AVALANCHE PROPERTIES AND RANDOMNESS OF THE TWOFISH CIPHER

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY
BY

ÖMER EL

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICS ELECTRONICS ENGINEERING

DECEMBER 2004

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. İSMET ERKMEN

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. Mete SEVERCAN	(METU, EE)	_____
Assoc.Prof.Dr. Melek D.YÜCEL	(METU, EE)	_____
Prof.Dr. Yalçın TANIK	(METU, EE)	_____
Prof.Dr. Ersan AKYILDIZ	(METU, EE)	_____
Savaş ARIKAN (M.Sc)	(ASELSAN)	_____

PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Ömer EL

Signature :

ABSTRACT

AVALANCHE PROPERTIES AND RANDOMNESS OF THE TWOFISH CIPHER

EL, Ömer
M.S., Department of Electrical and Electronics Engineering
Supervisor: Assoc.Prof. Dr. Melek D. YÜCEL
December 2004, 82 pages

In this thesis, one finalist cipher of the Advanced Encryption Standard (AES) block cipher contest, Twofish proposed by Schneier et al, is studied in order to observe the validity of the statement made by Arıkan about the randomness of the cipher, which contradicts National Institute of Standards and Technology (NIST)'s results. The strength of the cipher to cryptanalytic attacks is investigated by measuring its randomness according to the avalanche criterion. The avalanche criterion results are compared with those of the Statistical Test Suite of the NIST and discrepancies in the second and third rounds are explained theoretically.

Keywords: Block cipher, Twofish, avalanche criterion, Walsh-Hadamard transform, propagation, nonlinearity.

ÖZ

TWOFISH ŞİFRESİNİN ÇIĞ ÖZELLİĞİ VE RASTGELELİĞİ

EL, Ömer
Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Melek D. YÜCEL
Aralık 2004, 82 Sayfa

Bu tezde, AES blok şifre yarışmasının finalistlerinden biri olan, Schneier ve arkadaşlarının sunduğu Twofish şifresi, rastgeleliği hakkında Arıkan tarafından yapılan ve NIST'in sonuçları ile çelişen açıklamanın geçerliliğini incelemek için çalışıldı. Şifrenin kriptanalitik ataklara karşı dayanıklılığı, rastgeleliğini çığ kriterine göre ölçerek araştırıldı. Çığ kriteri sonuçları, NISTin test süit sonuçları ile karşılaştırıldı ve ikinci ve üçüncü turlardaki farklılıklar teorik olarak açıklandı.

Anahtar Sözcükler: Blok şifreler, Twofish, çığ kriteri, Walsh-Hadamard dönüşümü, yayılma, doğrusal olmama.

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my supervisor Assoc. Prof. Dr. Melek D. Yücel for her valuable guidance and helpful suggestions. This thesis would not have been completed without her guidance. I also thank for her tolerance and understanding as a supervisor.

I wish to thank ASELSAN Inc. for the facilities provided and to my colleagues at this company for their patience and encouragement in this effort.

I am grateful to all my friends, who were with me at every moment, for their patience and understanding during my thesis and their friendship throughout my life.

Last, but most, I would like to thank my mother Rabia, my father Ali, and my brother Serdar for their moral support, encouragement, patience and endless love that bring me up today.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES:

CHAPTER 1

INTRODUCTION

1.1 Background

Cryptography is the process of combining some input data, called the plaintext, with a user-specified key to generate an encrypted output, called the ciphertext. Cryptographic security requires that given the ciphertext, no one can recover the original plaintext without the key. The algorithms that combine the keys and plaintexts are called ciphers. Cryptanalysis is the science of breaking ciphers, i.e., retrieving the plaintext from the ciphertext without knowing the proper key. The branch of mathematics encompassing both cryptography and cryptanalysis is called cryptology.

There are two kinds of cryptographic algorithms: symmetric and asymmetric. Symmetric algorithms use the same key (the secret key) to encrypt and decrypt a message, and asymmetric algorithms use one key (the public key) to encrypt a message and a different key (the private key) to decrypt it. Asymmetric algorithms are also called public key algorithms. Symmetric algorithms, also called secret-key algorithms, require the sender and receiver to agree on a key before they pass messages back and forth. This key must be kept secret. The security of a symmetric algorithm rests entirely in the key.

Symmetric key algorithms can be divided into two categories, stream ciphers and block ciphers. Stream ciphers encrypt a single bit of plaintext at a time; whereas block ciphers operate on the plaintext in group of bits, called blocks. Most of the

block ciphers are composed of usually 8 to 32 iteration rounds, where each iteration contains nonlinear substitution boxes (S-Boxes) followed by linear permutations. Such ciphers are named as Substitution Permutation Networks (SPN) [Feistel, 1973]

Block ciphers can be used to design stream ciphers with a variety of synchronization and error extension properties, one-way hash functions, message authentication codes, and pseudo-random number generators. Because of this flexibility, they are the workhorse of modern cryptography.

Feistel ciphers [Feistel, 1973], which are also called DES-like ciphers, are a special class of iterated SPN ciphers where the ciphertext is calculated from the plaintext by repeated application of the same transformation or round function. Furthermore, in a Feistel cipher, the ciphertext being encrypted is split into two halves. The round function *f is* applied to one half using a subkey and the output of *f* is XORed with the other half. The two halves are then swapped. Each round follows the same pattern except for the last round, for which there is no swap. A nice feature of a Feistel cipher is that encryption and decryption are structurally identical, though the subkeys used during encryption in each round are taken in reverse order during encryption.

National Institute of Standards and Technology (NIST) announced the Advanced Encryption Standard (AES) program in 1997 to replace Data Encryption Standard (DES). 15 algorithms were submitted and five algorithms were selected as AES candidates. Finally, Rijndael algorithm was selected as the new encryption standard October 2, 2000. Twofish cipher was one of the finalist algorithms.

1.2 Scope and Objective of Thesis

In this thesis, in order to observe the validity of the statement made by Arıkan [Arıkan, 2003] about the randomness of the cipher, which contradicts NIST's results [Soto, 2000], the avalanche characteristics of the Twofish algorithm are investigated.

In Chapter 2, some mathematical definitions related to Boolean functions are reviewed to form a background [Yücel, 2001] and then some properties of Boolean functions are given [Preneel, 1994]. We prove that a dyadic shift in the Walsh-Hadamard domain, i.e., addition of linear terms to a function in the original domain, does not change the propagation characteristics of a function [Preneel, 1991]. Then we give an example to observe the effects of addition of linear and non-liner terms to a function on the propagation characteristics.

In Chapter 3, the building blocks and encryption algorithm of the Twofish cipher are given.

In Chapter 4, the 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 criterion and nonlinearity measure. Then we give the results of avalanche criterion test and compare them with the results found by Arıkan [Arıkan, 2003]. For better understanding of the avalanche test results, we derive the outputs for round 2 and 3 of the Twofish algorithm in terms of the input plaintext words. Avalanche test results are compared with three of 16 core tests of NIST statistical Test Suite, which are monobit, frequency test within a block and runs test. Also in chapter 4, the nonlinearity of the S-boxes of the Twofish cipher and the effects of keywords on the nonlinearity measure are investigated.

Finally Chapter 5 summarizes the work of the thesis.

CHAPTER 2

PROPAGATION CHARACTERISTICS OF BOOLEAN FUNCTIONS

2.1 Introduction

In this chapter some mathematical definitions related to Boolean functions $f : Z_2^n \to Z_2$, which map n bits to a single bit, are given [Yücel, 2001]. Autocorrelation function and Walsh-Hadamard transform, which is also called the spectrum of Boolean functions, are defined [Preneel, 1994] and some of their characteristics are given. The Strict Avalanche Criterion and perfect non-linearity are defined [Preneel, 1991]. The behaviour of a Boolean function for more than one input bit complementation which is defined as the propagation criterion of a Boolean function is studied [Preneel, 1991]. We prove that a dyadic shift in the Walsh-Hadamard domain implies adding linear terms to a function in the original domain. Finally we give an example of a Boolean function of degree 4 which shows that addition of linear terms to a function does not change the propagation characteristics of a Boolean function.

2.2 Definitions

A Boolean function $f(x)$ is a function whose domain is the vector space Z_2^n of binary *n*-tuples $\mathbf{x} = (x_1.....x_n) \in Z_2^n$ that takes values 0 or 1. In some cases it will be more convenient to work with functions that take the values $\{-1,1\}$. The function $\hat{f}(\mathbf{x})$ is defined as:

$$\hat{f}(\mathbf{x}) = 1 - 2f(\mathbf{x}) = (-1)^{f(\mathbf{x})}$$

Definition 2.1 ***(Affine & Linear Functions)*** A Boolean function $f(\mathbf{x})$ is called an affine function of $\mathbf{x} = (x_1.....x_n) \in Z_2^n$ if it is in the form

$$f(\mathbf{x}) = a_1 \otimes x_1 \oplus a_2 \otimes x_2 \oplus ..... \oplus a_n \otimes x_n \oplus c = \mathbf{w}.\mathbf{x} \oplus c, \qquad (2.1)$$

where $a_1, a_2....., a_n$, c belong to Z_2 , $\mathbf{w} = (a_1.....a_n) \in Z_2^n$, and $\oplus$, $\otimes$ & respectively denote addition, multiplication and inner product operations in Z_2.
$f(\mathbf{x})$ is called linear if c=0.

Definition 2.2 ***(Truth Table)*** The truth table f_t of a Boolean function $f(x)$ is found by evaluating $f(x)$ for all possible values of $\mathbf{x} = \mathbf{a}_i$; where $\mathbf{a}_i$ is the *n*-bit vector corresponding to binary representation of the integer $i = 0,1,....,2^n - 1$. So:

$$f_t = \{f(\mathbf{a}_0), ..., f(\mathbf{a}_{2^n-1})\} \qquad (2.2)$$

Definition 2.2 ***(Sequences)*** The sequence $\boldsymbol{f}_\mathbf{s}$ of a Boolean function $f(\mathbf{x})$ is defined for all possible values $\mathbf{x} = \mathbf{x}_i$ as:

$$\boldsymbol{f}_\mathbf{s} = \{(-1)^{f(\mathbf{x}_0)}, (-1)^{f(\mathbf{x}_1)}, ...(-1)^{f(\mathbf{x}_{2^n-1})}\}$$
$$= \{(-1)^{f(\mathbf{x})}\}|_{\mathbf{x}=\mathbf{x}_i}$$

Where $\mathbf{x}_i$ is the *n*-bit vector corresponding to the binary representation of the integer i= 0… 2^n–1.

Definition 2.4 ***(Correlation and Autocorrelation)*** Correlation coefficient C(f, g) between two functions $f : Z_2^n \to Z_2$ and $g : Z_2^n \to Z_2$ is

$$C(f, g) = 2^{-n} \sum_{\mathbf{x}} (-1)^{f(\mathbf{x}) \oplus g(\mathbf{x})} = 2^{-n} \sum_{\mathbf{x}} (-1)^{f(\mathbf{x})} (-1)^{g(\mathbf{x})} = 2^{-n} \boldsymbol{f}_{\mathbf{s}} \bullet \boldsymbol{g}_{\mathbf{s}} \qquad (2.3)$$

where the summation Σ and inner product $\bullet$ operations are defined in the field of real numbers R.

The autocorrelation function $r_f(\mathbf{d})$ of f is

$$r_f(\mathbf{d}) = 2^{-n} \sum_{\mathbf{x}} (-1)^{f(\mathbf{x}) \oplus f(\mathbf{x} \oplus \mathbf{d})} = 2^{-n} \sum_{\mathbf{x}} (-1)^{f(\mathbf{x})} (-1)^{f(\mathbf{x} \oplus \mathbf{d})} = 2p(\mathbf{d}) - 1.$$

p($\mathbf{d}$) is the probability that $\{f(\mathbf{x}) = f(\mathbf{x} \oplus \mathbf{d})\}$, which is computed as $\#\{\mathbf{x} \mid f(\mathbf{x}) = f(\mathbf{x} \oplus \mathbf{d})\} / 2^n$, where #{.} denotes the number of occurrences of an event.

Definition 2.3 ***(Distance Between Functions)*** Hamming distance $d_H(f, g)$ between two functions $f : Z_2^n \to Z_2$ and $g : Z_2^n \to Z_2$ is defined as the Hamming distance $d_H(\boldsymbol{f}_{\mathbf{s}}, \boldsymbol{g}_{\mathbf{s}})$ between their 2^n-bit sequences $\boldsymbol{f}_{\mathbf{s}}$ and $\boldsymbol{g}_{\mathbf{s}}$ (which is equal to the number of places, where these two vectors differ).

Definition 2.4 ***(Hadamard Matrix)*** A hadamard matrix H is an $n \times n$ matrix with entries $+1$ or -1, such that all rows and 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 Sylvester-Hadamard matrix of order 2^n denoted by H_n is generated by the following recursive relation:

$$H_0 = 1, \qquad H_n = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix} \qquad (2.4)$$

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 $2^3 x 2^3$ 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) I_i of a $2^n x 2^n$ Sylvester-Hadamard matrix H_n, and the sequence of a linear function $I : Z_2^n \to Z_2$ defined by $I_i(x) = \mathbf{w} \bullet \mathbf{x}$, where the subscript *i* takes 2^n different values corresponding to 2^n possible weighting vectors **w.**

Definition 2.5 ***(Walsh-Hadamard Transforms)*** In the space of Boolean functions, sequences of all linear functions form an orthogonal basis with respect to the inner product operation ♦. The representation of a Boolean function *f*(**x**) with respect to this basis is called the Walsh-Hadamard transform , or the *spectrum* of *f*(**x**):

$$W\{\hat{f}(\mathbf{x})\} = \hat{F}(\mathbf{w}) = \sum_{\mathbf{x} \in Z_2^n} (-1)^{f(\mathbf{x})} (-1)^{\mathbf{w}.\mathbf{x}} = \boldsymbol{f}_{\mathbf{s}} \blacklozenge (\mathbf{w}.\mathbf{x})_{\mathbf{s}}. \qquad (2.5)$$

Since the summation Σ and inner product ♦ operations in (2.5) are defined in the field of real numbers R, the Walsh-Hadamard transform $\hat{F}(\mathbf{w})$ takes even integer values in the interval [-2^n, 2^n]. For 2^n different values of $\mathbf{w}$, $\hat{F}(\mathbf{w})/2^n$ is the normalized component of $\boldsymbol{f}_s$ along the linear sequence (**w.x**)$_s$, which is also equal to the correlation coefficient (2.3) between $f(\mathbf{x})$ and **w.x**.

As the Walsh-Hadamard transform is linear, an alternative definition [Preneel, 1991] based on a matrix product is possible. The function values of $\hat{f}(\mathbf{x})$ and $\hat{F}(\mathbf{w})$ are written in the column matrices $\left[\hat{f}\right]$ and $\left[\hat{F}\right]$ respectively

$$\left[\hat{F}\right] = H_n \cdot \left[\hat{f}\right],$$

where H_n is the Walsh-Hadamard matrix of order n that can be recursively defined as

$$H_n = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes H_{n-1}\ , \quad H_0 = 1\,.$$

Here $\otimes$ denotes the Kronecker product between matrices. It is easily seen that $H_n^2 = 2^n \cdot I_n$

Fact 2.5 ***(The Relation Between the Walsh-Hadamard Transforms of*** $f(x)$ ***and*** $\hat{f}(x)$***)*** The relation between the Walsh-Hadamard transforms of $f(x)$ and $\hat{f}(x)$ is given by

$$\begin{aligned}\hat{F}(\mathbf{w}) &= \sum_{x} \hat{f}(\mathbf{x})(-1)^{\mathbf{w}\bullet\mathbf{x}} \\ &= \sum_{x} (-1)^{f(\mathbf{x})}(-1)^{\mathbf{w}\bullet\mathbf{x}} \\ &= \sum_{x} (1-2f(\mathbf{x}))(-1)^{\mathbf{w}\bullet\mathbf{x}} \\ &= \sum_{x} (-1)^{\mathbf{w}\bullet\mathbf{x}} - 2\sum_{x} f(\mathbf{x})(-1)^{\mathbf{w}\bullet\mathbf{x}} \\ &= -2F(\mathbf{w}) + 2^{n}\delta(\mathbf{w}) \qquad (2.6)\end{aligned}$$

where $\delta(\mathbf{w})$ denotes the Kronecker delta ($\delta(\mathbf{0}) = 1,\ \delta(\mathbf{k}) = 0$ for $\forall \mathbf{k} \neq \mathbf{0}$) function.

Definition 2.6 (*Hamming Weight*) The hamming weight w_H of an element of Z_2^n is the number of components equal to 1. Similarly, the hamming weight w_H of a Boolean function f is the number of components of its sequence equal to 1.

Definition 2.7 ***(Nonlinearity Measure)*** Nonlinearity of a Boolean function $f(\mathbf{x})$ is defined as the minimum number of cases, over all input vectors, that it differs from an affine function. One can find this distance by comparing the truth table of the Boolean function to all rows of the Hadamard matrix.

$$\begin{aligned}N_f &= min_{\mathbf{w},c} \#\{\mathbf{x} \in Z_2^n \mid f(\mathbf{x}) \neq \mathbf{w}\bullet\mathbf{x} \oplus c\} \\ &= min_{\mathbf{w},c}\ \mathrm{d}_H(f(\mathbf{x}),\ (\mathbf{w}\bullet\mathbf{x} \oplus c)) \qquad . \qquad (2.7)\end{aligned}$$

There are also other definitions, which can be shown to be equivalent to (2.7):

$$N_f = 2^{n-1} - \frac{1}{2} \max_{w=0,1,\ldots,2^{n-1}} \left| f_s \bullet (w\bullet x)_s \right| = 2^{n-1} - \frac{1}{2}\ max(\left|\hat{F}(\mathbf{w})\right|) \qquad (2.8)$$

Here $\hat{F}(\mathbf{w})$ is the Walsh-Hadamard transform of a $\hat{f}(\mathbf{x})$

2.3 Properties of Boolean Functions

1. *Balancedness*

A Boolean function is said to be balanced if its truth table contains as many 0 as 1 entries.

It is easy to show that this is equivalent to $\hat{F}(\mathbf{0}) = 0$

$$\hat{F}(\mathbf{w}) = \sum_{\mathbf{x}} (-1)^{f(\mathbf{x})} (-1)^{\mathbf{w} \cdot \mathbf{x}}$$

$$\hat{F}(\mathbf{0}) = \sum_{\mathbf{x}} (-1)^{f(\mathbf{x})} (-1)^{\mathbf{0} \cdot \mathbf{x}}$$

$$\hat{F}(\mathbf{0}) = \sum_{\mathbf{x}} (-1)^{f(\mathbf{x})}$$

$\hat{F}(\mathbf{0}) = 0$, so $F(\mathbf{0}) = \sum_{\mathbf{x}} f(\mathbf{x})(-1)^{0 \cdot \mathbf{x}} = 2^{n-1}$ by (2.6).

2. *Correlation Immunity*

A Boolean function $f(\mathbf{x})$ is *m*'th order correlation immune if $f(\mathbf{x})$ is statistically independent of any subset of *m* input variables [Preneel, 1994]. This can be shown to be equivalent to be,

$$\hat{F}(\mathbf{w}) = 0 \quad 1 \le w_H(\mathbf{w}) \le m, \tag{2.9}$$

and a necessary condition is $ord(f) \le n - m$ [Preneel, 1994].

If $f(\mathbf{x})$ is also balanced, then $ord(f) \le n - m - 1$ unless $m = n - 1$.

3. *Strict Avalanche Criterion*

A Boolean function $f(\mathbf{x})$ satisfies the Strict Avalanche Criterion (*SAC*) if and only if $f(\mathbf{x})$ changes with a probability of one half whenever a single input bit of $\mathbf{x}$ is complemented [Preneel, 1991].

That is, if $f(\mathbf{x})$ satisfies *SAC* then $p(\mathbf{d})$, the probability that $\{f(\mathbf{x}) = f(\mathbf{x} \oplus \mathbf{d})\}$, should be equal to ½ for $w_H(\mathbf{d}) = 1$. Then, the corresponding values of the autocorrelation function will be

$r_f(\mathbf{d}) = 2p(\mathbf{d}) - 1 = 0$ for all $\mathbf{d} \in Z_2^n \,|\, w_H(\mathbf{d}) = 1$.

4. *Higher Order SAC*

A Boolean function $f(\mathbf{x})$ satisfies the Strict Avalanche Criterion of order *m* (*SAC* of order *m*) if any function obtained from $f(\mathbf{x})$ by keeping *m* of its input bits constant satisfies the *SAC* [Preneel, 1991].

5. *Perfect Non-linearity*

A Boolean function $f(\mathbf{x})$ is perfect non-linear (with respect to linear structures) if $f(\mathbf{x})$ changes with a probability of one half whenever i $(1 \le i \le n)$ bits of $\mathbf{x}$ are complemented [Preneel, 1991]. That is,

$r_f(\mathbf{d}) = 2p(\mathbf{d}) - 1 = 0$ for all $\mathbf{d} \in Z_2^n \,|\, 1 \le w_H(\mathbf{d}) \le n$.

These two definitions can be generalized in a natural way as follows,

6. *Propagation Criterion*

A Boolean function $f(\mathbf{x})$ satisfies the propagation criterion of degree k (*PC* of degree *k*) if $f(\mathbf{x})$ changes with a probability of one half whenever i $(1 \le i \le k)$ bits of $\mathbf{x}$ are complemented [Preneel, 1994]. That is,

$$r_f(\mathbf{d}) = 0 \quad for \quad 1 \le w_H(\mathbf{d}) \le k, \ \mathbf{d} \in Z_2^n \tag{2.10}$$

Note that *SAC* is *PC* of degree 1 and perfect non-linear is *PC* of degree *n*.

The propagation criterion is defined as the non-linearity of f with respect to linear structure.

7. *Higher Order Propagation Criterion*

A Boolean function $f(\mathbf{x})$ of n variables satisfies the propagation criterion of degree k and order m ($PC(k)$ and order m) if any function obtained from $f(\mathbf{x})$ by keeping m of its input bits constant satisfies the $PC(k)$ [Preneel, 1991].

Here $k+m \le n$, if m bits are kept constant at most $n-m$ bits can be changed.
A dyadic shift in the original domain generates a Boolean function with the same autocorrelation function [Preneel, 1991].

Theorem 2.1 If $g(\mathbf{x}) = f(\mathbf{x} \oplus \mathbf{s})$, then $g(\mathbf{x})$ is called dyadically shifted form of $f(\mathbf{x})$ and the autocorrelation function of $g(\mathbf{x})$ is equal to that of $f(\mathbf{x})$.

Proof

$$\begin{aligned} r_g(\mathbf{d}) &= \sum_{\mathbf{x}} g(\mathbf{x}) \bullet g(\mathbf{x} \oplus \mathbf{d}) \\ &= \sum_{\mathbf{x}} f(\mathbf{x} \oplus \mathbf{s}) \bullet f(\mathbf{x} \oplus \mathbf{d} \oplus \mathbf{s}) \\ &= \sum_{\mathbf{x}'} f(\mathbf{x}') \bullet f(\mathbf{x}' \oplus \mathbf{d}) \\ &= r_f(\mathbf{d}) \end{aligned}$$

When dealing with propagation properties it is important to be able to construct different functions that satisfy the same property starting from one function. One method is the dyadic shift in the Walsh-Hadamard domain.

Theorem 2.2 Let $f(\mathbf{x})$ be a Boolean function. Then the function $g(\mathbf{x})$, with the Walsh-Hadamard transform $\hat{G}(\mathbf{w}) = \hat{F}(\mathbf{w} \oplus \mathbf{s})$, is dyadically shifted form of $\hat{F}(\mathbf{w})$ and $g(\mathbf{x})$ is also Boolean for all $\mathbf{s}$. Moreover, the autocorrelation function of $g(\mathbf{x})$ has the same absolute values (and thus the same zeroes) as the autocorrelation function of $f(\mathbf{x})$ and for $\mathbf{s} \neq \mathbf{0}$, the distance $d(f,g)$ equals 2^{n-1} [Preneel, 1991]. In short,

$\hat{G}(\mathbf{w}) = F(\hat{\mathbf{w}} \oplus \mathbf{s}) \;\Rightarrow\; r_g(\mathbf{d}) = r_f(\mathbf{d})(-1)^{\mathbf{s}\bullet\mathbf{d}}$, and $d(f,g) = 2^{n-1}$.

Proof

$$
\begin{aligned}
r_g(\mathbf{d}) &= \sum_{x} (-1)^{g(\mathbf{x})}(-1)^{g(\mathbf{x}\oplus\mathbf{d})} \\
&= 2^{-n} \sum_{\mathbf{w}} \hat{G}^2(\mathbf{w})(-1)^{\mathbf{w}\bullet\mathbf{d}} \\
&= 2^{-n} \sum_{\mathbf{w}} F^2(\hat{\mathbf{w}} \oplus \mathbf{s})(-1)^{\mathbf{w}\bullet\mathbf{d}} \\
&= 2^{-n} \sum_{\underline{\mathbf{w}'}} F^2(\hat{\mathbf{w}}')(-1)^{(\mathbf{w}'\oplus\mathbf{s})\bullet\mathbf{d}} \\
&= 2^{-n} \sum_{\mathbf{w}'} F^2(\hat{\mathbf{w}}')(-1)^{\mathbf{w}'\bullet\mathbf{d}}(-1)^{\mathbf{s}\bullet\mathbf{d}} \\
&= (-1)^{\mathbf{s}\bullet\mathbf{d}}\; 2^{-n} \sum_{\mathbf{w}'} F^2(\hat{\mathbf{w}}')(-1)^{\mathbf{w}'\bullet\mathbf{d}} \\
&= (-1)^{\mathbf{s}\bullet\mathbf{d}}\; r_f(\mathbf{d})
\end{aligned}
$$

To evaluate the distance of $g(\mathbf{x})$ and $f(\mathbf{x})$, and for $G(\mathbf{w}) = F(\mathbf{w} \oplus \mathbf{s})$

$$
\begin{aligned}
(-1)^{g(\mathbf{x})} &= 2^{-n} \sum_{\mathbf{w}} \hat{G}(\mathbf{w})(-1)^{\mathbf{w}\bullet\mathbf{x}} \\
&= 2^{-n} \sum_{\mathbf{w}} F(\hat{\mathbf{w}} \oplus \mathbf{s})(-1)^{\mathbf{w}\bullet\mathbf{x}} \\
&= 2^{-n} \sum_{\mathbf{w}'} F(\hat{\mathbf{w}}')(-1)^{(\mathbf{w}'\oplus\mathbf{s})\bullet\mathbf{x}} \\
&= 2^{-n} \sum_{\mathbf{w}'} F(\hat{\mathbf{w}}')(-1)^{\mathbf{w}'\bullet\mathbf{x}}(-1)^{\mathbf{s}\bullet\mathbf{x}} \\
&= (-1)^{\mathbf{s}\bullet\mathbf{x}}\; 2^{-n} \sum_{\mathbf{w}'} F(\hat{\mathbf{w}}')(-1)^{\mathbf{w}'\bullet\mathbf{x}} \\
&= (-1)^{\mathbf{s}\bullet\mathbf{x}}\; (-1)^{f(\mathbf{x})} \\
&= (-1)^{f(\mathbf{x})\oplus(\mathbf{s}\bullet\mathbf{x})} \\
g(\mathbf{x}) &= f(\mathbf{x}) \oplus (\mathbf{s}\bullet\mathbf{x})
\end{aligned}
$$

Since $\mathbf{s}\bullet\mathbf{x}$ is a linear function, for $\mathbf{s} \neq \mathbf{0}$, $\mathbf{s}\bullet\mathbf{x}$ takes the values 1 for 2^{n-1} times and 0 for 2^{n-1} times. So, the distance between f and g, $d(f,g)$ is equal to 2^{n-1}.

$$\hat{G}(\mathbf{w}) = F(\hat{\mathbf{w}} \oplus \mathbf{s}) \quad \Rightarrow g(\mathbf{x}) = f(\mathbf{x}) \oplus (\mathbf{s} \cdot \mathbf{x}) \tag{2.11}$$

Corollary 2.2.1 A dyadic shift and a complementation in the Walsh-Hadamard domain mean adding linear terms to a function in the original domain [Preneel, 1991].

$G(\mathbf{w}) = F(\mathbf{w} \oplus \mathbf{s}) \quad \Rightarrow g(\mathbf{x}) = f(\mathbf{x}) \oplus (\mathbf{s} \cdot \mathbf{x})$.

Corollary 2.2.2 Adding right linear terms to a function, i.e., if **s** is chosen properly, with at least one zero in the Walsh spectrum will result in a balanced function with the same propagation properties [Preneel, 1991].

In the following example, we define a simple function, $f(x)$, and evaluate the Walsh-Hadamard transform $\hat{F}(\mathbf{w})$ and autocorrelation function $r_f(\mathbf{d})$ of $f(x)$. Then we define 4 more functions, $f_1(x)$, $f_2(x)$, $f_3(x)$ and $f_4(x)$ where f_1, f_2 and f_3 are obtained with the addition of linear terms to f and f_4 is obtained with the addition of a nonlinear term. We show that $f(x)$ and 3 functions, $f_1(x)$, $f_2(x)$, $f_3(x)$, obtained with the addition of linear terms to $f(x)$, have the same *PC* where as $f_4(x)$ has different *PC*.

Example:

$$f(\mathbf{x}) = x_1x_2x_3x_4 + x_1x_2x_3x_5 + x_1x_2x_4x_5 + x_1x_3x_4x_5 + x_2x_3x_4x_5 + x_1x_4 + x_1x_5 + x_2x_3 + x_2x_5 + x_3x_4$$

Sequence of $f(\mathbf{x})$ is calculated as,

$\boldsymbol{f}_{\mathbf{s}}$ = (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)

In Fig 2.1 the Walsh-Hadamard transform of f is sketched. We see that f is correlation immune of order 1 because $F(\mathbf{w}) = 0$ for all $\mathbf{w}$ with hamming weight of $\mathbf{w}$ equal to 1. Also note that f is not balanced because $F(\mathbf{w}) \neq 0$ for $\mathbf{w} = 0$.

$F(\mathbf{w})$

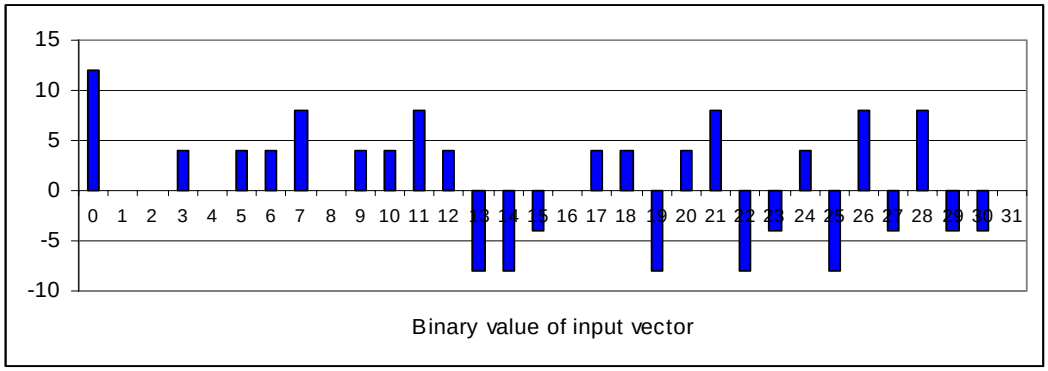


Figure 2.1 The Walsh-Hadamard transform of $f(x)$

Fig 2.2 shows the autocorrelation function $r_f(\mathbf{d})$ of f. From the figure we see that f satisfies $PC(2)$ because from (2.10) we know that $f(x)$ is *PC(k)* if $r_f(\mathbf{d}) = 0$ for $1 \leq w_H(\mathbf{d}) \leq k$.

$r_f(\mathbf{d})$

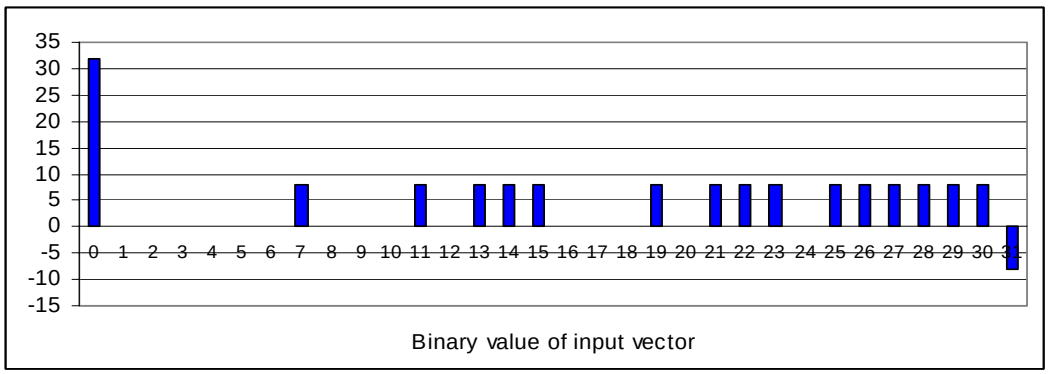


Figure 2.2 The autocorrelation function of $f(x)$

Now let's see what happens when we add $x_1 + x_2 + x_3 + x_4 + x_5$ to f. The function $f_1(x)$ is defined as $f(x) + x_1 + x_2 + x_3 + x_4 + x_5$.

Fig 2.3 shows the Walsh-Hadamard transform of $f_1(\mathbf{x})$. From the figure it is clear that $f_1(\mathbf{x})$ is balanced because $F_1(\mathbf{w}) = 0$ for $\mathbf{w} = 0$. We know from (2.11) that adding linear terms to a function results in a dyadic shift in the Walsh-Hadamard domain. From Fig. 2.3 it can be concluded that $F_1(\mathbf{w}) = F(\mathbf{w} \oplus (11111))$

$F_1(\mathbf{w})$ (Dyadic shift 11111 of F)

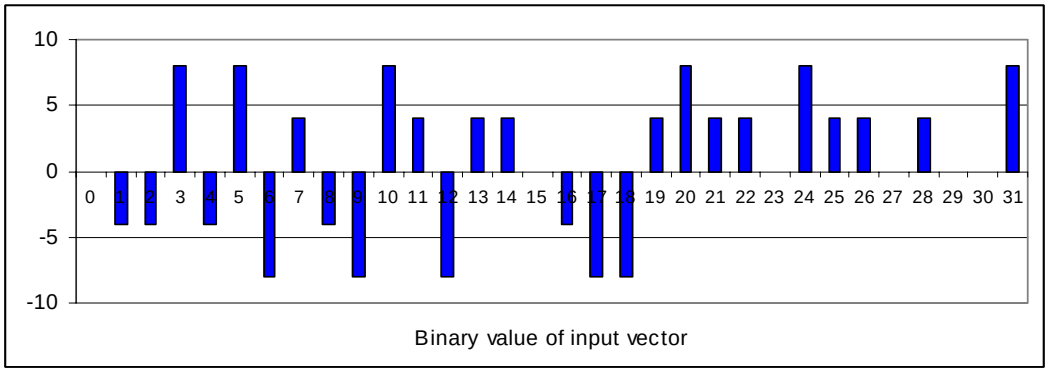


Figure 2.3 The Walsh-Hadamard transform of $f_1(x)$

$r_{f_1}(\mathbf{d})$

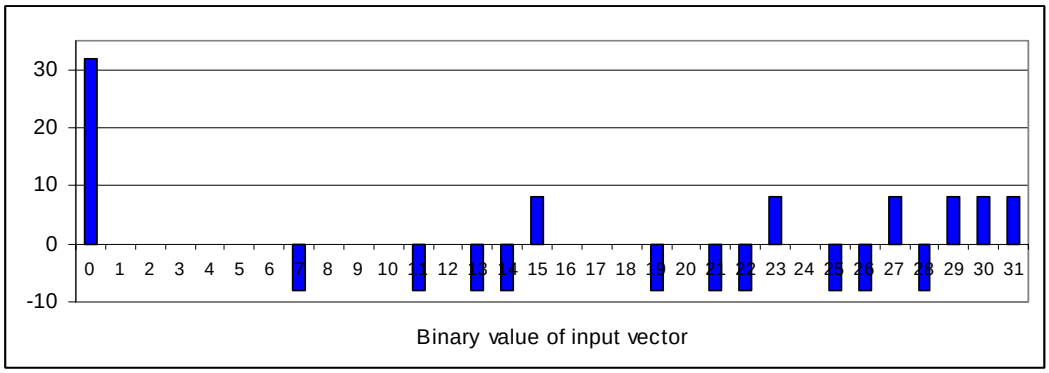


Figure 2.4 The autocorrelation function of $f_1(x)$

One expects that the propagation characteristics of $f(\mathbf{x})$ remain unaffected because from Theorem 2.2, we know that adding linear terms to a function does not affect the propagation characteristics. Fig. 2.4 demonstrates the autocorrelation function of $f_1(\mathbf{x})$. We see that $f_1(\mathbf{x})$ is $PC(2)$ because $r_{f_1}(\mathbf{d}) = 0$ for $1 \le w_H(\mathbf{d}) \le 2$. That is, the propagation characteristics of $f(\mathbf{x})$ and $f_1(\mathbf{x})$ are the same.

Let's define three more functions one of which is formed with the addition of a non-linear term and the other two are formed with the addition of linear term to $f(\mathbf{x})$.

$f_2(\mathbf{x}) = f(\mathbf{x}) + x_1 + x_2 + x_3 + x_4$ (Linear terms are added)

$f_3(\mathbf{x}) = f(\mathbf{x}) + x_2 + x_3 + x_5$ (Linear terms are added)

$f_4(\mathbf{x}) = f(\mathbf{x}) + x_1 x_2$ (Non-linear term is added)

In Fig 2.5, Fig 2.6, Fig 2.7 and Fig 2.8 the Walsh-Hadamard transforms and the autocorrelation functions of $f_2(\mathbf{x})$ and $f_3(\mathbf{x})$ are sketched respectively. From Fig. 2.6 and Fig 2.8 we see that $f_2(x)$ and $f_3(x)$ satisfy the propagation criterion of degree 2, ($PC(2)$).

$F_2(\mathbf{w})$ (Dyadic shift 11110 of F)

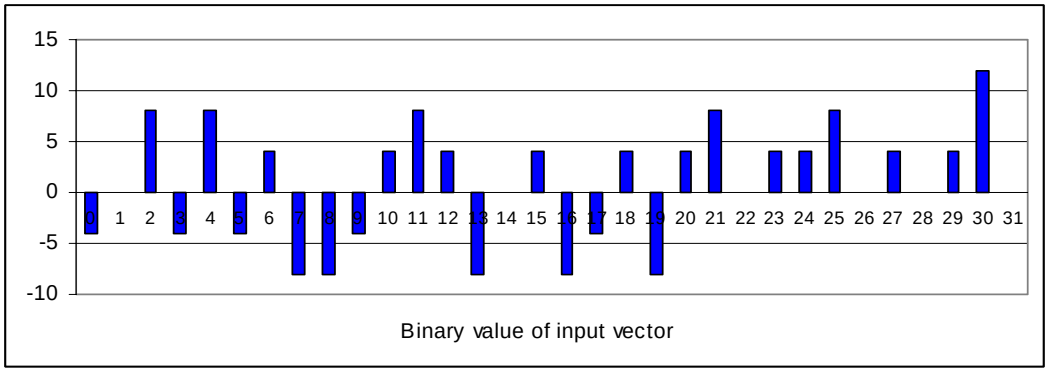


Figure 2.5 The Walsh-Hadamard transform of $f_2(x)$

$r_{f_2}(\mathbf{d})$

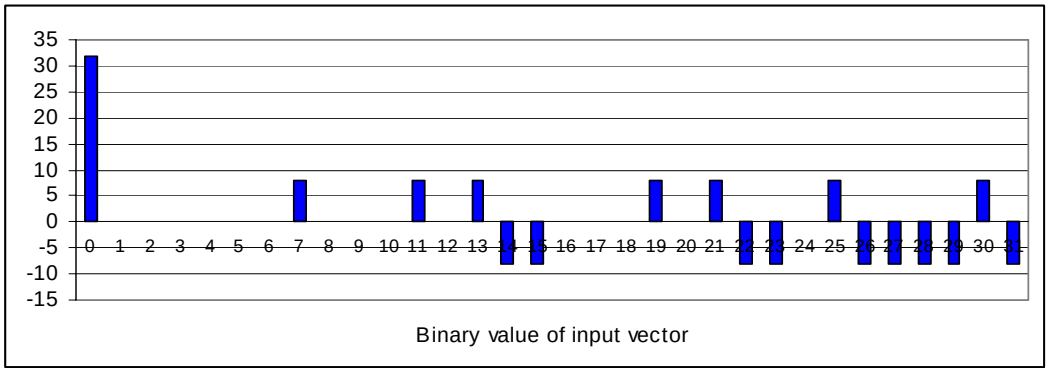


Figure 2.6 The autocorrelation function of $f_2(x)$

$F_3(\mathbf{w})$ (Dyadic shift 01101 of F)

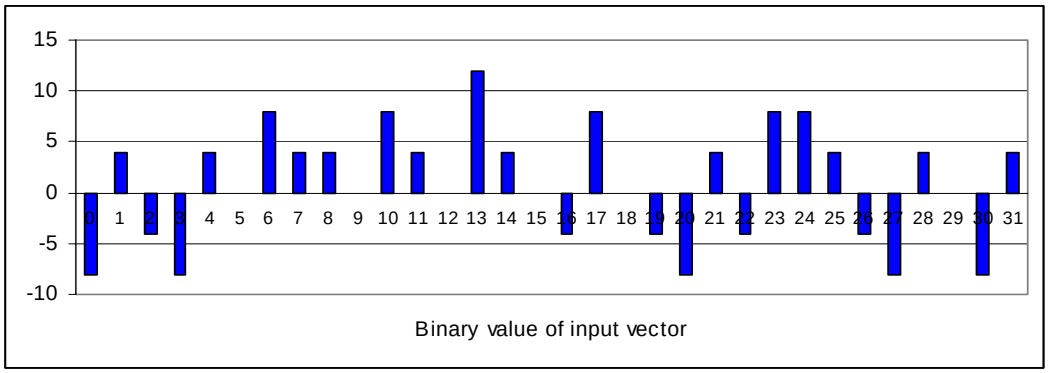


Figure 2.7 The Walsh-Hadamard transform of $f_3(x)$

$r_{f_3}(\mathbf{d})$

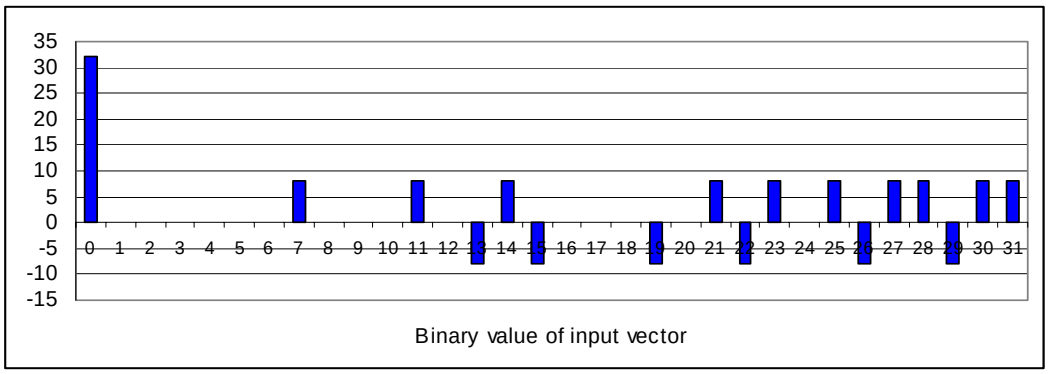


Figure 2.8 The autocorrelation function of $f_3(x)$

The function $f_4(\mathbf{x})$ is obtained with the addition of a non-linear term to $f(\mathbf{x})$. Fig. 2.9 and Fig. 2.10 show the Walsh-Hadamard transform and the autocorrelation function of $f_4(\mathbf{x})$. We see from Fig. 2.10 that $f_4(\mathbf{x})$ does not satisfy $PC(2)$.

$F_4(\mathbf{w})$

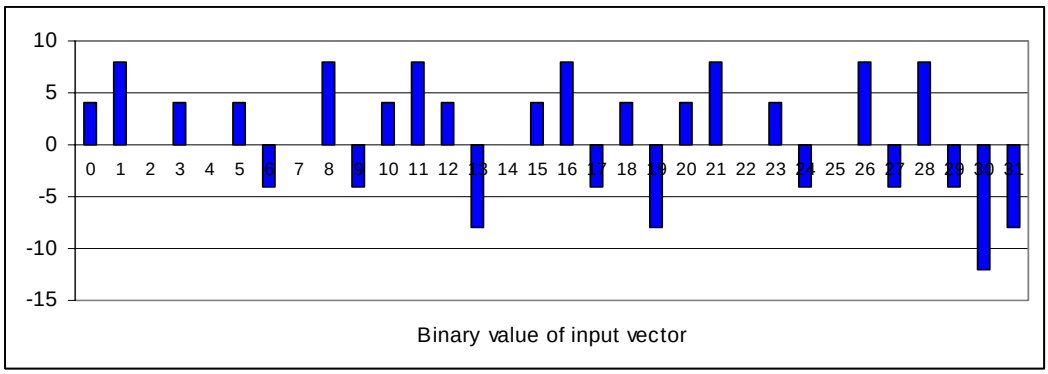


Figure 2.9 The Walsh-Hadamard transform of $f_4(x)$

$r_{f_4}(\mathbf{d})$

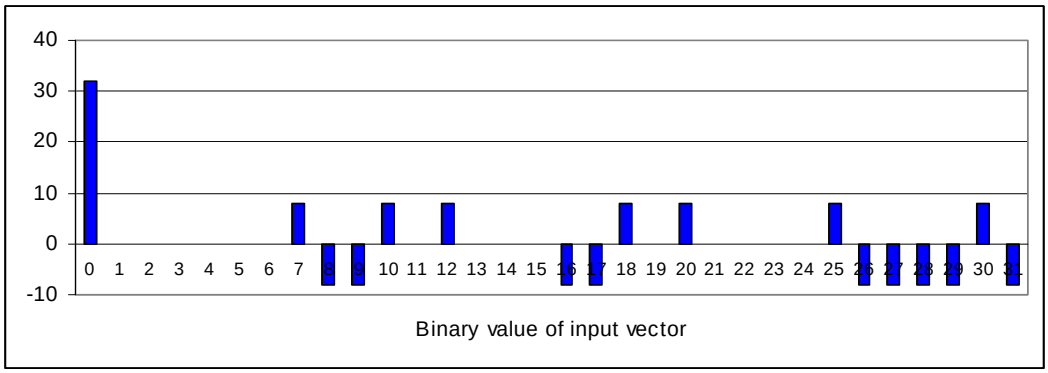


Figure 2.10 The autocorrelation function of $f_4(x)$.

In the figures, we observe that the resultant function will be balanced if dyadic shift of the spectrum is chosen accordingly. The propagation characteristics of a function may change with the addition of non-linear terms. Also note that $F(\mathbf{w})$ has 5 zeroes which means that 10 balanced functions can be obtained through addition of affine terms.

CHAPTER 3

THE TWOFISH ALGORITHM

3.1 Introduction

In response to a growing desire to replace DES (early encryption algorithm), National Institute of Standards and Technology (NIST) announced the Advanced Encryption Standard (AES) program in 1997. NIST solicited comments from the public on the proposed standard, and eventually issued a call for algorithms to satisfy the standard. The intention of NIST was to make all submissions public and eventually, through a process of public review and comment, choose a new encryption Standard to replace DES.

NIST specified several design criteria: a longer key length, larger block size, faster speed, and greater flexibility. Twofish is one of the submissions to the AES selection process. It meets all the required NIST criteria, 128-bit block; 128-, 192-, 256-bit key; efficient on various platforms; etc [Schneier, 1998].

In this chapter, we give the description of the building blocks of the Twofish algorithm.

3.2 TheTwofish Algorithm

Twofish is one of the submissions to the AES selection process. It meets all the required NIST criteria; 128-bit block, 128, 192, 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 [Schneier, 1998]. 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.

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

Feistel Networks: 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-depended 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):

$$F : \{0,1\}^{n/2} \times \{0,1\}^{N} \rightarrow \{0,1\}^{n/2}$$

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 a 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 bijective F function.

S-boxes: An S-box is a table-driven 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 $a \times b$ elements. Reed-Solomon (RS) error-correcting codes are known to be MDS. A necessary and sufficient condition for a $a \times b$ 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:

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

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

Twofish uses a 32-bit PHT to mix the outputs from its two parallel 32-bit *g* function.

Whitening: Whitening, the technique of XOR'ing key material before the first round and after the last round, substantially increases the difficulty of keysearch attacks against the remainder of the cipher. Twofish XORs 128 bits of subkey before the first Feistel round. These subkeys are calculated in the same manner as the round subkeys, but are not used any-where 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.

Figure 3.1 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 used as 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 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 prepares a total of forty 32-bit subkeys.

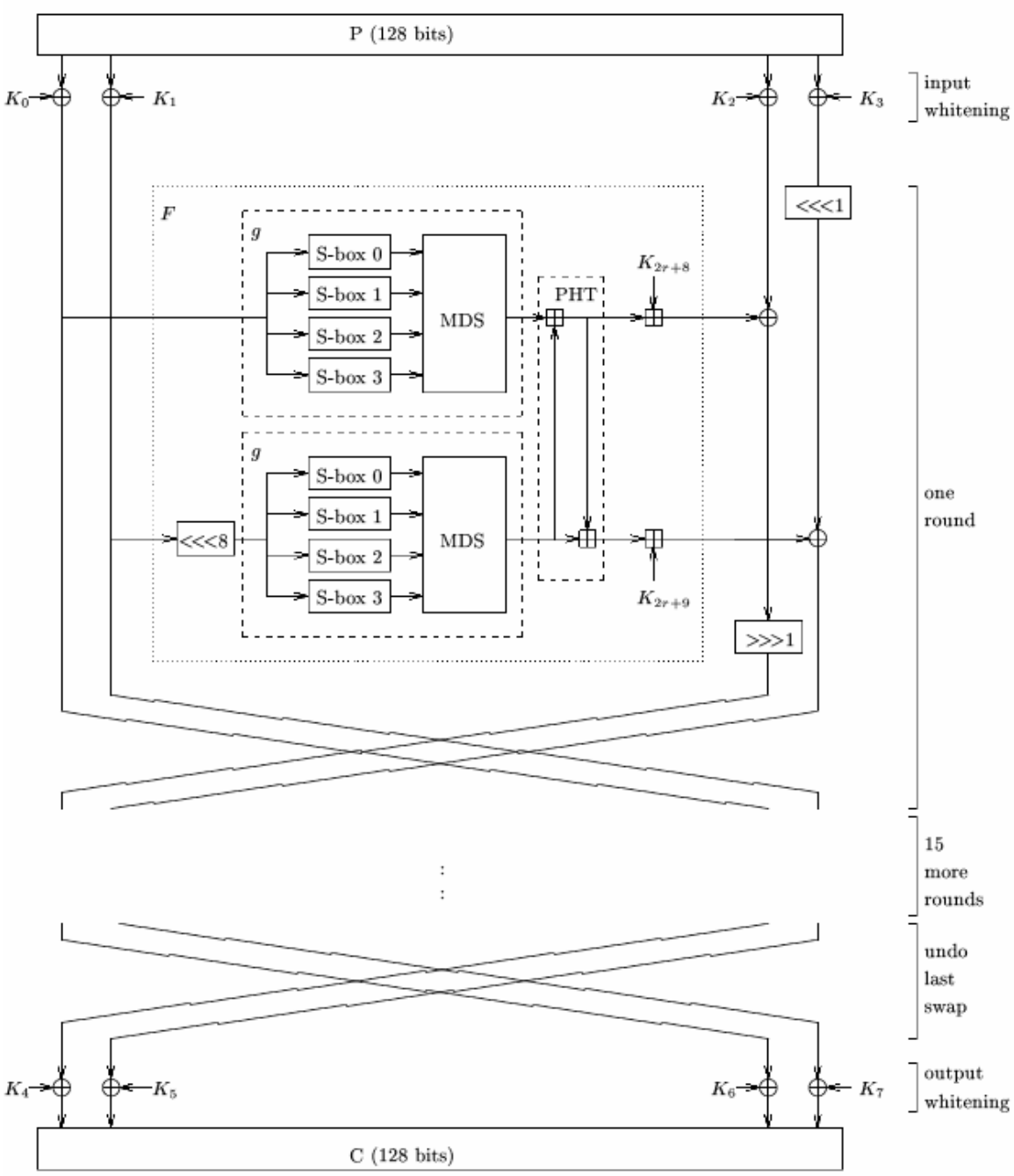


Figure 3.1 The Twofish encryption algorithm block.

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 byte of the plaintext) are first split into 4 words $P_0,.....,P_3$ of 32 bits each using the little-endian convention.

$$P_i = \sum_{j=0}^{3} p_{(4i+j)}.2^{8j} \qquad i = 0,.....,3 \tag{3.1}$$

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

$$R_{0,i} = P_i \oplus K_i \qquad i = 0,.....,3 \tag{3.2}$$

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. 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 inputs words R_{r+1} of the successive round are found as:

$$\begin{aligned}
&(F_{r,0}, F_{r,1}) = F(R_{r,0}, R_{r,1}, r) \\
&R_{r+1,0} = ROR(R_{r,2} \oplus F_{r,0}, 1) \\
&R_{r+1,1} = ROL(R_{r,3}, 1) \oplus F_{r,1} \\
&R_{r+1,2} = R_{r,0} \\
&R_{r+1,3} = R_{r,1}
\end{aligned} \tag{3.3}$$

for $r = 0,...,15$ 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

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

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

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

3.2.1 Main Functions of the Twofish Algorithm

a. The Function *g*

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

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

$$y_i = s_i[x_i] \qquad i = 0,.....,3 \tag{3.4}$$

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \tag{3.5}$$

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

In 3.4 s_i are the key-dependent S-boxes (S-box0 to S-box3) and the elements of the second 64-bit input $L = (I_{0,0}, I_{0,1}, I_{0,2}, I_{0,3}, I_{1,0}, I_{1,1}, I_{1,2}, I_{1,3})$ are used as the S-box constants, which are indicated in (3.8). The vector *L* is obtained from the keys. For MDS matrix multiplication (3.5) 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 $a = \sum_{i=0}^{7} a_i x^i$ with $a_i \in GF(2)$ is identified with the byte value $\sum_{i=0}^{7} a_i 2^i$. This is in some sense the "natural" mapping; addition in $GF(2^8)$ corresponds to a XOR of the bytes.

b. The Function *F*

The function *F* mentioned in (3.3) is a key-dependent permutation on 64-bit values. It takes three arguments, two input words R_0 and R_1 and the round number *r* used to select the appropriate subkeys. R_0 is passed through the *g* function, which yields $T_{r,0}$. 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 (3.13), 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 describes the details of *F* function.

$$\begin{aligned} T_{r,0} &= g(R_{r,0}, S) \\ T_{r,1} &= g(ROL(R_{r,1}, 8), S) \\ F_{r,0} &= (T_{r,0} + T_{r,1} + K_{2r+8}) \bmod 2^{32} \\ F_{r,1} &= (T_{r,0} + 2T_{r,1} + K_{2r+9}) \bmod 2^{32} \end{aligned} \tag{3.6}$$

where $(F_{r,0}, F_{r,1})$ is the result of *F*. Fig 3.2 shows the *F* function in detail, where (3.6) 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 (3.14). The round keys K_{2r+8} and K_{2r+9} used in (3.6) are produced in the upper part of Fig 3.2, as explained in (3.14). 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 odd indexed segments, so respectively, $M_e = (M_0, M_2)$ and $M_o = (M_1, M_3)$ as shown in (3.12).

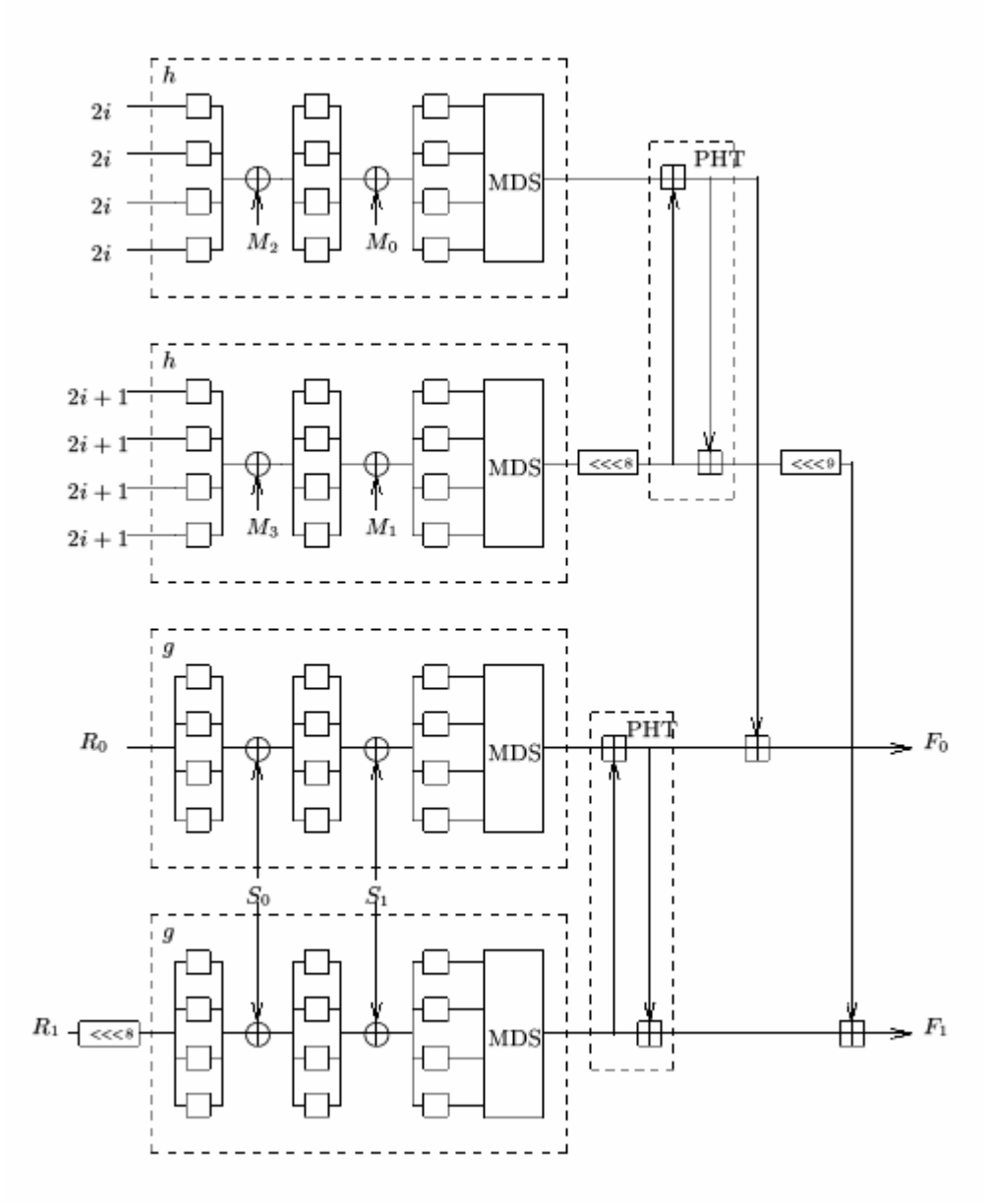


Figure 3.2 A view of a single round *F* function (128-bit key)

c. The Function *h*

The function $h(X, L)$ is used to obtain expanded keywords of the Twofish algorithm. *h* function is very similar to the function *g,* therefore equations (3.4) and (3.5) 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 32-bit word.

Note that the *h* and *g* functions are exactly sane 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 $p = (i\,i\,i\,i)$ where i is the 8-bit vector corresponding to $i = 0,....,39$. 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 3.2.2

d. Key-dependent S-boxes

The 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 (3.2.2). As can be seen from Fig 3.2 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 $p = X$ and $L = M_e$ or $L = M_o$. For $g(X, L)$ function the input parameters are $X = R_{r,0}$ or $X = ROL(R_{r,1}, 8)$ and $S = L$.

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

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

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

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

$$\begin{aligned} y_0 &= s_0[x_0] = q_1[q_0[q_0[x_0] \oplus l_{1,0}] \oplus l_{0,0}], \text{ (S-box0 formulation)} \\ y_1 &= s_1[x_1] = q_0[q_0[q_1[x_1] \oplus l_{1,1}] \oplus l_{0,0}], \text{ (S-box1 formulation)} \\ y_2 &= s_2[x_2] = q_1[q_1[q_0[x_2] \oplus l_{1,2}] \oplus l_{0,0}], \text{ (S-box2 formulation)} \\ y_3 &= s_3[x_3] = q_0[q_1[q_1[x_3] \oplus l_{1,3}] \oplus l_{0,0}], \text{ (S-box3 formulation)} \end{aligned} \tag{3.8}$$

The output of the S-boxes is the 32-bit word Y in the form of $y_3 y_2 y_1 y_0$. Figure 3.3 shows the S-box formulation of 128 bit the Twofish cipher.

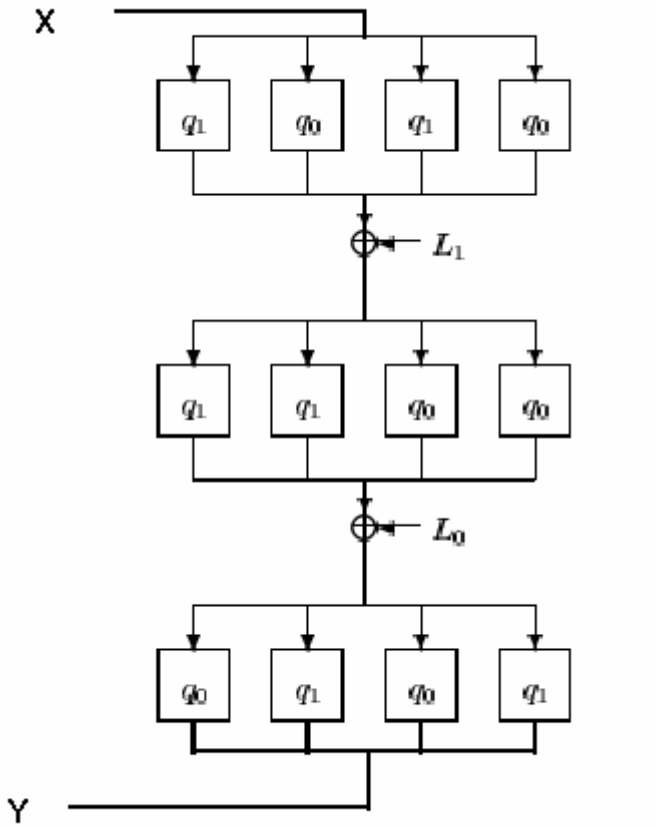


Figure 3.3 S-box formulation of the Twofish algorithm

3.2.2 Sub-functions of the Twofish Algorithm

a. 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:

$$a_0 = \lfloor x/16 \rfloor$$
$$b_0 = x \bmod 16$$
$$a_1 = a_0 \oplus b_0$$
$$b_1 = a_0 \oplus ROR(b_0, 1) \oplus (8a_0 \bmod 16)$$
$$a_2 = t_0(a_1)$$
$$b_2 = t_1(b_1)$$
$$a_3 = a_2 \oplus b_2$$
$$b_3 = a_2 \oplus ROR(b_2, 1) \oplus (8a_2 \bmod 16)$$
$$a_4 = t_2(a_3)$$
$$b_4 = t_3(b_3)$$

$$y = 16b_4 + a_4 \tag{3.9}$$

First, the byte is split into two nibbles. These 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 look-up. Finally, the two nibbles are recombined into a byte.

For the permutation q_0, look-up tables are given by

$$\begin{aligned}
t_0 &= [8\ \ 1\ \ 7\ \ D\ \ 6\ \ F\ \ 3\ \ 2\ \ 0\ \ B\ \ 5\ \ 9\ \ E\ \ C\ \ A\ \ 4] \\
t_1 &= [E\ \ C\ \ B\ \ 8\ \ 1\ \ 2\ \ 3\ \ 5\ \ F\ \ 4\ \ A\ \ 6\ \ 7\ \ 0\ \ 9\ \ D] \\
t_2 &= [B\ \ A\ \ 5\ \ E\ \ 6\ \ D\ \ 9\ \ 0\ \ C\ \ 8\ \ F\ \ 3\ \ 2\ \ 4\ \ 7\ \ 1] \\
t_3 &= [D\ \ 7\ \ F\ \ 4\ \ 1\ \ 2\ \ 6\ \ E\ \ 9\ \ B\ \ 3\ \ 0\ \ 8\ \ 5\ \ C\ \ A]
\end{aligned} \tag{3.10}$$

where each look-up 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 look-up tables are given by

$$\begin{aligned}
t_0 &= [2\ \ 8\ \ B\ \ D\ \ F\ \ 7\ \ 6\ \ E\ \ 3\ \ 1\ \ 9\ \ 4\ \ 0\ \ A\ \ C\ \ 5] \\
t_1 &= [1\ \ E\ \ 2\ \ B\ \ 4\ \ C\ \ 3\ \ 7\ \ 6\ \ D\ \ A\ \ 5\ \ F\ \ 9\ \ 0\ \ 8] \\
t_2 &= [4\ \ C\ \ 7\ \ 5\ \ 1\ \ 6\ \ 9\ \ A\ \ 0\ \ E\ \ D\ \ 8\ \ 2\ \ B\ \ 3\ \ F] \\
t_3 &= [B\ \ 9\ \ 5\ \ 1\ \ C\ \ 3\ \ D\ \ E\ \ 6\ \ 4\ \ 7\ \ F\ \ 2\ \ 0\ \ 8\ \ A]
\end{aligned} \tag{3.11}$$

b. Key Schedule

The key schedule has to provide 40 words of expanded key $K_0, \ldots\ldots K_{39}$, 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 $m_0, ..., m_{8k-1}$. To obtain the constant vectors for key dependent S-boxes, the bytes are first converted into 2k words of 32 bits each

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

and then into two word vectors of length k.

$$M_e = (M_0, M_2, ..., M_{2k-2})$$
$$M_o = (M_1, M_3, ..., M_{2k-1}) \tag{3.12}$$

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 $K_0, ..., K_{39}$. 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 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 RS code. Each result S_i of 4 bytes is then interpreted as a 32-bit word.

$$\begin{pmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,3} \end{pmatrix} = \begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & 5A & 58 & DB & 9E & 03 \end{pmatrix} \cdot \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix} \tag{3.13}$$

Using $S_i = \sum_{j=0}^{3} s_{i,j}.2^{8j}$ 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 multiply, $GF(2^8)$ 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 form the S-box constants. M_e and M_o are used in the *h* function which produces the expanded key; where as *S* is used in *g* function which encrypts the plaintext.

c. Expanded Key Words

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 $p = 2^{24} + 2^{16} + 2^8 + 2^0$. To evaluate the 40 keywords, one computes for all values of $i = 0,...,19$.

$$\begin{aligned} A_i &= h(2ip, M_e) \\ B_i &= ROL(h((2i+1)p, M_o), 8) \\ K_{2i} &= (A_i + B_i) \bmod 2^{32} \\ K_{2i+1} &= ROL((A_i + 2B_i) \bmod 2^{32}, 9) \end{aligned} \tag{3.14}$$

Notice that for producing A_i, the first argument of *h* function has all bytes values equal to 2*i,* 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 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 32-bit words of the expanded key.

CHAPTER 4

AVALANCHE CHARACTERISTICS OF TWOFISH

4.1 Avalanche Criteria

In this chapter, after stating the results of our avalanche tests, we compare them to the results of NIST. The analysis given in Section 4.3 explains the differences between these results.

The idea of avalanche was introduced by Feistel [Feistel, 1973]. 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 $n \times n$ function f satisfies this requirement, a large amount of plaintext pairs, P and P_i, such that P and P_i differ only in bit i are used to calculate the difference vectors, $\Delta C = f(P) \oplus f(P_i)$. These XOR sums are referred to as the avalanche vectors, each of which contains n bits, called avalanche variables.

If a function f has good avalanche characteristics, then whenever the i'th input bit complemented, each avalanche variable a_j, $1 \le j \le n$, should be equal to 1 or 0 with almost equal probabilities. Moreover, this property should be satisfied for all input bits i, $1 \le i \le n$ [WebTav, 1985]. The avalanche curve of a function is obtained using sufficiently large sets of input pairs (P, P_i), for a fixed i. Then the percentage of

changes (i.e., $a_j = 1$) of each avalanche variable, is sketched versus the output index *j*. This figure is called the avalanche curve of *f* corresponding to the change of the *i*'th input bit.

The following steps are used in the evaluation of the avalanche curves:

1. Set $i = 1$.
2. Choose a random key.
3. Set $k = 0$ and $\Delta C_k = (0, 0, ..., 0)_{1xn}$.
4. $k = k + 1$.
5. Choose a random plaintext P & $P' = P \oplus e_i$, where $e_i = (0...010...0)$ and 1 corresponds to i'th bit.
6. Encipher *P* and *P'* to obtain $\Delta C_k^i = f(P) \oplus f(P')$
7. $\Delta C_k^i = \Delta C_k^i \oplus \Delta C_{k-1}^i$ is the sum of avalanche vectors.
8. If $k < 10000$, go to 4.
9. $\Delta C_k^i = \frac{\Delta C_k^i}{10000} \times 100$ is the change percentage of the avalanche vector.
10. Sketch the avalanche curve, which is composed of the elements a_j^i of the $1 \times n$ vector $\Delta C_k^i = (a_1^i, ..., a_n^i)$ sketched versus the index $j = 1, ..., n$.
11. $i = i + 1$.
12. If $i \leq 128$, go to 2.
13. Stop.

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.

4.2 Avalanche Test Results of Twofish

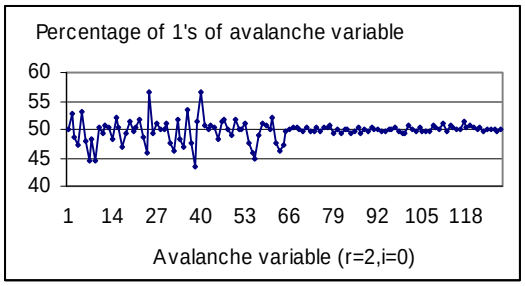
In this section, the avalanche characteristics of the Twofish cipher are investigated and its avalanche curves are sketched according to the algorithm given in Sec. 4.1.

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* is complemented. The avalanche behaviour of the cipher can be categorized into four cases, depending on the the position of the complemented input bit. These input intervals are found as $i \in [0,31],\ [32,63],\ [64,95],\ [96,127]$.

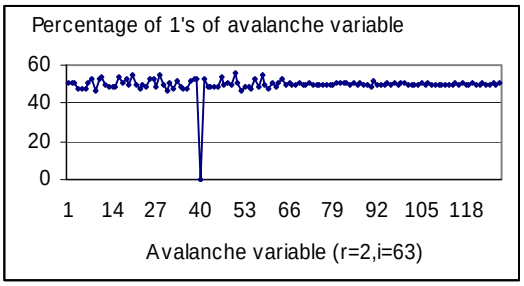
While sketching these curves at the output of the first round, it is observed that the number of average changes of each avalanche variable is very small, so there is no need to sketch this nonrandom characteristic for the first round.

In Fig. 4.1, the avalanche curves of Twofish for round 2 corresponding to four input intervals are sketched. For each input interval, the changed input bit *i* is chosen as the representative of the whole interval, i.e., *i*=0 for the first, *i*=63 for the second, *i*=64 for the third and *i*=127 for the last input intervals. Very similar curves are obtained when other input bits in the related intervals are changed. Fig. 4.2 and Fig. 4.3 are sketched in the same way, for rounds 3 and 4 of Twofish respectively.

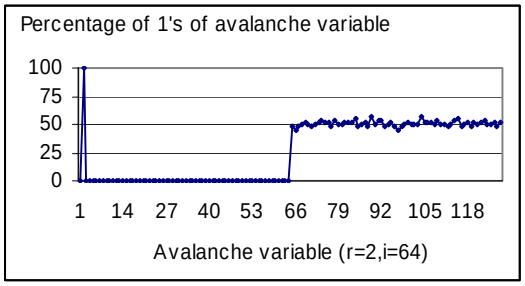
Fig. 4.1 part (i) shows that the round 2 output of Twofish is random if any bit *i* in the first input interval is complemented. Fig. 4.1, part (iii) and part (iv) show that for round 2 of Twofish, when a single bit *i* in the third and fourth input intervals is complemented, approximately half of the output bits never change. Fig. 4.1 part (ii) and Fig 4.2 part (iv) demonstrate that the avalanche curves for round 2 of Twofish corresponding to second input interval is similar to the avalanche curves for round 3 corresponding to fourth input interval. From Fig. 4.3, it can be understood that for round 4 of Twofish, all of the output bits change with a probability close to ½ whenever a single input bit *i* in any one of four input intervals is complemented. The only difference between Fig. 4.2 and Fig. 4.3 is that, for round 3, 39'th bit of the output from Twofish never changes for a single input bit complementation in the fourth input interval.



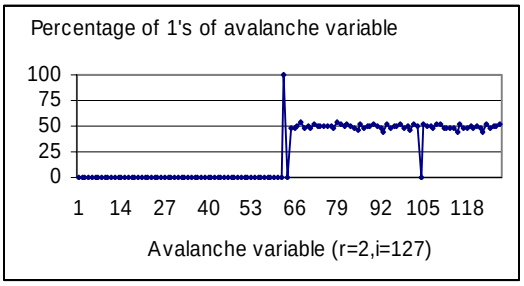
(i)



(ii)

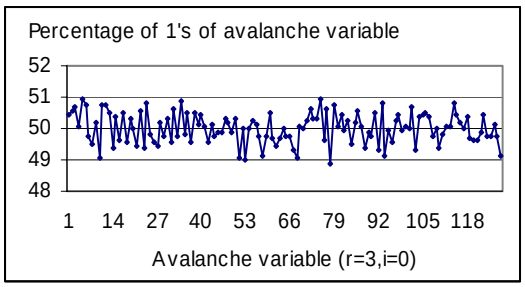


(iii)

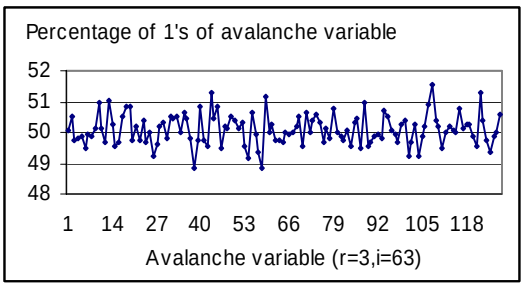


(iv)

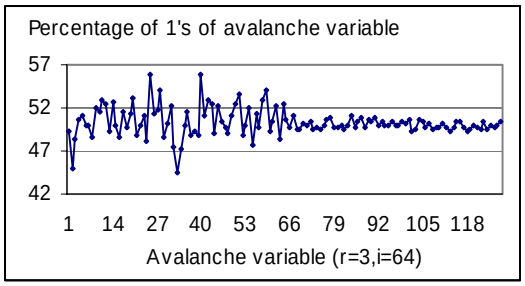
Figure 4.1 Avalanche curves of Twofish for round 2 and chosen error bit position in the (i) first (ii) second (iii) third (iv) fourth input intervals.



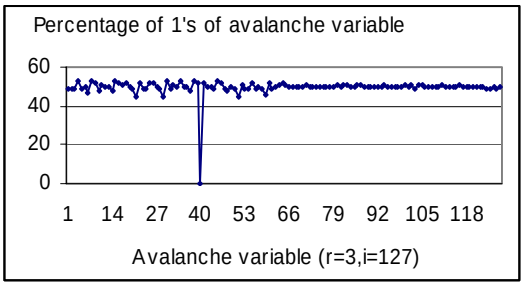
(i)



(ii)



(iii)



(iv)

Figure 4.2 Avalanche curves of Twofish for round 3 and chosen error bit position in the (i) first (ii) second (iii) third (iv) fourth input intervals.

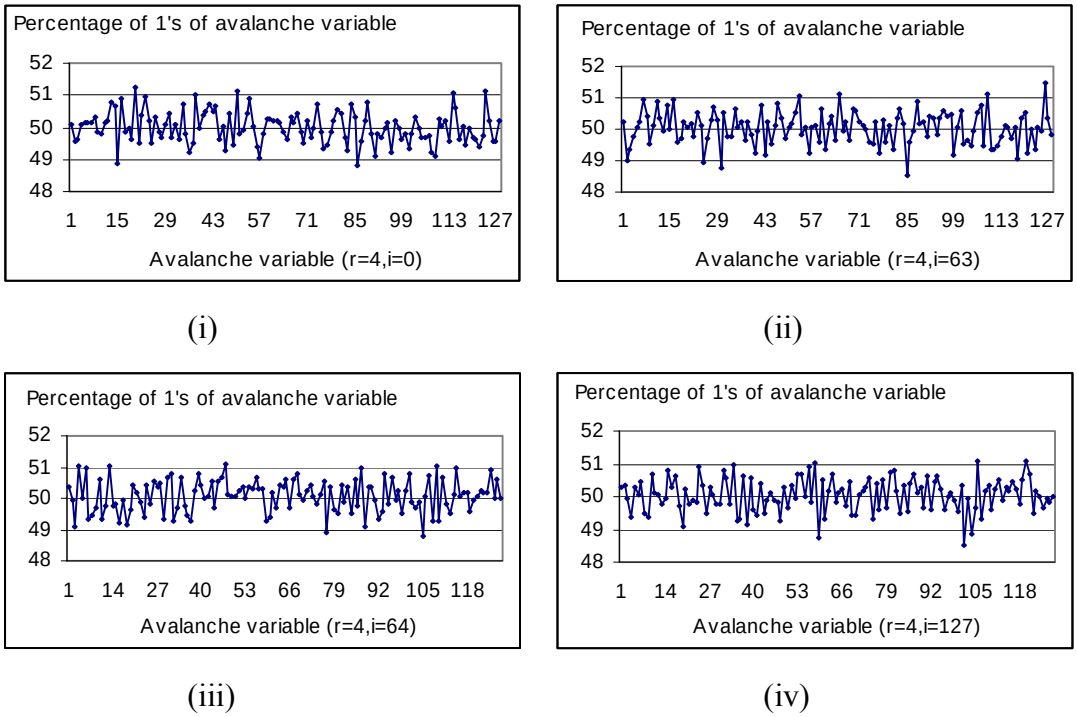


Figure 4.3 Avalanche curves of Twofish for round 4 and chosen error bit position in the (i) first (ii) second (iii) third (iv) fourth input intervals.

The avalanche test results we obtain confirm those calculated by Arıkan [Arıkan, 2003], which show that the output of Twofish is not random for rounds 2 and 3. However, the curves we obtain (Fig. 4.1 and Fig. 4.2) for the second and third round outputs of Twofish are completely different from those calculated by Arıkan [Arıkan03]. The main difference is that the avalanche curves calculated by Arıkan for round 2 outputs of Twofish are the same as those calculated for round 3, whereas we see from Fig. 4.1 and Fig. 4.2 that the randomness of the second round output is much worse than that of round 3.

Moreover, Fig. 4.1 part (i) shows that the round 2 output of Twofish is random if any bit *i* in the first input interval is complemented whereas the results calculated by Arıkan shows that the round 2 output of Twofish is not random if any bit *i* in the first input interval is complemented [Arıkan, 2003].

For better understanding of the differences between our results and those calculated by Arıkan [Arıkan, 2003], we derive the outputs for round 2 and 3 of the Twofish

algorithm in terms of the input plaintext words in Sec 4.3. The detailed explanations of the Figures 4.1, 4.2 and 4.3 and comparison of avalanche tests with the results of NIST Statistical Test Suite are given in Sec 4.3 and Sec 4.4 respectively.

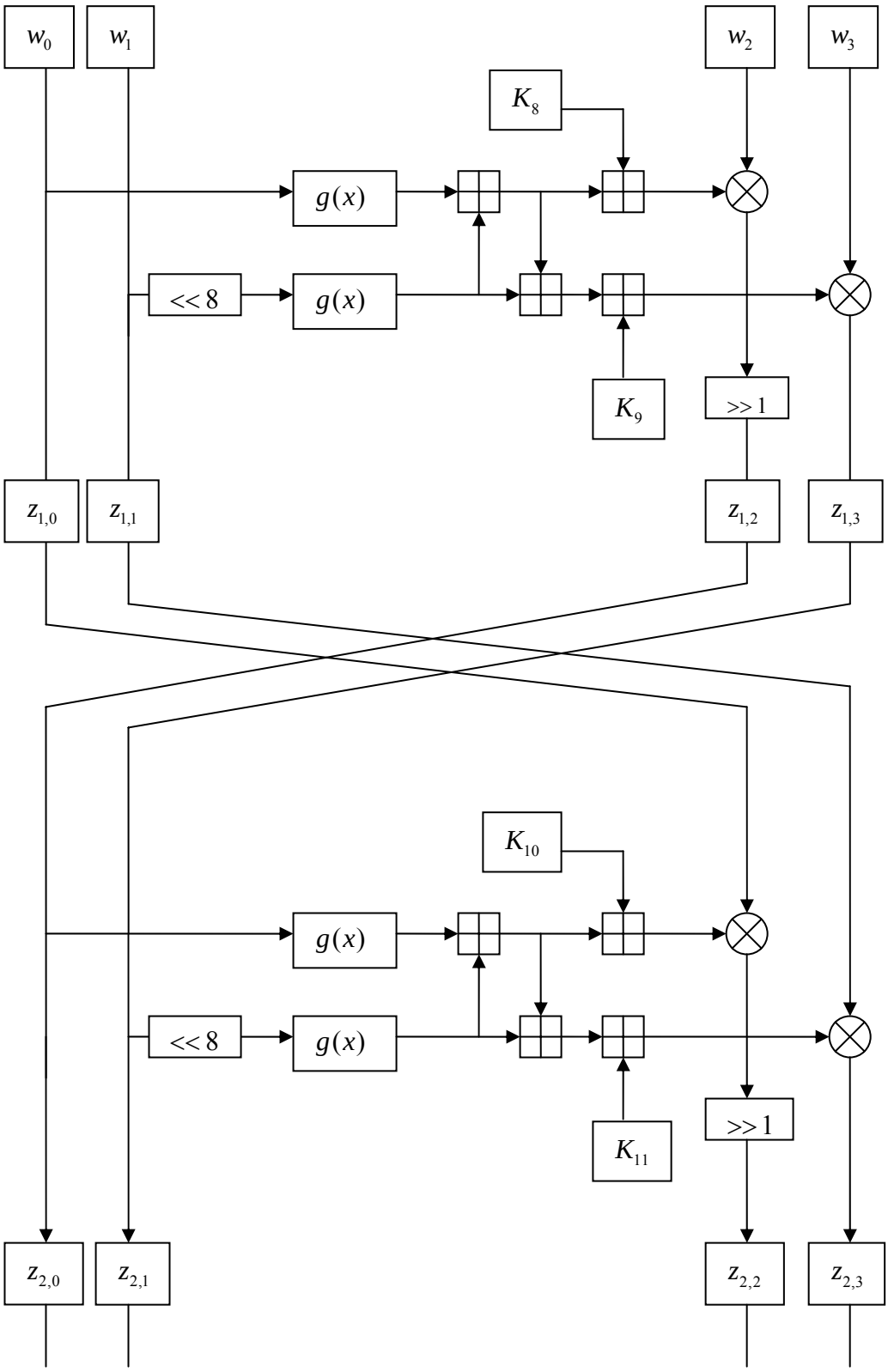
4.3 Analysis of the Test Results for Rounds 2 and 3

In this section we try to explain why the 2^{nd} and 3^{rd} round outputs of Twofish fail in randomness tests. For example, round 2 results of Fig. 4.1 part (ii) demonstrate that the avalanche variable is not random at a single point. We show that this point corresponds to the 39'th avalanche variable. The randomness of the second round output is much worse, in parts (iii) and (iv) of Fig. 4.1, where almost half of the output bits are not changed at all, with a complemented input bit at position *i*, where $64 \le i \le 127$.

Moreover, round 3 results of Fig 4.2 part (iv) show that the avalanche variable at position 39 is not random when a bit at position *i* is complemented, where $96 \le i \le 127$.

For better understanding of the nonrandomness of some avalanche variables, one needs to derive the outputs for round 2 and 3 of the Twofish algorithm in terms of the input plaintext words. Fig. 4.4 is sketched to demonstrate the details of the first 3 rounds of Twofish.

We notice that the function $g : F_2^{32} \to F_2^{32}$ is crucial in determining the randomness of the round output. Therefore, we first apply randomness tests to the function *g*, (see section 4.3.1) and observe that its output is completely random.


w_0
w_1
w_2
w_3
K_8
$g(x)$
$<<8$
$g(x)$
K_9
$>>1$
$z_{1,0}$
$z_{1,1}$
$z_{1,2}$
$z_{1,3}$
K_{10}
$g(x)$
$<<8$
$g(x)$
K_{11}
$>>1$
$z_{2,0}$
$z_{2,1}$
$z_{2,2}$
$z_{2,3}$

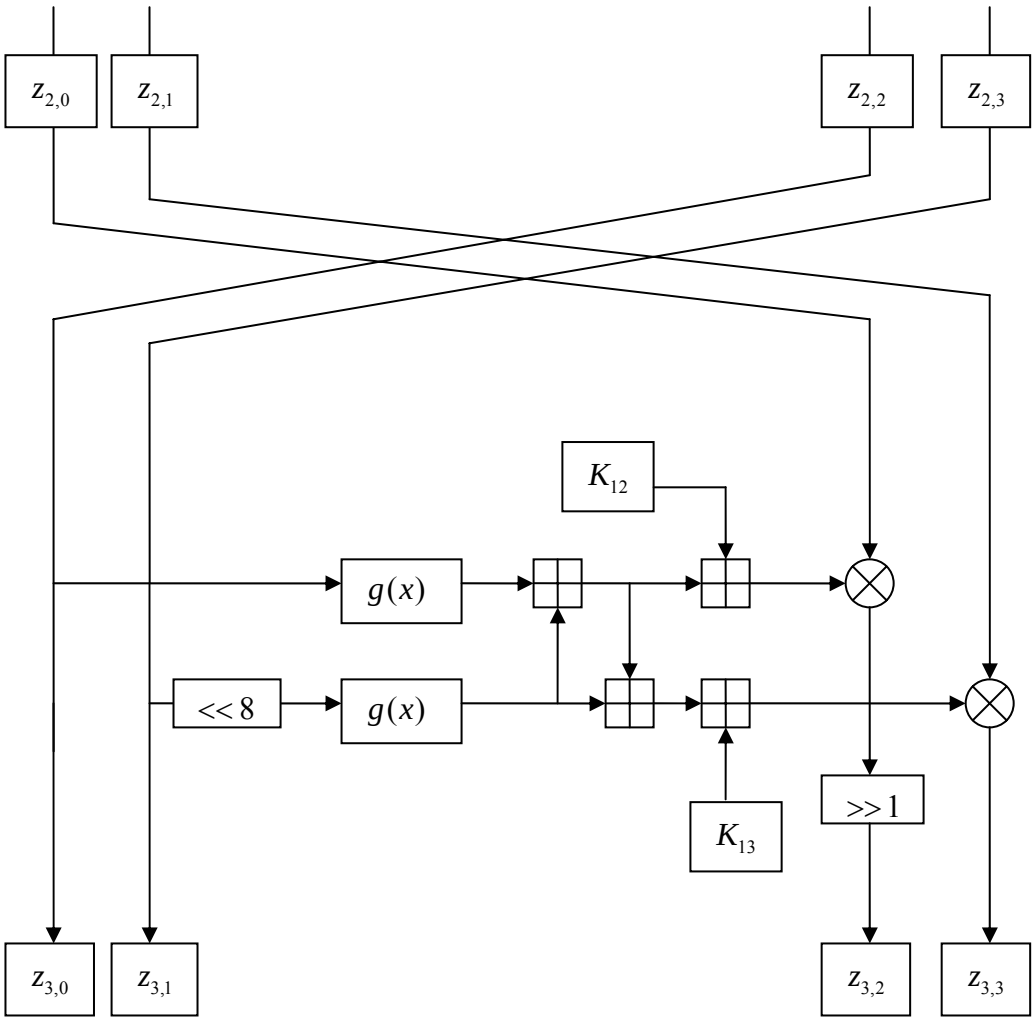


Figure 4.4 The Twofish algorithm for rounds 1,2 and 3.

Let $P = m_0 m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8 m_9 m_{10} m_{11} m_{12} m_{13} m_{14} m_{15}$ (m_0 is the most significant byte of the plaintext, and m_{15} is the least significant byte of the plaintext) be 128-bit input plaintext of the Twofish algorithm. The plaintext *P* is splitted into 32-bit words by using little-endian conversion (3.1) and XORed with constant keys (which are obtained by 3.14)**,** which is called input whitening step (3.2).

$$
\begin{aligned}
w_0 &= m_3 m_2 m_1 m_0 \oplus K_0 \\
w_1 &= m_7 m_6 m_5 m_4 \oplus K_1 \\
w_2 &= m_{11} m_{10} m_9 m_8 \oplus K_2 \\
w_3 &= m_{15} m_{14} m_{13} m_{12} \oplus K_3
\end{aligned} \tag{4.1}
$$

After input whitening step the round operation (3.6) is applied on the input words. Then round 1 output words of Twofish will be:

$$
\begin{aligned}
z_{1.0} &= w_0 \\
z_{1.1} &= w_1 \\
z_{1.2} &= ROR\left(\left[(g(w_0) + g(ROL(w_1,8)) + K_8) \ \oplus \ w_2\right],1\right) \\
z_{1.3} &= \left[g(w_0) + 2g(ROL(w_1,8)) + K_9\right] \ \oplus \ ROL(w_3,1)
\end{aligned} \tag{4.2}
$$

where *ROR* and *ROL* are the functions that rotate their first argument right or left by the number of bits indicated by their second argument.

The swap operation is applied to round 1 output words and then the swapped words are used as the input words of the second round, i.e., $w_{2.0} = z_{1,2}$, $w_{2.1} = z_{1,3}$, $w_{2.2} = z_{1,0}$ and $w_{2.3} = z_{1,1}$.

The output words of round 2 are obtained as,

$$
\begin{aligned}
z_{2.0} &= ROR\left(\left[(g(w_0) + g(ROL(w_1,8)) + K_8) \ \oplus \ w_2\right],1\right) \\
z_{2.1} &= \left[g(w_0) + 2g(ROL(w_1,8)) + K_9\right] \ \oplus \ ROL(w_3,1)
\end{aligned}
$$

$$z_{2.2} = ROR\left(\left[\begin{array}{l} g\left(ROR\left(\left[(g(w_0)+g(ROL(w_1,8))+K_8) \ \oplus \ w_2\right],1\right)\right) \\ \quad + \ g\left(ROL\left(\begin{array}{l}\left[g(w_0)+2g(ROL(w_1,8))+K_9\right] \\ \oplus \ ROL(w_3,1),8\end{array}\right)\right)+K_{10} \\ \oplus \ w_0 \end{array}\right],1\right)$$

$$z_{2.3} = \left(\left[\begin{array}{l} g\left(ROR\left(\left[(g(w_0)+g(ROL(w_1,8))+K_8) \ \oplus \ w_2\right],1\right)\right) \\ +2g\left(ROL\left(\begin{array}{l}\left[g(w_0)+2g(ROL(w_1,8))+K_9\right] \\ \oplus \ ROL(w_3,1),8\end{array}\right)\right)+K_{11} \end{array}\right] \right) \qquad (4.3)$$
$$\oplus \ ROL(w_1,1)$$

Round 2 output words are swapped and then used as the input words of the 3^{rd} round, i.e, $w_{3.0} = z_{2,2}$, $w_{3.1} = z_{2,3}$, $w_{3.2} = z_{2,0}$ and $w_{3.3} = z_{2,1}$.

After another round operation, the output words of the third round of Twofish will be:

$$z_{3.0} = ROR\left(\left[\begin{array}{l} g\left(ROR\left(\left[(g(w_0)+g(ROL(w_1,8))+K_8) \ \oplus \ w_2\right],1\right)\right) \\ \quad + \ g\left(ROL\left(\begin{array}{l}\left[g(w_0)+2g(ROL(w_1,8))+K_9\right] \\ \oplus \ ROL(w_3,1),8\end{array}\right)\right)+K_{10} \\ \oplus \ w_0 \end{array}\right],1\right)$$

$$z_{3.1} = \left(\left[\begin{array}{l} g\left(ROR\left(\left[(g(w_0)+g(ROL(w_1,8))+K_8) \ \oplus \ w_2\right],1\right)\right) \\ +2g\left(ROL\left(\begin{array}{l}\left[g(w_0)+2g(ROL(w_1,8))+K_9\right] \\ \oplus \ ROL(w_3,1),8\end{array}\right)\right)+K_{11} \end{array}\right] \right)$$
$$\oplus \ ROL(w_1,1)$$

$$z_{3.2} = ROR\left(\left[\begin{array}{l} g\left(\begin{array}{l} ROR\left(\left[\begin{array}{l} g\left(\begin{array}{l} ROR\left(\left[\begin{array}{l} g\left(\begin{array}{l} ROR\left(\left[\begin{array}{l} (g(w_0)+g(ROL(w_1,8))) \\ +K_8 \\ \oplus\ w_2 \end{array}\right],1\right) \\ +g\left(\begin{array}{l} ROL\left(\left[\begin{array}{l} g(w_0)+2g(ROL(w_1,8)) \\ +K_9 \end{array}\right] \\ \oplus\ ROL(w_3,1),8 \end{array}\right) \\ +K_{10} \\ \oplus\ w_0 \end{array}\right),1\right) \\ +g\left(\begin{array}{l} ROL\left(\left[\begin{array}{l} g\left(\begin{array}{l} ROR(\left[\begin{array}{l} (g(w_0)+g(ROL(w_1,8))) \\ +K_8 \\ \oplus\ w_2 \end{array}\right],1) \\ +2g\left(\begin{array}{l} ROL(\left[\begin{array}{l} g(w_0)+2g(ROL(w_1,8)) \\ +K_9 \end{array}\right] \\ \oplus\ ROL(w_3,1),8) \end{array}\right) \\ +K_{11} \\ \oplus\ ROL(w_1,1),8 \end{array}\right]\right) \\ +\mathrm{K}_{12} \end{array}\right) \\ \oplus\ ROR\left(\left[(g(w_0)+g(ROL(w_1,8))+K_8)\ \oplus\ w_2\right],1\right) \end{array}\right),1\right)$$

$$
z_{3.3} = \left(\begin{bmatrix} g\left(ROR\left(\begin{bmatrix} g\left(ROR\left(\begin{bmatrix} g\left(ROR\left(\begin{bmatrix} (g(w_0) + g(ROL(w_1,8)) + K_8) \\ \oplus \quad w_2 \end{bmatrix},1 \right) \right) \\ + \; g\left(ROL\left(\begin{bmatrix} g(w_0) + 2g(ROL(w_1,8)) \\ +K_9 \end{bmatrix} \right) \right) \\ \oplus \quad ROL(w_3,1),8 \end{bmatrix} \right) \\ +K_{10} \\ \oplus \quad w_0 \end{bmatrix},1 \right) \right) \\ +2g\left(ROL\left(\begin{bmatrix} g\left(ROR\left(\begin{bmatrix} (g(w_0) + g(ROL(w_1,8)) + K_8) \\ \oplus \quad w_2 \end{bmatrix},1 \right) \right) \\ +2g\left(\begin{bmatrix} ROL(\left[g(w_0) + 2g(ROL(w_1,8)) + K_9 \right] \\ \oplus \quad ROL(w_3,1),8) \end{bmatrix} \right) \\ +K_{11} \\ \oplus \quad ROL(w_1,1),8 \end{bmatrix} \right) \right) \\ +\mathrm{K}_{13} \end{bmatrix} \\ \left[g(w_0) + 2g(ROL(w_1,8)) + K_9 \right] \quad \oplus \quad ROL(w_3,1) \end{bmatrix} \right) \tag{4.4}
$$

4.3.1 Randomness of the *g* Function

The theoretical explanations given in Sec. 4.3.2 are valid only if the *g* function has completely random characteristics. In this section, we investigate the random characteristics of the *g* function according to the avalanche criterion. We obtain the avalanche curves of the *g* function by counting the number of changes at each position of the output vector, when a specific bit at position *i* if complemented for a set of *N*=100000 different plaintexts. The test procedure is repeated for each of the 32 input bits of the *g* function. To satisfy the avalanche criterion, a single bit change in the input should yield the output of each bit change with a probability of one half.

In Fig. 4.5, avalanche curve of the *g* function of Twofish, with the complemented input bit position chosen as 0, is sketched. Very similar curves are obtained when the remaining 31 input bits are changed separately.

From Fig. 4.5, it can be understood that the output from *g* function changes with a probability close to ½ whenever a single input bit *i* in any one of 32 input bits is complemented. It is clear that *g* function has completely random characteristics, with respect to the avalanche criterion.

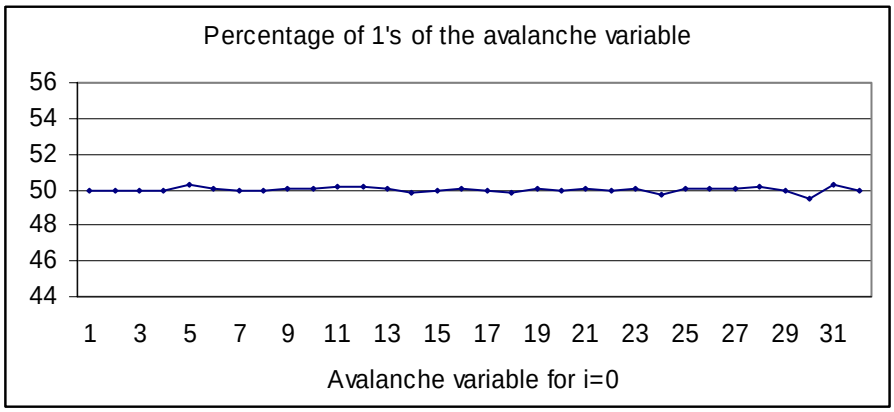


Figure 4.5 Avalanche curve of the *g* function for the complemented input bit at position 0.

4.3.2 Analysis of the Avalanche Test Results of Twofish for Round 2

Fig. 4.1 part (ii) shows that for round 2, when a plaintext bit in the second input interval is complemented, the avalanche variable at position 39 of ciphertext never changes. Similarly, Fig. 4.1 part (iii) shows that when a plaintext bit in the third input interval is complemented, the avalanche variable at position 2 changes with a probability of 1. So, for round 2, Twofish does not satisfy the avalanche criterion of randomness.

To explain this behaviour, let C_2 and C_2' be the round 2 outputs of Twofish when P and P' are the corresponding inputs, respectively. The ciphertext C_2 for round 2 will be obtained from output words of round 2 by using the same little-endian conversion used for the plaintext.

$$
\begin{aligned}
c_0 &= z_{2,0} \oplus K_4 = m'_3\, m'_2\, m'_1\, m'_0 \\
c_1 &= z_{2,1} \oplus K_5 = m'_7\, m'_6\, m'_5\, m'_4 \\
c_2 &= z_{2,2} \oplus K_6 = m'_{11}\, m'_{10}\, m'_9\, m'_8 \\
c_3 &= z_{2,3} \oplus K_7 = m'_{15}\, m'_{14}\, m'_{13}\, m'_{12} \\
C_2 &= m'_0\, m'_1\, m'_2\, m'_3\, m'_4\, m'_5\, m'_6\, m'_7\, m'_8\, m'_9\, m'_{10}\, m'_{11}\, m'_{12}\, m'_{13}\, m'_{14}\, m'_{15}
\end{aligned}
\tag{4.5}
$$

where m'_0 is the most significant byte, and m'_{15} is the least significant byte of the ciphertext.

The analysis of the avalanche test results for round 2 of Twofish is given below for four different intervals of the chosen error bit position, which are $i \in [0,31]$, $[32,63]$, $[64,95]$, $[96,127]$.

Interval i ($0 \le i \le 31$)

For this case, error bit position corresponds to the first input word $w_0 = m_3 m_2 m_1 m_0 \oplus K_0$ given by (4.1). Then the round 2 output of the Twofish algorithm will be random as observed in Fig. 4.1 (i) because all output words $z_{2.0}, z_{2.1}, z_{2.2}$ and $z_{2.3}$ given by (4.3) depend on $g(w_0)$. So the percentage of 1's will be approximately ½ for all 128 avalanche variables.

We see that if even one bit in w_0 is complemented, all 128 bits of the ciphertexts C_2 and C_2' will change randomly, with probability ½.

Interval ii $(32 \le i \le 63)$

For this case, the error bit position is chosen in the second input word $w_1 = m_7 m_6 m_5 m_4 \oplus K_1$ and Fig. 4.1 (ii) is obtained as the result of the avalanche tests. Then from (4.3),

1. All 32 bits of $z_{2.0}$ differ randomly for C_2 and C_2' because $z_{2.0}$ depends on $g(ROL(w_1, 8))$ and one bit of w_1 is different for P and P'.

2. $z_{2.1}$ changes randomly in all 32 bits for C_2 and C_2', except for the bit 0, because;
   - $g(w_0)$ is the same for C_2 and C_2' because w_0 is the same for P and P'.
   - $ROL(w_3, 1)$ is the same for C_2 and C_2' because w_3 is the same for P and P'.
   - $g(ROL(w_1, 8))$ changes randomly for C_2 and C_2' in all 32 bits because w_1 differs in 1 bit for P and P'. But because of multiplication by 2, bit 0 of $2g(ROL(w_1, 8))$ is the same for C_2 and C_2', because the least significant bit of even numbers is always 0.

   So $c_1 = m'_7\, m'_6\, m'_5\, m'_4 = z_{2,1} \oplus K_5$ is the same in bit 0 of m'_4 for C_2 and C_2' and all other 31 bits change randomly. From (4.5), one can see that bit 0 of m'_4 corresponds to bit 39 of C_2 and C_2'.

3. All 32 bits of $z_{2.2}$ changes randomly for C_2 and C_2' because $z_{2.2}$ is directly related to $g\left(ROR\left(\left[(g(w_0) + g(ROL(w_1, 8)) + K_8) \oplus w_2\right], 1\right)\right)$ which depends on $g(ROL(w_1, 8))$.

4. All 32 bits of $z_{2.3}$ will be random for C_2 and C_2' because $z_{2.3}$ is directly related to $g\left(ROR\left(\left[(g(w_0) + g(ROL(w_1, 8)) + K_8) \oplus w_2\right], 1\right)\right)$ which depends on $g(ROL(w_1, 8))$.

So the percentage of 1's will be approximately ½ for all avalanche variables except the 39^{th} avalanche variable, which is equal to bit 0 of m'_4.

Interval iii $(64 \le i \le 95)$

Error bit position is chosen in the third input word $w_2 = m_{11}m_{10}m_9m_8 \oplus K_2$. In this case, Figure 4.1 (iii) can be explained as follows;

1. Only one bit of $z_{2.0}$ will be different for C_2 and C_2' and all other 31 bits of $z_{2.0}$ will be the same, because from (4.3) we see that $z_{2.0}$ does not depend on $g(w_2)$. From $z_{2.0}$, one can calculate the position of this bit, which will be different for C_2 and C_2', as $n = (8m + k + 1) \bmod 32$ where,
   k = remainder of $(i - 64)$ divided by 8.
   m = fractional part of $(i - 64)$ divided by 8.
   n = position of the output difference bit.

2. $z_{2.1}$ given by (4.3) is the same for C_2 and C_2' in all 32 bits because it does not depend on w_2.

3. All 32 bits of $z_{2.2}$ will be random for C_2 and C_2' because $z_{2.2}$ given by (4.3) is directly related to $g\left(ROR\left(\left[(g(w_0) + g(ROL(w_1,8)) + K_8) \oplus w_2\right],1\right)\right)$ which depends on w_2 and 1 bit of w_2 is different for P and P'.

4. All 32 bits of $z_{2.3}$ will change randomly for C_2 and C_2' because $z_{2.3}$ is directly related to $g\left(ROR\left(\left[(g(w_0) + g(ROL(w_1,8)) + K_8) \oplus w_2\right],1\right)\right)$ which depends on w_2. Notice that all 32 bits of the term $2g\left(ROL\left(\left[\, g(w_0) + 2g(ROL(w_1,8)) + K_9\right] \oplus ROL(w_3,1),8\right)\right)$ in (4.3) are the same for both C_2 and C_2'. But because of the first term of $z_{2,3}$ which contains

the term $g(w_2)$, all 32 bits of $c_3 = m'_{15}\, m'_{14}\, m'_{13}\, m'_{12}$ will change randomly for C_2 and C_2'.

Because of parts 3 and 4, the percentage of 1's will be approximately ½ for 64 (64-127) avalanche variables. Only one avalanche variable considered in part 1 is always 1, whose position depends on the complemented plaintext bit, and other 63 avalanche variables are always equal to 0.

Interval iv ($96 \le i \le 127$)

Error bit position is chosen in the fourth input word $w_3 = m_{15}m_{14}m_{13}m_{12} \oplus K_3$, and Fig. 4.1 (iv) is obtained. Because, a close inspection of (4.3) shows that:

1. $z_{2.0}$ is the same for C_2 and C_2' in all 32 bits because it does not depend on w_3.

2. Only one bit of $z_{2.1}$ is different for C_2 and C_2'. From $z_{2.1}$, one can calculate the position of this bit as $n = (8m + k - 1) \bmod 32 + 32$ where,
   k = remainder of $(i - 96)$ divided by 8.
   m = fractional part of $(i - 96)$ divided by 8.
   n = position of the output difference bit.

3. $z_{2.2}$ will change randomly for C_2 and C_2' in all 32 bits because $z_{2.2}$ is directly related to $g\left(ROL\left(\left[g(w_0) + 2g(ROL(w_1,8)) + K_9\right] \oplus ROL(w_3,1),8\right)\right)$ which depends on w_3 and one bit of w_3 is different for P and P'.

4. $z_{2.3}$ will change randomly in all 32 bits for C_2 and C_2' except bit 0, because of the $2g(w_3)$ term it contains. That is, in (4.3),
   - $g\left(ROR\left(\left[(g(w_0) + g(ROL(w_1,8)) + K_8) \oplus w_2\right],1\right)\right)$ will be the same for C_2 and C_2' because w_0, w_1, w_2 are the same for P and P'.
   - $ROL(w_1,1)$ is the same for C_2 and C_2' because w_1 is the same for P and P'.

- $g\left(ROL\left(\left[g(w_0)+2g(ROL(w_1,8))+K_9\right] \oplus ROL(w_3,1),8\right)\right)$ will change randomly for C_2 and C_2' in all 32 bits because w_3 differs in 1 bit for P and P'. But $2\,g\left(ROL\left(\left[g(w_0)+2g(ROL(w_1,8))+K_9\right] \oplus ROL(w_3,1),8\right)\right)$ is the same only in bit 0 because of multiplication by 2.

As a result, we see that for round 2, Twofish does not satisfy the avalanche criterion. Because, if the i'th ($64 \le i$) bit of the plaintext is complemented, the second round output is not random as seen in parts (ii), (iii) and (iv) of Fig. 4.1 and explained above.

4.3.3 Analysis of the Avalanche Test Results of Twofish for Round 3

Fig. 4.2 part (vi) shows that for round 3, when a plaintext bit in the second input interval is complemented, the avalanche variable at position 39 of ciphertext never changes.

To explain this behaviour, let C_3 and C_3' be the round 2 outputs of Twofish when P and P' are the corresponding inputs. The ciphertext C_3 for round 3 will be obtained from output words using the same little-endian conversion used for the plaintext.

$$
\begin{aligned}
c_0 &= z_{3,0} \oplus K_4 = m'_3\, m'_2\, m'_1\, m'_0 \\
c_1 &= z_{3,1} \oplus K_5 = m'_7\, m'_6\, m'_5\, m'_4 \\
c_2 &= z_{3,2} \oplus K_6 = m'_{11}\, m'_{10}\, m'_9\, m'_8 \\
c_3 &= z_{3,3} \oplus K_7 = m'_{15}\, m'_{14}\, m'_{13}\, m'_{12} \\
C_3 &= m'_0\, m'_1\, m'_2\, m'_3\, m'_4\, m'_5\, m'_6\, m'_7\, m'_8\, m'_9\, m'_{10}\, m'_{11}\, m'_{12}\, m'_{13}\, m'_{14}\, m'_{15}
\end{aligned}
\tag{4.6}
$$

where m'_0 is the most significant byte m'_{15} is the least significant byte of the ciphertext.

Intervals i, ii, iii $(0 \le i \le 95)$

For this case, error bit position corresponds to first, second or third input words $w_0 = m_3 m_2 m_1 m_0 \oplus K_0$, $w_1 = m_7 m_6 m_5 m_4 \oplus K_1$ or $w_2 = m_{11} m_{10} m_9 m_8 \oplus K_2$ given by (4.1) respectively.

Then the round 3 output words of Twofish will be random as observed in Fig 4.2 (i), (ii) and (iii) because all output words $z_{3,0}, z_{3,1}, z_{3,2}$ and $z_{3,3}$ given by (4.4) depend on $g(w_0)$, $g(w_1)$ and $g(w_2)$. So the percentage of 1's will be approximately ½ for all 128 avalanche variables.

Interval iv $(96 \le i \le 127)$

Error bit position is chosen in fourth input word $w_3 = m_{15} m_{14} m_{13} m_{12}$. In this case, Fig 4.2 (iv) can be explained as follows:

1. $z_{3.0}$ will be random for C_3 and C_3' in all 32 bits because it depends on $g\left(ROL\left(\begin{matrix} [g(w_0) + 2g(ROL(w_1,8)) + K_9] \\ \oplus \quad ROL(w_3,1),8 \end{matrix} \right) \right)$ which depends on w_3.
2. $z_{3.1}$ will be random for C_3 and C_3' in all 32 bits except for the bit 0, because:
   - $g\left(ROR\left([(g(w_0) + g(ROL(w_1,8)) + K_8) \ \oplus \ w_2], 1 \right) \right)$ is the same for C_3 and C_3' because w_0, w_1, w_2 are the same for P and P'.
   - $g\left(ROL\left(\begin{matrix} [g(w_0) + 2g(ROL(w_1,8)) + K_9] \\ \oplus \ ROL(w_3,1),8 \end{matrix} \right) \right)$ changes randomly for C_3 and C_3' in all 32 bits because w_3 differs in 1 bit for P and P'. But because of the multiplication by 2, bit 0 of $2g\left(ROL\left(\begin{matrix} [g(w_0) + 2g(ROL(w_1,8)) + K_9] \\ \oplus \ ROL(w_3,1),8 \end{matrix} \right) \right)$ is the same for C_3 and C_3'.
   - $ROL(w_1,1)$ is same for C_3 and C_3'.

So $z_{2,1}$ ($c_1 = m'_7\, m'_6\, m'_5\, m'_4$) will be random for C_3 and C_3' in all 32 bits except bit 0 of m'_4. From (4.6), one can see that bit 0 of m'_4 corresponds to bit 39 of C_3 and C_3'.

3. $z_{3.2}$ will be random for C_3 and C_3' in all 32 bits because it depends on $g\left(ROL\left(\begin{bmatrix} g(w_0) + 2g(ROL(w_1,8)) \\ +K_9 \\ \oplus\ ROL(w_3,1),8 \end{bmatrix} \right) \right)$ which depends on w_3.

4. $z_{3.3}$ changes randomly for C_3 and C_3' in all 32 bits because it depends on $g\left(ROL\left(\begin{bmatrix} g(w_0) + 2g(ROL(w_1,8)) \\ +K_9 \\ \oplus\ ROL(w_3,1),8 \end{bmatrix} \right) \right)$ which depends on w_3.

Because of part 2, only bit 39 of C_3 and C_3' is the same for the change of any plaintext bit in the interval ($96 \le i$) and the remaining 127 bits will be random. As a result, we see that for round 3, Twofish does not satisfy the avalanche criterion.

In Avalanche test, we have computed 128 avalanche curves each of which corresponds to changes of *i*'th bit of Plaintext. That is for each of 10000 random plaintexts we first complemented 1st bit of plaintext and computed the avalanche curve. Then we have complemented 2nd bit of plaintext and computed another avalanche curve and so on until bit 128.

To remove the effect of the position of the complemented bit, we collect data in a different way. We chose 78 random plaintext, and complemented *i*'th bit for $1 \le i \le 128$ and for each plaintext pair (78×128 = 9984) we calculate 9984 ciphertext pairs. Then from the original ciphertext and the 9984 ciphertext pairs we compute a single avalanche curve. Fig. 4.6 shows avalanche curves for rounds 2 to 5. We expect these results, because each of them is the average of the 128 avalanche curves

obtained by the first method. From the figures, it is obvious that the output from the Twofish algorithm is not random for round 2 and round 3, but for round 4 it is.

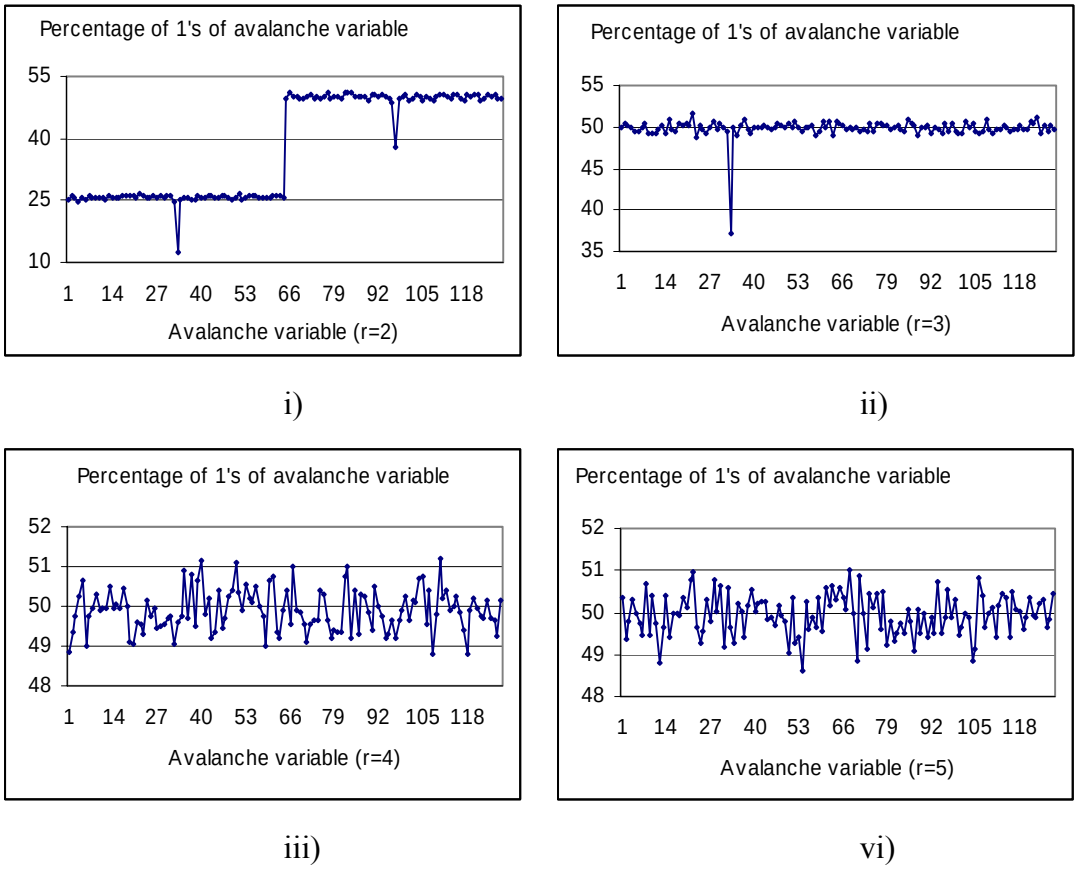


Figure 4.6 Avalanche curves for rounds 2 to 5 of the Twofish algorithm.

4.4 Comparison of Our Results with Those of NIST Statistical Test Suite

Randomness testing of AES candidates was based on NIST Statistical Test Suite which consists of 16 core statistical tests [Soto, 2000]. NIST test results showed that, by the end of second round, the output from the algorithm appears to be random for

all 189 statistical tests [Rukhin, 2000]. However, the results we found and those given by Arıkan [Arıkan, 2003] are different from the results of NIST tests. Fig. 4.1 and Fig. 4.2 show that by the end of the third round, the output from the Twofish algorithm does not satisfy the avalanche criterion. It meets the conditions at the end of the fourth round as seen from Fig. 4.3.

Within these 16 core tests, Frequency (Monobit) Test, Frequency Test Within a Block and Runs Test, whose explanations are given in Sec. 4.4.1, are mostly related with the avalanche criterion studied in this thesis. So comparing the results of these tests ours will be meaningful. This is why we have also applied these 3 tests to the Twofish algorithm.

4.4.1 Description of the Tests

a. Frequency (Monobit) Test

The focus of the test is the proportion of zeroes and ones for the entire sequence. The purpose of this test is to determine whether the number of ones and zeroes in a sequence are approximately the same as would be expected for a truly random sequence [Rukhin, 2000].
Test Description:

1. Convert the zeroes and ones of the input sequence to (−1) and (+1) and add together to produce $S_n = X_1 + X_2 + ... + X_n$.
2. Compute the test statistic $s_{obs} = \frac{|S_n|}{\sqrt{n}}$.
3. Compute the $P\text{-}value = erfc\left(\frac{s_{obs}}{\sqrt{2}}\right)$

where *erfc* is the complementary error function.

4. If the computed *P-value* is < 0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

b. Frequency Test within a Block

The focus of the test is the proportion of zeroes and ones within *M*-bit blocks. The purpose of this test is to determine whether the frequency of ones in an *M*-bit block is approximately *M*/2, as would be expected under an assumption of randomness [Rukhin, 2000].

Test Description:

1. Partition the input sequence into $N = \left\lfloor \frac{n}{M} \right\rfloor$ non-overlapping blocks. Discard any unused bits.
2. Determine the proportion π_i of ones in each *M*-bit block using the equation

$$\pi_i = \frac{\sum_{j=1}^{M} \varepsilon_{(i-1)M+j}}{M}, \text{ for } 1 \le i \le N .$$

3. Compute the χ^2 statistic: $\chi^2(obs) = 4M \sum_{i=1}^{N} (\pi_i - 1/2)^2$.
4. Compute $P\text{-}value = igamc\left(N/2, \chi^2(obs)/2\right)$

   where *igamc* is the incomplete gamma function.
5. If the computed *P-value* is < 0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

c. Runs Test

The focus of this test is the total number of runs in the sequence, where a run is an uninterrupted sequence of identical bits. A run of length *k* consists of exactly *k* identical bits and is bounded before and after with a bit of the opposite value. The purpose of the runs test is to determine whether the number of ones and zeroes of various lengths is as expected for a random sequence [Rukhin, 2000].

Test Description:

1. Compute the pre-test proportion π of ones in the input sequence: $\pi = \frac{\sum_j \varepsilon_j}{n}$.
2. Determine if the prerequisite Frequency test is passed.
3. Compute the test statistic $v_n(obs) = \sum_{k=1}^{n-1} r(k) + 1$

   where $r(k) = 0$ if $\varepsilon_k = \varepsilon_{k+1}$, and $r(k) = 1$ otherwise.
4. Compute $P\text{-}value = erfc\left(\frac{|v_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2}\pi(1-\pi)}\right)$.
5. If the computed $P\text{-}value$ is < 0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

4.4.2 Description of the Data Type

To examine the sensitivity of the Twofish algorithm to changes in the plaintext, 300 binary sequences were analyzed by NIST. We also constructed 300 sequences in one of the following ways, the first one being the same as the data type used in the NIST tests [Soto, 2000]. The second data type indicates the performance versus each complemented input bit separately, whereas the first data type used by NIST is an average over all complemented output bits. Description below is for 128-bit input blocks, so each of the 300 binary sequences is of length 1,048,576 bits.

Plaintext Avalanche-1: Given a fixed key of all zeroes and 19200 random 128-bit plaintext blocks (or 19200 input vectors), the i^{th} avalanche vector is found for each plaintext. The i^{th} avalanche vector, which is also a 128-bit vector, is equal to the XOR of "the ciphertext formed using the plaintext" and "the ciphertext formed using the perturbed plaintext complementing its i^{th} bit". For each random plaintext, complementing the input bit i, $1 \le i \le 128$, 128 avalanche vectors are formed. Concatenating all derived 128-bit avalanche vectors of 19200 random plaintexts, a total of 2,457,600 (=19200×128) avalanche vectors result, which is a sequence of

2,457,600×128=314,572,800 bits. Then, 300 subsequences of 1,048,576 (=314,572,800/300) bits are parsed from the concatenated derived blocks [Soto, 2000].

Plaintext Avalanche-2: The data type explained above is prepared for a fixed value of the complemented input bit *i*. Since the number of avalanche vectors is now one 128^{th} of the previous case, the length of 300 subsequences is kept the same by using 128 times as much plaintext blocks.

We have implemented the three tests given above and applied them to the Twofish algorithm. If 11 of the 300 sequences have the P-value < 0, 01 then the test is considered to fail in accordance with NIST criteria of at least 96.6%. The sequences used in the tests, whose results are given in Fig. 4.7 to Fig. 4.18, are constructed by using the data type "plaintext avalanche-1", and Fig. 4.19 to Fig. 4.27 are the results obtained by "plaintext avalanche-2".

4.4.3 Test Results

4.4.3.1 Monobit Test

Figures 4.7 to 4.10 show the results of the monobit test for rounds 2 to 5 of Twofish. In Fig. 4.7 we see that none of the 300 sequences have the *P*-value greater than 0.01, which means that Twofish failed the monobit test for round 2. Similarly, round 3 output of Twofish failed the monobit test because Fig 4.8 show that 257 of 300 sequences have the *P*-value greater than 0.01. But from Fig. 4.9 and 4.10, we see that 295 of 300 sequences have the *P*-value greater that 0.01, which means that Twofish passes the monobit test for rounds 4 and 5. We conclude from the results that, according to the monobit test, the output from Twofish is random at the end of round 4.

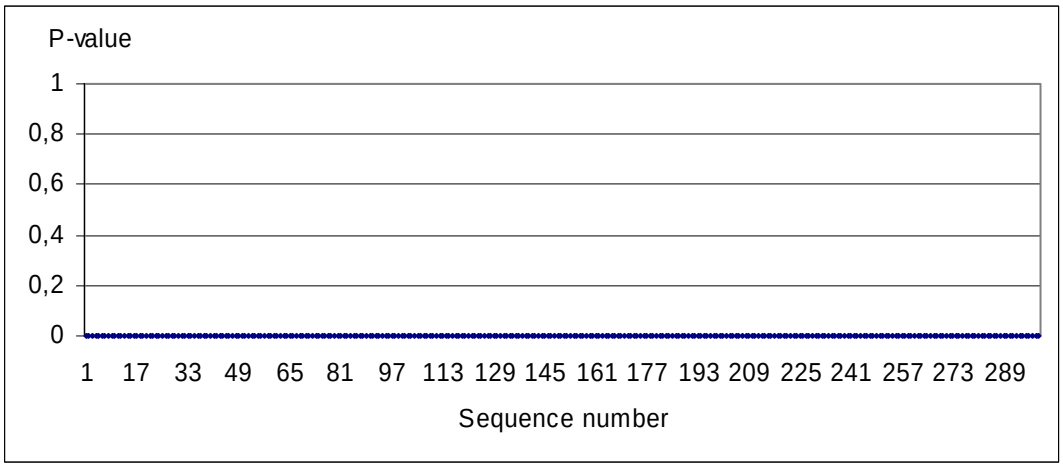


Figure 4.7 *P*-values of the monobit test for round 2 with the first data type (0 of 300 passes = %0).

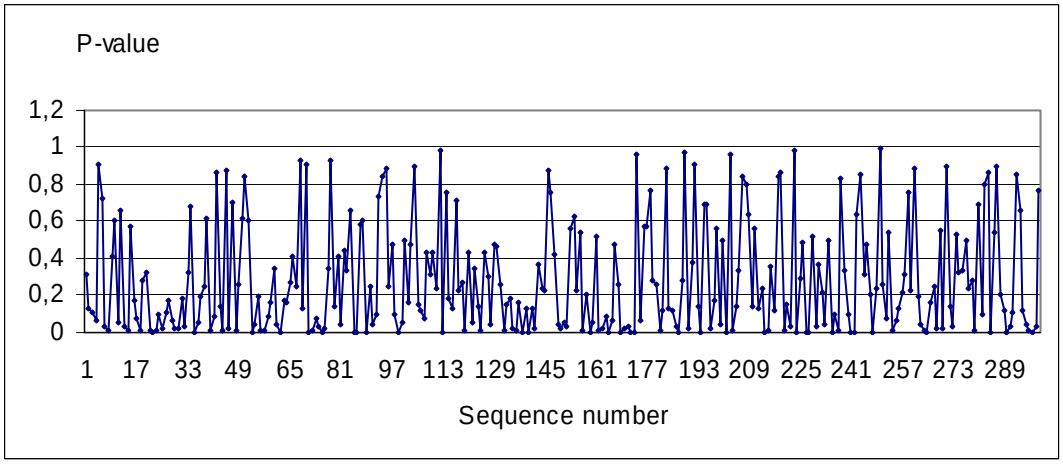


Figure 4.8 *P*-values of the monobit test for round 3 with the first data type (257 of 300 passes = %85, 6).

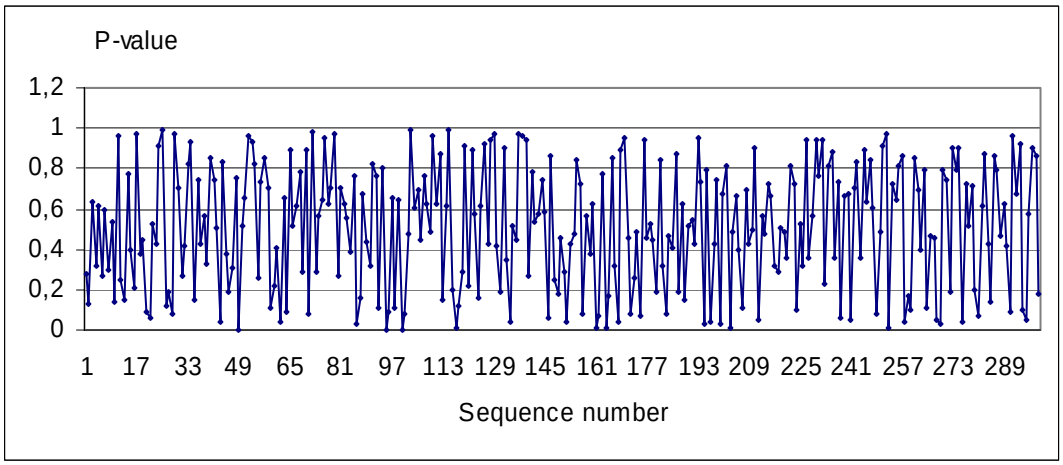


Figure 4.9 *P*-values of the monobit test for round 4 with the first data type (295 of 300 passes = %98, 3).

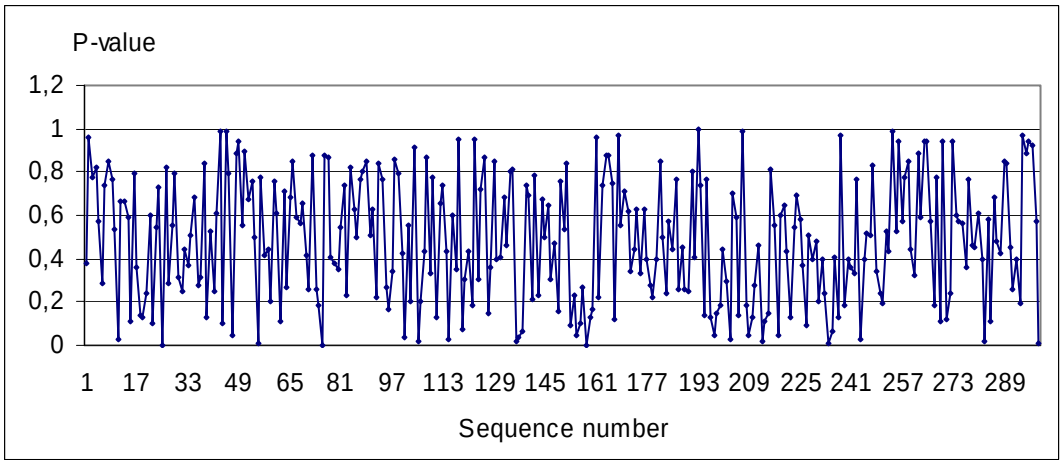


Figure 4.10 *P*-values of the monobit test for round 5 with the first data type (295 of 300 passes = %98, 3)

4.4.3.2 Frequency Test within a Block

Figure 4.11 to 4.14 are the results of Frequency test within a Block for rounds 2 to 5. The test results are similar to monobit test results. It can be seen that the output from the Twofish algorithm is not random until round 4. Because for round 2, none of the

300 sequences has the *P*-value greater then 0.01, so by the end of round 2, Twofish failed the Frequency test within a block. At the end of round 3, 275 of 300 sequences has the *P*-value greater than 0.01, so Twofish failed the Frequency test within a block for round 3. But by the end of round 4, 297 of 300 sequences has a *P*-value greater than 0.01. We conclude from the results that, according to Frequency test within a block, the output from Twofish appears to be random at the end of round 4.

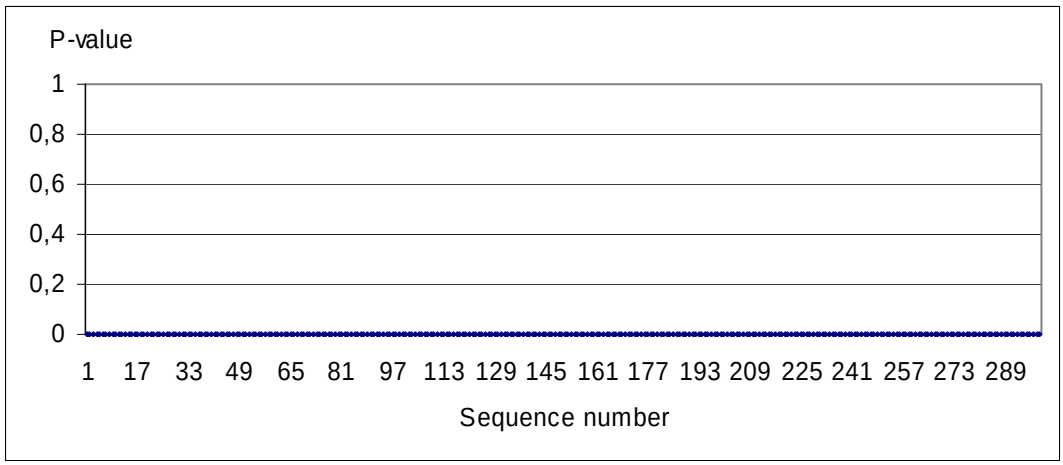


Figure 4.11 *P*-values of the frequency test within a block for round 2 with the first data type (0 of 300 passes = %0).

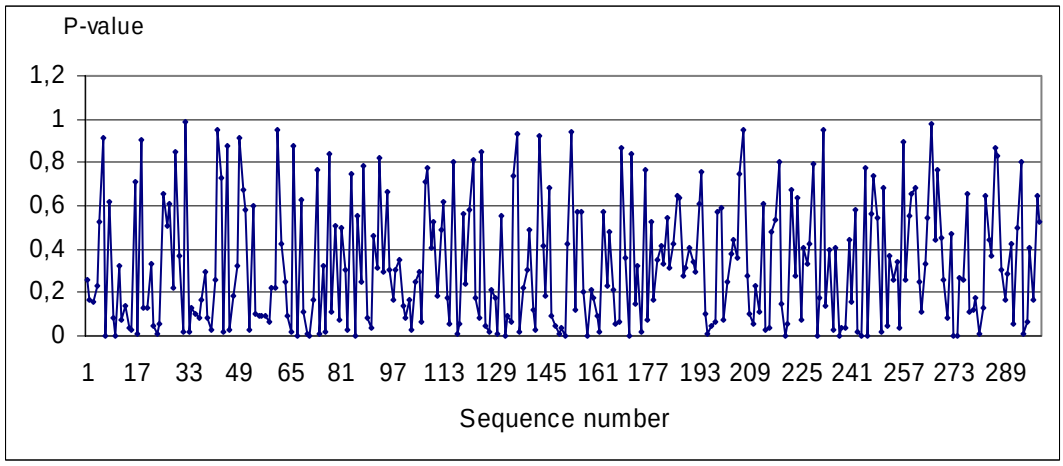


Figure 4.12 *P*-values of the frequency test within a block for round 3 with the first data type (275 of 300 passes = %91, 6).

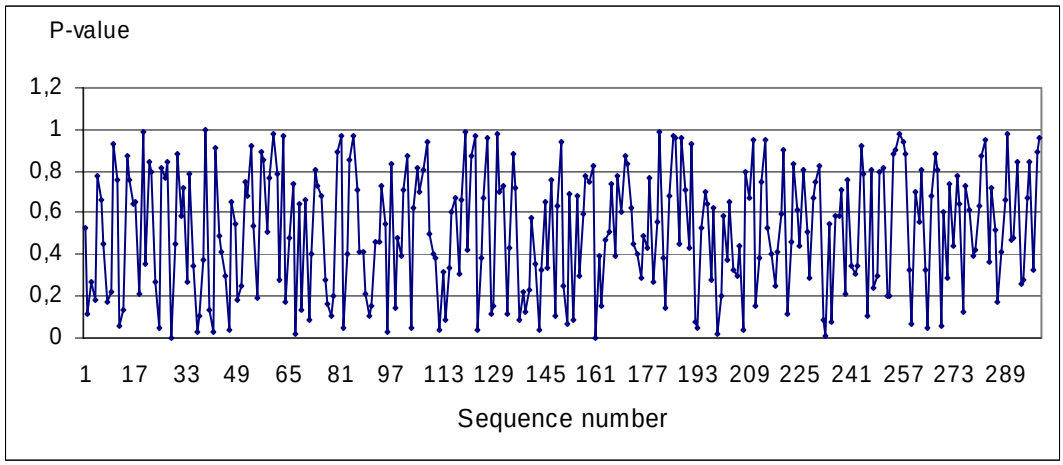


Figure 4.13 *P*-values of the frequency test within a block for round 4 with the first data type (297 of 300 passes = %99).

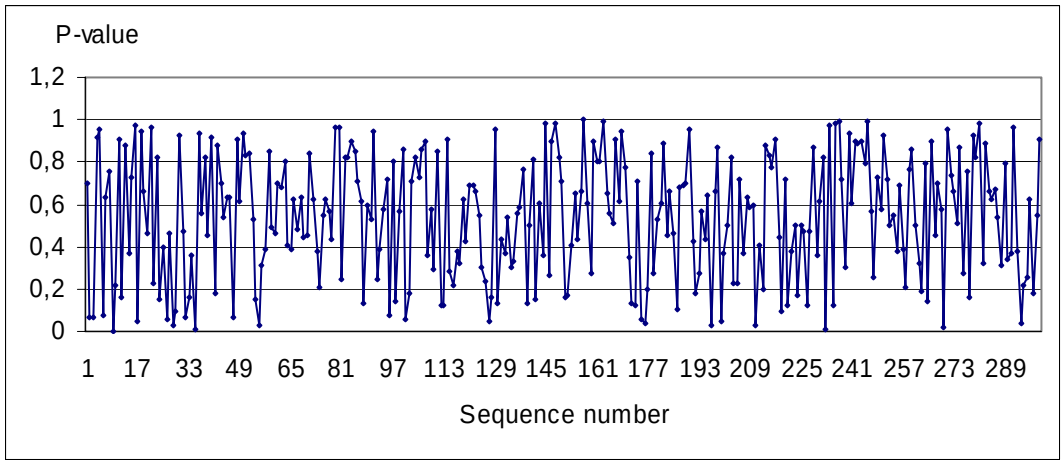


Figure 4.14 *P*-values of the frequency test within a block for round 5 with the first data type (297 of 300 passes = %99).

4.4.3.3 Runs Test

Figure 4.15 to 4.18 are the results of Runs test for rounds 2 to 5. The test results are similar to monobit test and frequency test within a block results. It can be seen that the output from the Twofish algorithm is not random until round 4. Because for round 2, none of the 300 sequences has the *P*-value greater then 0.01, so by the end

of round 2, Twofish failed the Runs test. At the end of round 3, 263 of 300 sequences has the *P*-value greater than 0.01, so Twofish failed the Runs test for round 3. But by the end of round 4, 296 of 300 sequences has a *P*-value greater than 0.01. We conclude from the results that, according to Runs test, the output from Twofish appears to be random at the end of round 4.

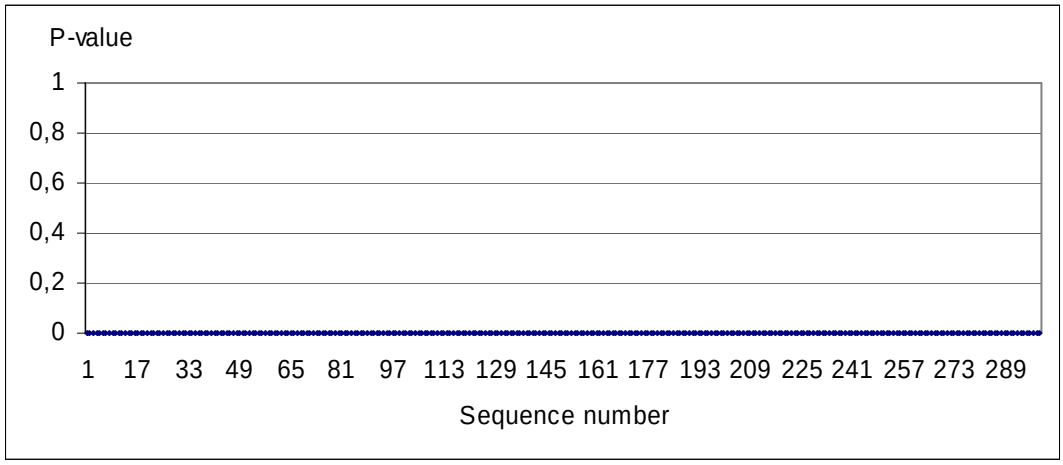


Figure 4.15 *P*-values of the runs test for round 2 with the first data type (0 of 300 passes = %0).

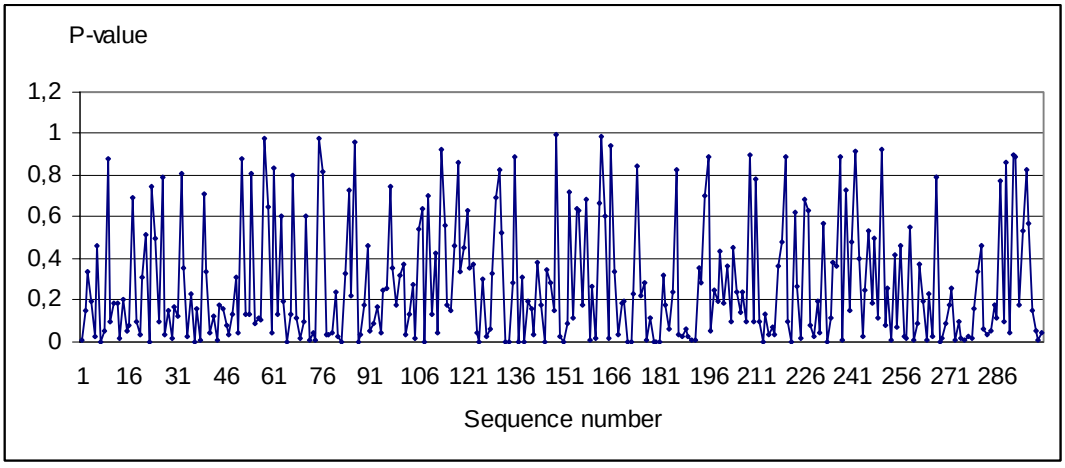


Figure 4.16 *P*-values of the runs test for round 3 with the first data type (263 of 300 passes = %87, 6).

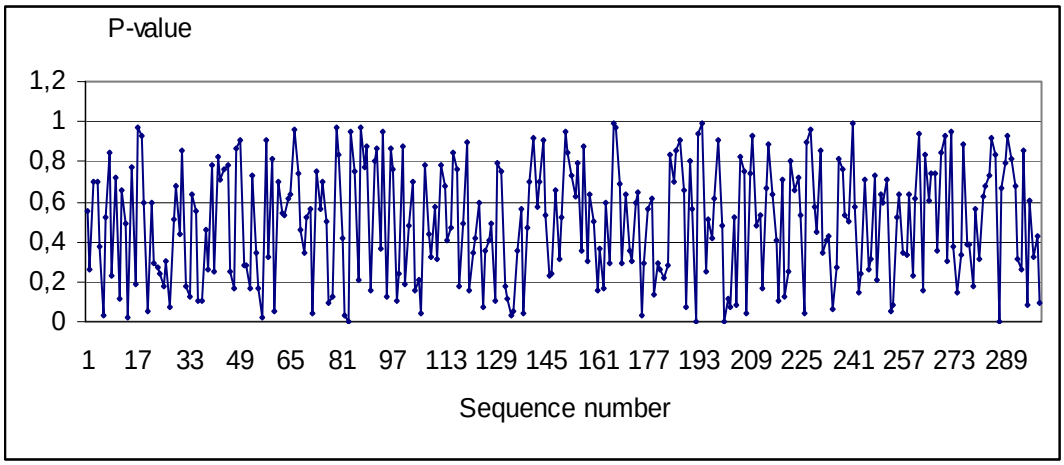


Figure 4.17 *P*-values of the runs test for round 4 with the first data type (296 of 300 passes = %98, 6).

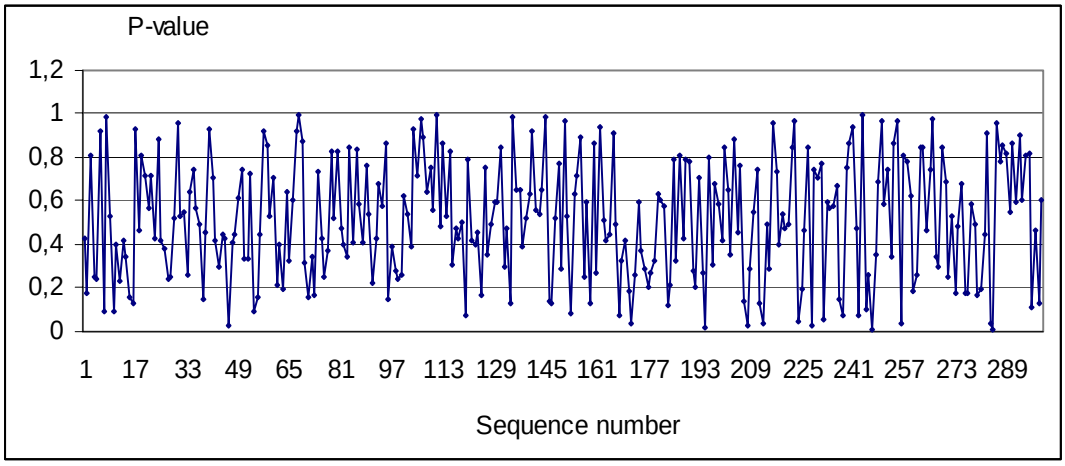


Figure 4.18 *P*-values of the runs test for round 5 with the first data type (296 of 300 passes = %98, 6).

To examine the sensitivity of the Twofish algorithm to separate changes of each of 128 bits in the plaintext, these three tests have been repeated using, “plaintext avalanche-2” as the data type.

In Fig. 4.19, monobit test results of Twofish for round 2 are sketched. Figure 4.19 shows the number of sequences that has the *P*-value > 0.01 for each input bit complementation. Fig. 4.20 and Fig. 4.21 are sketched in the same way, for rounds 3 and 4 of Twofish respectively.

Fig. 4.19 shows that the second round output of Twofish is not random because when a single input bit *i* in the second interval is complemented, none of 300 sequences has the *P*-value > 0.01. Also for some error bit positions chosen in the first input interval, less than 289 of 300 sequences has the *P*-value > 0.01.

Fig. 4.20 demonstrates that the third round output of Twofish is not random because for some error bit positions chosen in the second input interval, less than 289 of 300 sequences meets the condition.

The output from the Twofish algorithm appears to be random at the end of fourth round because from Fig. 4.21 we see that for each input bit complementation, the number of sequences that has the *P*-value > 0.01 is greater than 289.

Fig. 4.22 to 4.24 and 4.25 to 4.27 are the Frequency test within a block and runs test results of Twofish respectively. Similar results are obtained for all three tests.

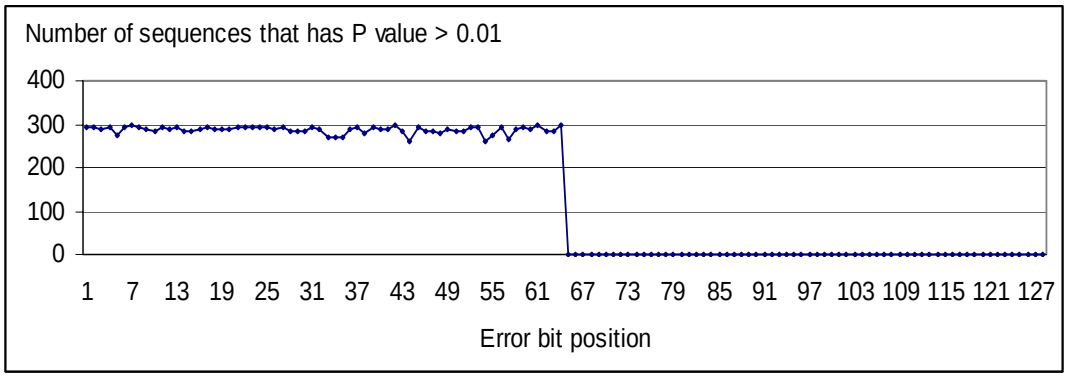


Figure 4.19 Number of sequences that have *P*-value > 0.01 at the end of round 2 for 128 monobit tests, each of which are made with different input error bit position *i*, $0 \le i \le 127$

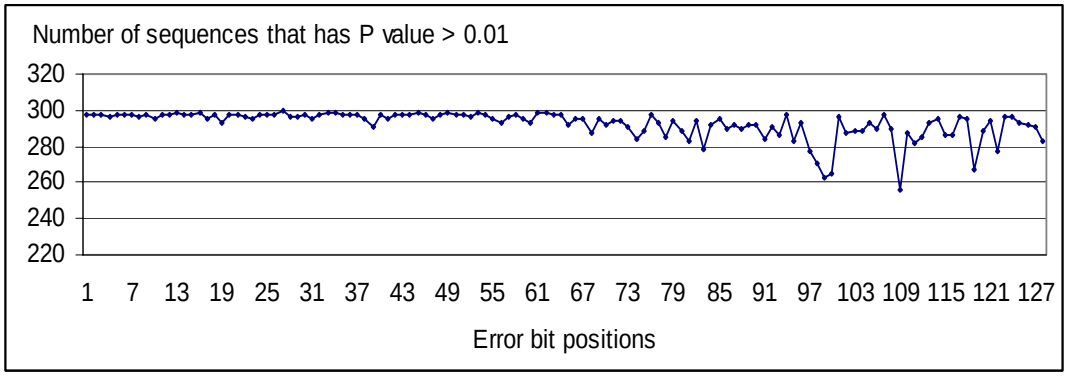


Figure 4.20 Number of sequences that have *P*-value > 0.01 at the end of round 3 for 128 monobit tests, each of which are made with different input error bit position *i*, $0 \le i \le 127$

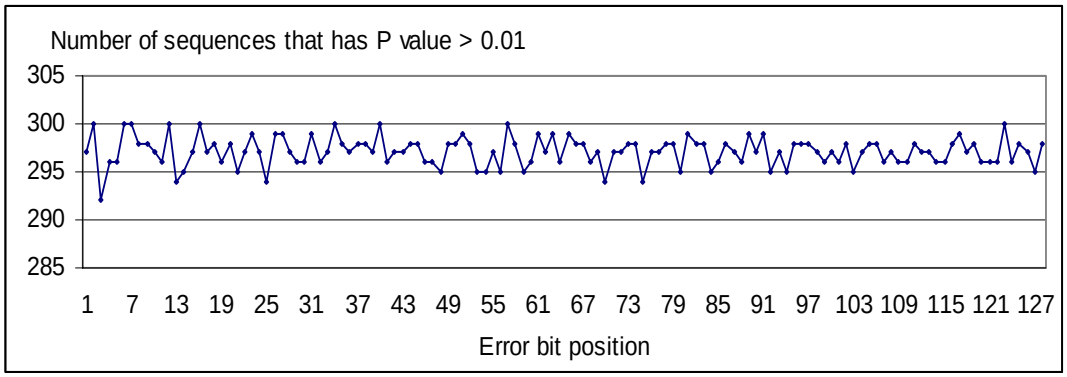


Figure 4.21 Number of sequences that have *P*-value > 0.01 at the end of round 4 for 128 monobit tests, each of which are made with different input error bit position *i*, $0 \le i \le 127$

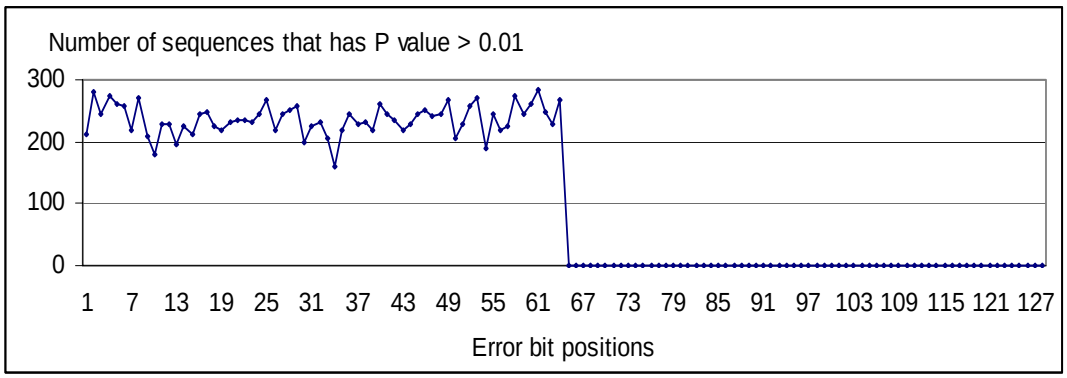


Figure 4.22 Number of sequences that have *P*-value > 0.01 at the end of round 2 for 128 frequency tests, each of which are made with different input error bit position *i*, $0 \le i \le 127$

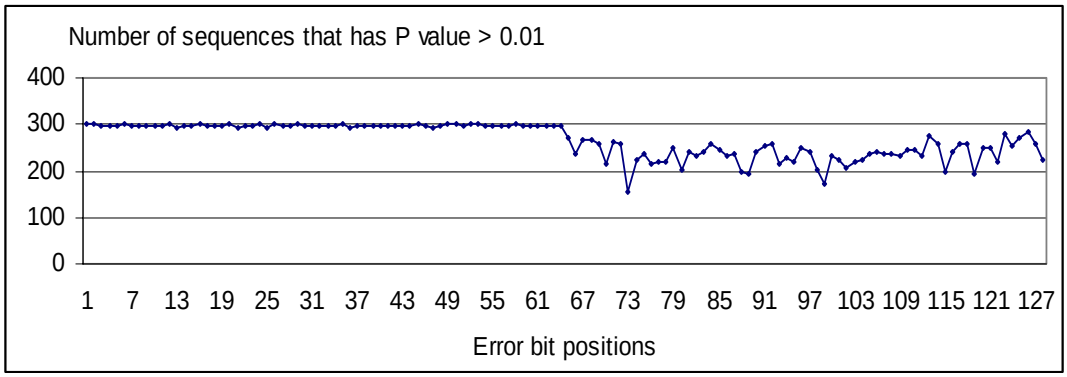


Figure 4.23 Number of sequences that have *P*-value > 0.01 at the end of round 3 for 128 frequency tests, each of which are made with different input error bit position *i*, $0 \le i \le 127$

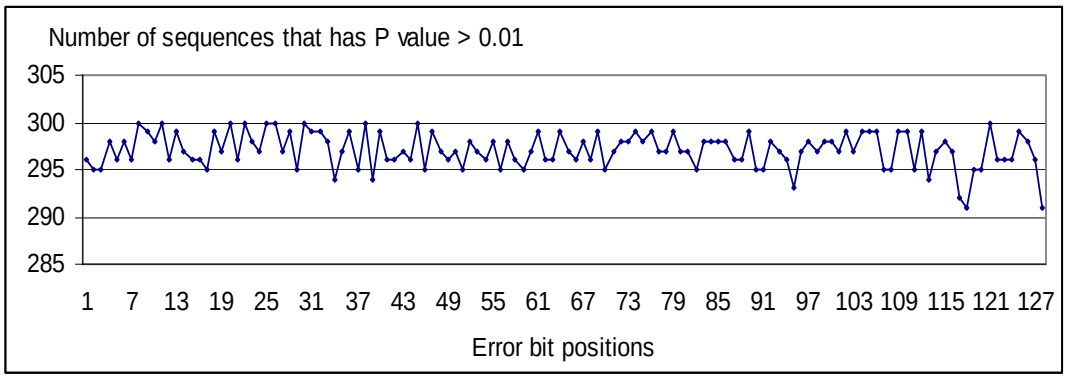


Figure 4.24 Number of sequences that have *P*-value > 0.01 at the end of round 4 for 128 frequency tests, each of which are made with different input error bit position *i*, $0 \le i \le 127$

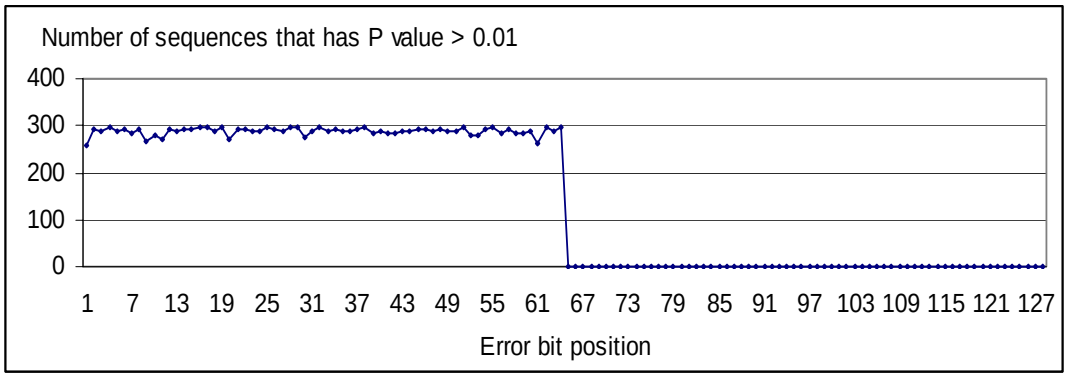


Figure 4.25 Number of sequences that have *P*-value > 0.01 at the end of round 2 for 128 runs tests, each of which are made with different input error bit position *i*, $0 \le i \le 127$

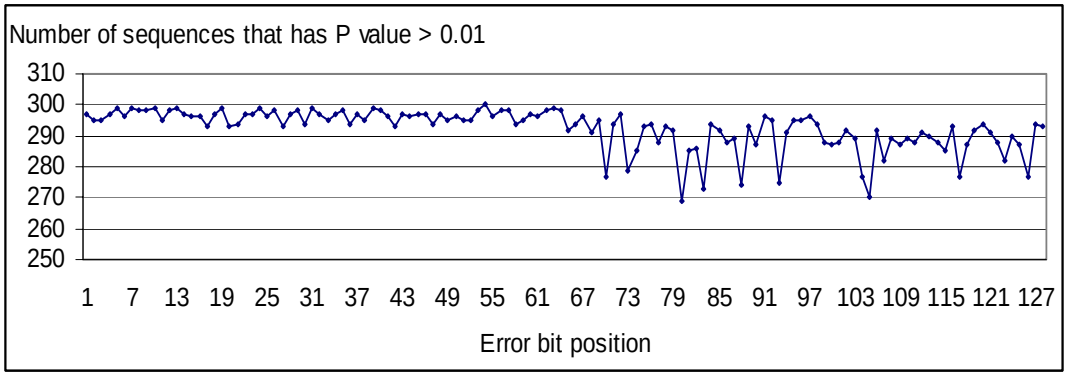
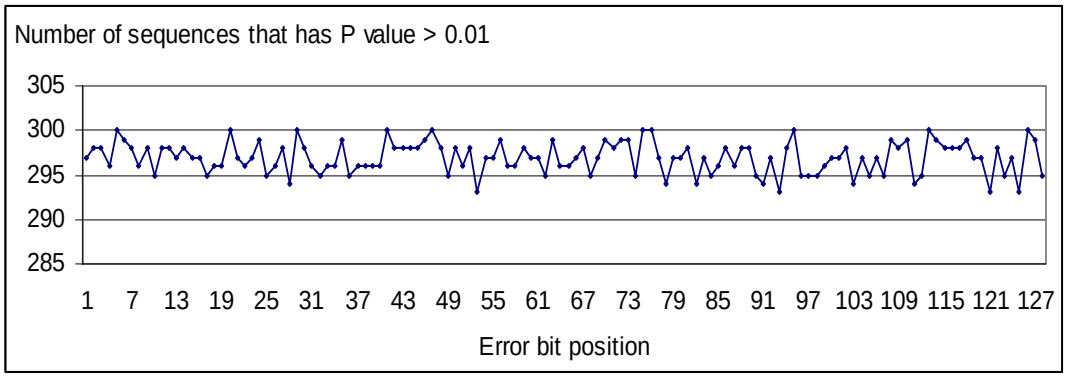


Figure 4.26 Number of sequences that have *P*-value > 0.01 at the end of round 3 for 128 runs tests, each of which are made with different input error bit position *i*, $0 \le i \le 127$



iii)

Figure 4.27 Number of sequences that have *P*-value > 0.01 at the end of round 4 for 128 runs tests, each of which are made with different input error bit position *i*, $0 \le i \le 127$

4.5 Cryptanalysis of Twofish

Bruce Schneier et al cryptanalyzed Twofish and a summary of these attacks on their own cipher is as follows [Schneier, 1999]:

- On Twofish with fixed S-boxes, no 1-bit rotations, and no whitening, They [Schneier, 1999] have a meet-in-the-middle attack on eleven rounds requiring 2^{225} memory, 256 known plaintexts, and 2^{232} work, and a differential attack breaking nine rounds, requiring 2^{41} memory, 2^{41} chosen plaintexts, and 2^{254} work. I.e., they break nine rounds of Twofish with differential attack and eleven rounds of it with meet-in-the-middle attack.
- On standard Twofish, they [Schneier, 1999] have a 4-round meet-in-the-middle attack requiring 256 known plaintexts, but 2^{225} memory and 2^{232} work. They also have a differential attack which breaks five rounds of full Twofish with 2^{232} work and 2^{41} chosen-plaintext queries.
- They [Schneier, 1999] have a chosen-key attack. This attack involves choosing 160 bits of a pair of keys, *K*, *K**, with the remaining bits to be

found. This attack requires 2^{34} work, 2^{32} chosen-plaintext queries, and 2^{12} adaptive chosen-plaintext queries, in order to break 10 rounds without the whitening.

- They [Schneier, 1999] have a related key attack against 10-round Twofish without whitening. This attack requires 2^{155} related-key queries, 2^{187} work, and for each of the 2^{155} keys it requires 2^{32} chosen plaintexts and 2^{12} adaptive chosen plaintexts.

From the attacks we see that NIST's results also conflict with the results given by Bruce Schneier et al [Schneier, 1999]. Because, they broke the 4-round of full Twofish with meet-in-the-middle attack and 5-round of full Twofish with differential attack. However, NIST claims that the output from Twofish is random at the end of the second round.

4.6 Nonlinearity Measure of Twofish

If a ciphertext bit c_i is described by the Boolean function f_i then it is generally required that each f_i should possess a combination of the properties such as balancedness, nonlinearity, completeness, correlation immunity, the strict avalanche criterion.

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 components of the cipher.

4.6.1 Nonlinearity of the S-boxes

Nonlinearity of the S-box can be defined in terms of the 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 $f_i : Z_2^n \to Z_2$ is defined by

$$N_{f_j} = 2^{n-1} - \frac{1}{2}\max_{i=1,\ldots,2^n} \left| f_{j,s} \bullet I_{i,s} \right| = 2^{n-1} - \frac{1}{2}\max \left| F_j(\mathbf{w}) \right|$$

Perfect nonlinearity condition implies $|F_{opt}(\mathbf{w})|=2^{n/2}$ for all **w** [Yücel, 2001],

So; $N_{f_j} \le 2^{n-1} - \frac{1}{2}\left|F(\mathbf{w})\right| = 2^{n-1} - 2^{(n/2)-1}$ (4.7)

This maximum is not achievable if *n* is odd; because for odd values of *n*, $2^{n/2}$ is not even (and not rational), hence it is not a proper spectral coefficient. This is why there is no perfectly nonlinear Boolean function for odd values of *n*. However, if $\lceil 2^{n/2} \rceil$ denotes the smallest *even* integer which is larger than $2^{n/2}$, it should be possible to find spectra with $|F(\mathbf{w})|_{max}=\lceil 2^{n/2} \rceil$; so that for all values of *n*, N_f has a maximum value of

$$| N_f |_{max} = 2^{n-1} - \lceil 2^{n/2} \rceil 2^{-1}.$$

4.6.2 Nonlinearity Criterion

To calculate the nonlinearity of *nxn* s-boxes, we first find the truth tables 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 $(2^n - 1)$ truth tables corresponding to all $(2^n - 1)$ linear combinations of the output bits. Each row of the truth table matrix is then compared to all rows of the $2^n x 2^n$ 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 the corresponding *nxn* S-boxes.

We find the $(2^n - 1)x(2^n)$ truth table matrix with the following algorithm.

1. Define a Boolean vector of $\mathbf{F} = \{f_1, f_2, ..., f_n\}$ where f_x are the result bits of the S-boxes while $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ is the input vector, having the integer value, $0 \le x \le 2^n - 1$.
2. Define the Boolean function to be $\mathbf{a} \circ \mathbf{F} = a_1.f_1 \oplus a_2.f_2 \oplus .....\oplus a_n.f_n$, where $\mathbf{a} = \{a_1, a_2, ..., a_n\}$ with the integer value $0 \le a \le 2^n - 1$.
3. Find the truth table of each Boolean function by using all available coefficient vectors, **a**.

Table 4.1 shows the form of a $(2^3 - 1)x2^3$ truth table matrix.

Table 4.1: A simple form of a $(2^3 - 1)x2^3$ truth table matrix

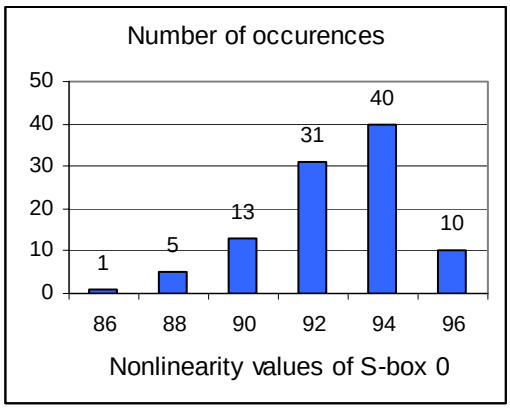
		$x_1x_2x_3$							
$a_1a_2a_3$		000	001	010	011	100	101	110	111
001	f_3	…	…	…	…	…	…	…	…
010	f_2	…	…	…	…	…	…	…	…
011	$f_2 + f_3$	…	…	…	…	…	…	…	…
100	f_1	…	…	…	…	…	…	…	…
101	$f_1 + f_3$	…	…	…	…	…	…	…	…
110	$f_1 + f_2$	…	…	…	…	…	…	…	…
111	$f_1 + f_2 + f_3$	…	…	…	…	…	…	…	…

In table 4.1, f_1, f_2 and f_3 are the result bits of the 3x3 S-box while $x = \{x_1, x_2, x_3\}$ is the input vector.

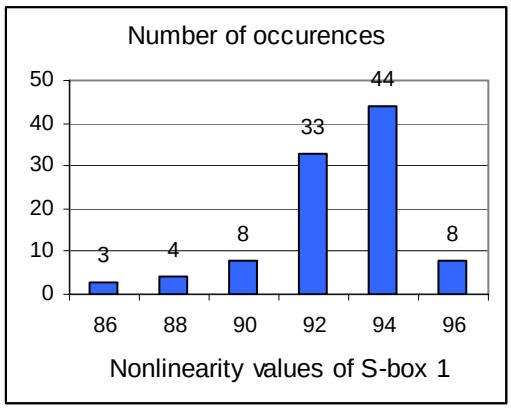
4.6.3 Nonlinearities of the S-boxes of Twofish

In this section, the graphical results of nonlinearity criterion are given for the 8x8 S-boxes of the 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. 3.5). The two boxes q_0 and q_1 are simple 8×8 permutations. Their algorithms are the same but only their look-up tables given by (3.10) and (3.11) are different from each other. The nonlinearities of q_0 and q_1 are found as 96. The S-boxes of the Twofish algorithm, which employ q_0 and q_1, have key-dependent coefficients as indicated by the elements $I_{i,j}$ in (3.8). 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 8×8 S-boxes of Twofish, the distribution of the nonlinearity values for 100 keywords corresponding to 100 random choices of the coefficients $I_{i,j}$ in (3.8) is sketched.

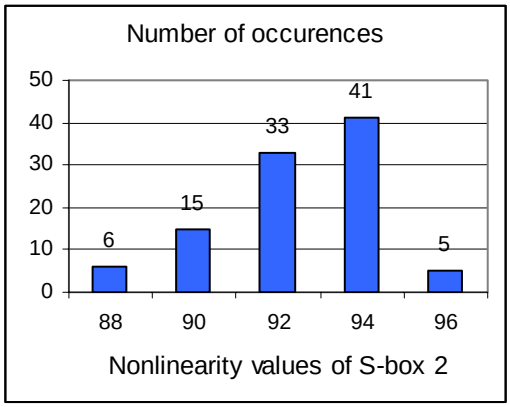
In Fig. 4.28 (i), Fig. 4.28 (ii), Fig. 4.28 (iii), and Fig. 4.28 (iv), the nonlinearity distributions of S-boxes of Twofish are given. Although the number of occurrences of nonlinearity values different from each other, the curves are similar to each other and the average of nonlinearity values is almost same for different keywords.

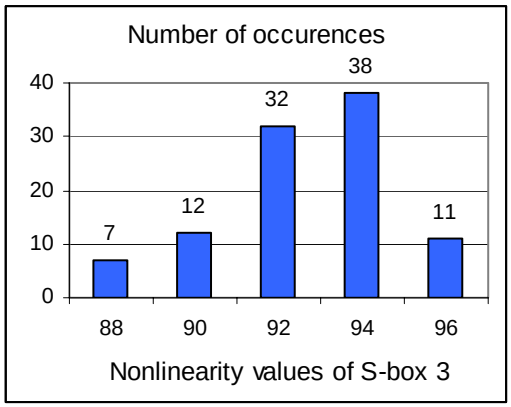


(i)



(ii)





(iii) (iv)

Figure 4.28 Nonlinearity of S-boxes of Twofish i) S-box 0 ii) S-box 1 iii) S-box 2 iv) S-box 3

As the above figure states, most often the nonlinearity is around 94 for different keywords. From (4.4) it can be calculated that for $n = 8$ if S-boxes of Twofish were perfectly nonlinear, the nonlinearity would be 120. For Twofish, highest value is 96.

The nonlinearity values of Twofish we obtain are very similar to those calculated by Arıkan [Arıkan, 2003].

CHAPTER 5

CONCLUSION

In this thesis, Twofish, one of the finalists of the Advanced Encryption Standard (AES) contest, is studied. The strength of the cipher against cryptanalytic attacks is measured according to the avalanche criterion.

Our results show that, Twofish becomes random at the output of the fourth round. The results obtained by Arıkan [Arıkan, 2003] and those we calculate claim the same thing: the output of Twofish is not random for rounds 2 and 3. However, there are some differences between the curves that we obtain for the second and third round outputs of Twofish and those calculated by Arıkan. To understand the discrepancies, we analyse our results in detail.

We derive in Chapter 4 the second and third round outputs of Twofish in terms of plaintext bytes and we show that the output of the Twofish algorithm will not be completely random at the end of second and third rounds.

The nonlinearities of the S-boxes of the Twofish cipher are calculated. The nonlinearity of the permutation boxes q_0 and q_1 are found as 96. The nonlinearity distributions of four 8×8 S-boxes are computed over many different sets of keys. Since these S-boxes have key dependent coefficients, their nonlinearities change in the range [86, 96] for different keys, the average value being around 94. One can argue that dynamic behaviour of key dependent S-boxes may increase the security of Twofish.

The most important result is that, although NIST results assume randomness of Twofish at end of the second round, the avalanche criterion that we use as well as their monobit, frequency and runs tests, indicate that the second round outputs are completely nonrandom especially when a bit change is made in the third and fourth intervals of the plaintext (for $i = 64,...,127$). 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. To remove the effect of the preparation methods of the test data, we use "plaintext avalanche-1", among the data types of NIST, as the test data. From figure 4.5 it is seen that when we use "plaintext avalanche-1" as the test data, the output from Twofish becomes random at the end of round 4 and at the end of the second round, Twofish seems to be non-random.

We have also implemented the three core tests of NIST which are monobit test, frequency test within a block and runs test using both "plaintext avalanche-1" and "plaintext avalanche-2" as the test data. The test results are similar for all three tests. From the figures (4.6 to 4.26) it is seen that the output from the Twofish algorithm is not random until the end of round 4.

REFERENCES

1. [Arıkan, 2003] Savaş Arıkan, "Propagation Characteristics of RC5, RC6 and Twofish Ciphers", December 2003.

2. [Feistel, 1973] H.Feistel, "Cryptography and computer privacy", Scientific American, v.228, n5, pp.15-23, May 1973.

3. [Hirose, 1994] S. Hirose, K. Ikeda, "Nonlinearity Criteria for Boolean Functions", July 14, 1994.

4. [Rukhin, 2000] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Simid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, San Vo. "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications", September 2000.

5. [Preneel, 1991] B.Preneel, W. Van Leekwijck, L. Van Linden, R. Govaerts and J. Vandewalle, "Propagation Characteristics of Boolean Functions", Advances in Cryptology, Proc. Eurocrypt 90, Lecture Notes in Computer Science 473, Springer Verlag, 1991. pp. 161-173.

6. [Preneel, 1994] B.Preneel, Rene Guvaerts and Joos Vandewalle, "Boolean Functions Satisfying Higher Order Propagation Criteria".

7. [Schneier, 1998] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, "Twofish: A 128-Bit Block Cipher", June 1998.

8. [Schneier, 1999] Bruce Schneier, John Kelsey, Dough Whiting, David Wagner, Chris Hall, Niels Ferguson, “The Twofish Encryption Algorithm”, 1999.

9. [Soto, 2000] Juan Soto, Lawrence Bassham, “Randomness Testing of the Advanced Encryption Standard Finalist Candidates”, March 2000.

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

11. [WebTav, 1985] A.F.Webster and S.E.Tavares, “On the Design of S-boxes”, CRYPTO’85, Springer-Verlag, pp.523-534, 1985.

12. [Yücel, 2001] M.D.Yücel, “Nonlinearity Indices for Boolean Functions”, METU Electrics-Electronics Engineering Dep. Memorandum, No: 2001-1, Jan. 2001.