numerous other block ciphers has solidified the pre-eminence of both cryptanalysis techniques in the consideration of the security of all block ciphers.

3.3 Linear Cryptanalysis

Linear cryptanalysis tries to take advantage of high probability occurrences of linear expressions involving plaintext bits, "ciphertext" bits (actually we shall use bits from the 2nd last round output), and subkey bits. It is a known plaintext attack: that is, it is premised on the attacker having information on a set of plaintexts and the corresponding ciphertexts. However, the attacker has no way to select which plaintexts (and corresponding ciphertexts) are available. In many applications and scenarios it is reasonable to assume that the attacker has knowledge of a random set of plaintexts and the corresponding ciphertexts. The basic idea is to approximate the operation of a portion of the cipher with an expression that is linear where the linearity refers to a mod-2 bit-wise operation (i.e., exclusive-OR denoted by "⊕"). Such an expression is of the form:

$$X_{i1} \oplus X_{i2} \oplus \ldots \oplus X_{iu} \oplus Y_{i1} \oplus Y_{i2} \oplus \ldots \oplus Y_{iv} = 0 \qquad (3.1)$$

where X_i represents the i-th bit of the input $X = [X_1, X_2, ...]$ and Y_j represents the j-th bit of the output $Y = [Y_1, Y_2, ...]$. This equation is representing the exclusive-OR "sum" of u input bits and v output bits. The approach in linear cryptanalysis is to determine expressions of the form above which have a high or low probability of occurrence. If a cipher displays a tendency for equation (3.1) to hold with high probability or not hold with high probability, this is evidence of the cipher's poor randomization abilities. Consider that if values for $u + v$ bits are randomly selected and placed into the equation above, the probability that the expression would hold

would be exactly 1/2. It is the deviation or bias from the probability of 1/2 for an expression to hold that is exploited in linear cryptanalysis: the further away that a linear expression is from holding with a probability of 1/2, the better the cryptanalyst is able to apply linear cryptanalysis. The amount by which the probability of a linear expression holding deviates from 1/2 is referred as the *linear probability bias*. Hence, if the expression above holds with probability p_L for randomly chosen plaintexts and the corresponding ciphertexts, then the probability bias is $p_L - 1/2$. The higher the magnitude of the probability bias, $|p_L - 1/2|$, the better the applicability of linear cryptanalysis with fewer known plaintexts required in the attack.

3.4 Differential Cryptanalysis

Differential cryptanalysis exploits the high probability of certain occurrences of plaintext differences and differences into the last round of the cipher. For example, consider a system with input $X = [X_1 X_2 ... X_n]$ and output $Y = [Y_1 Y_2 ... Y_n]$. Let two inputs to the system be X' and X'' with the corresponding outputs Y' and Y'', respectively. The input difference is given by $\Delta X = X' \oplus X''$ where "$\oplus$" represents a bit-wise exclusive-OR of the n-bit vectors and, hence,

$$\Delta X = [\Delta X_1, \Delta X_2, ..., \Delta X_n]$$

where $\Delta X = X_i' \oplus X_i''$ with X_i' and X_i'' representing the i-th bit of X' and X'', respectively. Similarly, $\Delta Y = Y' \oplus Y''$ is the output difference and

$$\Delta Y = [\Delta Y_1, \Delta Y_2, ..., \Delta Y_n]$$

where $\Delta Y = Y_i' \oplus Y_i''$.

In an ideally randomizing cipher, the probability that a particular output difference ΔY occurs given a particular input difference ΔX is $1/2^n$ where n is the number of bits of X. Differential cryptanalysis seeks to exploit a scenario where a particular ΔY occurs given a particular input difference ΔX with a very high probability p_D (i.e., much greater than $1/2^n$). The pair (ΔX, ΔY) is referred to as a *differential*.

Differential cryptanalysis is a chosen plaintext attack, meaning that the attacker is able to select inputs and examine outputs in an attempt to derive the key. For differential cryptanalysis, the attacker will select pairs of inputs, X' and X'', to satisfy a particular ΔX, knowing that for that ΔX value, a particular ΔY value occurs with high probability. As with linear cryptanalysis, to construct highly likely differential characteristics, the properties of individual S-boxes are examined and these properties are used to determine the complete differential characteristic. Specifically, the input and output differences of the S-boxes are considered in order to determine a high probability difference pair. Combining S-box difference pairs from round to round so that the nonzero output difference bits from one round correspond to the non-zero input difference bits of the next round, enables finding a high probability differential, consisting of the plaintext difference and the difference of the

input to the last round. The subkey bits of the cipher end up disappearing from the difference expression because they are involved in both data sets and, hence, considering their influence on the difference involves XORing subkey bits with themselves, the result of which is zero.

CHAPTER 4

TEST CRITERIA FOR BLOCK CIPHERS

4.1 Avalanche Criteria

The idea of avalanche [12] was introduced by Feistel. For a given transformation to exhibit the avalanche effect, an average of one half of the output bits should change whenever a single input bit is complemented. In order to determine whether a given nxn function f satisfies this requirement, the 2^n plaintext pairs, P and P_i. Such that P and P_i differ only in bit i are used to calculate the 2^n difference vectors, $\Delta C = f(P) \oplus f(P_i) = e_i$. These XOR sums are referred as avalanche vectors, each of which contains n bits, called avalanche variables. If this procedure is repeated for all i such that $1 \leq i \leq n$, and one half of the avalanche variables are equal to 1 for each i, then the function f has good avalanche effect. Avalanche properties and avalanche weight distribution characteristics of block ciphers help us analyze diffusion and confusion properties of block ciphers.

The principle of diffusion and confusion was introduced by Shannon [31] in 1949 and simply can be stated as:

Diffusion: Diffusion tries to distribute the redundancy of the plaintext over the cipher text. Every bit of the ciphertext should depend on every bit of the plaintext. Good diffusion spreads the influence of individual plaintext characters over as much of the ciphertext as possible, thereby hiding the statistical features of the plaintext.

Confusion: Confusion is described as being "the use of ciphering transformations that complicate the determination of how the statistics of the ciphertext depend on the statistics of the plaintext" [33] or, more briefly, to make the relation between the key and the ciphertext as complex as possible. The objective is to hide redundancies in plaintext. Every bit of the ciphertext should depend on every bit of the key.

Completeness: Completeness is the result of diffusion and was introduced by Kam and Davida [15]. If a cryptographic transformation is *complete*, then each ciphertext bit must depend on all plaintext bits. Thus, if it were possible to find the simplest boolean expression for each ciphertext bit in terms of plaintext bits, each of those expressions would have to contain all of the plaintext bits if the function was complete.

4.1.1 Avalanche Weight Distribution

Avalanche weight distribution (AWD) [3] criterion can be stated as follows: Even for quite similar plaintext pairs (P_1, P_2), i.e., when the hamming weight of the

differences of plaintext pairs (P_1, P_2) is small, the distribution of the hamming weight of the differences of corresponding ciphertext pairs (C_1, C_2) should be close to a binomial distribution around $n/2$ for a good block cipher with a block length of n. This criterion reveals the diffusion property of block ciphers.

It should be noted that the AWD of an ideal algorithm satisfying the diffusion property, the probability of finding any particular number of i ciphertext bit changes in a ciphertext of n bits is:

$$B(i) = \frac{\binom{n}{i}}{2^n}, \quad 0 \le i \le n \tag{4.1}$$

which is the binomial expression. Also notice that

$$\sum_{i=0}^{n} B(i) = 1 \tag{4.2}$$

In Fig. 4.1, ideal binomial distribution curve is sketched for 128 bits

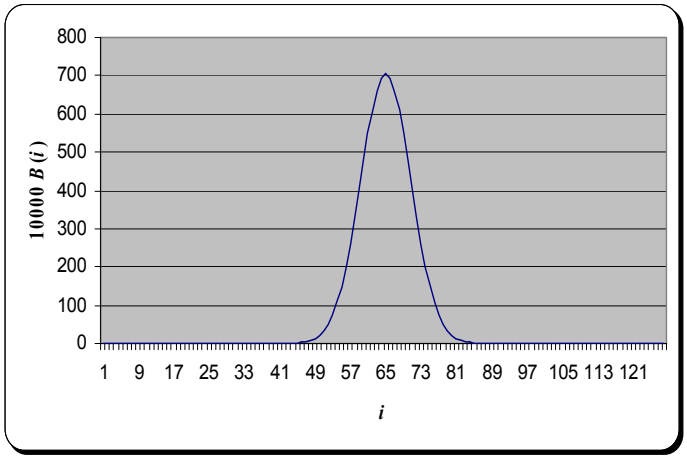

Figure 4.1: Binomial Distribution Curve for n = 128 bits

The distortion measure D, which gives the distortion between the actual AWD of the cipher and the ideal distribution $B(j)$ is calculated using N pairs of plaintexts (P, $P \oplus \Delta P$) with a fixed difference ΔP, and corresponding ciphertexts (C, $C \oplus \Delta C$). ΔC of weight j increments the array element $AWD(j)$ by 1. Then, deviation of the cipher from the ideal binomial distribution is found for a specific plaintext difference ΔP of hamming weight 1 as

$$D^i = \frac{1}{2N} \sum_{j=0}^{n} \left| AWD(j) - NB(j) \right| \tag{4.3}$$

where i corresponds to the index of "1" in the plaintext difference (ΔP) vector.

Corresponding resemblance parameter R to a binomial distribution is then given by

$$R^i = 1 - D^i \tag{4.4}$$

While defining the distortion by (4.3) the magnitude of [$AWD(j)$ - $NB(j)$] is used in order not to make a distinction between positive or negative errors. The normalization coefficient of $1/2N$ is added to restrict the worst case value of R to 0. (Notice that the worst case occurs when non-zero values of $AWD(j)$ correspond to zero values of $B(j)$ and vice versa)

If $R^i = 1$ then the actual AWD is exactly the same as the ideal binomial distribution. In the worst case $R^i = 0$, and the AWD of the corresponding block cipher shows no resemblance to the ideal binomial distribution.

4.1.2 Avalanche Criteria Analysis Procedures

In this section, the test procedures of avalanche criterion, AWD and resemblance parameter analysis are given. These procedures are applied to the studied cipher in Chapter 5.

a Avalanche Criterion

The following steps are used in this avalanche criterion test procedure:

1. A key is chosen randomly.

2. A plaintext P is chosen at random and the pair of that plaintext P_i is calculated so that the difference between P and P_i is, i.e. $P_i = P \oplus e_i$ and P and P_i differ only in bit i, where e_i is a n-bit unit vector with a position i, and $i \in \{1, 2,\ldots, n\}$,

3. P and P_i are submitted to r-rounds of cipher for encryption under the key chosen in step 1,

4. From the resultant ciphertexts C and C_i, the avalanche vector $\Delta C = C \oplus C_i$ is calculated,

5. The avalanche vector is summed up to an avalanche sum array,

6. The above steps 2-5 are repeated N (typically 10000) times and the values in the avalanche sum array are sketched versus its index.

It is expected due to the avalanche criterion that an average of one half of the output bits should change whenever a single input bit is changed, so if we use 10000

sample plaintexts all n entries in the avalanche sum array should be around 5000. So we expect a straight line around 5000 as the result of avalanche criterion.

b Avalanche Weight Distribution

To determine the diffusion properties of ciphers a derivative of avalanche criterion, avalanche weight distribution (AWD) curves are helpful. The criterion can be stated as: even for small hamming weight differences at the input (plaintext or keybits), the distribution of the hamming weight of the ciphertext differences (avalanche vectors) should be close to a binomial distribution around $n/2$ for a good block cipher with a block length of n.

To investigate the *diffusion* properties of cipher the following test procedure is used for the criterion of avalanche weight distribution (AWD).

1. A key is chosen randomly,

2. A plaintext P is chosen at random and the pair of that plaintext P_i is calculated so that the difference between P and P_i is, i.e. $P_i = P \oplus e_i$ and P and P_i differ only in bit i, where e_i is a n-bit unit vector with a position i, and $i \in \{1, 2,\ldots, n\}$,

3. P and P_i are submitted to r-round of cipher for encryption under the key chosen at step 1,

4. From the resultant ciphertexts C and C_i, the hamming weight of the avalanche vector $wt(\Delta C) = wt(C \oplus C_i) = j$ is calculated, where $j \in \{1, 2,\ldots, n\}$,

5. The value of the j^{th} element of an AWD array with a size of n is incremented by 1, i.e. $AWD[j] = AWD[j] + 1$,

6. The steps 2-5 are repeated N (typically 10000) times and the values in the AWD array are sketched versus its index, as the AWD curve corresponding to the input difference $\Delta P = e_i$.

c Resemblance Parameter Analysis

After the AWD array is found distortion measure (D^i) and resemblance parameter (R^i) can be found with the following procedure:

1. Obtain the AWD curve corresponding to $\Delta P = e_i$,

2. Calculate the binomial distribution function $B(j)$ where $j \in \{1, 2,\ldots, n\}$,

3. Find the sum of absolute difference of $AWD[j]$ and $B(j)$ for each j where $j \in \{1, 2,\ldots, n\}$,

4. Calculate D^i and R^i using equations (4.3) and (4.4) respectively.

In Chapter 5, avalanche and AWD curves to investigate diffusion properties of RC5, RC6 and Twofish ciphers with random plaintext or keyword and differences at different positions i are presented. The results of avalanche criteria and NIST' s statistical test results are compared in this chapter. The nonlinearity of the S-boxes of Twofish cipher is also evaluated in Chapter 5.

4.2 Nonlinearity Measure

Encryption mappings are often designed to satisfy a set of chosen criteria which have been established either formally or empirically as essential to the security of the cipher. Two basic criteria due to Shannon suggest that a cipher should be constructed using the notions of *diffusion* and *confusion*. As described in the preceding sections, diffusion refers to the dissipation of the statistical properties of the plaintext, while confusion refers to the internal operations of the cipher that produce complex relations between the plaintext, key and ciphertext.

If a ciphertext bit c_i is described by the boolean function f_i then it is generally accepted that each f_i should possess a combination of the properties such as balance, nonlinearity, completeness, correlation immunity, the strict avalanche criterion, or be bent.

The nonlinearity of many block ciphers depend directly on the selection of the *S*-boxes since, typically, the *S*-boxes are the only non-affine component of the cipher. So one can state that, if the *S*-boxes are affine then the entire mapping is affine [10]. In the following section, basic definitions of the nonlinearity criteria are given.

4.2.1 Basic Definitions of Nonlinearity Criteria

Affine Function: A boolean function $f(\mathbf{x})$ is called a affine function of $\mathbf{x} = (x_1 \ldots\ldots x_n) \in Z_2^n$, if it is in the form

$$f(\mathbf{x}) = a_1 x_1 \oplus a_2 x_2 \oplus \ldots\ldots \oplus a_n x_n \oplus c = \mathbf{w}.\mathbf{x} \oplus \mathbf{c} \qquad (4.5)$$

where $a_1, a_2 ....., a_n, c$ belong to Z_2 , $\mathbf{w} = (a_1, ..., a_n) \in Z_2^n$, and **w.x** denotes the inner product of vectors **w** and **x**.

In the boolean field, the coefficients a_i simply enable or disable the associated variable x_i. If $c = 0$, affine function is also linear.

Truth Table: The truth table $\boldsymbol{f}_\mathbf{t}$ of the boolean function $f(\mathbf{x})$ is found by evaluating $f(\mathbf{x})$ for all possible values of $\mathbf{x} = \boldsymbol{\alpha}_\mathbf{i}$; where $\boldsymbol{\alpha}_\mathbf{i}$ is the n-bit vector corresponding to binary representation of the integer $i = 0, ..., 2^n - 1$. So:

$$\boldsymbol{f}_\mathbf{t} = \{f(\boldsymbol{\alpha}_\mathbf{0}), ..., f(\boldsymbol{\alpha}_{\mathbf{2^n-1}})\} \tag{4.6}$$

Notice that the truth table of the boolean function $f : Z_2^n \to Z_2$ is a binary sequence of length 2^n.

Sequence of a Boolean Function: The sequence of a boolean function $f : Z_2^n \to Z_2$ is defined as:

$$\boldsymbol{f}_\mathbf{s} = \{(-1)^{f(\alpha_0)}, (-1)^{f(\alpha_1)}, ..., (-1)^{f(\alpha_{2^n-1})}\} \tag{4.7}$$

where $\boldsymbol{\alpha}_i$ is the *n*-bit vector corresponding to binary representation of the integer $i = 0, ..., 2^n - 1$. So, 0's and 1's of the truth table $\boldsymbol{f}_\mathbf{t}$ given by simply turn into +1's and –1's in the sequence $\boldsymbol{f}_\mathbf{s}$ given by (4.7). The sequence of a linear (affine) function is called a linear (affine) sequence.

Hamming Distance: Hamming distance between two functions $f : Z_2^n \to Z_2$ and $g : Z_2^n \to Z_2$ is defined as the hamming weight of the truth table of the difference function $f(\mathbf{x}) \oplus g(\mathbf{x})$.

$$d_H(f,g) = w_H(f(\mathbf{x}) \oplus g(\mathbf{x}))_t \tag{4.8}$$

where $d_H(f,g)$ is the hamming distance and $w_H(f(\mathbf{x}) \oplus g(\mathbf{x}))_t$ is the hamming weight of the truth table corresponding to the function $f(\mathbf{x}) \oplus g(\mathbf{x})$. Notice that the distance between boolean functions *f* and *g* is also equal to the hamming distance between their 2^n-bit truth tables $\boldsymbol{f}_\mathbf{t}$ and $\mathbf{g}_\mathbf{t}$; and the hamming distance between their 2^n-bit sequences $\boldsymbol{f}_\mathbf{s}$ and $\mathbf{g}_\mathbf{s}$. Hence:

$$d_H(f,g) = d_H(\boldsymbol{f}_\mathbf{t}, \mathbf{g}_\mathbf{t}) = d_H(\boldsymbol{f}_\mathbf{s}, \mathbf{g}_\mathbf{s}) \tag{4.9}$$

Hadamard Matrix: A Hadamard matrix *H* is an $n \times n$ matrix with entries +1 or -1, such that all rows and all columns are orthogonal, i.e., $HH^T = nI_n$ where H^T is the transpose of the Hadamard matrix and I_n is the identity matrix of order n. A special kind of Hadamard matrix, called the Sylvester-Hadamard matrix of order 2^n denoted by H_n is generated by the following recursive relation:

$$H_0 = 1, \qquad H_n = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix} \tag{4.10}$$

So; $H_1 = \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix}$ $H_2 = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix}$

and $2^3 \times 2^3$ Sylvester-Hadamard matrix H_3 can be obtained as follows:

$$H_3 = \begin{bmatrix} +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 \\ +1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 \\ +1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\ +1 & +1 & -1 & -1 & -1 & -1 & +1 & +1 \\ +1 & -1 & -1 & +1 & -1 & +1 & +1 & -1 \end{bmatrix}$$

It can be shown that each row (or column) of H_n is a linear sequence of length 2^n, i.e., it corresponds to the sequence of a linear function. There is a one to one mapping between each row (or column) l_i of a $(2^n \times 2^n)$ Sylvester-Hadamard matrix H_n, and the sequence of a linear function $l : Z_2^n \rightarrow Z_2$ defined by l_i (x)= **w.x**, where the subscript i takes 2^n different values corresponding to 2^n possible weighting vectors **w**.

Non-linearity: The non-linearity of a boolean function is formulated with equation (4.11)

$$N_f = \min_{\mathbf{w}, c} \#\{\mathbf{x} \in Z_2^n \mid f(\mathbf{x}) \neq \mathbf{w} \cdot \mathbf{x} \oplus c\}, \quad (4.11)$$

which can be stated as the minimum hamming distance of this function from an affine function. One can find this minimum distance by comparing the truth table of the boolean function to all rows of the Hadamard matrix. This definition of nonlinearity shown [38] to be equivalent to equation (4.12)

$$N_f = 2^{n-1} - \frac{1}{2} \max_{w=0,1,..,2^{n-1}} \left\{ \left| f_s \bullet (w \bullet x)_s \right| \right\} \qquad (4.12)$$

Walsh Transform: The walsh transform of a function $f(\mathbf{x})$ is defined as [21]:

$$F(\mathbf{w}) = \sum_{\mathbf{x} \in Z_2^n} (-1)^{f(\mathbf{x})} (-1)^{\mathbf{w}.\mathbf{x}} . \qquad (4.13)$$

Notice that for 2^n different values of the n bit vector $\mathbf{w}$, one obtains 2^n different linear functions

$$\boldsymbol{l}_w(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} \qquad (4.14)$$

and the walsh transform defined in (4.13) is nothing but the inner product of the sequences of $f(\mathbf{x})$ and $l_w(\mathbf{x})$:

$$F(\mathbf{w}) = \boldsymbol{f}_\mathbf{s} \,.\, \boldsymbol{l}_\mathbf{ws} = \boldsymbol{f}_\mathbf{s} \,.\, (\,\mathbf{w}.\mathbf{x}\,)_\mathbf{s} \qquad (4.15)$$

The walsh transform given by (4.13) takes integer values in the interval [-2^n , 2^n].

Bent Function: A function $f : Z_2^n \to Z_2$ is called a bent function [37] if,

$$2^{-\frac{n}{2}}\sum(-1)^{f(\mathbf{x})\oplus(\mathbf{w}\cdot\mathbf{x})} = \pm 1 \quad , \quad \text{for all } \mathbf{w} \in Z_2^n . \qquad (4.16)$$

Notice that the walsh transform defined by (4.13) can also be written as:

$$F(\mathbf{w}) = \sum_{\mathbf{x}\in Z_2^n}(-1)^{f(\mathbf{x})}(-1)^{\mathbf{w}.\mathbf{x}} = \sum_{\mathbf{x}\in Z_2^n}(-1)^{f(\mathbf{x})\oplus(\mathbf{w}\cdot\mathbf{x})}$$

Hence, one can express (4.16) in the form: $2^{-n/2}F(\mathbf{w}) = \pm 1$

So, the magnitude of the walsh transform for bent functions is found as:

$$\left|F(\mathbf{w})\right| = 2^{n/2}, \qquad \text{for all } \mathbf{w}. \qquad (4.17)$$

Bent functions only exist for even values of *n*.

4.2.2 Nonlinearity of *S*-boxes

Nonlinearity of the *S*-box can be defined in terms of nonlinearities of the individual components f_i which are the output bit functions of the *S*-boxes. The worst case nonlinearity over all output bit positions and their linear combinations; where the nonlinearity factor for each function $f_j : Z_2^n \to Z_2$ is defined by

$$N_{f_j} = 2^{n-1} - \frac{1}{2}\max_{i=1,\ldots,2^n} |\boldsymbol{f}_{j,\mathbf{s}} \cdot \boldsymbol{l}_{i,\mathbf{s}}| = 2^{n-1} - \frac{1}{2}\max |F_j(\mathbf{w})| \quad (4.18)$$

It was shown by Rothaus [25] that the class of perfectly nonlinear functions coincides with the class of bent functions. Using (4.17) and (4.18),

$$N_f \le 2^{n-1} - \frac{1}{2}\left|F(\mathbf{w})\right|_{Bent} = 2^{n-1} - 2^{(n/2)-1} \quad , \quad \text{for } n \text{ even} \qquad (4.19)$$

Rothaus [25] also showed that for odd values of n, there are no perfect nonlinear functions, and maximum nonlinearity is equal to $2^{n-1} - 2^{(n+1)/2}$. So,

$$N_f \leq 2^{n-1} - 2^{(n+1)/2}, \quad \text{for } n \text{ odd} \tag{4.20}$$

4.2.3 Nonlinearity Criterion

To calculate the nonlinearity of nxn S-boxes we have first found the truth tables of permutation boxes of the S-boxes. Each output bit has a truth table of 2^n bits. After obtaining n truth tables for n output bits, we find all 2^n truth tables corresponding to all 2^n linear combinations of the output bits. Each row of the truth table matrix is then compared to all rows of 2^nx2^n Sylvester-Hadamard matrix, to find the minimum distance. Nonlinearity values are obtained for each of the 2^n boolean functions. The smallest of all is the nonlinearity parameter of nxn S-boxes.

We find the 2^nx2^n truth table matrix with the following algorithm:

1. Define a boolean vector of $F = \{f_1, f_2, \ldots, f_n\}$ where f_x are the result bits of the S-boxes while $x = \{x_1, x_2, \ldots, x_n\}$ is the input vector, $0 < x < 2^n\text{-}1$

2. Define the boolean function to be $f(f_x) = a_1 \bullet f_1 \oplus a_2 \bullet f_2 \oplus \ldots \oplus a_8 \bullet f_8$, where $a = \{a_1, a_2, \ldots, a_n\}$ $0 < a < 2^n\text{-}1$

3. Use all available input x values to the permutation where the boolean vector is found and then by using this vector, the boolean functions truth table is found by using all available coefficient vectors, **a**.

Notice that, in the first row of the truth table, the coefficient vector $\boldsymbol{a}$ is equal to all zero which results to an all zero row. And in the first column the input vector $\boldsymbol{x}$ is equal to all zero.

In Chapter 5 the nonlinearity values for the *S*-boxes of Twofish are given after presentation of avalanche and AWD curves for RC5, RC6 and Twofish.

CHAPTER 5

EVALUATION RESULTS

In this chapter we give the evaluation results of the studied ciphers, RC5, RC6, and Twofish. The avalanche curves and avalanche weight distribution (AWD) curves are sketched and analyzed according to the steps given in Chapter 4. Together with resemblance parameter analysis, the nonlinearity of Twofish *S*-boxes is investigated.

5.1 Avalanche Characteristics of RC5 Cipher

5.1.1 Avalanche Curves of RC5 Cipher

The avalanche curves of RC5 are obtained by counting the number of changes at each position of the round output vector, when a specific plaintext bit at position i is complemented for a set of $N = 10000$ different plaintexts. The keyword is usually chosen as all-zero keyword, unless it is specified as something else. Other parameters, such as magic words, are also not changed and the same as the original code. For RC5

algorithm three intervals can be identified for the position of the complemented input bit, i.e. error bit where the avalanche behavior is similar. These intervals are found as $i \in [1..35], [36..40], [41..63]$.

In Fig. 5.1 (a) and Fig. 5.1 (b), avalanche curves of 1-round RC5 are given for different error bit positions at the input vector, i.e. the plaintext. For the first round in the interval of $i = 36,...,40$ the average number of changes in avalanche variable is more than 4970 (which is expected to be 5000 ideally) and this can be said to be a very good diffusion value, because it is in the 0.6% vicinity of the ideal value and maximum change is within 7%. On the other hand, in the other two regions, the average number of changes in avalanche variable is within 1100 and 1600, which is much more less than the desired value.

In Fig. 5.1 (c) and Fig. 5.1 (d), avalanche curves of 2-round RC5 are given for different error bit positions *i*. For the second round, the average number of changes in avalanche variable differs only 0.2% from the ideal value of 5000 in the interval of $i \in [36..40]$. Also in the other intervals the average avalanche value is improved to the range 3400-4200.

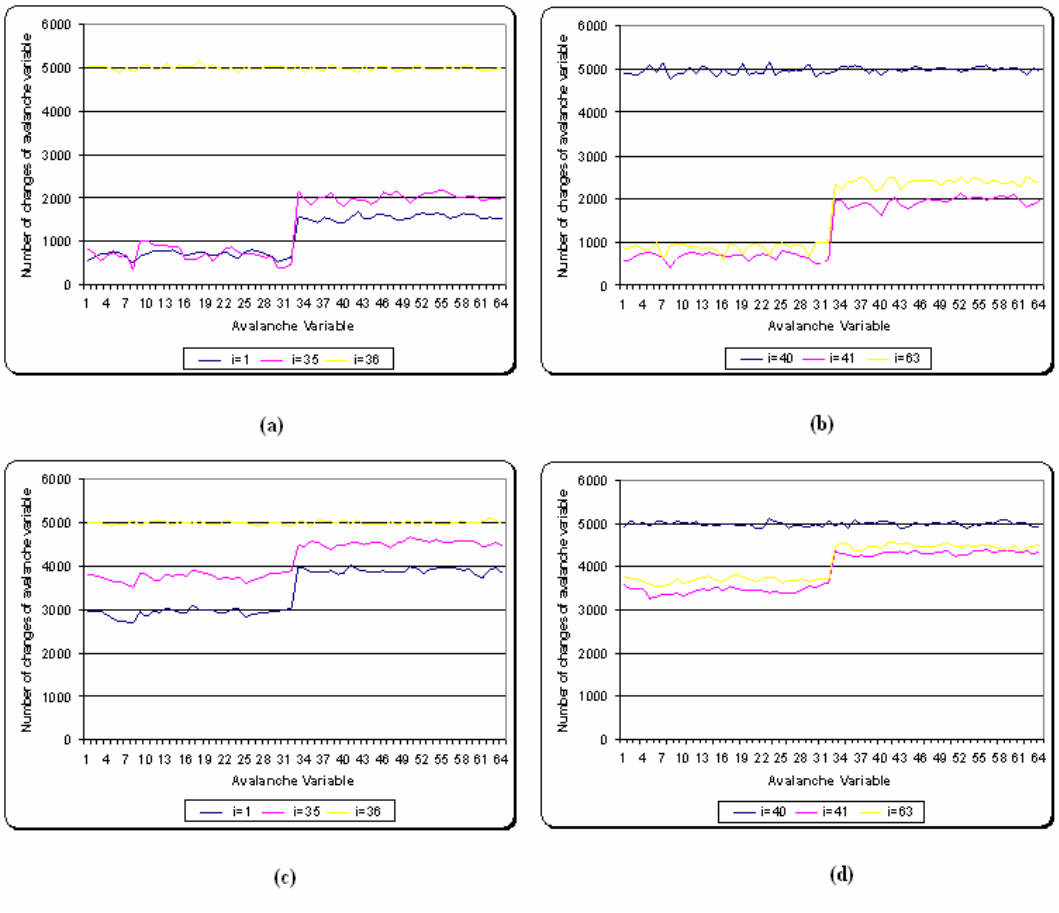

Figure 5.1 Avalanche curves of RC5 for the first and second round (r=1, r=2) and chosen error bit positions (i), which represents different cases

The avalanche curves for 3-rounds of RC5 are sketched in Fig. 5.2 (a) and Fig. 5.2 (b). As can be observed from the figures for all intervals the characteristics are improved and in the worst case, i.e. $i \in \{(1..35), (41..63)\}$, the number of change of avalanche variable is more than 4400.

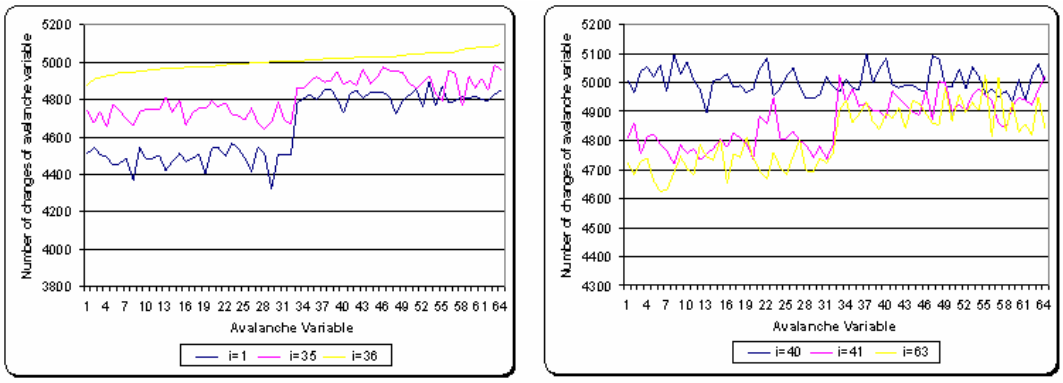

Figure 5.2 Avalanche curves of RC5 for the third round (r=3) and chosen error bit positions (i), which represents different cases

5.1.2 Avalanche Wight Distribution (AWD) Curves of RC5 Cipher

The AWD curves of RC5 are obtained by calculating the hamming weight of the round output vector, when a specific plaintext bit at position i if complemented for a set of N = 10000 different plaintexts. The keyword is usually chosen as all-zero keyword, unless it is specified as something else. Other parameters, such as magic words, are also not changed and same as the original code. The intervals explained in section 5.1.1 are used for the analysis.

In Fig. 5.3 (a) and Fig. 5.3 (b), AWD curves of 1-round RC5 are given for different error bit positions at the input vector, i.e. the plaintext. For the first round in the intervals $i \in \{(1..31), (41..63)\}$ the resemblance parameter (R^i) is below 0.045 and if a single bit of the plaintext is changed, first round of RC5 changes less than 31 bits

of the ciphertext and the change is mostly around 5 bits. Notice that the resemblance parameter R^i defined by (4.4) measures how close the avalanche weight distributions are to the ideal binomial curve, i.e., how random the avalanche variables are. For the first round in the interval $i \in [36..40]$ the resemblance parameter (R^i) is higher than 0.950 and if a single bit of the plaintext is changed, first round of RC5 changes less than 46 bits of the ciphertext and more importantly the change is mostly around 32 bits.

Notice that in the interval $i \in [36..40]$, the AWD curves are very close to the ideal curve given in Fig. 4.1. One can argue that bits in that region are not suitable for differential cryptanalysis based attacks.

In Fig. 5.3 (c) and Fig. 5.3 (d), AWD curves of 2-rounds of RC5 are given for different error bit positions i at the input vector, i.e., the plaintext. For the first round in the intervals $i \in \{(1..31), (41..63)\}$ the resemblance parameter (R^i) is below 0.680 and if a single bit of the plaintext is changed, first round of RC5 changes less than 46 bits of the ciphertext and the change is mostly around 25 bits. Besides, for the first round in the interval $i \in [36..40]$ the resemblance parameter (R^i) is higher than 0.980 and if a single bit of the plaintext is changed, first round of RC5 changes less than 48 bits of the ciphertext and more importantly the change is mostly around 32 bits.

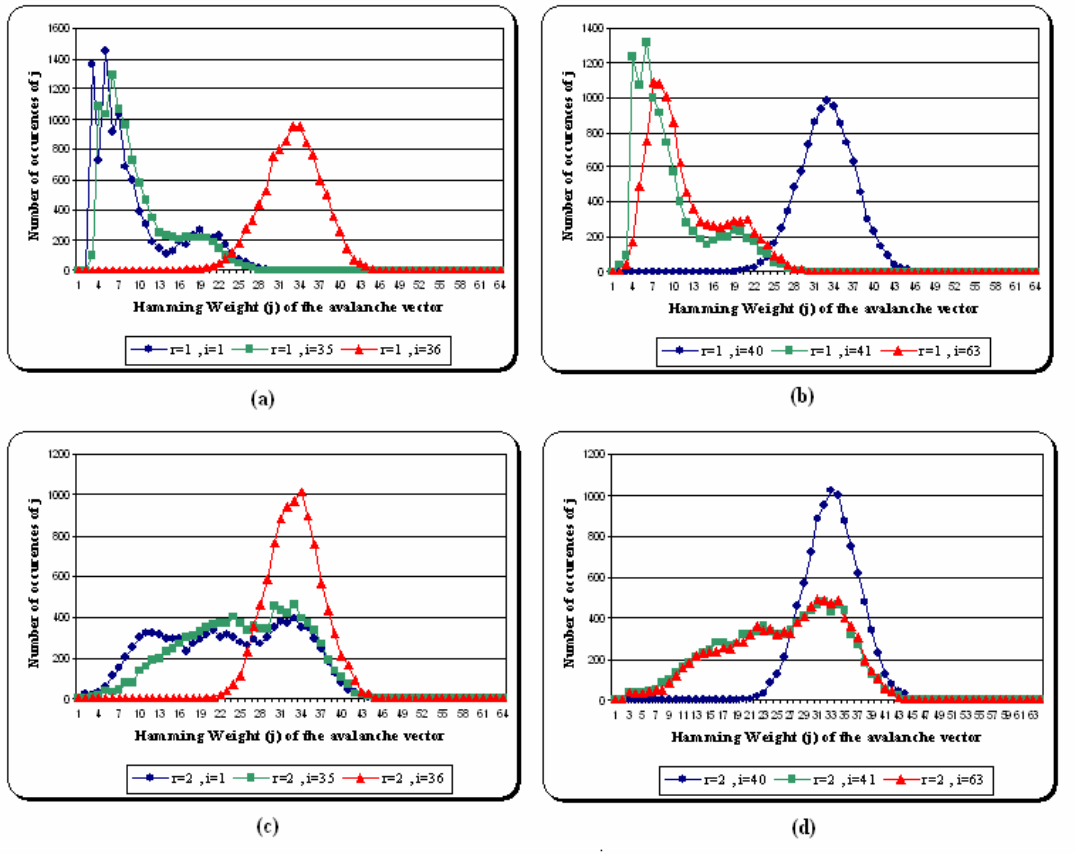

Figure 5.3 Avalanche weight distribution curves of RC5 for the first and second round (r=1, r=2) and chosen error bit positions (i), which represents different cases

In Fig. 5.4 (a) and Fig. 5.4 (b) AWD curves of 3-rounds RC5 are given for different error bit positions i at the input vector. In either interval resemblance parameter (R^i) is higher than 0.850 but on the other hand the difference of the histograms sketched in the predefined intervals can be observed clearly. With the start of third round the AWD curves become more similar to the ideal curve in all intervals

and it is clear that cryptanalysts need much more given plaintexts in known-plaintext based attacks.

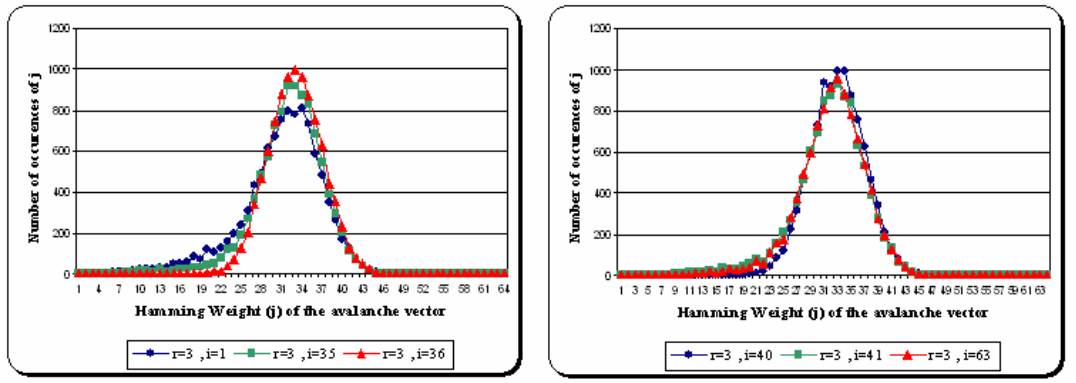

Figure 5.4 Avalanche weight distribution curves of RC5 for the third round (r=3) and chosen error bit positions (i), which represents different cases

5.1.3 Resemblance Parameters for RC5 Cipher

Resemblance parameters (R^i) of RC5 are obtained by finding the absolute difference between the AWD curves of RC5 and the binomial distribution. Indeed resemblance parameter variations according to different bit positions i reveals the intervals that have same avalanche characteristics. The histograms are sketched for first, second and third round of RC5 Cipher. As can be seen from the figures below: the characteristics of RC5 Cipher gives better results within the interval $i \in 36,...,40$. The curves are sketched according to the steps given in Chapter 4. All of the curves are sketched with the parameters N = 10000 (number of sample plaintexts), all-zero keyword, 64 bits of plaintext and 64 bits of keyword.

Notice that the interval on Y-axis of the Fig. 5.5 (c) and Fig. 5.5 (d) are different from the others to focus on the characteristic detailed.

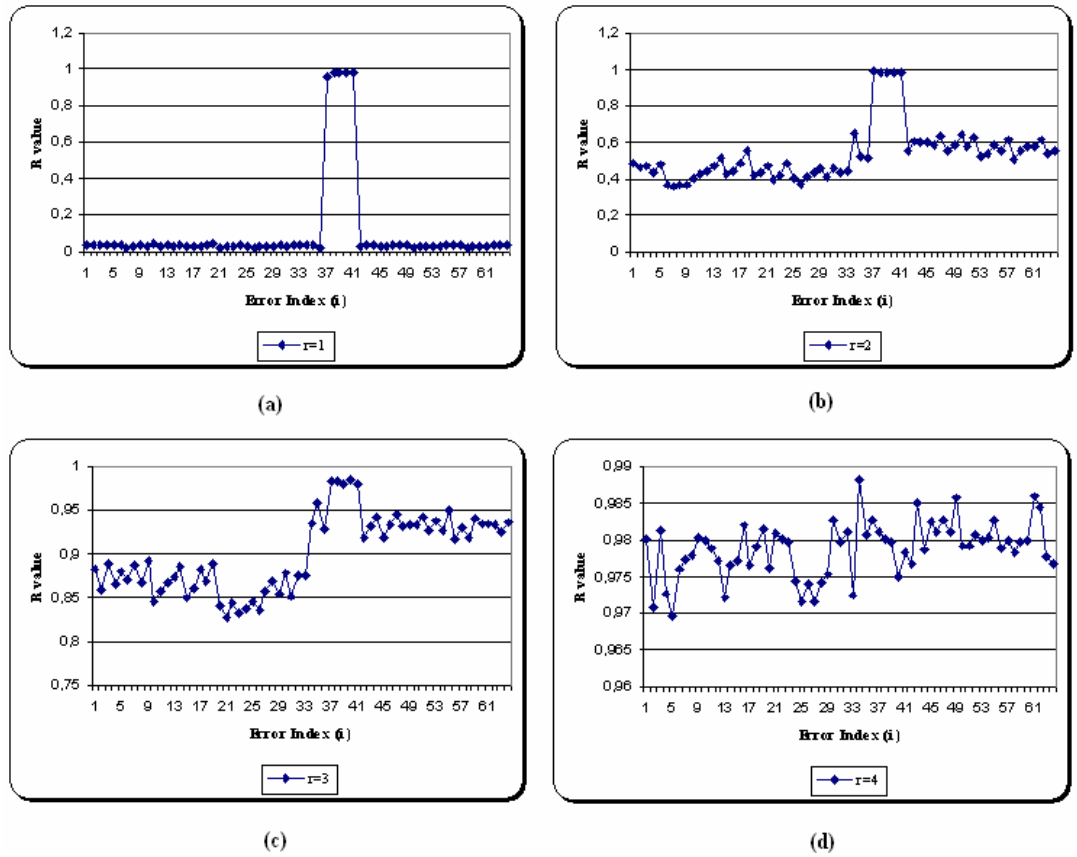

Figure 5.5 Resemblance parameter curves of RC5 for different rounds (*r*): a) *r*=1 b) *r*=2 c) *r*=3 d) *r*=4

As can be seen from the above figures after 4 rounds RC5 algorithm becomes invulnerable to the 1-bit changes in the plaintext values, i.e. R^i value is lower than 0.02 . But there is an important issue to be noted when the figures are investigated

detailed, there appears an extreme point that when the bit positions i=36, 37, 38, 39, 40 without looking at the round number, R^i value is near to one. But nevertheless we can conclude that RC5 Cipher achieves acceptable and desired diffusion after the fourth round.

5.2 Avalanche Characteristics of RC6 Cipher

5.2.1 Avalanche Criterion for RC6 Cipher

The avalanche curves of RC6 are obtained by counting the number of changes at each position of the round output vector, when a specific plaintext bit at position i if complemented for a set of N = 10000 different plaintexts. The keyword is manually chosen as all-zero keyword, unless it is specified as something else. Other parameters, such as magic words, are also not changed and the same as the original code. For RC6 algorithm four intervals can be identified for the position of the complemented input bit, i.e. error bit where the avalanche behavior is similar. These are intervals are found as $i \in$ [1..31], [32..63], [64..95], [96..127].

In Fig. 5.6 (a) and Fig. 5.6 (b), the avalanche curves of 1-round RC6 are given for error bit positions i at the input vector, i.e., the plaintext. For the first round in the intervals $i \in$ {(32..63), (96..127)} the avalanche curves are better than the other intervals which will be the starting point of selecting these intervals. The average change in avalanche vectors in these regions are more than 10 times better than the values in the interval $i \in$ {(1..31), (64..95)}.

In Fig. 5.6 (c) and Fig. 5.6 (d), the avalanche curves of RC6 for second round with different error bit positions i at the input vector are given. In the intervals $i \in \{(32..63), (96..127)\}$ RC6 has better avalanche characteristics than the other regions as expected from the first round. In these regions the average of change of avalanche variables is 4100 and 4700 respectively.

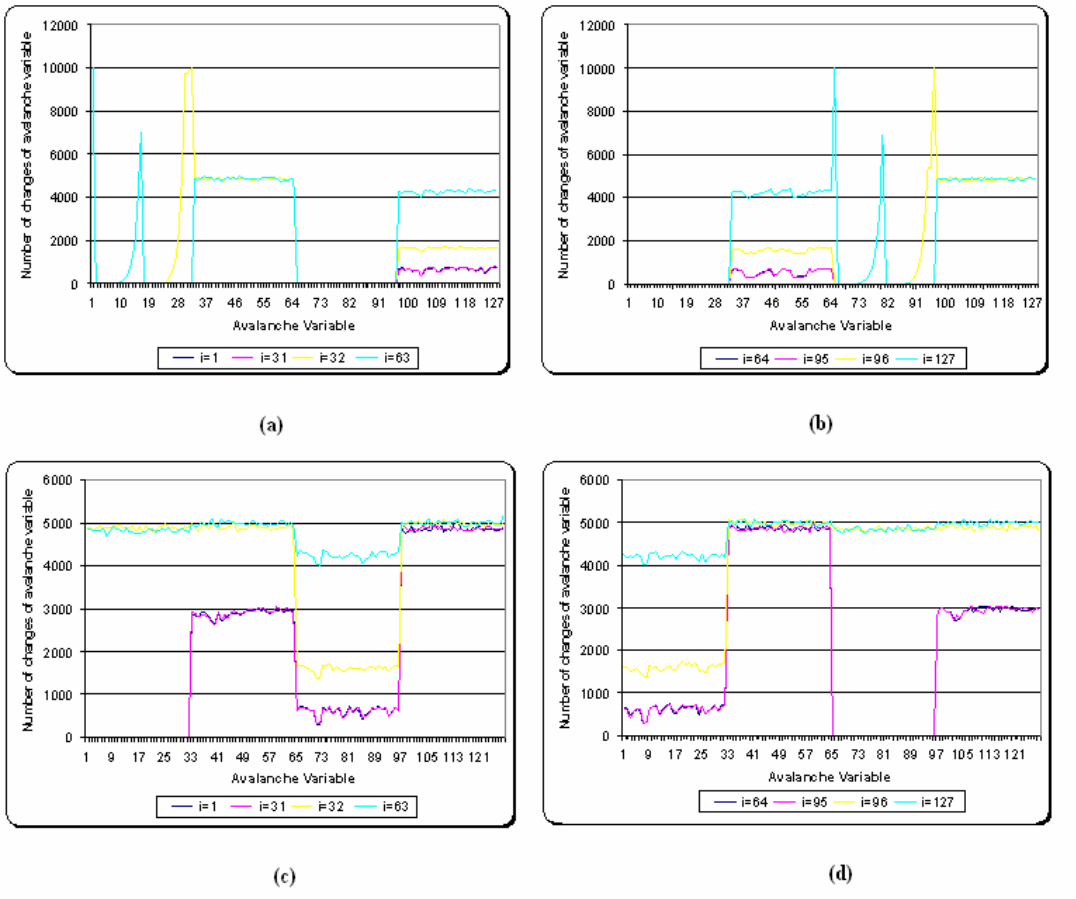

Figure 5.6 Avalanche curves of RC6 for the first and second round (r=1, r=2) and chosen error bit positions (i), which represents different cases

In Fig. 5.7 (a) and 5.7 (b), avalanche curves for 3-rounds of RC6 cipher for different error bit positions *i* at the input vector are sketched. The number of change of avalanche variables is improved by increasing the number of rounds and the average number of change of avalanche variable is more than 4450 for all intervals. Additionally in the intervals $i \in \{(32..63), (96..127)\}$ the average number of change of avalanche variable is varying between 4960 and 5000.

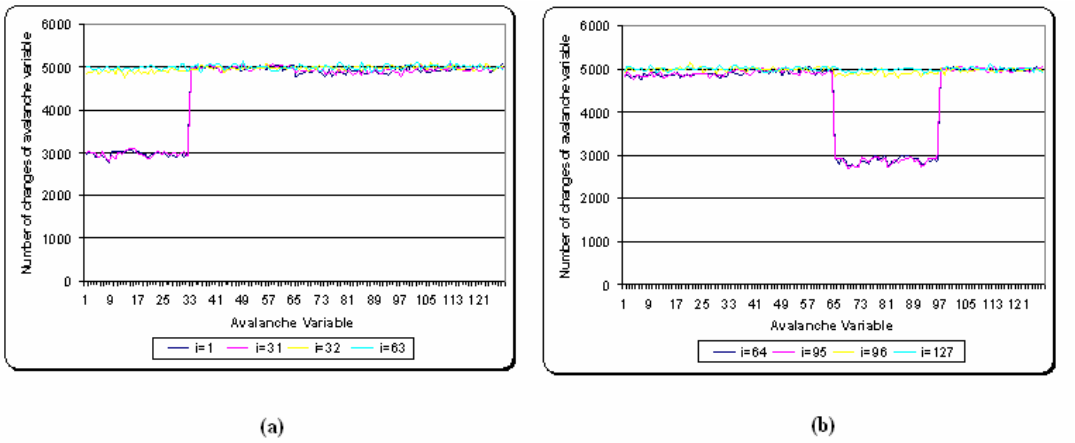

Figure 5.7 Avalanche curves of RC6 for the third round (r=3) and chosen error bit positions (i), which represents different cases

5.2.2 Avalanche Wight Distribution (AWD) Curves of RC6 Cipher

The AWD curves of RC6 are obtained by calculating the hamming weight of the round output vector, when a specific plaintext bit at position *i* if complemented for a set of N = 10000 different plaintexts. The keyword is manually chosen as all-

zero keyword, unless it is specified as something else. Other parameters, such as magic words, are also not changed and same as the original code. The intervals explained in section 5.2.1 are used for the analysis.

In Fig. 5.8 (a) and Fig. 5.8 (b), AWD curves of 1-round RC6 are given for different error bit positions *i* at the input vector. For the first round in the intervals of $i \in \{(1..31), (64..95)\}$ the resemblance parameter (R^i) is below 0.001 and if a single bit of the plaintext is changed, first round of RC6 changes less than 10 bits of the ciphertext but the change is mostly around 3 bits. Besides, for the first round in the intervals of $i \in \{(32..63), (96..127)\}$ the resemblance parameter (R^i) is less than 0.004 and if a single bit of the plaintext is changed, first round of RC6 changes less than 48 bits of the ciphertext and more importantly the change is varies around 25 to 32 bits.

In Fig. 5.8 (c) and Fig. 5.8 (d) AWD curves of 2-rounds of RC6 are given for different error bit positions *i* at the input vector. For the second round in the intervals of $i \in \{(1..31), (64..95)\}$ the Resemblance Parameter (R^i) is below 0.0015 and if a single bit of the plaintext is changed, first round of RC6 changes less than 50 bits of the ciphertext and the change is mostly around 28 bits. Besides, for the first round in the intervals of $i \in \{(32..63), (96..127)\}$ the resemblance parameter (R^i) varies from 0.280 to 0.980 and if a single bit of the plaintext is changed, 2-rounds of RC6 changes less than 82 bits of the ciphertext and more importantly the change is varies around 55 to 63 bits. There is significant performance improvements in the histograms but the difference between the intervals can be observed clearly. Notice that between the

intervals $i \in \{(32..63), (96..127)\}$ AWD curves become very similar to the desired curve given in Fig. 4.1.

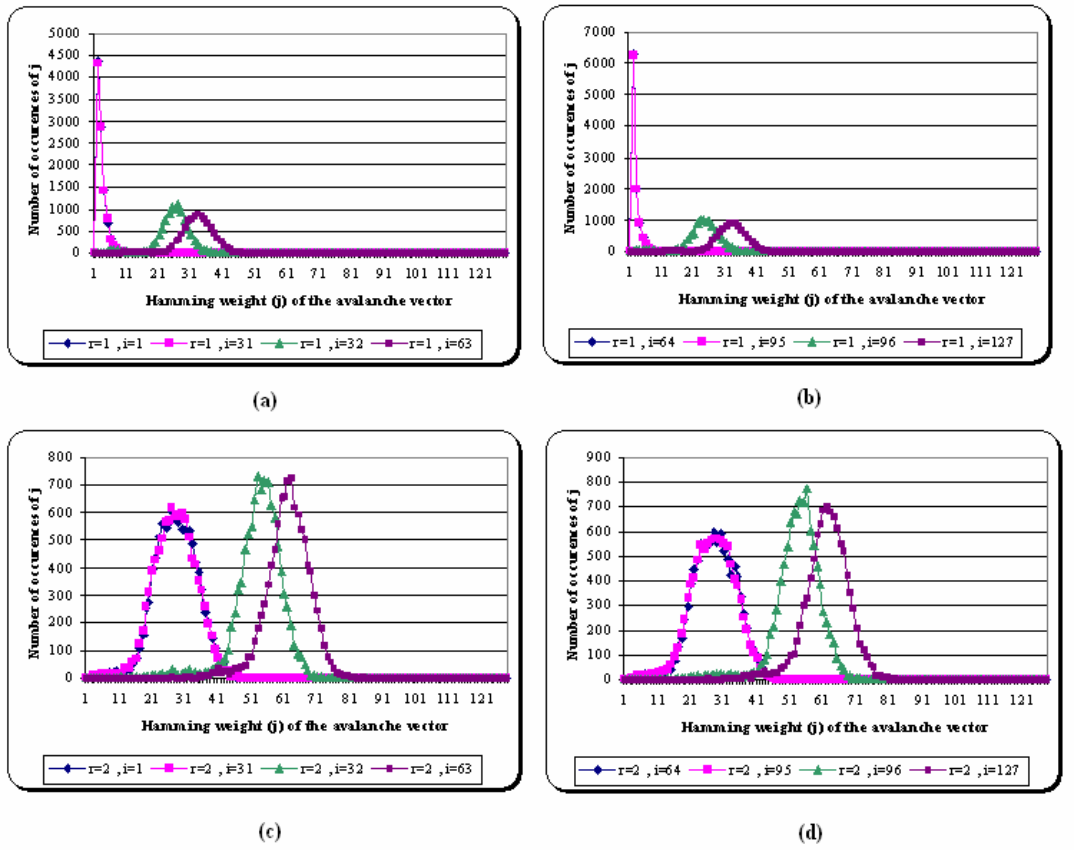

Figure 5.8 Avalanche weight distribution curves of RC6 for the first and second round (r=1, r=2) and chosen error bit positions (i), which represents different cases

In Fig. 5.9 (a) and Fig. 5.9 (b) AWD curves of 3-rounds of RC6 are given for different error bit positions i at the input vector. For the third round in the intervals of

$i \in \{(1..31), (64..95)\}$ the resemblance parameter (R^i) is below 0.600 and if a single bit of the plaintext is changed, third round of RC6 changes less than 85 bits of the ciphertext but the change is mostly around 58 bits. Besides, for the third round in the intervals of $i \in \{(32..63), (96..127)\}$ the resemblance parameter (R^i) is higher than 0.004 and if a single bit of the plaintext is changed, 3-rounds of RC6 changes less than 88 bits of the ciphertext and more importantly the change is varies around 64 bits.

It can be observed that between the intervals $i \in \{(32..63), (96..127)\}$ the AWD curves of RC6 give the desired diffusion characteristics, on the other hand this result can not be observed in the other intervals and if it is to be compared with RC5 it is clear that RC5 has better avalanche characteristics within the same round.

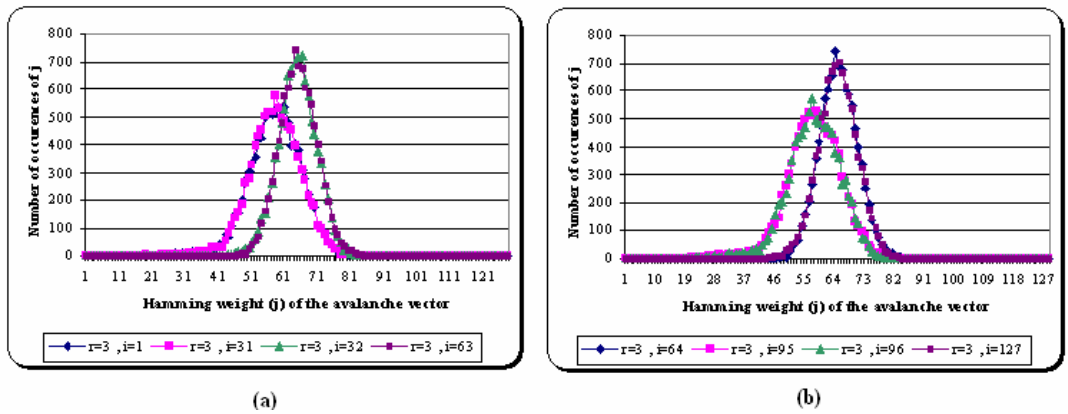

Figure 5.9 Avalanche weight distribution curves of RC6 for the third round (r=3) and chosen error bit positions (i), which represents different cases

5.2.3 Resemblance Parameter for RC6 Cipher

Resemblance parameters (R^i) described by (4.4) are obtained by finding the absolute difference between the AWD curves of RC6 and the binomial distribution, as shown in (4.3). Indeed resemblance parameter variations according to different bit positions i reveals the intervals that have the same avalanche characteristics. The histograms are sketched for the first, second, third and fourth rounds of RC6 Cipher. As can be seen from the figures below: the characteristics of RC6 Cipher gives better results within the interval $i \in \{(32..63), (96..127)\}$. The AWD curves are sketched according to the steps given in Chapter 4. All of the curves are sketched with the parameters $N = 10000$ (number of sample plaintexts), all-zero keyword, 128 bits of plaintext and 128 bits of keyword.

Notice that the interval on Y-axis of Fig. 5.10 (a) and Fig. 5.10 (d) is different from the others to focus on the characteristic detailed.

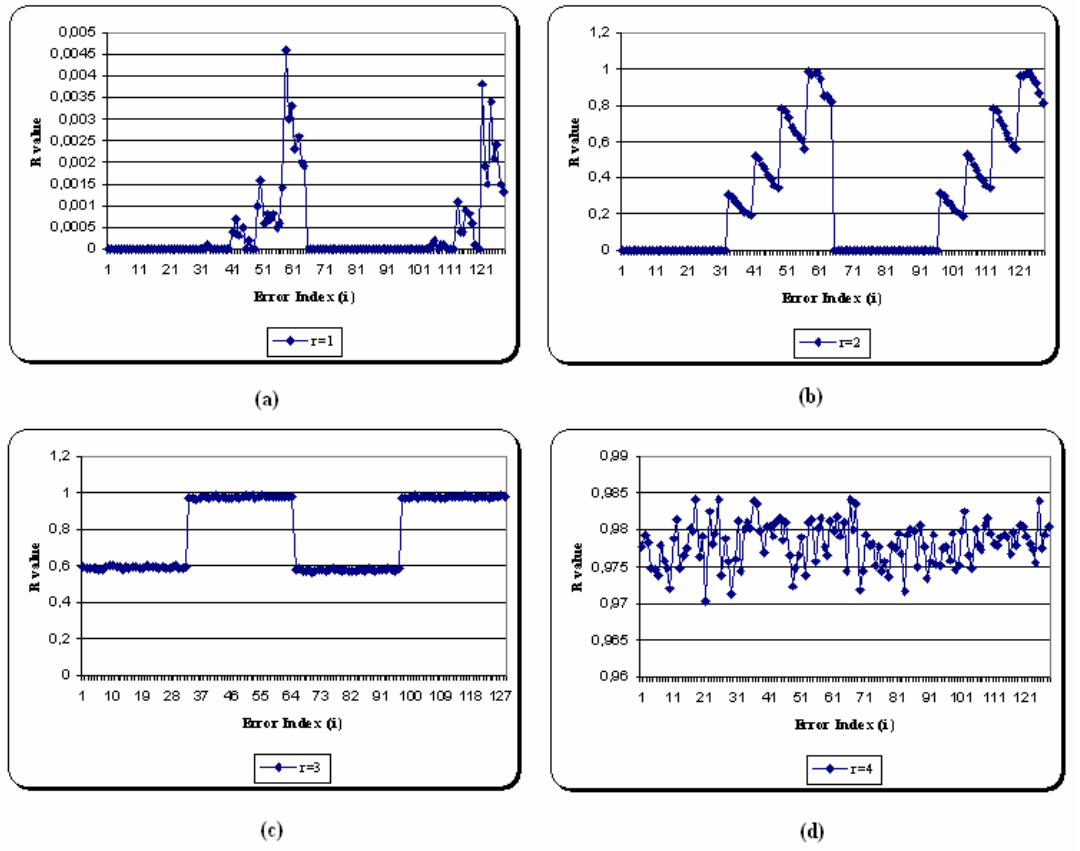

Figure 5.10 Resemblance parameter curves of RC6 for different rounds (*r*): a) *r*=1 b) *r*=2 c) *r*=3 d) *r*=4

As can be seen from the above figures after 4 rounds RC6 algorithm becomes invulnerable to the 1-bit changes in the plaintext values, i.e. R^i value is nearly equal to 1.

There is an important issue to be noted when the figures are investigated detailed, for 2-rounds of RC6 the R^i curve is such stepped-saw tooth waveform between the intervals $i \in \{(32..63), (96..127)\}$ and for 3-rounds of RC6 in these

intervals the R^i value is nearly equal to 1 where for the other intervals R^i value is equal to nearly 0.55. This shows that the bits in the interval intervals $i \in \{(32..63), (96..127)\}$ are less vulnerable to diffusion based cryptanalytic attacks. Indeed this result can also be revealed by examining the encryption algorithm of RC6.

It is clear that these bits correspond to the bits of *A* and *C* which are two of the four *w*-bit registers. And also the *B* and *D* registers are not too much changed in the encryption algorithm, only operation is the subtraction of secret key register's 0^{th} and 1^{st} indexed values from these registers. Furthermore the operations, quadratic function and data-dependent shifting, which increase the complexity of algorithm are not applied to the registers *B* and *D*. On the other hand the registers *A* and *C* are exposed to data-dependent shifting where the value of shift is found by quadratic function.

5.3 Avalanche Criteria and Derivations Applied to Twofish Cipher

5.3.1 Avalanche Criterion for Twofish Cipher

In this section, the avalanche characteristics of Twofish cipher are investigated and avalanche curves are sketched according to the steps given in Chapter 4. The avalanche curves of Twofish are obtained by counting the number of changes at each position of the round output vector, when a specific plaintext bit at position *i* if complemented for a set of $N = 10000$ different plaintexts. The keyword is manually chosen as all-zero keyword, unless it is specified as something else. For Twofish algorithm tow intervals can be identified for the position of the complemented input

bit, i.e. error bit where the avalanche behavior is similar. These are intervals are found as $i \in [1..63], [64..127]$.

While sketching these curves it is observed that the number of average change of avalanche variable of Twofish cipher is very near to zero so there is no need to sketch the characteristics for the first round.

In Fig. 5.11 (a), Fig. 5.11 (b), Fig. 5.11 (c), Fig. 5.11 (d) the avalanche curves of 2-round Twofish cipher are sketched for different error positions *i* at the input vector. The average number of change of avalanche variable 2500 and 5000 in intervals [1..63] and [64..127] respectively. This is said to be a better result from RC6 Cipher because almost 64 avalanche variables has number of changes near to ideal.

In Fig. 5.12 (a), Fig. 5.12 (b), Fig. 5.12 (c), Fig. 5.12 (d), the avalanche curves for 3-round Twofish cipher are sketched. Its waveform is similar to 2-round Twofish.

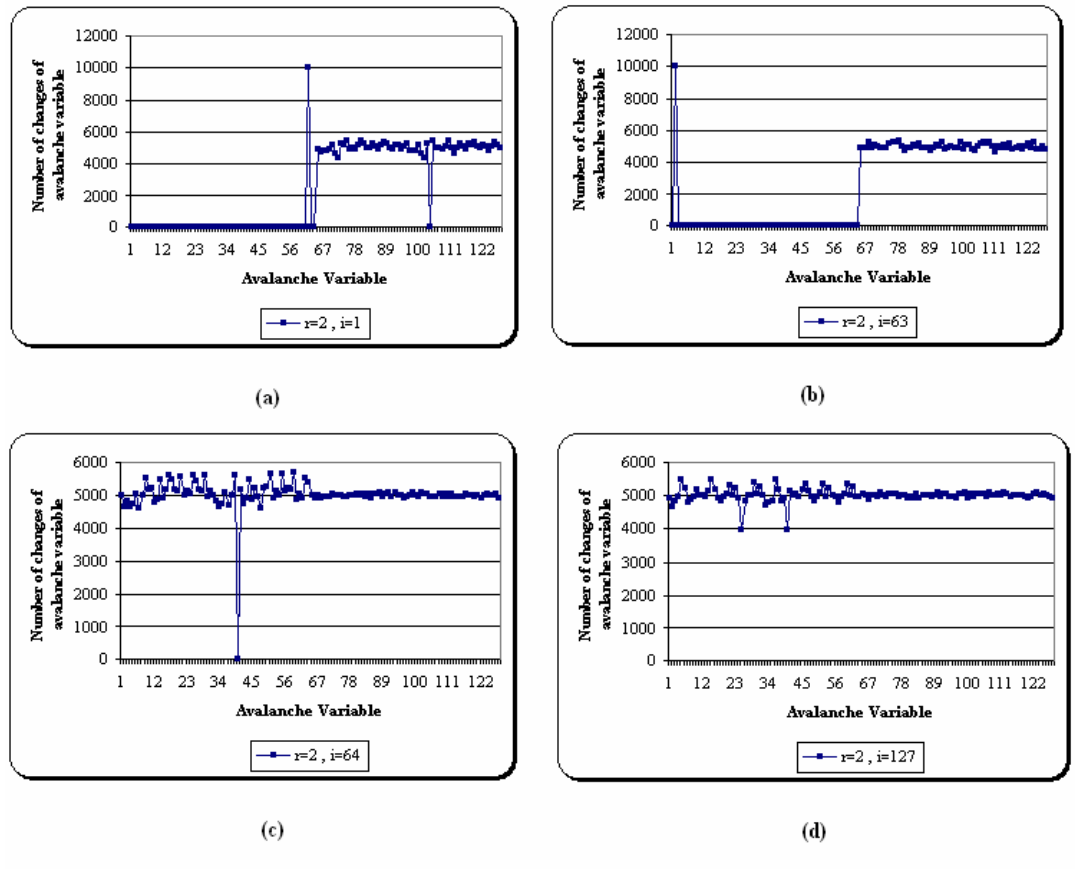

Figure 5.11 Avalanche curves of Twofish for the second round (r=2) and chosen error bit positions (i), which represents different cases

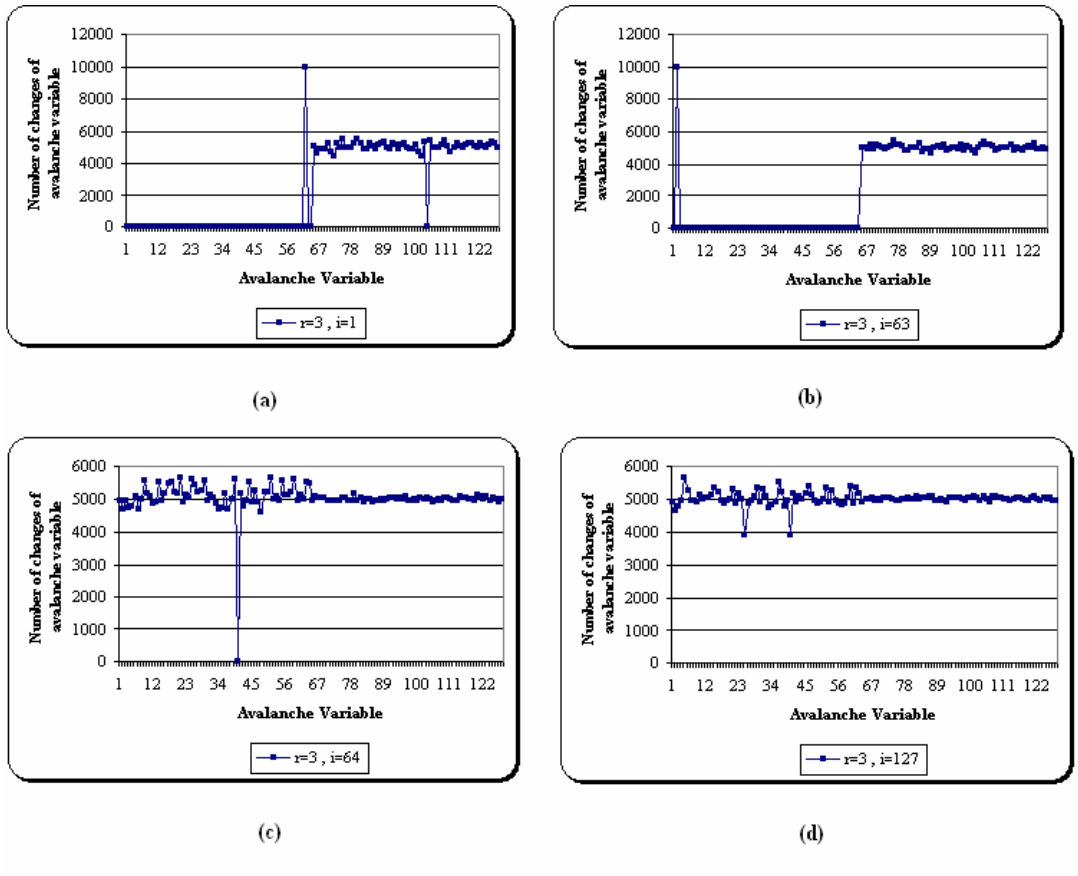

Figure 5.12 Avalanche curves of Twofish for the third round (r=3) and chosen error bit positions (i), which represents different cases

5.3.2 AWD Test for Twofish Cipher

The AWD curves of Twofish are obtained by calculating the hamming weight of the round output vector, when a specific plaintext bit at position i if complemented for a set of N = 10000 different plaintexts. The keyword is manually chosen as all-

zero keyword, unless it is specified as something else. The intervals explained in section 5.2.1 are used for the analysis.

Differing from other studied cipher algorithms for the first round of Twofish the AWD curves resemble to a line of origin 0 so there is no need to sketch the figure. On the other hand, as shown in Fig. 5.13 (a), 2-rounds Twofish Cipher has better performance than the others when observed in whole 128-bit wide.

In Fig. 5.13 (a) and Fig. 5.13 (b) AWD curves of 2-rounds and 3-rounds of Twofish are given for different error bit positions *i* at the input vector. For the second and third rounds in the interval of [1..63] the resemblance parameters (R^i) are below 0,0025 and if a single bit of the plaintext is changed, first round of Twofish changes less than 51 bits of the ciphertext and the change is mostly around 34 bits. Besides, for the second and third rounds in the interval of [64..127] the resemblance parameter (R^i) is higher than 0.920 and if a single bit of the plaintext is changed, second and third rounds of Twofish changes less than 88 bits of the ciphertext and more importantly the change is varies around 64 bits.

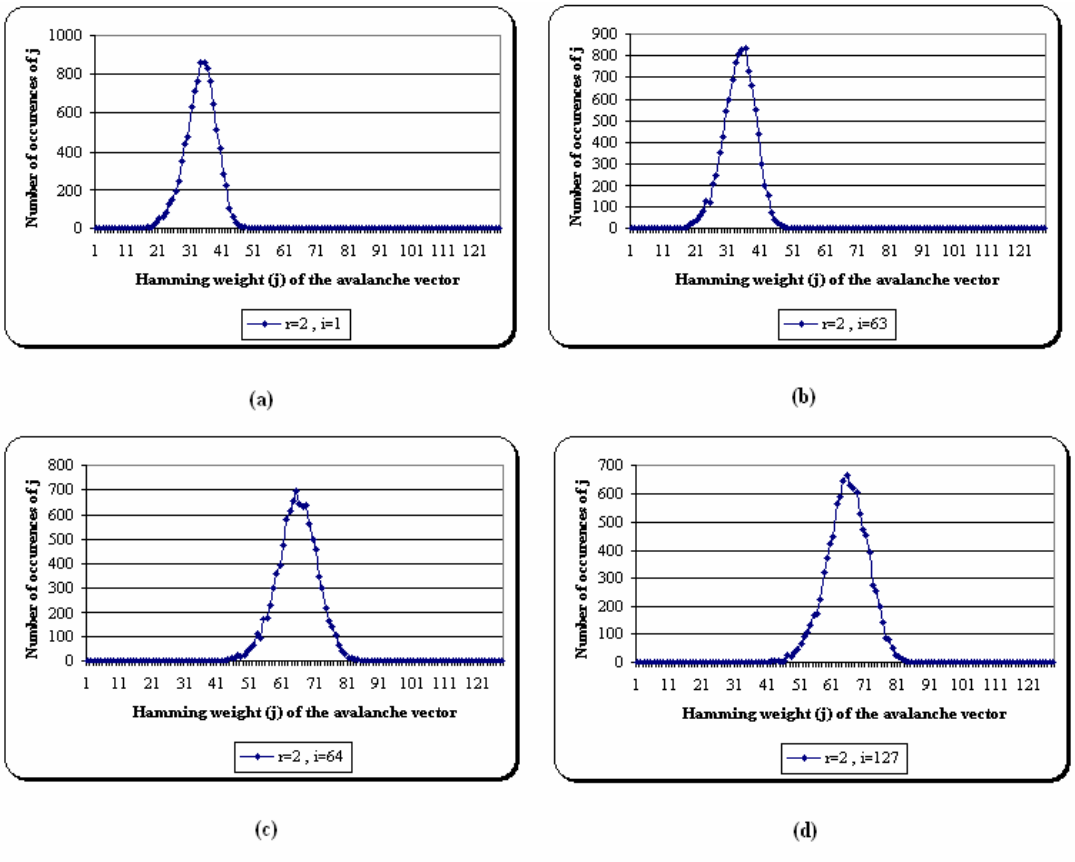

Figure 5.13 Avalanche weight distribution curves of Twofish for second round (r=2) and chosen error bit positions (i), which represents different cases

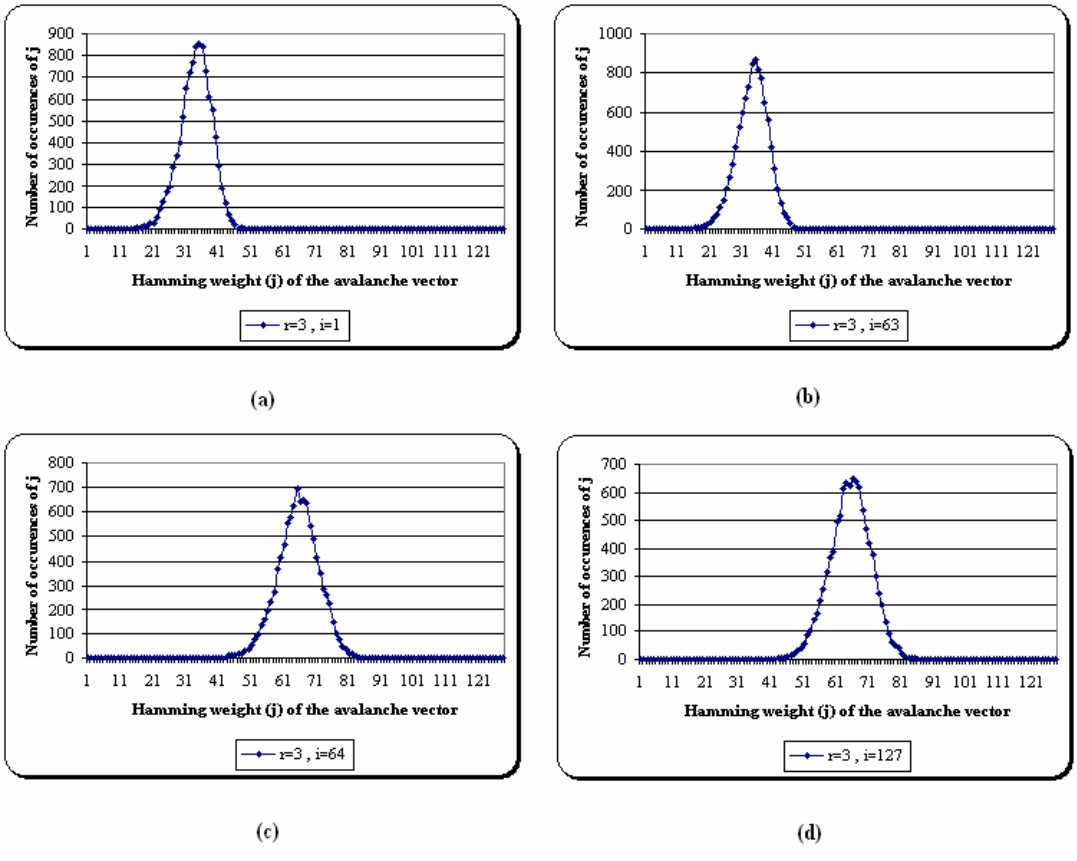

Figure 5.14 Avalanche weight distribution curves of Twofish for third round (r=3) and chosen error bit positions (i), which represents different cases

5.3.3 Resemblance Parameter Analysis Applied to Twofish Cipher

An avalanche criteria derivative, resemblance parameters (R^i) of Twofish are obtained by finding the absolute difference between the AWD curves of Twofish and the binomial distribution. Indeed resemblance parameter variations according to different bit positions i reveals the intervals that have same avalanche characteristics.

The histograms are sketched for second, third and fourth rounds of Twofish Cipher. As can be seen from the figures below: the characteristics of Twofish gives better results within the interval $i \in (64..127)$. The AWD curves are sketched according to the steps given in Chapter 4. All of the curves are sketched with the parameters N = 10000 (number of sample plaintexts), all-zero keyword, 128 bits of plaintext and 128 bits of keyword.

As in RC5 Cipher these curves are the origin of the idea that we should investigate the AWD curves for different regions.

Notice that the interval on Y-axis of Fig. 5.15 (c) is different from the others to focus on the characteristic detailed.

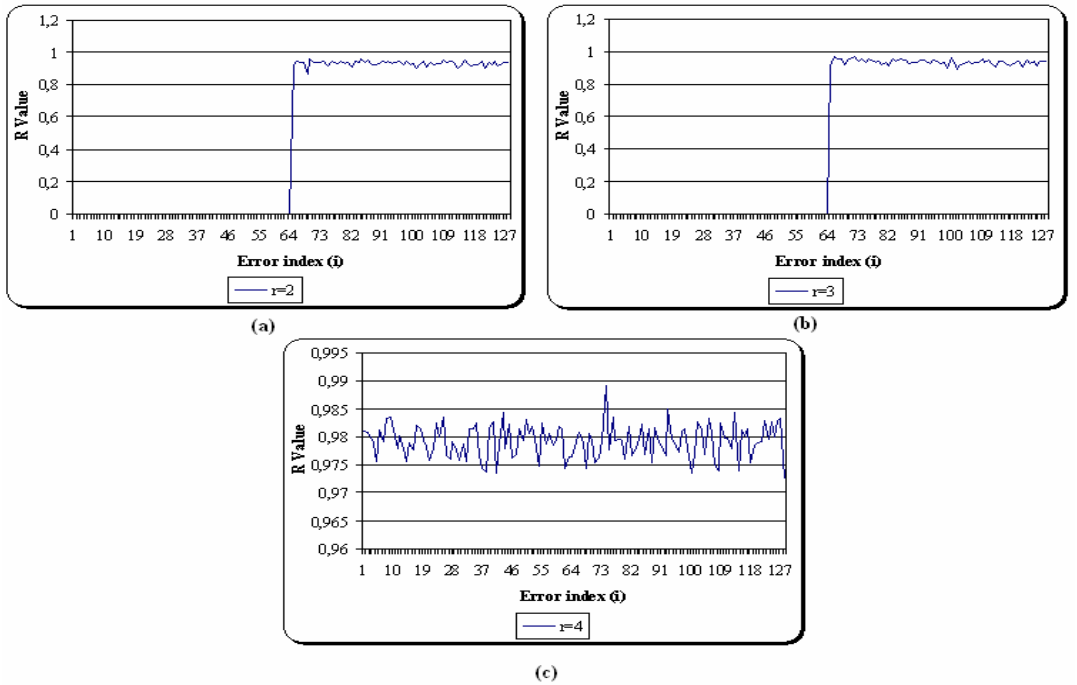

Figure 5.15 Resemblance parameter curves of Twofish for different rounds (r): a) r=2 b) r=3 c) r=4

As can be seen from the above figures after 4 rounds Twofish algorithm becomes invulnerable to the 1-bit changes in the plaintext values, i.e. R^i value is nearly equal to 1.

But there is an important issue to be noted when the figures are investigated detailed, for 2-rounds and 3-rounds the R^i values rises from around 0 to around 1 after the index 63. This shows that these bits are less vulnerable to diffusion based cryptanalytic attacks. Indeed this result can also be revealed by examining the encryption algorithm of Twofish. In each round the value of the first 2 words are the same with the previous last 2 words. This fact is clear from Fig. 2.3.

The resemblance parameter of the AWD curves of RC5, RC6 and Twofish to ideal binomial curve are summarized in the following tables.

Table 5.1 Resemblance parameter (R^i) of RC5 for different error vector bits (i)

Location of the plaintext change, *i*	**Range of R^i at the 1st round**	**Range of R^i at the 2nd round**	**Range of R^i at the 3rd round**
1-35	0,0220-0,0380	0,4450-0,5220	0,8550-0,9200
36-40	0,9820-09690	0,9780-0,9850	0,9860
41-64	0,029-0,0400	0,552-0,555	0,9200-0,9310

Table 5.2 Resemblance parameter (R^i) of RC6 for different error vector bits (i)

Location of the plaintext change, *i*	Range of R^i at the 1st round	Range of R^i at the 2nd round	Range of R^i at the 3rd round
1-35	0	0,0010-0,0020	0,5840-0,5990
36-63	0.0001-0,0030	0,309-0,815	0,9740-0,9770
64-95	0	0,0010-0,0020	0,571-0,574
96-128	0,0001-0,0030	0,182-0,682	0,9730-0,9780

Table 5.3 Resemblance parameter (R^i) of Twofish for different error vector bits (i)

Location of the plaintext change, *i*	Range of R^i at the 1st round	Range of R^i at the 2nd round	Range of R^i at the 3rd round
1-63	0	0,0010-0,0025	0,0010-0,0025
64-127	0	0,9250-0,9450	0,8650-0,9280

5.4 Nonlinearity Measure of Twofish Cipher

Nonlinearity value of Twofish is found by finding the minimum distances between all affine functions and 2^n possible linear combinations of the output bits. In this section the graphical results of nonlinearity criterion are given for the 8x8 *S*-boxes of Twofish Cipher. Also the nonlinearity values of permutation boxes of

Twofish are calculated because these permutation boxes form the "heart" of the *S*-boxes (refer to Fig. 2.5). The two boxes q_0 and q_1 are simple 8 by 8 permutations. Their algorithms are the same but only their look-up tables given by (2.9) and (2.10) are different from each other. Although one may think that such small difference in the lookup tables does not affect nonlinearity values much; this is not the case, and the nonlinearity of q_0 is found as 82, whereas the nonlinearity of q_1 is 72. The *S*-boxes of the Twofish algorithm which employ q_0 and q_1 have key-dependent coefficients as indicated by the elements $l_{i,j}$ in (2.7). So nonlinearity values of *S*-boxes are calculated for 100 random keywords to examine the effect of the keywords. After evaluating the nonlinearity values of the 8x8 *S*-boxes of Twofish, the distribution of the nonlinearity values for 100 keywords corresponding to 100 random choices of the coefficients $l_{i,j}$ in (2.7) is sketched.

In Fig. 5.16 (a), Fig. 5.16 (b), Fig. 5.16 (c), Fig. 5.16 (d), the nonlinearity distributions of *S*-boxes of Twofish are given. Although the number of occurrences of nonlinearity values is different from each other the curves are similar to each other and the average of non-linearity values is almost same for different keywords.

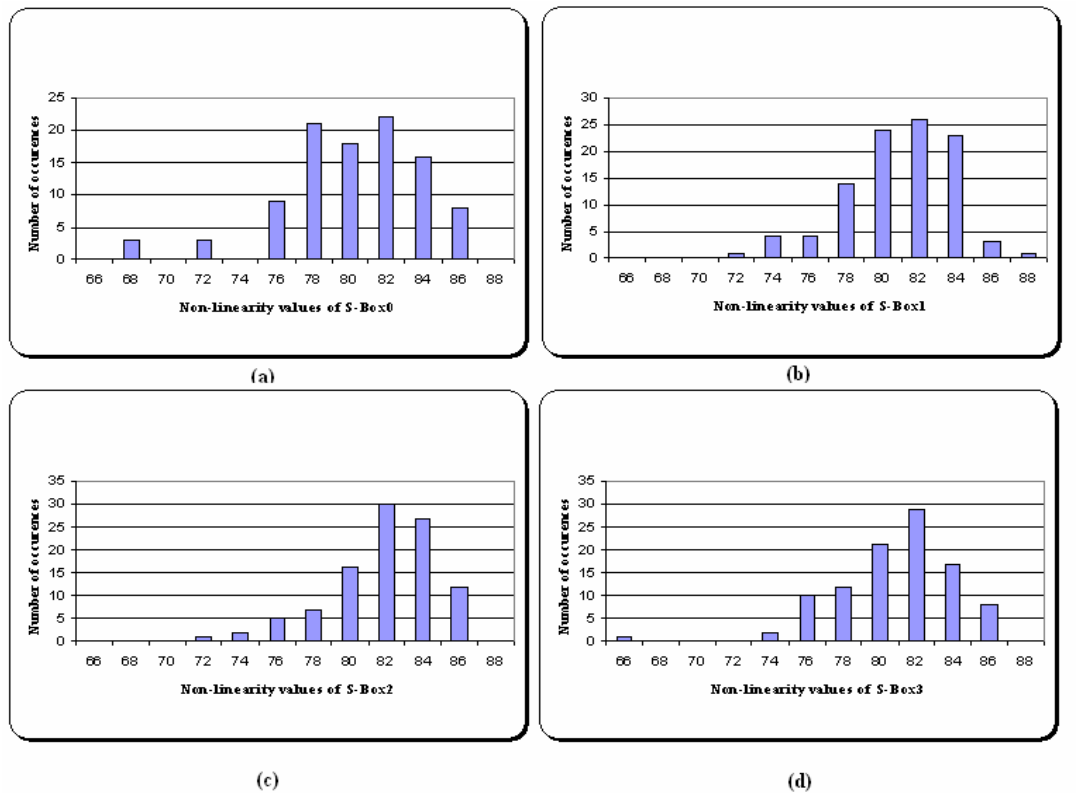

Figure 5.16 Nonlinearity of *S*-boxes of Twofish: a) *S*-box0 b) *S*-box1 c) *S*-box2 d) *S*-box3

After sketching the nonlinearity values of *S*-boxes of Twofish individually, the distribution of total nonlinearity values of the four *S*-boxes is sketched over 400 random keywords. By that observation, the effect of a single keyword on all *S*-boxes is investigated and the question whether or not the *S*-boxes compensate each others' nonlinearity values is tried to be answered. In this experiment the total nonlinearity of *S*-boxes of Twofish for the same keyword is found. The aim is to find weak keys that cause the minimum nonlinearity value while the nonlinearity values of *S*-boxes of

Twofish are summed for each keyword. In Fig.5.17 through Fig. 5.19 the non-linearity distribution of four *S*-boxes is given for three set of 400 random keywords. The total nonlinearity values of the *S*-boxes are divided by four to find the average to observe the similarities between the nonlinearity values of individual *S*-boxes more clearly.

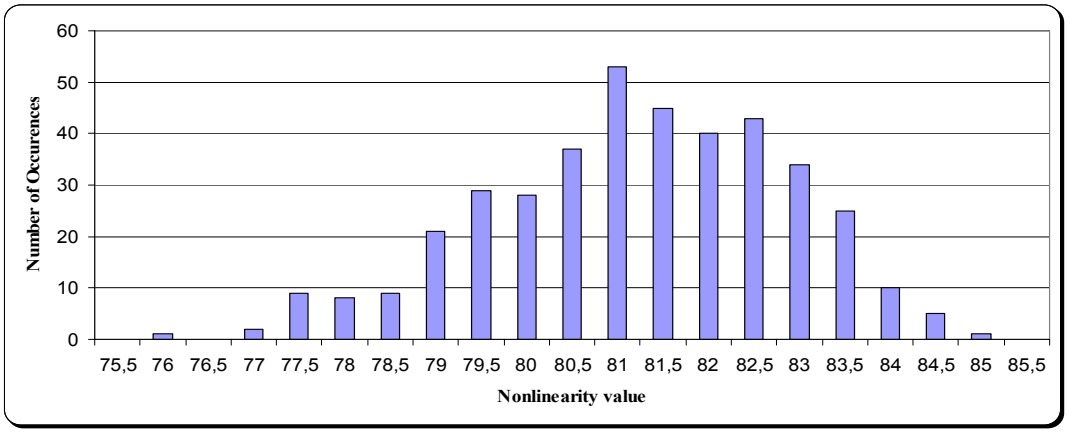

Figure 5.17 Nonlinearity values of *S*-boxes of Twofish (Average: 81,215)

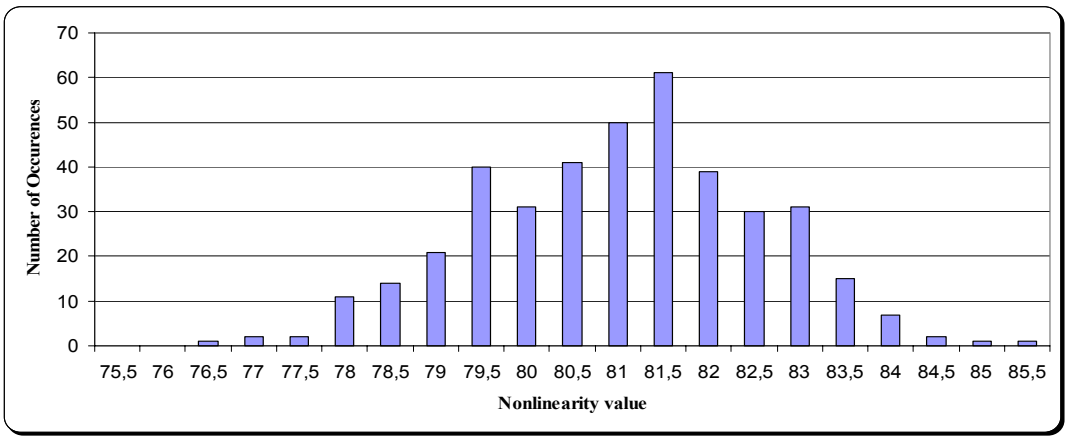

Figure 5.18 Nonlinearity values of *S*-boxes of Twofish (Average: 81,02625)

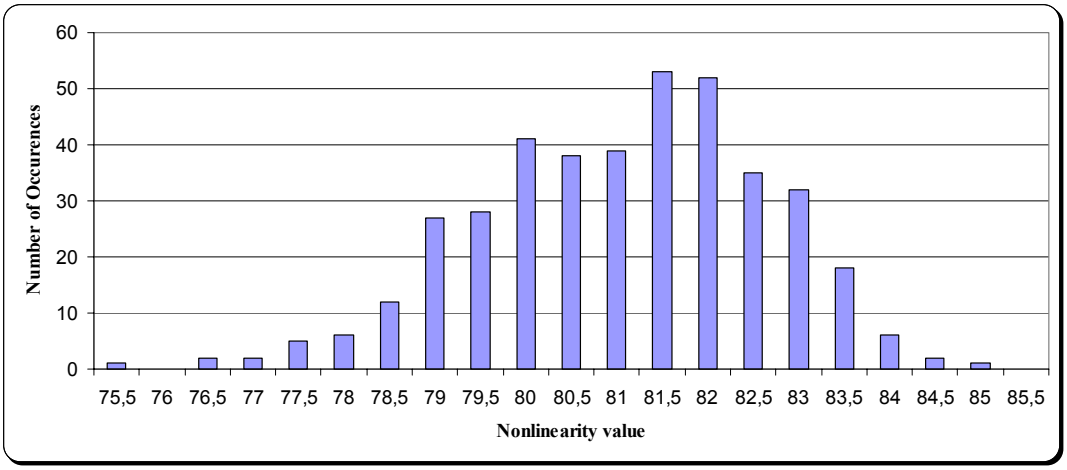

Figure 5.19 Nonlinearity values of *S*-boxes of Twofish (Average: 81,0625)

As the above figures state, there may be weak and strong keywords but most often the nonlinearity is around 81 for different keywords. From (4.19) it can be calculated that for n=8 if *S*-boxes of Twofish were perfectly nonlinear, the nonlinearity would be 120. The highest nonlinearity achieved for n=8 balanced functions is 116, and 8x8 *S*-box of Rijndael has a nonlinearity of 112. For Twofish, highest value is 88.

5.5 Comparison of Avalanche Criteria with NIST Statistical Test Suite

Randomness testing of AES candidates was based on NIST Statistical Test Suite [24] which consists of 16 core statistical tests. These tests are explained briefly in the following section, to form a basis for comparison with our results.

5.5.1 Description of the Statistical Tests

Frequency Test: The purpose of this test is to determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence.

Block Frequency Test: The purpose of this test is to determine whether the frequency of m-bit blocks in a sequence appears as often as would be expected for a truly random sequence.

Cumulative Sums Forward (Reverse) Test: The purpose of this test is to determine whether the maximum of the cumulative sums in a sequence is too large or too small; indicative of too many ones or zeroes in the early (late) stages.

Runs Test: The purpose of this test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such substrings is too fast or too slow.

Long Runs of Ones Test: The purpose of this test is to determine whether the distribution of long runs of ones agrees with the theoretical probabilities.

Rank Test: The purpose of this test is to determine whether the distribution of the rank of 32x32 bit matrices agrees with the theoretical probabilities.

Spectral (Discrete Fourier Transform) Test: The purpose of this test is to determine whether the spectral frequency of the binary sequence agrees with what would be expected for a truly random sequence.

Non-periodic Templates Test: The purpose of this test is to determine whether the number of occurrences for a specified nonperiodic template agrees with the number expected for a truly random sequence.

Overlapping Template Test: The purpose of this test is to determine whether the number of occurrences for a template of all ones agrees with what is expected for a truly random sequence.

Universal Statistical Test: The purpose of this test is to determine whether a binary sequence does not compress beyond what is expected of a truly random sequence.

Approximate Entropy Test: The purpose of this test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and m+1) against the expected result for a normally distributed sequence. In short, it determines whether a sequence appears more regular than is expected from a truly random sequence.

Random Excursion Test: The purpose of this test is to examine the number of cycles within a sequence and determine whether the number of visits to a given state, [-4, -1] and [1, 4], exceeds the expected for a truly random sequence.

Random Excursion Variant Test: The purpose of this test is to determine if the total number of visits to states, between [-9, -1] and [1, 9] exceeds the expected for a truly random sequence.

Serial Test: The purpose of this test is to determine whether the number of occurrences of *m*-bit overlapping patterns is approximately the same as would be expected for a random sequence.

Lempel-Ziv Complexity Test: The purpose of this test is to determine whether or not the sequence compresses no more than a truly random sequence.

Linear Complexity Test: The purpose of this test is to determine whether or not the sequence is complex enough to be considered truly random.

5.5.2 Statistical Test Results and Comparison with Avalanche Criteria

These sixteen tests applied under different parameter inputs, can be viewed as 189 statistical tests [36], while some of the tests are repeated many times by changing the parameters. Table 5.4 gives the indices of the applied 189 tests, and should be used as reference for the horizontal axes of Fig. 5.20 and 5.21

Table 5.4 Breakdown of the 189 statistical tests applied during randomness test applied by J. Soto [36]

Statistical Test	No. of P-values	Test ID	Statistical Test	No. of P-values	Test ID
Monobit	1	1	Periodic Template	1	157
Block Frequency	1	2	Universal Statistical	1	158
Cusum	2	3-4	Approximate Entropy	1	159
Runs	1	5	Random Excursions	8	160-167
Long Runs of Ones	1	6	Random Excursions Variant	18	168-185
Rank	1	7	Serial	2	186-187

Table 5.4 cont'd Breakdown of the 189 statistical tests applied during randomness test applied by J. Soto [36]

Spectral DFT	1	8	Lempel-Ziv Compression	1	188
Aperiodic Templates	148	9-156	Linear Complexity	1	189

Within these 16 core tests, Frequency (Monobit) Test, Frequency Test within a Block, Runs Test, Test for the Longest Run of Ones in a Block are mostly related with avalanche criteria studied in this thesis. In avalanche weight distribution criterion, the effect of 1 bit changes in the plaintext on the ciphertext are investigated, and the weight of avalanche vector, which is indeed the number of ones in the sequence, is found. As described in the preceding section the indicated tests are also investigating the number of ones in the sequence. As a result comparing the results of these tests with AWD criterion results will be convenient.

For Twofish, by the end of the second round, the output appears to be random for these 4 tests, but according to the avalanche criterion and avalanche weight distribution criterion, it is clear that Twofish meets the conditions within the fourth round. So it seems that if the AWD test of this study was used as the 190^{th} test, Twofish would fail to pass it until the end of the fourth round.

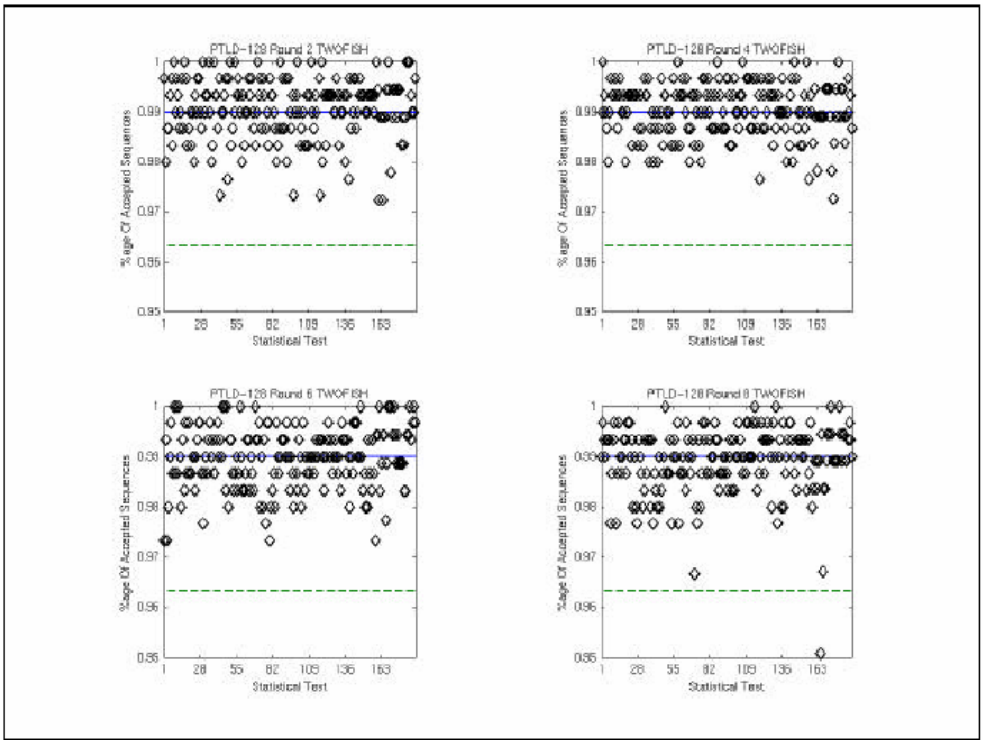

Figure 5.20 Results of 189 statistical tests applied to Twofish, first result on top-left refers to the end of the second round, and others refer to the outputs of the fourth, sixth and eighth rounds, respectively.

The statistical test results for RC6 seem to be more similar to the results obtained from this study. As can be observed from Fig. 5.21. RC6 satisfies the randomness at the end of fourth round. This result is also the same for AWD criterion, where the AWD curves of RC6 are nearly identical to the ideal binomial distribution function within the fourth round.

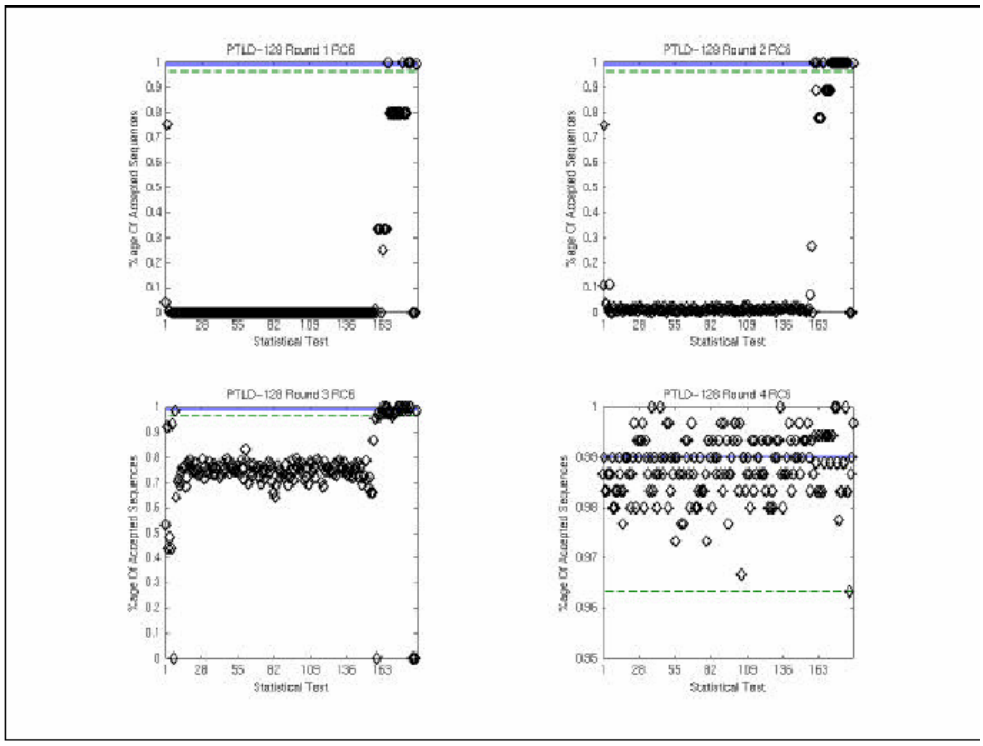

Figure 5.21 Results of 189 statistical tests applied to RC6, where the first result on top-left refers to the end of the first round, and others refer to the outputs of the second, third and fourth rounds, respectively.

CHAPTER 6

CONCLUSION

In this thesis, two finalists of the AES (Advanced Encryption Standard) contest, RC6 developed by Rivest et al, Twofish proposed by Schneier et al, and preceding algorithm of RC6 cipher, RC5, are studied. The strength of ciphers to cryptanalytic attacks is measured according to different criteria. The studied evaluation criteria are the avalanche criterion and its derivations. After the implementation of the algorithms and the test procedures, they are compared with each other.

Firstly, RC5 algorithm is analyzed according to the avalanche criterion and the avalanche weight distribution criterion. It is concluded that RC5 becomes random at the end of the third round. Resemblance parameters in Fig. 5.5 show that RC5 is at least 85% similar to the ideal case after the third round, and the similarity is more than 97%, after the fourth round.

Secondly, RC6 algorithm is analyzed and RC6 seems to be random after the fourth round. Fig 5.10 shows that the resemblance of RC6 avalanche vectors to an ideal random sequence may be as low as 60% at the end of the third round, but it becomes more than 97% at the end of fourth round.

Thirdly, Twofish algorithm is analyzed and similar to RC6; Twofish seems to be random at the fourth round according to avalanche criteria. Fig. 5.15 and Table 5.3 indicate that the resemblance of Twofish avalanche vectors to a true random sequence is as low as 0.1% after fourth round, but it quickly increases to more than 97.3% at the end of the fourth round.

Finally, the nonlinearities of the *S*-boxes of Twofish cipher are calculated. The nonlinearity of the permutation boxes are found as 82 and 72 for the boxes q_0 and q_1 respectively. The nonlinearity distributions of four 8x8 *S*-boxes are computed over many different sets of keys. Since these *S*-boxes have key dependent coefficients, their nonlinearities change in the range [66, 88] for different keys, the average value being around 80 or 82. Although such a nonlinearity parameter is much less than nonlinearity of Rijndael *S*-box, which is 112, one can still argue that dynamic behavior of key dependent *S*-boxes may increase the security of Twofish.

The most important conclusion of this thesis study is the fact that, although NIST results given in Fig. 5.20 assume randomness of Twofish at the end of the second round, the avalanche criteria that we use, indicate that second round outputs are completely nonrandom, especially when a bit change is made in the first part of the plaintext (for i=1,…,63) as observed form Fig. 5.15. Complete randomness

according to our tests can be achieved at the end of the fourth round, where the avalanche vectors of Twofish become similar to random vectors, with a resemblance parameter greater than 97.3%. The difference between NIST results and ours, is most probably coming from the difference between the preparation methods of the test data. Among the data types of NIST [36], the "plaintext avalanche" type is the kind which is the most similar one to our data type. However, there is still a large difference: NIST data is prepared considering all input bit differences for i=1,…,128 for a single plaintext, followed by thousands of other plaintexts, whereas our data is prepared considering a single input bit difference (say i=1) for thousands of plaintexts. After the test is performed i is incremented by 1 and another set of data is prepared using thousands of plaintexts. NIST test data of "plaintext avalanche" type can be considered as an "average" over the data types used in this study; therefore it looses some details related to specific values of i. Future work, we think that NIST tests and our tests should be compared for exactly the same data types. We also propose our data type as an additional data type for NIST Statistical Test Suite.

REFERENCES

[1] C. M. Adams, “A Formal and Practical Design for Substitution-Permutation Network Cryptosystems”, PhD Thesis, Queen’ s University, Kingston, Canada, 1990.

[2] R. Anderson, E. Biham, and L. Knudsen, “Serpent: A Proposal for the Advanced Encryption Standard”, AES algorithm submission, June 1998.

[3] E. Aras, “Analysis of Security Criteria for Block Ciphers”, M.S. Thesis, Middle East Technical University, Turkey, September, 1999.

[4] E. Biham and A. Shamir. Differential Cryptanalysis of the Data Encryption Standard. Springer-Verlag, 1993

[5] L. Brown, J. Pieprzyk and J. Seberry, “LOKI- A Cryptographic Primitive for Authentication and Secrecy Applications”, Advances in Cryptology: Proceedings of CRYPTO’98, Springler-Verlag, 1998, pp.229-236.

[6] C. Burwick, et al., “MARS – A Candidate Cipher for AES”, AES algorithm submission, August 20, 1999.

[7] J. Daemen, V. Rijmen, “AES Proposal: Rijndael”, AES Algorithm Submission, September 3, 1999.

[8] H. Feistel, “Block Cipher Cryptographic System”, U.S. Patent No. 3,798,359, 1974.

[9] J. FOTI, Advanced Encryption Standard (AES): Selection and Plans October 16, 2000

[10] S. Hirose, K. Ikeda, “Nonlinearity Criteria for Boolean Functions”, July 14, 1994.

[11] M.H. Howard, “A Tutorial on Linear and Differential Cryptanalysis”.

[12] I. Vergili, M. D. Yücel, “Avalanche and Bit Independence Properties for the Ensembles of Randomly Chosen *n*x*n* *S*-Boxes”, Turk J Elec Engin, Vol.9, No.2, TÜBİTAK, 2001, pp. 137-145.

[13] B.S. Kaliski Jr. and Y.L. Yin, “On Differential and Linear Cryptanalysis of the RC5 Encryption Algorithm”, In D. Coppersmith, editor, Advances in Cryptology - Crypto '95, pp 171-183, Springer, 1995.

[14] B. S. Kaliski Jr., Y. L. Yin, “On the Security of the RC5 Encryption Algorithm”, RSA Laboratories Technical Report TR-602, Version 1.0, September, 1998.

[15] J. B. Kam, G. I. Davida, “Structured Design of Substitution-permutation Encryption Networks”, IEEE Transactions on Computers, Vol. C-28, No. 10, 1979, pp. 747-753.

[16] L.R. Knudsen and W. Meier, “Improved Differential Attacks on RC5”, In N. Koblitz, editor, Advances in Cryptology - Crypto '96, pp. 216-228, Springer, 1996.

[17] P.C. Kocher, “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”, In N. Koblitz, editor, Advances in Cryptology - Crypto '96, pp. 104-113, Springer, 1996.

[18] X. Lai, J. L. Massey, “A Proposal For A New Block Encryption Standard”, In I.B. Damgärd, Editor, Advances in Cryptology – Eurocrypt’90, Volume 473 of “Lecture Notes in Computer Science”, pages 17-38, Springer-Verlag, Berlin, 1992.

[19] J. L. Massey, “SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm”, Fast Software Encryption, Proc. Cambridge Security Workshop, Cambridge, U.K., LNCS 09, Springler-Verlag, 1994, pp.1-17.

[20] M. Matsui, “Linear Cryptanalysis Method for DES Cipher”, Advances in Cryptology, Proc. Eurocrypt’93, LNCS 765, Springer-Verlag, 1994, pp.386-397.

[21] W. Meier, O. Staffelbach, “Nonlinearity Criteria for Cryptographic Functions”, Advances in Cryptology: Proceedings of EUROCRYPT’89, Springler-Verlag, pp.549-562, 1989.

[22] S. Moriai, K. Aoki, and K. Ohta, “Key-dependency of Linear Probability of RC5”, March 1996.

[23] National Institute of Standards and Technology (NIST), “FIPS Publication 46: Announcing the Data Encryption Standard”, January 1977.

[24] National Institute of Standards and Technology (NIST), “Special Publication 800-22”, “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications”, May 15, 2001

[25] K. Nyberg, “On the Construction of Highly Nonlinear Permutations”, Advances in Cryptology: Proc.EUROCRYPT’92, pp.93-99, 1993.

[26] R. L. Rivest, “The RC2 Encryption Algorithm”, RSA Data Security Inc., March 12, 1992.

[27] R. L. Rivest, “The RC5 Algorithm”, Fast Encryption Algorithm, volume 1008 of Lecture Notes in Computer Science, Springer-Verlag, 1995, pp. 89-96.

[28] R. L. Rivest, M. J. B. Robshaw, R. Sidney, Y. L. Yin, “The RC6 Block Cipher”, August 20, 1998.

[29] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, “Twofish: A 128-Bit Block Cipher”, June 15, 1998.

[30] J. Seberry, X. Zhang, Y. Zheng, “Nonlinearity and Propagation Characteristics of Boolean Functions”, November, 1993.

[31] C. E. Shannon, “Communication Theory of Secrecy Systems”, Bell Systems Technical Journal, Vol.28, 1949, pp.656-715.

[32] A. Shimizu, S. Miyaguchi, “Fast Data Encipherment Algorithm FEAL” In D. Chaum and W.L. Price, editors, Advances in Cryptology – Eurocypt’87,

volume 304 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1988, pp. 267-280.

[33] G.J. Simpsons, editor. Contemporary Cryptology, "The Science of Information Integrity", IEEE Press, New York, 1992.

[34] "SKIPJACK and KEA Algorithm Specifications", May 29, 1998

[35] A. Sorkin, "Lucifer, A Cryptographic Algorithm". Cryptologia, 8 (1):22-41 1984.

[36] J. Soto, L. Bassham "Randomness Testing of the Advanced Encryption Standard Finalist Candidates", Computer Security Division National Institute of Standards and Technology, March 28, 2000.

[37] X. Zhang, "On the Difficulty of Constructing Cryptographically Strong Substitution Boxes", Journal of Universal Computer Science, vol.2, n.3, pp.147-162, 1996.

[38] X. Zhang, Y. Zheng and H. Imai, "Relating Differential Distribution Tables to Other Properties of Substitution Boxes", Designs, Codes and Cryptography, vol.19, pp.45-63, 1998.