Algebraic Immunity in Boolean Functions (Methods)

By
Rizwan Muhammad
2011-NUST-MS-CCS-16

Supervisor
Dr. Abdul Ghafoor Abbasi
Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of Masters in Computer and Communication Security (MS CCS)

In
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(February 15, 2015)

Approval

It is certified that the contents and form of the thesis entitled "**Algebraic Immunity in Boolean Functions (Methods)**" submitted by **Rizwan Muhammad** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Abdul Ghafoor Abbasi**
Signature: _____

Date: _____

Committee Member 1: **Dr. Mehreen Afzal**

Signature: _____
Date: _____

Committee Member 2: **Dr. Zahid Anwar**

Signature: _____
Date: _____

Committee Member 3: **Dr. Adnan Khalid Kiani**

Signature: _____
Date: _____

Abstract

Boolean functions play an important role in designing any modern symmetric cipher. They can be utilized either as filter/combiner functions in LFSR based stream ciphers, or as s-box component functions in block ciphers. In order to design strong crypto-systems, cryptographers over the years have identified some basic cryptographic criteria for Boolean functions, which are required to be fulfilled before applying them in practical systems. These basic cryptographic criteria include balanced-ness, algebraic degree, non-linearity, correlation immunity and algebraic immunity. Therefore, constructing Boolean functions, along with fulfilling basic cryptographic criteria, has become a vital task for cryptographers. Considerable work has been achieved over the last few years for constructing Boolean functions, mainly focused on achieving optimal algebraic immunity. However, all existing methods lacks in fulfilling all the cryptographic criteria other than algebraic immunity, due to some essential trade offs among various cryptographic criteria. Mostly construction methods are iterative in nature, which require more number of existing Boolean functions with at least optimal algebraic immunity, as their initial functions. Moreover, only theoretical constructions are found in literature, with very less or no implementation results.

In this thesis, we have carried out comparative analysis of four existing

methods for constructing Boolean functions with maximum algebraic immunity. These methods are not only efficiently implemented to construct Boolean functions, but are also analyzed in terms of fulfilling basic cryptographic criteria. Additionally, these methods are evaluated in terms of higher order non-linearity up to n=5 variables. We have also presented a method to extend existing construction methods and construct 2^n more number of Boolean functions with maximum algebraic immunity by using existing single Boolean function. We got successful results for two existing constructions, which include construction of Boolean functions using majority functions and construction through primitive polynomials. We have proved our results through experiments, up to n=12 variables; however no mathematical proof has been given and is left as future work.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Rizwan Muhammad**

Signature: _____

Acknowledgment

First of all, I am extremely thankful to Almighty Allah, for His countless blessing bestowed on me throughout my life. I duly acknowledge all the help and support extended by complete supervision of my co-advisor Dr. Mehreen Afzal, for her continuous encouragement, enlightening advice and most willing help which made it possible for me to complete my research work. I would also like to acknowledge the thorough guidance and help extended to me by my supervisor Dr. Abdul Ghafoor, every time I needed it. I am also thankful to Dr. Zahid Anwar and Dr. Adnan Khalid Kiani for their support and time. Lastly, I am extremely grateful to my parents and wife for their unwavering support and encouragement throughout my master studies.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

1.1 Introduction

In todays digital world, information security has become extremely important and an essential part in our everyday life, for example online banking, credit cards, internet shopping, mobile phones, etc. Cryptography, which is the field of designing strong algorithms/ciphers for information security, is mainly divided into three broad categories; symmetric key ciphers (involve block and stream ciphers), asymmetric key ciphers (involve public key ciphers and signatures) and unkeyed ciphers (involve hash functions). Symmetric key ciphers, which use same key for encryption and decryption, are further divided into stream ciphers and block ciphers.

Stream ciphers received special attention and their designs are widely studied in last few years, due to their simple, fast and efficient design as well as implementation in vast number of applications especially in wireless communication. They are extremely efficient against real time applications,

like mobile communication, telephonic voice, Bluetooth, VoIP, etc. However, LFSR based stream ciphers have one big disadvantage of their linearity, which stops them to be utilized directly in any cryptosystem. The cryptographers provide its solution by introducing nonlinear Boolean functions as combiner or filter generator in LFSR based stream ciphers.

The main purpose of using Boolean functions in stream ciphers is to introduce non-linearity in linear LFSRs, in order to avoid linear attacks against LFSRs based crypto systems. However, the utilization of Boolean functions is not limited to have non-linearity only; they can be affectively design/construct in a way to fulfill other cryptographic properties as well. In order to design strong crypto-systems, cryptographers over the years have identified basic cryptographic criteria for Boolean functions, which are required to be fulfilled before applying them in practical systems. These basic cryptographic criteria include balanced-ness, algebraic degree, non-linearity, correlation immunity and algebraic immunity. Algebraic Immunity, which is the most recent and discovered in last decade, is extremely important as it characterizes resistance of a Boolean function against algebraic attacks and fast algebraic attacks.

This thesis deals with existing methods of constructing Boolean functions and their analysis in terms of fulfilling basic cryptographic criteria in order to effectively utilize them in practical crypto-systems.

1.2 Background for the Thesis

In this section, we have mentioned the motivation of this research, the problem statement and objectives of the thesis.

1.2.1 Motivation

Constructing Boolean functions, such that they fulfill all the basic cryptographic criteria had become a vital task for cryptographers in the last decade, especially after the introduction of algebraic attacks in 2003. In this regard many construction methods have been proposed in literature, mainly focus on achieving maximum algebraic immunity to avoid algebraic attacks. D. K. Dalai, for the first time present the basic theory for constructing Boolean functions with maximum algebraic theory in his world renowned paper [11]. The constructed functions are the majority functions under the class symmetric Boolean functions. However, the author did not get optimal values of other cryptographic properties (like nonlinearity, correlation immunity, etc). Moreover, his construction provides only one Boolean function for each value of n (number of variables). In [21], E. Pasalic introduced an iterative concatenation method for construction of Boolean functions utilizes four initial functions with maximum algebraic immunity and one out of them should fulfill algebraic attack resistant property, which author defined as mandatory for resistant against fast algebraic attacks. The proposed infinite class of Boolean function optimizes almost all the cryptographic criteria except achieving very high non-linearity, also his construction required large set of initial functions with maximum algebraic immunity, to construct Boolean

functions with higher value of n. In [24], the author proposed two methods to obtain maximum AI by using concatenation. The author also studies the balancedness, algebraic immunity and nonlinearity of constructed functions, showing that concatenation does not degrade these important cryptographic properties. However, his construction also required large set of initial functions with maximum algebraic immunity to start with. In [22], the author proposed a new method of constructing Boolean functions using primitive polynomials and achieved Balancedness, optimal values of algebraic immunity and algebraic degree, and a very high value of non-linearity. However, the author does not talk about correlation immunity of his constructed functions and his method can construct only one Boolean function for each value of n. In [6], Claude Carlet proposed an infinite class of constructing Boolean functions with maximum algebraic immunity and high non-linearity checked for small number of variables. The proposed method produces Boolean functions with maximum AI and a very high value of non-linearity; however the author has not discussed anything about their degree and correlation immunity. Few other construction methods with optimal algebraic immunity are also proposed in [16, 17, 25, 26], but somehow all these methods fail to fulfill all the cryptographic criteria along with maximum algebraic immunity.

This brings the motivation for this research to conduct a comparative analysis/study of few existing methods of constructing Boolean functions with maximum algebraic immunity and evaluate them in terms of fulfilling other cryptographic criteria. Additionally, in literature, only theoretical constructions are given, but their actual implementations are not found, therefore efficient implementations of each construction method would also

be carried out in this research. Moreover, existing construction methods are not evaluated in terms of higher order non-linearity. Identification of any improvement in existing methods, such that to construct more number of Boolean functions with maximum algebraic immunity is also a main motivation of this thesis.

1.2.2 Problem Statement

As it is always desirable to construct Boolean functions such that they fulfill all the cryptographic criteria including optimal algebraic immunity. Existing construction methods, although construct Boolean functions with optimal algebraic immunity, but somehow lacked in fulfilling rest of the cryptographic criteria. Moreover, theoretical constructions of good Boolean functions are available in literature but their actual construction or implementation is not available. These methods are also not analyzed in terms of their higher order non-linearity.

Additionally, mostly existing construction methods are iterative in nature, i.e. utilize existing Boolean functions with maximum algebraic immunity as their initial functions, therefore more number of Boolean functions with maximum algebraic immunity are required when $n > 10$. However, existing methods e.g. [11, 22] provide only one Boolean function for each value of n / primitive polynomial respectively.

1.2.3 Objectives

The aim of the research is to carryout comparative study of four existing methods of constructing Boolean functions with optimal algebraic immunity and to analyze them in terms of higher order non-linearity, in order to further optimize them or to enhance these existing construction methods.

The main objectives of this research include: -

- Computing cryptographic properties efficiently which include balancedness, algebraic degree, non-linearity, correlation immunity and algebraic immunity.
- Efficient implementation of existing construction methods, to get respective Boolean functions and to analyze them in terms of their cryptographic properties.
- Analysis of Boolean functions construction methods in terms of Higher Order Non-Linearity.
- Extend existing construction methods to obtain more number of Boolean functions with maximum algebraic immunity.

1.3 Priliminaries

Hamming weight of a binary string or vector is the number of nonzero elements present in it.

Hamming distance between two equal binary strings or vectors is the number of corresponding unequal elements present in it.

A Boolean function is called **Affine** function, denoted by A (n), if it has no term of degree strictly greater than 1 in its ANF.

An affine function in which constant term equal to zero is called **Linear** function and all non-constant affine functions (i.e. linear functions) are balanced. The **Support** of a Boolean function, denoted by supp (f), is the set of all input vectors for which the output of the Boolean function is 1.

In nonlinear **Filter Generator**, the n stages of a single, usually longer, LFSR are filtered by a nonlinear Boolean function which produces keystream.

In nonlinear **Combiner Function**; the output of several LFSRs is combined by using a nonlinear Boolean function that produces the keystream.

Symmetric Boolean functions are those which output the same values for all the inputs of same weight. Thus in its ANF, the symmetric Boolean functions contain all the terms of same degree monomial or no term of that degree.

1.4 Organization of the Thesis

This thesis report is organized as follows: chapter 1 gives a brief introduction of the thesis, its background, motivation, problem statement and objectives. Chapter 2 provides a detailed description of Boolean functions, their desirable cryptographic properties and how to compute them. Chapter 3 consists of literature review. Chapter 4 gives an efficient implementation of existing methods for constructing Boolean functions, along with comparison matrix at the end. Chapter 5 provides analysis of existing construction methods in terms of higher order non-linearity. Chapter 6 includes proposed method for extending existing construction methods to get more number of Boolean functions with optimal AI, and finally at the end conclusion and contribu-

tions are presented in Chapter 7.

- **System Requirements:** All implementations (including implementation of existing construction methods and experimental results), are performed in computer, with Processor Intel(R) Core(TM) i3-2310M @ 2.10 GHz, 4 GB RAM and 64-bit Operating System. Microsoft Visual Studio 2012 has been used for implementation of algorithms and for generation of experimental results. However, for verification of results Maple-16 (64-bit) has been used.

Chapter 2

Overview of Boolean Functions

A Boolean function $f(x_1, ..., x_n)$ of n-variables is a multivariate polynomial over GF(2) used as a combiner or filter function in stream ciphers or as s-box component in block ciphers. However, before utilizing them in practical crypto systems, cryptographers over the years have determined certain cryptographic criteria for constructing cryptographic secure Boolean functions. These cryptographic criteria include balancedness, algebraic degree, non-linearity, correlation immunity and algebraic immunity.

The number of Boolean functions with respect to number of variables rises exponentially. The total number of Boolean functions constructed for any value of n (where n is number of variables) are equal to 2^{2^n}. Therefore, it is not possible to construct all Boolean functions for higher values of n (i.e. if n $>$ 6), due to computational limits. Table 2.1 shows the number of Boolean functions up to 10 variables.

A Boolean function can be represented in any of two most commonly forms; i.e. Algebraic Normal Form (ANF) or Truth Table (TT) Form. ANF

No of Variables (n)	**2^n**	**No of Boolean functions (2^{2^n})**
1	2	4
2	4	16
3	8	256
4	16	65536
5	32	4294967296
6	64	18446744073709551616
7	128	3.402e+38
8	256	1.157e+77
9	512	1.340e+154
10	1024	1.797e+308

Table 2.1: Total Numbers of Boolean Functions (up to n=10)

is the representation of a Boolean function in polynomial form over GF2, which is the natural form. Mathematically it can be represented as:

$$f_{ANF}(x_1, x_2..., x_n) = a_0 + \sum_{i=0}^{n} a_i x_i + \sum_{1 \le i < j \le n} a_{i,j} x_i x_j + ... + \sum_{1 \le i_1 \le ... \le i_{n-1} \le n} a_{i_1,...i_{n-1}} x_{i_1} ... x_{i_{n-1}} + a_{1,...n} x_1 ... x_n$$

where $a_0, a_1, a_2, a_3, ..., a_{1,...,n}$ are called the co-efficients of the respective monomials. Boolean functions most commonly represented in TT form, which contains output column values in natural order of input vectors in GF2, denoted by TT, by carrying binary string of length 2^n as shown below:

$$f_{TT} = f(a_0).f(a_1)...f(a_{2^n-1})$$

Example:-

$f_{ANF} = x_1 + x_2 + x_3 + x_1x_2x_4 + x_3x_4$

$f_{TT} = 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0$

2.1 Basic Cryptographic Properties of Boolean Functions

The use of linear feedback shift register (LFSR) based stream ciphers have one main problem of linearity. To overcome this, cryptographers use non-linear Boolean functions as filter generators or combiner functions, due to which the security of whole cryptosystem is mainly dependent on the cryptographic characteristics of these Boolean functions. Therefore various design criteria's have been identified over the years to construct cryptographically strong Boolean functions. These basic cryptographic criteria (or cryptographic properties) required for designing/construction of these Boolean functions are briefly explained below.

- **Balanced-ness:** Balanced-ness is the first and foremost requirement that a good crypto-system should have. A Boolean function is called balanced if it outputs equal number of zeros and ones, i.e. its hamming weight equals 2^{n-1}. Balancedness is required to prevent the system from leaking statistical information on the plaintext, when the ciphertext is known. Moreover, balanced-ness is mandatory to get Boolean functions (for odd n only) with maximum AI, as shown in [10]. The total number of balanced functions, out of total space of a Boolean function of n-variable, is $\begin{pmatrix} 2^n \\ 2^{n-1} \end{pmatrix}$.

- **Algebraic Degree:** Algebraic degree, denoted by Deg (f), of a Boolean function is the number of variables in the highest order monomial with nonzero coefficient in its algebraic normal form (ANF). A high algebraic degree is required to achieve cryptographically secure Boolean functions, in

order to prevent them from Masseys Attack, applied by using Berlekamp-Massey algorithm, as shown in [15] . Additionally, the maximum (or optimal) value of algebraic degree of an n-variable Balanced Boolean function would be n-1, as shown in [12]

- **Non-Linearity:** The non-linearity, denoted by N_f, of a Boolean function is the number of bits to change in the truth-table of the function to make it an affine function. In other words, nonlinearity is the minimum hamming distance from the set of all n-variable affine functions, i.e. N_f = min (d (f, g)), where g $\in$ set of affine functions. A high value of non-linearity is required to resist against affine approximation attacks and linear attacks. However, it is known that there is an upper limit for maximum nonlinearity which is $2^{n-1} - 2^{\lceil \frac{n}{2} \rceil - 1}$.

- **Correlation Immunity (Resiliency):** A Boolean function is said to be correlation immune of order m, if distribution of its truth table is unaltered while fixing any m inputs, denoted by CI (f), or in other words, the output of Boolean function must be independent of combination of any m inputs. If the Boolean function is correlation immune of order m and is also balanced then it is called m-resilient. A Boolean function should have high order of correlation immunity to resist correlation attack.

- **Algebraic Immunity (AI):** The algebraic immunity (AI) was first introduced by W. Meier [19] as a measure of resistance against algebraic attacks. Algebraic Attacks [8], exploits the structure of the underlying functions to construct an over-defined system of nonlinear multivariate equations that will allow to determine the secret key. The AI of a Boolean function f is defined as the lowest degree of the function g for which f. g = 0, or (f+1).g

=0; the function g for which f. g = 0 is called an annihilator of f. In other words we can understand algebraic immunity in terms of algebraic attacks, i.e. in algebraic attacks, instead of setting up a system of equations of degree determined by the degree of Boolean function f, the attacker can consider a lower degree system if there either exists a low degree function g (known as Annihilator) such that f . g = 0 or (1 + f). g = 0. The minimum degree of this nonzero annihilators g is called algebraic immunity (AI).

For any n-variable Boolean function, its maximum algebraic immunity achievable is the upper bound of its number of variables divided by 2, i.e. $AI(f) = \lceil \frac{n}{2} \rceil$. As shown in [8] that there is always an annihilator g exists with degree at most $\lceil \frac{n}{2} \rceil$ such that f.g is of degree atmost $\lceil \frac{n}{2} \rceil$. If the bound is achieved, we say the Boolean function has optimal algebraic immunity.

2.2 Attacks against Boolean Functions

In case of stream ciphers, most commonly attacks are linear cryptanalysis, differential cryptanalysis, statistical attacks, correlation attacks, distinguishing attacks, etc which are all statistical in nature. However, in the last decade algebraic attacks have been extensively used against LFSR based stream ciphers implemented using nonlinear Boolean functions. Algebraic attacks are successfully applied on Toyocrypt and LILI-128 as shown in [8], due to low algebraic immunity of their Boolean functions. Fast algebraic attacks further reduce the complexity of algebraic attacks by reducing the degree of equations. Due to their importance, in targeting crypto-systems based on Boolean functions they are further explained below.

• **Algebraic Attacks (AA)** Algebraic Attacks are first introduced by Courtois and Meier [8]. It is a known plaintext attack against stream ciphers, where attacker has the knowledge of keystream. The main aim is to recover the internal state bits (secret-key) of the LFSRs by considering the whole cipher as a large system of multivariate algebraic equations. The attacker constructs the system of equations by collecting a large number of bits from keystream i.e. each new bit of the key stream provides a new equation on initial state, then converts the attack as solving a system of polynomial equations. It consists of two steps; finding the system of polynomial equations and solving the system of equations using any efficient method. The complexity of the attack is mainly dependent on the degree of the algebraic system, the efficiency of generating algebraic equations and finally to solve these multivariate algebraic equations. Various efficient methods are used to solve the algebraic equations like Linearization, XL, Groebner bases, etc; however there efficient solution is available if degree of equations is low.

• **Fast Algebraic Attacks (FAA)** Algebraic attacks may be further improved by exploiting linear relations among the keystream bits; this approach is called Fast Algebraic Attack, first proposed by N, Courtois in [9]. Fast algebraic attacks are much more efficient than the usual algebraic attacks. In the fast algebraic attacks, the attacker tries to decrease the degree d of the system of equations even further by searching for relations between the initial state of the LFSR and several bits of the output function simultaneously. Fastest attack against Toyocrypt, LILI-128, E0 (Bluetooth), Summation generator, Sober-t32, etc.

2.3 Computing Cryptographic Properties

In this section, we explained the major modules/tools to compute the cryptographic properties of a Boolean function. We started with Boolean function representation in which Moebius Transform technique is used to convert a Boolean function from truth table form into ANF form, then code for finding balancedness and algebraic degree is explained. Walsh Transform is implemented next, which is used to compute Non-Linearity and Correlation Immunity. Finally, we have implemented algorithm for computing Algebraic Immunity. All the modules/tools are written in C-language, which is chosen because of its fast processing speed in handling/computing large amount of data.

2.3.1 Moebius Transform for ANF Representation

The Boolean functions can be represented in various forms, e.g. ANF, truth table, etc. But for implementation point of view most desirable form for taking the Boolean function as input is truth table representation, as it contains 1 and 0s so easily usable form in machine or C-language. However, the desirable form for the output is ANF, as it is more human readable than truth table. In our research, we take the Boolean function in truth table form (as input) however there is an option that user might put input in ANF form and the program then converts it internally to generate its truth table and later on utilize it to compute cryptographic properties. The output or constructed Boolean functions are in ANF form as it is more convenient to express the results. The conversion from truth table form to ANF form is accomplished

by using Moebius Transform technique which is explained below:

Moebius Transform: This following program will convert truth table of a Boolean Function into its Algebraic Normal Form (ANF) by using Moebius Transform Technique. The program is tested and verified up to n = 18 variables. For example if input of a Boolean function of n = 3 variables is given in truth table form as TT=10100110 then the Moebius Transform output would be ANF = $1 + x^1 + x^3 + x^2x^3$.

```
//*****CONVERTING TRUTH-TABLE INTO ANF (MOEBIUS TRANSFORM)
void Moebius_Transform(unsigned char *truth_table)
{
  unsigned int i=0,i0=0,i1=0,step=0;
  for (step=1;step<size_TT;step<<=1)
  {   for (i1=0;i1<size_TT;i1+=2*step)
      {    for (i0=0;i0<step;i0++)
           {
                i=i1+i0;
                truth_table[i+step]^=truth_table[i];
  }   }    }
}// End Moebious Transfor Function
```

2.3.2 Algebraic Degree and Balancedness

Computing balancedness is not too difficult, to see whether a Boolean function is balanced or not, simply calculate the weight of its truth table i.e. count the number of 1s in the truth table and compare it with half the num-

ber of truth table entries i.e. $wt(f) = 2^{n-1}$, if the answer is 'True' means function is balanced otherwise Not. On the other hand, algebraic degree can easily be computed, if we know the ANF (which can be calculated through Moebious Transform) of the Boolean funciton. The code below calculates the degree of a Boolean function (using ANF) and stored its value in the variable 'degree'.

```
void Moebius_Transform(unsigned char *);
unsigned char truth_table[256]={0}; //max=256 for n=8 variables
Moebius_Transform(truth_table);
terms_BF = 0;
degree=0;
for(i=0;i<size_TT;i++)
{ if(truth_table[i])
  { not_count = 0;
    terms_BF++;
    if(terms_BF > 1)  fprintf(fwrite,"+");
    for(j=0;j<size_n;j++)
      {   //Printing ANF in an external file
      if(i>>j & 0x01) { fprintf(fwrite,"x%d",j+1);  d++; }
      else not_count++;
      }
    if(not_count == size_n)   fprintf(fwrite,"1");
    if(d>degree) degree = d;   //Computing Degree of BFn
    d=0;
   }
}
```

2.3.3 Walsh Transform for Non-Linearity and Correlation Immunity

Walsh Transform is the best tool to compute Non-linearity of a Boolean Function. It is fast and efficient algorithm, but compute only first order non-linearity. Moreover the maximum value of Walsh transform is further utilized to compute Correlation Immunity of a Boolean function. The source code of computing both non-linearity and correlation immunity of a Boolean function is mentioned in appendix A.1.

2.3.4 Computing Algebraic Immunity

Algebraic Immunity is the most important part of this whole research and computing AI is the most difficult task, since it requires a lot of computation power. As during computation of algebraic immunity, it is required to solve a matrix of size $2^n \times 2^n$ and then transform it into reduced echelon form, due to which complexity and computation increases manifold with each increment on the value of n. In literature, few methods are found in [2, 13, 14, 18] for computing algebraic immunity, but are very complex and difficult to implement directly from given information. Moreover online tools on Internet are readily available for computing other cryptographic properties but no such tools are available for computing AI directly. This is also a contribution of this research that an efficient algorithm for computing algebraic immunity has been written and implemented successfully for up to n=16 variables. The algorithm, i.e. source code, of all the processing steps for computing Algebraic Immunity of a Boolean function are mentioned in appendix A.2.

2.4 Searching Optimal Boolean Functions by Exhaustive Search up to n=4 variables

The next step, after writing and implementing necessary algorithms for computing cryptographic properties of a Boolean function would be to start constructing Boolean functions. Instead of implementing existing methods of constructing Boolean functions, we started the brute force method to implement all possible Boolean functions of n=3 and n=4 variables. The number of Boolean functions increases exponentially very large with each increment on the value of n i.e. 2^{2^n}. Therefore, its not possible to go beyond n=4 and constructed 255 Boolean functions for n=3 and 65535 Boolean functions for n=4. So in this research, we have computed cryptographic properties of all the Boolean functions of n=3 and n=4.

- **Exhaustive Search for n=3 variables:** For n=3 variables, there are 255 possible Boolean functions that one can construct. After construction we have calculated all the cryptographic properties of these constructed Boolean funcitons. The number of Boolean Functions found for optimal value of each cryptographic property are mentioned in Table 2.2 in decending order i.e. first we write number of functions with max AI, then Balanced functions within these max AI functions, then max N_f and Degree funcitons within max AI and balanced functions and so on. In the rest of the section, B means Balanced, D is Degree, N_f is Non-Linearity, CI is Correlation Immunity and AI is Algebraic Immunity.

Boolean Functions Search	No. of BFns	Percentage (%)
Total BFns for n=3 variables	255	100 %
BFns with Max AI	190	74.5%
BFns Balanced	65	25.5%
BFns with Max $N_f = 2$	57	22.4%
BFns with Max Degree = 2	57	22.4%
BFns with Max CI	8	3.1%

Table 2.2: No of Boolean functions found with respect to optimal cryptographic properties for n=3 variables

The maximum optimal values we found for cryptographic properties (B, D, N_f, CI, AI) are (1, 2, 2, 2, 2) for eight Boolean functions, i.e. four Boolean functions as mentioned in Table 2.3 and four their respective complements, here B is Balanced, D is Degree, N_f is Non-Linearity, CI is Correlation Immunity and AI is Algebraic Immunity.

S.No	TT	ANF(f)	B	D	N_f	CI	AI
1	10001110	$x_1 + x_2 + x_1x_2 + x_1x_3 + x_2x_3$	1	2	2	2	2
2	10110010	$x_1 + x_1x_2 + x_3 + x_1x_3 + x_2x_3$	1	2	2	2	2
3	11010100	$x_2 + x_1x_2 + x_3 + x_1x_3 + x_2x_3$	1	2	2	2	2
4	11101000	$x_1x_2 + x_1x_3 + x_2x_3$	1	2	2	2	2

Table 2.3: Most optimal Boolean functions found in brute force search for n=3 variables

- **Exhaustive Search for n=4 variables:** For n=4 variables, there are 65535 possible Boolean functions that one can construct. After construc-

tion we have calculated all the cryptographic properties of these constructed Boolean funcitons. The number of Boolean Functions found for optimal value of each cryptographic property are mentioned in Table 2.4 in decending order i.e. first we write number of functions with max AI, then Balanced functions within these max AI functions, then max N_f and Degree funcitons within max AI and balanced functions and so on. In the last row of the table, we have mentioned number of Boolean functions found for max correlation immunity which are 200, although degree is one less than optimal, i.e. degree = 2, while other cryptographic properties are (B, D, N_f, CI, AI) = (1, 2, 4, 3, 2).

Boolean Functions Search	No. of BFns	Percentage (%)
Total BFns for n=4 variables	65535	100 %
BFns with Max AI	62613	95.5%
Balanced BFns	12857	19.6%
BFns with Max $N_f = 4$	10920	16.6%
BFns with Max Degree = 3	10080	15.4%
BFns with Max CI = 3 (D=2)	200	0.3%

Table 2.4: No of Boolean functions found with respect to optimal cryptographic properties for n=4 variables

The maximum optimal values we found for cryptographic properties (B, D, N_f, CI, AI) are (1, 3, 4, 0, 2) for 10080 Boolean functions. Moreover we also found around 200 Boolean functions with maximum correlation immunity equal to three, i.e. (B, D, N_f, CI, AI) = (1, 2, 4, 3, 2), however the

degree is one less than optimal value, i.e. Degree = 2. This also shows that there is a trade off between algebraic degree and correlation immunity. One doesnot get optimal values of both of these cryptographic properties at the same time. In Table 2.5, few examples out of these optimal Boolean funcitons for both optimal degree and optimal correlation immunity are mentioned.

S.No	Truth Table	ANF(f)	B	D	N_f	CI	AI
1	1001010111001010	$x_1 + x_1x_3 + x_2x_3 + x_4 + x_1x_3x_4$	1	3	4	0	2
2	1001010111001100	$x_2 + x_4 + x_1x_4 + x_2x_4 + x_2x_3x_4$	1	3	4	0	2
3	1001010111100010	$x_1 + x_1x_2 + x_2x_3 + x_4 + x_1x_2x_4$	1	3	4	0	2
4	1001010111110000	$x_3 + x_4 + x_1x_4 + x_3x_4 + x_2x_3x_4$	1	3	4	0	2
5	1111101000001010	$x_1 + x_1x_3 + x_3x_4$	1	2	4	3	2
6	1111101001010000	$x_3 + x_1x_3 + x_1x_4$	1	2	4	3	2
7	1111110000001100	$x_2 + x_2x_3 + x_3x_4$	1	2	4	3	2
8	1111110000110000	$x_3 + x_2x_3 + x_2x_4$	1	2	4	3	2

Table 2.5: Four examples of Boolean functions, each of optimal algebraic degree (S.No 1-4) and optimal correlation immunity (S.No 5-8) found in exhaustive search for n=4 variables

Chapter 3

Literature Review

3.1 Introduction

As discussed in previous sections, Boolean functions play an important role in designing any modern symmetric cipher. Due to their imperative role, a large number of methods have been published in recently past for constructing Boolean functions with optimal algebraic immunity along with fulfilling other cryptographic criteria. However, these methods lacks in fulfilling all the cryptographic criteria at the same time along with getting optimal algebraic immunity.

3.2 Existing Construction Methods

This section provides literature review of some important existing methods of constructing Boolean functions with maximum algebraic immunity, mostly iterative and concatenated in nature. In this research, out of these existing

methods, four methods have been selected for comparative study; in which the said methods are constructed by practically implementing them and further analyzed each of them in detail to identify some better/improved results.

3.2.1 Method-1 : Basic Theory by D. K. Dalia

For the first time, the basic theory for constructing Boolean functions with optimal algebraic theory was presented by D. K. Dalai in his world renowned paper [11]. The constructed functions are the majority functions under the class symmetric Boolean functions. These symmetric Boolean functions can easily be transformed into non-symmetric Boolean functions, to cover the complete class, using linear transformation (as it does not affect other properties like algebraic degree, nonlinearity, etc). The basic motivation of this construction was a systematically study of Boolean functions that can achieve maximum algebraic immunity, although the author did not get optimal values of few other cryptographic properties (like nonlinearity, correlation immunity, etc). Therefore the author discourages the use of these functions in practical crypto systems and left it as an open problem to work further. A detailed analysis of the said method has been carried out in this research by practically implementing it and then computing all the major cryptographic properties of constructed Boolean functions including algebraic immunity. It is further referred as Construction-1 in this report.

3.2.2 Method-2 : Iterative Concatenation Method by E. Pasalic

In [21], E. Pasalic introduced an iterative concatenation method for construction of Boolean functions which can also protect fast algebraic attack other than protecting classical algebraic attacks. The iterative method takes four initial functions with optimal algebraic immunity and after concatenation the constructed Boolean function has degree increased by 2 and algebraic immunity increased by 1, i.e. optimal. He further showed that the functions that resist fast algebraic attacks must have maximum algebraic immunity, although there are some additional requirements also. His constructed functions, he called them Algebraic Attack Resistant (AAR) functions are also extremely efficient in hardware implementation and fulfill most of cryptographic criteria including high non-linearity and maximum algebraic degree. The main motivation of the author was due to the fact that for the time being there is no method that fulfills all the cryptographic criteria including maximum algebraic immunity. Moreover, existing methods do not resist against fast algebraic attacks and do not have efficient hardware implementation. The proposed infinite class of Boolean function optimizes almost all the cryptographic criteria except achieving very high non-linearity but they do resist fast algebraic attacks with efficient hardware implementation. The author left it as future work to find out further improvement in terms of non-linearity of his proposed AAR functions. This method is further referred at Construction-2 in this research, and a detailed analysis of the said method has been carried out by practically implementing it and then computing all

the major cryptographic properties of constructed Boolean functions including algebraic immunity.

3.2.3 Method-3 : Boolean Functions by Concatenation

In [24], the author proposed two methods to obtain n-variable Boolean functions from (n-1) variables or (n+1) variables functions, i.e. according to its construction-1, one can obtain two even Boolean functions of (n-1) variables with maximum Algebaric Immunity(AI) by decomposing one odd Boolean function of n-variable which must be of maximum algebraic immunity or in its second construction-2, one can construct an odd Boolean function of (n+1) variables with maximum AI by concatenating two even Boolean functions of n-variable which must be of maximum AI. Moreover, the author also studies the balancedness, algebraic immunity and nonlinearity of constructed functions, showing that concatenation does not degrade these important cryptographic properties. Moreover the author also suggested it as future work to study the properties of concatenation, which can further improve these cryptographic properties of constructed Boolean functions.

3.2.4 Method-4 : Boolean Functions using Primitive Polynomials

In [22], the author proposed a new method of constructing Boolean functions using primitive polynomials and achieved Balancedness, optimal values of algebraic immunity and algebraic degree, and a very high value of nonlinearity. The author in fact exploits the built in properties of primitive

polynomials, by constructing Boolean functions using the output of these primitive polynomials. Since the output of primitive polynomial is always Balanced, i.e. $wt(f) = 2^{n-1}$ therefore one property is achieved. Next the output of primitive polynomials is always random with a very high value of linear complexity, this helps in achieving a very high value of non-linearity. Similarly, the author found that this constructed Boolean functions have maximum algebraic immunity along with optimal value of algebraic degree.

3.2.5 Other Construction Methods

As discussed in previous sections, Boolean functions play an important role in designing any modern symmetric cipher, due to this a large number of new attacks were also introduced in last decade. The most important and successful attacks were Algebraic Attacks and Fast Algebraic Attacks, introduced by N. Courtois in [8], [9] for LFSR based stream ciphers. In the later half of last decade, various methods of constructing Boolean functions (to resist these attacks) with optimal algebraic immunity have been proposed, few of them are briefly mentioned below.

In [5], Dalai for the first time put an effort to introduce a method which mainly focuses at algebraic immunity of constructed Boolean functions and then look at other cryptographic properties. The proposed method is based on iteration, in which four initial functions are concatenated as $f = E \parallel F \parallel G \parallel H$ and at every step two variables are added and AI increased by one. However, the author here only focus on algebraic immunity and has not

given any values for other cryptographic properties. In [11], the same author, i.e. Dalai presented the famous Basic Theory which is also implemented (as Method-1) in this research and explained in much detail in this research. In [20], the author presented construction for a class of 1-resilient functions with optimal AI for n=even only, which is based on concatenation of two balanced functions. For some values of n, the author claimed to achieve optimal values of non-linearity and algebraic degree. However, the author couldn't provide exact values of non-linearity for his complete construction; moreover the larger class of 1-resilient Boolean functions are constructed with sub-optimal algebraic immunity. functions with optimal algebraic immunity along with fulfilling other cryptographic criteria. However, these methods lacks in fulfilling all the cryptographic criteria at the same time along with getting optimal algebraic immunity.

In [6], Claude Carlet proposed an infinite class of constructing Boolean functions with maximum algebraic immunity and high non-linearity checked for small number of variables. The author also claimed that no infinite class with optimal AI and high non-linearity has been proposed so far. The porposed construction is defined in Theorem-1 of [5] as suppose f be a Boolean function on $\mathbb{F}_{2^n}$ whose support is $\{0, 1, \alpha, ..., \alpha^{2^{n-1}-2}\}$, then f has maximum algebraic immunity; where, n is any integer greater than 2, i.e. $n \geqslant 2$ and α is a primitive element. The values of non-linearity that author found for n=6, 7, 8, 9, 10 and 11 using Theorem-1 are 24, 54, 112, 232, 478 and 1001 respectively. Theses values are slightly less than the optimal non-linearity values computed using $2^{n-1} - 2^{\lceil \frac{n}{2} \rceil - 1}$. To conclude, the proposed method produces Boolean functions with optimal AI and a very high value of non-

linearity, however the author has not discussed anything about their degree and correlation immunity.

In [7], Su and Tang proposed two new constructions of infinite classes of rotation symmetric Boolean functions (RSBF) with optimal algebraic immunity on either odd variables or even variables. These functions have better nonlinearity then all the existing methods of constructing RSBF, and their algebraic degree is also high enough. The paper propose the construction for n $\geqslant$ 11 (odd) and n $\geqslant$ 10 for even, assume n = 2k+1 (for odd) and n = 2k (for even) where k $\geqslant$ 5, and give a new construction of balanced RSBF f(x) on n variables. This new constructed function f(x) has optimal algebraic immunity, high nonlinearity and high algebraic degree. The author also derived exact values of non-linearity and algebraic degree for any value of n$\geqslant$10 (for even case) and n$\geqslant$11 (for odd). In short, this paper present two new construction of RSBF and studies their cryptographic properties like nonlinearity and algebraic degree theoretically. However, the constructed functions are not checked to be robust against fast algebraic attacks.

Few other construction methods with optimal algebraic immunity are also proposed in [16, 17, 25, 26] , but somehow all these methods fail to fulfill all the cryptographic criteria along with optimal algebraic immunity.

Chapter 4

Efficient Implementation of Boolean Functions Construction Methods

4.1 Introduction

This chapter discussed the details of implementation and results of four existing methods [11, 21, 22, 24], for constructing Boolean functions with optimal algebraic immunity. It also present any shortcoming found in any of the methods during implementation. At the end a comparative study carried out, based on the results found during implementation. The study carried out here would be very helpful in utilizing any of these construction methods in practical crypto systems.

4.2 Implementation of Construction Methods

4.2.1 Implementation of Method-1 : Basic Theory

Method-1 includes basic theory presented by D. K. Dalai [11], to construct Boolean functions with maximum algebraic immunity. The author in his

paper proposed the basic theory as construction-1 and further presented construction-2 and construction-3 as its applications. The constructions presented by Dalai are explained below:-

- **Construction - 1:** According to construction-1 of Dalai, suppose there are three Boolean functions f, f_1 and f_2, then

(a) If f_1 and f_2 has max AI such that supp$(f) \supseteq$ supp(f_2) and supp$(1 + f) \supseteq$ supp(f_1) then$f$ also has max AI.

(b) If f has max AI, fulfilling above condition then f_1 and f_2 also have max AI, i.e. the opposite of part (a).

- **Construction - 2:** In construction-2 of Dalai, the author presented an application of basic theory in which he constructed functions, separately for odd and even n, with maximum algebraic immunity as mentioned below:-

(a) Suppose n is odd, then

$$f(x_1, ..., x_n) = 0 \ \ for \ \ wt(x_1, ..., x_n) \leq \lfloor \tfrac{n}{2} \rfloor$$

$$= 1 \ \ for \ \ wt(x_1, ..., x_n) \geq \lceil \tfrac{n}{2} \rceil$$

(b) Suppose n is even, then

$$f(x_1, ..., x_n) = 0 \ \ for \ \ wt(x_1, ..., x_n) < \tfrac{n}{2}$$

$$= 1 \ \ for \ \ wt(x_1, ..., x_n) > \tfrac{n}{2}$$

$$= \text{b} \in \{0, 1\} \ \ for \ \ wt(x_1, ..., x_n) = \tfrac{n}{2}$$

Here for odd n, we exactly got one Boolean functions, but for even n, we have to choose from remaining $\begin{pmatrix} n \\ \frac{n}{2} \end{pmatrix}$. For example: for n=4, we have one input value for weight 0 and 4, four input values for weight 1 and 3 and remaining 6 input values for weight 2, which can be chosen randomly, such that to get the resulted function balanced.

- **Construction – 3:** The construction-3 of Dalai deals with symmetric Boolean function only for both even and odd cases, as described below :-

$$f(x_1, ..., x_n) = 0 \ \ for \ \ wt(x_1, ..., x_n) \leq \lfloor \tfrac{n}{2} \rfloor$$

$$= 1 \ \ for \ \ wt(x_1, ..., x_n) > \lfloor \tfrac{n}{2} \rfloor$$

In this research, we have implemented Construction-3 of Dalai up to n=16 variables, for both odd and even n-variables (beyond that is out of computer hardware limitations). All the cryptographic properties like Balancedness, Algebraic Degree, Non-linearity, Correlation Immunity and Algebraic Immunity have been calculated. Some very interested and useful results extracted by implementing Construction-3 are mentioned below, which also answers some open problems presented in his paper. Note all results are computed in computer, with Processor Intel(R) Core(TM) i3-2310M @ 2.10 GHz, 4 GB RAM and 64-bit Operating System and Microsoft Visual Studio 2012 has been used for implementation.

(a) In the first step, Cryptographic properties of constructed Boolean functions have been computed as shown in Table 4.1 up to n = 16 variables. Due to space constraint the ANF and Truth Table, which is also computed, are not shown here.

n (var)	**Bal**	**Deg(f)**	**Non-Linearity**	**CI(f)**	**AI(f)**	**CPU Time**
3	1	2	2	2	2	$<$ 1 sec
4	0	4	5	0	2	$<$ 1 sec
5	1	4	10	4	3	$<$ 1 sec
6	0	4	22	0	3	$<$ 1 sec
7	1	4	44	6	4	$<$ 1 sec
8	0	8	93	0	4	$<$ 1 sec
9	1	8	186	8	5	$<$ 1 sec
10	0	8	386	0	5	$<$ 1 sec
11	1	8	772	10	6	1.96 sec
12	0	8	1586	0	6	2.18 sec
13	1	8	3172	12	7	5.89 sec
14	0	8	6476	0	7	19.93 sec
15	1	8	12952	14	8	126.22 sec
16	0	16	26333	0	8	296.17 sec

Table 4.1: Cryptographic properties of Boolean functions (upto n=16 variables) constructed by implementing construction-3 of Method-1 [11]

(b) The author in his paper claimed that for odd n, there is only one Symmetric Boolean function with maximum AI (computed up to n = 11) which is constructed through his construction-3 (see Page 52, observation-1, in [11]) and left it as open problem to prove or disprove it. In this research, we have calculated AI along with other cryptographic properties of all symmetric Boolean function of n=11 and found that there are at least two symmetric Boolean functions with maximum AI, which disproved authors claim of having only one Boolean function with max AI through his construction-3. The

ANF (in Hex form) of second function found with maximum AI is mentioned below:-

ANF(f) = 80000000000000010000000100010117000000010001011700010117011717 7F000000010001011700010117011717 7F000101170117177F0117177F177F7FFF000000010001011700010117011717 7F000101170117177F0117177F177F7FFF000101170117177F0117177F177F7FFF0117177F177F7FFF17 7F7FFF7FFFFFFF000000010001011700010117011717 7F000101170117177F0117177F177F7FFF000101170117177F0117177F177F7FFF0117177F177F7F FF177F7FFF7FFFFFFF000101170117177F0117177F177F7FFF0117177F17 7F7FFF177F7FFF7FFFFFFF0117177F177F7FFF177F7FFF7FFFFFFF17 7F7FFF7FFFFFFF7FFFFFFFFFFFFFFE

We have computed all 64 possible symmetric Boolean functions of n=11 variables; however due to space constraint few are shown in Table 4.2. Note in table there are total of FOUR functions found with maximum AI, i.e. AI = 6; however two are complement of other two, so actually there are two symmetric Boolean functions with max AI (i.e. S.No-0 constructed by construction-3 and S.No-1 found through our implementation). This is an important result that denied author claim of constructing symmetric Boolean functions with maximum AI by using only construction-3 for odd n. This work is also published in [23]

S. No	Bal	Deg(f)	Non-Linearity	CI(f)	AI(f)
0	**1**	**8**	**772**	**10**	**6**
1	**1**	**10**	**770**	**10**	**6**
2	1	10	790	10	5
3	1	4	792	10	5
4	1	10	662	10	4
.	.	.	.	.	.
59	1	10	662	10	4
60	1	4	792	10	5
61	1	10	790	10	5
62	**1**	**10**	**770**	**10**	**6**
63	**1**	**8**	**772**	**10**	**6**

Table 4.2: Cryptographic properties of all symmetric Boolean functions for n = 11 variables (shown in Bold are BFns with max AI)

(c) For n = Even, the author has not presented any concrete construction for non-symmetric Boolean functions and presented a heuristic construction by which one can make these function balanced by randomly selecting the outputs of those input whose weight is equal to n/2. On the other hand symmetric Boolean functions computed through construction-3 of author, are not balanced and their Correlation Immunity is Zero. In this research, we have computed cryptographic properties of n=Even symmetric Boolean functions, through exhaustive search, such that to get balanced and correlation immune Boolean functions as shown in Table 4.3 up to n=14 variables.

n-var	Bal	Deg(f)	Non-Linearity	CI(f)	AI(f)
4	1	2	4	2	2
6	1	4	20	4	3
8	1	4	88	6	4
10	1	8	372	8	5
12	1	8	1544	10	6
14	1	8	6344	12	7

Table 4.3: Results of heuristic construction for n=Even upto 14 variables

4.2.2 Implementation of Method-2 : Iterative Method

In [21], E. Pasalic proposed an iterative concatenation method to construct a function f with maximum algebraic immunity by concatenating four initial functions, i.e. $f = f_1 \parallel f_2 \parallel f_3 \parallel f_4$, where each f_i has maximum algebraic immunity. The ANF of constructed function f is given below:-

$$f = x_{n+1}x_{n+2}(f_1 + f_2 + f_3 + f_4) + x_{n+1}(f_1 + f_2) + x_{n+2}(f_1 + f_3) + f_1$$

The author here introduced Algebraic Attack Resistant (AAR) functions, having certain specific properties/criteria as defined below:-

Definition-3 : Let f be a Boolean function of n - variables, then f is called AAR function if f has maximum AI, i.e. $AI(f) = \lceil \frac{n}{2} \rceil$, moreover for any non-annihilating function g of degree e, where $e \in \{1, \lceil \frac{n}{2} \rceil - 1\}$, we should have $\deg(fg) = $ d, that satisfies e+d $\geq$ n .

(a) Constructing Algebraic Attack Resistant (AAR) Functions. In the first part, we have calculated all the AAR functions for n=3 and n=4, satisfying above criteria. Following are the main steps:-

(a) Search all functions of n=3 and n=4 var, with max algebraic immunity.

(b) Multiply each function f with all possible g's of degree ranges: $e \in [1, 1]$ for n=3 and $e \in [1, 1]$ for n=4

(c) Short list all those functions which satisfies the condition e+d $\geq$ n .

After implementing the above criteria for n=3 and n=4 variables, the re-

sults achieved are mentioned in Table 4.4. The basic cryptographic properties are also computed, where (B, D, N_f, CI, AI) stands for Balancedness, Algebraic Degree, Non-Linearity, Correlation Immunity and Algebraic Immunity respectively.

No of var(n)	**AAR - Functions**	**Total BFns**	**(B,D,N_f,CI,AI)**
n = 3	56	255	(1,2,2,0,2)
n = 4	6048	65535	(1,3,4,0,2)

Table 4.4: Number of Algebraic Attack Resistant (AAR) functions found for n=3 and n=4 variables along with their cryptographic properties

(b) Constructing n=6 variables Boolean functions by Concatenating four n=4 AAR-functions. In the second part of our implementation, we constructed Boolean functions of n=6 variables by concatenating four n=4 AAR functions as defined in Proposition-1 below:-

Proposition-1 : Let $f = f_1 \parallel f_2 \parallel f_3 \parallel f_4$ be a function with n Even, with its subfunctions having max AI, i.e. $AI(f_i) = \lceil \frac{n}{2} \rceil$. Moreover, let $f_3 = 1 + f_1$ and f_1 is such that for any function g of degree e, where $e \in \{1, \lceil \frac{n}{2} \rceil - 1\}$, we have deg($f_1 g$) = d $\geq AI(f_1)$, and e+d $\geq$ n .Then $AI(f) = \lceil \frac{n}{2} \rceil + 1$, i.e. f has max AI.

Here, the author has defined some additional criteria for f_1 only other than having maximum algebraic immunity. We have choosen n=4 (small even) to start of with this construction. After complete search of 65536 Boolean functions (i.e. total possible functions of n=4 variables), we have found total of 6048 functions that fulfill the condition of AAR functions writ-

ten in Definition-3. Few of these functions found are mentioned in Table 4.5, along with other cryptographic properties.

Although there are many possibilities to concatenate these 6048 AAR-functions with other Boolean functions of n=4 having maximum algebraic immunity, however we have used all these 6048 AAR-functions to construct 6048 functions of n=6 variables. Out of four initial functions of n=4 variables, f_1 would be the AAR-function while f_3 should be complement of f_1, i.e. $f_3 = 1 + f$ according to proposition-1. The remaining two functions, f_2 and f_4 would be any functions of n=4 with max AI.

S.No	TruthTable	ANF(f)	B	D	N_f	CI	AI
1	1100011011110000	$1+x_2+x_3+x_1x_3+x_2x_4+x_1x_3x_4$	1	3	4	0	2
2	1010011011110000	$1+x_1+x_3+x_2x_3+x_1x_4+x_2x_3x_4$	1	3	4	0	2
3	1010110111110000	$x_1+x_2x_3+x_4+x_1x_4+x_3x_4+x_2x_3x_4$	1	3	4	0	2
4	1101101111110000	$x_2+x_1x_3+x_4+x_2x_4+x_3x_4+x_1x_3x_4$	1	3	4	0	2
.	...	...	.	.	.	.	.
6048	0100101000011111	$x_1+x_1x_2+x_3+x_1x_2x_3+x_1x_4$	1	3	4	0	2

Table 4.5: Algebraic Attack Resistant (AAR) functions for n=4 variables

After successful implementation, we got all constructed Boolean functions of n=6 variables with maximum AI. Other cryptographic properties are also calculated, one example of n=6 variable Boolean function constructed using Proposition-1 is mentioned below:-

Example:-

Initial Functions :

$ANF(f_1) = x_1 + x_3 + x_1x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4$

$TT(f_1) = 0101111101001000$

$ANF(f_2) = x_1 + x_2 + x_1x_3 + x_1x_2x_4 + x_3x_4 + x_1x_3x_4$

$TT(f_2) = 0110001101111000$

$ANF(f_3) = 1 + x_1 + x_3 + x_1x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4$

$TT(f_3) = 1010000010110111$

$ANF(f_4) = 1 + x_1 + x_2 + x_3 + x_4 + x_1x_3 + x_1x_2x_4 + x_1x_3x_4$

$TT(f_4) = 1001001101111000$

where cryptographic properties of above four initial functions are (B,D,N_f,CI,AI) = (1,3,4,0,2), however only f_1 satisfies AAR function criteria.

Next we construct f, by concatenating these four initial functions, i.e. $f = f_1 \| f_2 \| f_3 \| f_4$ and then compute cryptographic properties of f, as mentioned below:-

$ANF(f) = x_1 + x_3 + x_1x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4 + x_2x_5 + x_3x_5 +$

$x_3x_4x_5 + x_2x_3x_4x_5 + x_6 + x_3x_5x_6 + x_4x_5x_6 + x_3x_4x_5x_6$

TruthTable(f) = 0101111101001000011000110111100010100000101101111001001101111000

Balancedness = Yes

Degree(f) = 4

Nonlinearity(f) = 24

Correlation Immunity(f) = 0

Algebraic Immunity (f) = 3

(c) Constructing n=8 variables Boolean functions by Concatenating four n=6 AAR-functions. In the next part of our implementation, we constructed Boolean functions of n=8 variables by concatenating four n=6 AAR functions. We take total of 6048 functions of n=6 variables with max AI and after concatenation construct 2016 Boolean functions of n=8 variables. All functions constructed have maximum algebraic immunity, balanced, algebraic degree varies from 5-7, non-linearity varies from 92-112 and correlation immunity equal to zero.

4.2.3 Implementation of Method-3 : Concatenation

In [24], the author proposed a very simple method of concatenation through which one can get Boolean functions with maximum AI. The author proposed two construction methods, i.e. Construction-1 by which one can get two functions with max AI by decomposing an (n+1) function with max AI and Construction-2 by which one can get one (n+1) function by concatenating two functions. In this research we have implemented Construction-2, according to which one can obtain a new n-variable Boolean funciton h_n, by concatenating two functions g_{n-1} and f_{n-1} that are of maximum AI.

Implementation-1 : (from n=3 to n=4) In the first part, we have taken n=3 variables Boolean functions with maximum AI and are Balanced. Then applied concatenation and construct Boolean functions of n=4 variables with maximum AI. Other cryptographic properties are also calculated of the constructed functions, the results for few functions constructed by

concatenation of two n=3 variables BFns are mentioned in Table 4.6.

S.No	**Truth Table**	**Bal**	**Deg**	N_f	**CI**	**AI**
1	TT=1110100011011000	1	3	4	0	2
2	TT=1011100001111000	1	3	4	0	2
3	TT=1110010011010100	1	3	4	0	2
4	TT=1011010001110100	1	3	4	0	2
5	TT=1010110001101100	1	3	4	0	2
6	TT=1001110001011100	1	3	4	0	2
7	TT=1110001011010010	1	3	4	0	2
8	TT=1011001001110010	1	3	4	0	2

Table 4.6: Cryptographic properties of Boolean functions after concatenation from n=3 to n=4

Implementation-2 : (from n=5 to n=6)

In the second part, we have first calculated a large set of Boolean functions of n=5 variables with maximum AI and balanced. Then we applied same concatenation technique and construct Boolean functions of n=6 variables with max AI. The results we obtained are mentioned in Table 4.7.

n - var	No of Const BFns	BFns with Max AI & Bal	CI	N_f	CPU Time
from n=5	1530394	1530394	0-4	2-8	-
to n=6	765197	725338	0-5	24	53.28 sec

Table 4.7: Number of constructed Boolean functions constructed after concatenation from n=5 to n=6 variables and their cryptographic properties

4.2.4 Implementation of Method-4 : Primitive Polynomials

Lastly, we implement the construction of Boolean functions with max AI using Primitive polynomials proposed by Qichun Wang in [22]. In this paper,

the idea used was to utilize output sequence of primitive polynomials which already are highly nonlinear, i.e. its linear complexity is too high, moreover output sequence has equal number of 0s and 1s i.e. balanced, so the author used this output as truth table of a Boolean function, and calculate AI and other properties of constructed Boolean functions. It has been found that algebraic immunity found is always optimal; the function constructed is always balanced with optimal value of algebraic degree along with a very high value of nonlinearity. We have constructed such Boolean functions using Primitive Polynomials of degree n=6 till n=15, and also computed other cryptographic properties of these Boolean functions. The CPU time consumed in constructing and computing cryptographic properties is also computed and is mentioned with other results in Table 4.8.

var(n)	Primitive Polynomial	CPU Time	B	D	N_f	CI	AI
6	$1 + x + x^6$	< 1 sec	1	5	22	0	3
7	$1 + x^3 + x^7$	< 1 sec	1	6	48	0	4
8	$1 + x^2 + x^3 + x^4 + x^8$	< 1 sec	1	7	104	0	4
9	$1 + x^4 + x^9$	< 1 sec	1	8	220	0	5
10	$1 + x^3 + x^{10}$	< 1 sec	1	9	462	0	5
11	$1 + x^2 + x^{11}$	< 1 sec	1	10	940	0	6
12	$1 + x^1 + x^4 + x^6 + x^{12}$	1.26 sec	1	11	1940	0	6
13	$1 + x^1 + x^3 + x^4 + x^{13}$	12.95 sec	1	12	3916	0	7
14	$1 + x^1 + x^3 + x^5 + x^{14}$	67.44 sec	1	13	7910	0	7
15	$1 + x^1 + x^{15}$	736.6 sec	1	14	15988	0	8

Table 4.8: Constructing Boolean functions using primitive polynomials up to n = 12 variables

4.3 Comparison of Implemented Methods

The comparison matrix of all the four existing methods for constructing Boolean functions with maximum algebraic immunity is given in Table 4.9.

Note here, the Boolean functions constructed, using these four methods have optimal values of Algebraic Immunity and Balancedness, but somehow lacks in fulfilling other cryptographic criteria. Method-4, provides best results among these four constructions, as it provides optimal values of algebraic immunity, balancedness and algebraic degree and a very high value of non-linearity (close to its optimal value), which makes it the best choice to be utilized in practical crypto systems.

Methods / Cryptographic Criteria	**Method-1**	**Method-2**	**Method-3**	**Method-4**
Algebraic Immunity (AI)	Optimal AI	Optimal AI	Optimal / suboptimal AI	Optimal AI
Non-Linearity (N_f)	Not very Good	Not fully Optimized	Considerably High	Very High, near Optimal
Algebraic Degree	High but Not Optimized	Optimized	Optimized	Optimized
Correlation Immunity	Not Optimized	Not Optimized	Does not talked about it	Does not talked about it
Balancedness	Yes	Yes	Yes	Yes

Table 4.9: Comparison matrix of implementation methods with respect to cryptographic criteria

A comparison graph is shown in Figure 4.1, which depicts nonlinearity value with respect to n (variables) for construction Method-1 [11], Method-4 [22] for up to n = 12 variables. Clearly shows that nonlinearities of construtions method-4 are much better than method-1.

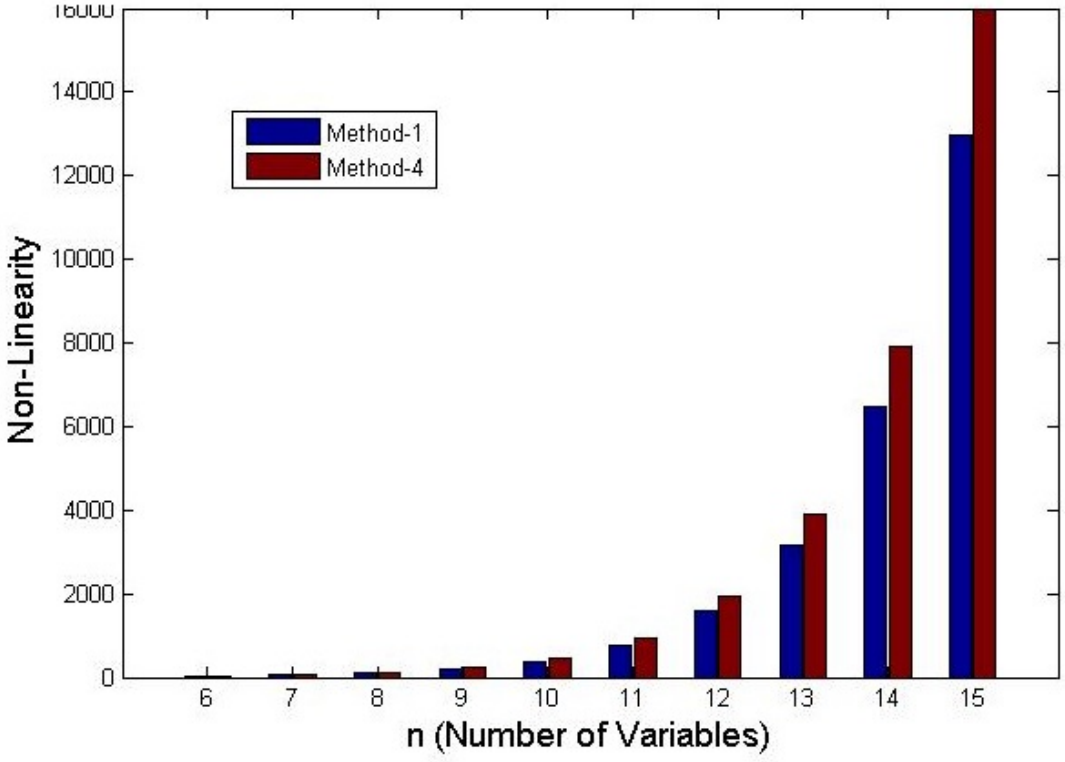

Figure 4.1: Comparison between Method-1 [11] and Method-4 [22] in terms of Nonlinearity up to n = 12 variables

Chapter 5

Analysis of Construction Methods in terms of Higher Order Non-Linearity

5.1 Introduction

The higher order non-linearity is a characteristic of Boolean functions which plays an important role with respect to the security of the crypto-systems in which they are involved. Their cryptographic relevance has been illustrated by several papers like [1,3,4]. However, computing the r-th order nonlinearity is a difficult task especially when the algebraic degree of a given Boolean function is greater than r, where r $>$ 1 and almost no paper has given general effective results on it, in literature. Even the second order nonlinearity is known only for a few particular functions and for functions in small numbers of variables. In this section, we have computed higher order non-linearity of existing four methods up to n=5 variables, as beyond that is computationally unfeasible.

Definition. The higher order nonlinearity of a Boolean function f is the minimum Hamming distance between f and all n-variable Boolean functions

of algebraic degree at most r, for every positive integer $r \leqslant n$, denoted by N_{f_r}. It is also called r-th order nonlinearity.

In [3] Claude Carlet has given the lower bound on the r-th order non-linearity defined in Proposition-5. We have practically implemented it, and found useful results to characterize any Boolean functions in terms of minimum higher order nonlinearity. The lower bound computed up to n=16 variables in mentioned in Table 5.1.

n - var	AI	N_{f_r} (for r=1)	N_{f_r} (for r=2)	N_{f_r} (for r=3)	N_{f_r} (for r=4)	N_{f_r} (for r=5)
5	3	10	2	-	-	-
6	3	12	2	-	-	-
7	4	44	12	2	-	-
8	4	58	14	2	-	-
9	5	186	58	14	2	-
10	5	260	74	16	2	-
11	6	772	260	74	16	2
12	6	1124	352	92	18	2
13	7	3172	1124	352	92	18
14	7	4760	1588	464	112	20
15	8	12952	4760	1588	464	112
16	8	19898	6946	2186	598	134

Table 5.1: Lower bound on r-th order nonlinearity computed upto n=16 variables

5.2 Searching Optimal Functions up to n=4

In this section, we have computed higher order non-linearity of all the Boolean functions of n=3 and n=4 variables. The results are shown in Table 5.2 and Table 5.3.

Boolean Functions Search w.r.t N_{f_r}	No. of BFns	Optimal Value of N_{f_r}	Percentage (%)
Total BFns for n=3 variables	255	-	100 %
1st - Order N_f	112	2	44 %
2nd - Order N_{f_2}	15	2	6%
3rd - Order N_{f_3}	127	1	50%

Table 5.2: Number of Boolean functions found with respect to optimal value of r-th order non-linearity for n=3 variables

Boolean Functions Search w.r.t N_{f_r}	No. of BFns	Optimal Value of N_{f_r}	Percentage (%)
Total BFns for n=4 variables	65535	-	100 %
1st - Order N_f	896	6	1.36 %
2nd - Order N_{f_2}	30	4	0.4%
3rd - Order N_{f_3}	2046	2	3.12%
4th - Order N_{f_4}	32766	1	50.00%

Table 5.3: Number of Boolean functions found with respect to optimal value of r-th order non-linearity for n=4 variables

5.3 Computing Higher Order Non-Linearity of Implemented Construction Methods

In this section, we have computed higher order nonlinearity (N_{f_r}) of four implemented construction methods up to n= 5 variables. Since in literature there is no efficient method is available for computing higher order nonlinearity, therefore we have implemented the basic method, in which the minimum hamming distance of the Boolean function is measured, with all the functions of degree up to 5. The complete source code of our implementation is attached as appendix A.3. In the subsequent paragraphs, the results of

computed N_{f_r} of four implemented methods are mentioned. Note time taken to compute higher order nonlinearity increases exponentially with each increment in n (number of variables), additionally time take to computer only 2nd-order nonlinearity is slightly reduced as compared to time taken for computing all r-th order nonlinearity (e.g. for n=5 variables, CPU time for 2nd-order is reduced to 3917 sec from 4171 sec). Therefore, computationally its unfeasible to compute only 2nd-order nonlinearity for values of n greater than 5.

- **Higher Order Non-Linearity (N_{f_r}) of Method-1.** In this method, Dalai [11] constructed Boolean functions with maximum AI by using majority functions. The N_{f_r} of his construction methods is computed up to n=5 variables, and is shown in Table 5.4

No of Var	Truth Table	N_{f_1}	N_{f_2}	N_{f_3}	N_{f_4}	N_{f_5}	Processing Time
n=3	11101000	2	0	1	-	-	0.00 sec
n=3	00010111	2	0	1	-	-	0.00 sec
n=4	1110100010000000	5	1	1	0	-	0.03 sec
n=4	0001011101111111	5	1	1	0	-	0.03 sec
n=5	11111110111010001110 100010000000	10	2	2	0	1	4171 sec
n=5	00000001000101110001 011101111111	10	2	2	0	1	4171 sec

Table 5.4: The r-th order non-linearity found up to n=5 variables Boolean functions constructed using Method-1 [11]

- **Higher Order Non-Linearity (N_{f_r}) of Method-2.** In this method, Pasalic [21] constructed Boolean functions by concatenating four initial func-

tions and after each iteration he constuct Boolean functions with AI increased by 1, and no of variables increased by 2. The N_{f_r} of his constructed Boolean functions up to n=5 variables, is shown in Table 5.5. Note, since we have implemented his construction for n=Even only so N_{f_r} for n=4 is calculated and shown below.

No of Var	Truth Table	N_{f_1}	N_{f_2}	N_{f_3}	N_{f_4}	Processing Time
n=4	0101111101001000	4	2	0	1	0.03 sec
n=4	0110001101111000	4	2	0	1	0.03 sec
n=4	1010000010110111	4	2	0	1	0.03 sec
n=4	1001001101111000	4	2	0	1	0.03 sec

Table 5.5: The r-th order non-linearity found for Boolean functions of n=4 variables of Construction Method-2 [21]

- **Higher Order Non-Linearity (N_{f_r}) of Method-3.** In method-3 [24], the author used the technique of concatenation, and use two initial functions (n=Odd) to construct a function (n=Even) with maximum algebraic immunity. The N_{f_r} of his constructed Boolean functions of n=4 variables, is shown in Table 5.6.

No of Var	Truth Table	N_{f_1}	N_{f_2}	N_{f_3}	N_{f_4}	Processing Time
n=4	1110100011011000	4	2	0	1	0.03 sec
n=4	1011100001111000	4	2	0	1	0.03 sec
n=4	1110010011010100	4	2	0	1	0.03 sec
n=4	1011010001110100	4	2	0	1	0.03 sec
n=4	1010110001101100	4	2	0	1	0.03 sec
n=4	1001110001011100	4	2	0	1	0.03 sec

Table 5.6: The r-th order non-linearity found for Boolean functions of n=4 variables of Construction Method-3 [24]

- **Higher Order Non-Linearity (N_{f_r}) of Method-4.** In method-4 [24], the author used primitive polynomials to construct Boolean functions of maximum AI and other good cryptographic properties. The N_{f_r} of his constructed Boolean functions up to n=5 variables, is shown in Table 5.7. In the last row of table below, only second order nonlinearity has been calculated, however processing time was not much reduced as large number of processing is consumed in computing algebraic degree.

No of Var	Truth Table	N_{f_1}	N_{f_2}	N_{f_3}	N_{f_4}	N_{f_5}	Processing Time
n=3	10010011	2	0	1	-	-	0.00 sec
n=3	10001110	2	0	1	-	-	0.00 sec
n=4	1000100110100111	4	2	0	1	-	0.03 sec
n=4	1000011110101100	4	2	0	1	-	0.03 sec
n=5	10000100101100011111000110111010	10	6	2	0	1	4171 sec
n=5	10000110010001111101110001010110	10	6	2	0	1	4171 sec
n=5	11111011100010101101000011001000	12	6	0	2	1	4171 sec
n=5	10000111001100111110100010010101	10	6	2	0	1	4171 sec
n=5	10000111001100111110100010010101	10	6	-	-	-	3660 sec

Table 5.7: The r-th order non-linearity found up to n=5 variables Boolean functions constructed using Method-4 [22]

5.4 Analysis of Boolean Functions Construction Methods

In light of above implementation results of existing construction methods [11, 21, 22, 24] with respective to their higher order non-linearities, it is concluded

that Method-4 [22] has much better 2nd and 3rd order non-linearity for n=4 and n=5 variables, as compared to Method-1 [11]. As far as, remaining two methods are concern (i.e. Method-2 and Method-3), they are iterative in nature and their cryptographic properties are based on their initial functions which are used for concatenation. We have also calculated r-th order non-linearity of their constructed Boolean functions for n=4 variables, which have optimal values at least for 2nd order non-linearity for both the methods [21, 24]. A comparison graph between rth-order nonlinearity of Method-1 [11], Method-4 [22] is shown in Figure 5.1 below, generated using MatLab R2009b.

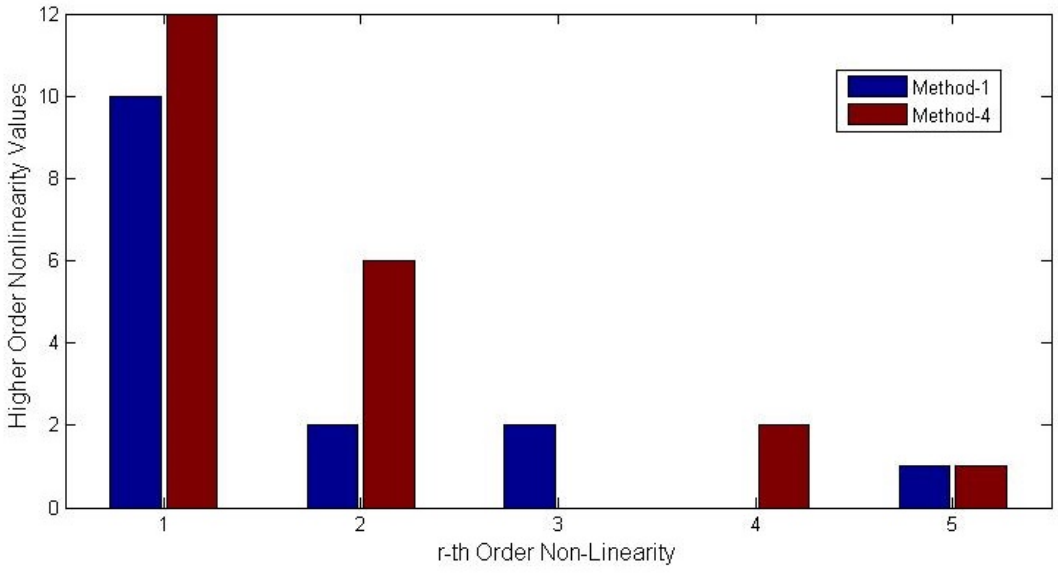

Figure 5.1: rth-order nonlinearity of Method-1 [11] and Method-4 [22] for n = 5 variables

Chapter 6

Extending Existing Construction Methods

6.1 Introduction

In this chapter, we will present a technique, which we have discovered through experimental results, and is novel till now up to our knowledge. By using this technique one can construct 2^n more number of Boolean functions, from existing single Boolean function with same cryptographic properties. Most importantly, the new constructed functions have same algebraic immunity as that of original one, i.e. maximal. We have successfully applied this technique in two existing construction methods, i.e. Method-1 [11] and Method-4 [22]. Our main focus was to achieve maximum algebraic immunity, although our technique does not disturb balanced-ness, non-linearity completely and algebraic degree up to some extent for Method-1. For construction method-4, we got 100% successful results for n=Even, and achieved all the cryptographic properties as that of existing single Boolean function. However, we did not get optimal AI always in case of n=Odd for construction method-4. The details are given in subsequent paragraphs.

6.2 Extending Method-1 (using Majority Functions)

During implementation of construction method proposed by D.K. Dalia [11], we identified that there is only one Boolean function that can be constructed using his proposed construction-3, for any value of n=ODD. This approach is not too much useful, as while implementing subsequent methods, which are mostly based on concatenation, utilize more number of existing Boolean functions with maximum AI. Therefore, its very difficult (almost impossible) to find a Boolean function with optimal cryptographic values (especially maximum algebraic immunity) when n is greater than 10, as space for number of Boolean functions becomes $2^{2^{10}}$.

Through experimental results, we have discovered a technique, which can be used to construct 2^n more number of Boolean functions using single existing Boolean function with maximum AI. The new constructed Boolean functions have same cryptographic properties (AI, balanced-ness and non-linearity) as that of existing one, except algebraic degree and correlation immunity, which may divert slightly for few Boolean functions. This technique is novel up to our best knowledge. We have tried/applied various different techniques, through experiments, to get more number of Boolean functions from existing functions, however only this technique showed successful results. In this technique, we apply cyclic shift on the truth table of constructed majority functions, i.e. the support set of the Boolean function is cyclic shifted by one to construct one new function. In this way by shifting total of 2^n times, we construct 2^n number of Boolean functions. We have suc-

cessfully applied this technique on both Odd and Even Majority functions, constructed using construction-3 of Dalai [11].

For n = Odd, we have implemented up to n = 11 variables, and found all constructed Boolean functions with same Algebraic Immunity, Non-linearity and Balancedness as that of original function, however algebraic degree and correlation immunity changes slightly, as shown in Table 6.1.

No of Variables	No of Constructed BFns	Bal	Deg	N_f	CI	AI
n = 3	$2^3 = 8$ BFns	1	2	2	0-2	2
n = 5	$2^5 = 32$ BFns	1	4	10	0-4	3
n = 7	$2^7 = 128$ BFns	1	4-6	44	0-6	4
n = 9	$2^9 = 512$ BFns	1	8	186	0-8	5
n = 11	$2^{11} = 2048$ BFns	1	8-10	772	0-10	6

Table 6.1: Number of Boolean functions constructed using our proposed extension in Method-1 [11] and their cryptographic properties for n = ODD

For n = Even, we have implemented up to n = 12 variables. Since using construction-3 of Dalai, the output functions are not balanced and the author has left it to us to make them balanced. We in our work have made them balanced, making first half of total (weight = n/2) outputs equal to ONE and rest half to ZERO, in this case we get all the cryptographic properties exactly the same as that of original boolean function, rather in some cases we achieve higher value of correlation immunity. The implementation results are mentioned in Table 6.2, for up to n=12 variables. Note balancedness, non-linearity and algebraic immunity is same for all the newly constructed Boolean functions.

No of Variables	No of Constructed BFns	Bal	Deg	N_f	CI	AI
n = 4	2^4 = 16 BFns	1	2	4	2-3	2
n = 6	2^6 = 64 BFns	1	4	20	0-5	3
n = 8	2^8 = 256 BFns	1	4-6	88	0-7	4
n = 10	2^{10} = 1024 BFns	1	8	372	0-9	5
n = 12	2^{12} = 4096 BFns	1	8-10	1544	0-11	6

Table 6.2: Number of Boolean functions constructed using our proposed extension in Method-1 [11] and their cryptographic properties for n = EVEN

6.3 Extending Method-4 (using Primitive Polynomials)

The construction method proposed by D.K.Dalai [11] is not useful for practical crypto systems, as suggested by the author; it can only be used for theoretical study. This is because of the fact that although the author get optimal value of algebraic immunity, however unable to get optimal values of other cryptographic properties, especially non-linearity.

The idea of constructing Boolean functions using primitive polynomials was presented in [22], which not only provide optimal values of algebraic immunity, algebraic degree and balancedness but also have very high value of non-linearity (very close to its optimal value). The problem with this construction [22] was also same, i.e. only one Boolean function can be constructed using one primitive polynomial. Therefore, in this research, we also applied our technique of cyclic shift to Method-4 [22], and construct successfully 2^n number of more Boolean functions from existing single Boolean function. We have applied our technique for both Even and Odd n, however got successful results for n=Enen only, as for n=Odd we did not get optimal value of AI always. The results up to n=12 are mentioned in Table 6.3.

No of Var	Primitive Polynomial	No of Constructed BFns	Bal	Deg	N_f	CI	AI
n=4	$1+x+x^4$	$2^4 = 16$	1	3	4	0	2
n=4	$1+x^3+x^4$	$2^4 = 16$	1	3	4	0	2
n=6	$1+x+x^6$	$2^6 = 64$	1	5	22-24	0	3
n=6	$1+x+x^4+x^5+x^6$	$2^6 = 64$	1	5	20-24	0	3
n=6	$1+x^5+x^6$	$2^6 = 64$	1	5	22-24	0	3
n=8	$1+x+x^3+x^5+x^8$	$2^8 = 256$	1	7	100-110	0	4
n=8	$1+x+x^2+x^3+$ $x^4+x^6+x^8$	$2^8 = 256$	1	7	102-110	0	4
n=8	$1+x^2+x^5+x^6+x^8$	$2^8 = 256$	1	7	100-110	0	4
n=10	$1+x^4+x^5+x^8+x^{10}$	$2^{10} = 1024$	1	9	436-472	0	5
n=10	$1+x^2+x^3+x^8+x^{10}$	$2^{10} = 1024$	1	9	448-468	0	5
n=10	$1+x+x^4+x^9+x^{10}$	$2^{10} = 1024$	1	9	440-468	0	5
n=12	$1+x+x^4+x^6+x^{12}$	$2^{12} = 4096$	1	11	1882-1952	0	6

Table 6.3: Number of Boolean functions constructed using our proposed extension in Method-4 [22] and their cryptographic properties up to n = 12 variables (EVEN only)

A comparison is carried out among number of Boolean functions constructed with each value of n (variable) for Method-1 [11], Method-4 [22] and our proposed extension in existing methods. Note for Method-1 [11], for each value of n, one can get only two Boolean functions, i.e. one majority function and other its complement, while for Method-4 [22], for each value of n, one can construct same number of Boolean functions as there are primitive polynomials exists for that particular value of n, while with our proposed extension, one can get at least 2^n number of Boolean functions with maximum algebraic immunity for each value of n, as shown in Figure 6.1 below, generated using MatLab R2009b.

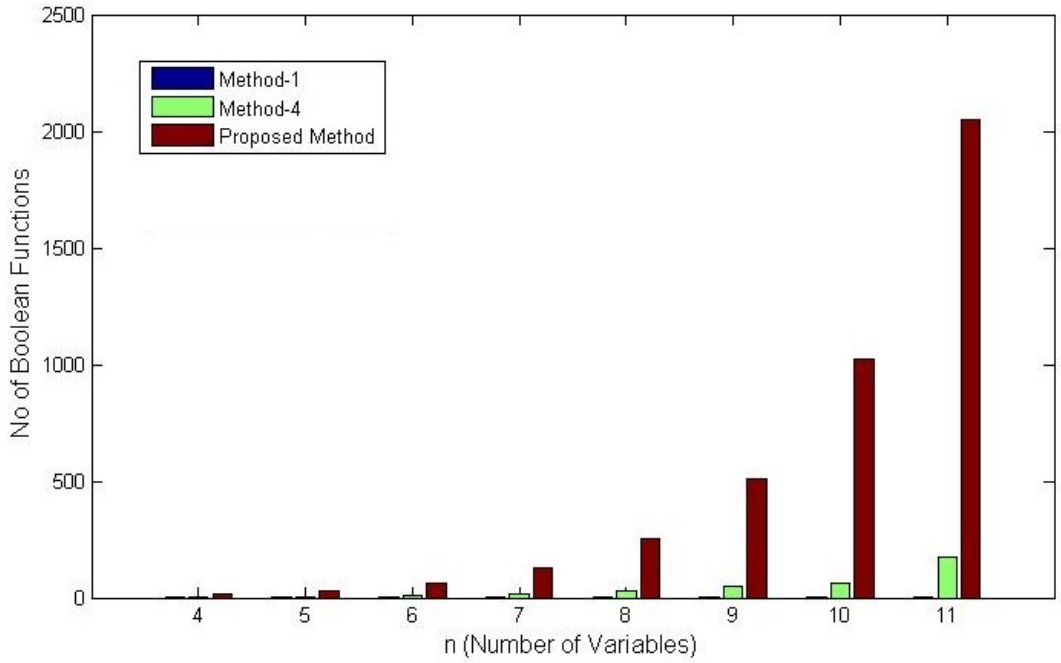

Figure 6.1: Comparison among number of Boolean functions constructed from Method-1 [11], Method-4 [22] and our Proposed Method

Chapter 7

Conclusion and Future Work

7.1 Conclusions

In this research, a comparative study of four existing methods has been carried out, in order to characterize these methods in terms of fulfilling basic cryptographic properties. Additionally, these methods are evaluated in terms of achieving optimal values of higher order non-linearity, for which there are no such study carried out in literature. Although every construction method achieved optimal value of algebraic immunity, which we verified practically, however, the real challenge was to achieve optimal values of other cryptographic criteria as well. In our detailed analysis, Method-4 is found most suitable as it provides balanced Boolean functions with optimal value of algebraic immunity, algebraic degree and a very high value of non-linearity. However, it does not provide high value of correlation immunity, which can be taken as a future work.

In the second part, an effective and efficient technique of constructing 2^n more number of Boolean functions from a single Boolean function has been developed. Each of the newly constructed function has been verified

by practically computing their cryptographic properties and all have optimal values of algebraic immunity, balanced-ness and algebraic degree. The value of non-linearity achieved is same as that of original Boolean function, which is close to optimal in case of Method-4.

7.2 Summary of Contributions

In this research, we have experimentally proved that one can construct 2^n more number of Boolean functions from single existing function. We applied our technique, in two existing methods, i.e. Method-1 [11] and Method-4 [22], and present our results for both even and odd cases.

The second contribution in this thesis is the comparative analysis of four existing methods of constructing Boolean functions with optimal algebraic immunity. These methods are not only evaluated on fulfilling basic cryptographic criteria but are also analyzed in terms of their higher order non-linearity, which has not been studied yet. The study concludes that out of these four methods, Method-4 is best which provides optimal value of algebraic degree, balancedness and a very high value of non-linearity other than achieving maximum algebraic immunity.

- **Future Directions:** The results we achieved for extending existing methods are experimentally verified and no theoretical proof has been given and is left as future work. Additionally, Boolean functions constructed using Method-4 could be utilized in designing a strong LFSR based stream cipher.

Bibliography

[1] M. Afzal and A. Masood, "Higher order non-linearity of cryptographically significant boolean functions and its importance."

[2] F. Armknecht, C. Carlet, P. Gaborit, S. Künzli, W. Meier, and O. Ruatta, "Efficient computation of algebraic immunity for algebraic and fast algebraic attacks," in *Advances in Cryptology-EUROCRYPT 2006*. Springer, 2006, pp. 147–164.

[3] C. Carlet, "On the higher order nonlinearities of algebraic immune functions," in *Advances in Cryptology-CRYPTO 2006*. Springer, 2006, pp. 584–601.

[4] C. Carlet, "On the higher order nonlinearities of boolean functions and s-boxes," in *ARES*, 2009, p. 768.

[5] C. Carlet, D. K. Dalai, K. C. Gupta, and S. Maitra, "Algebraic immunity for cryptographically significant boolean functions: analysis and construction," *Information Theory, IEEE Transactions on*, vol. 52, no. 7, pp. 3105–3121, 2006.

[6] C. Carlet and K. Feng, "An infinite class of balanced functions with optimal algebraic immunity, good immunity to fast algebraic attacks

and good nonlinearity," in *Advances in Cryptology - ASIACRYPT 2008*. Springer Berlin Heidelberg, 2008, vol. 5350, pp. 425–440.

[7] Y. Chen and P. Lu, "Two classes of symmetric boolean functions with optimum algebraic immunity: Construction and analysis," *Information Theory, IEEE Transactions on*, vol. 57, no. 4, pp. 2522–2538, 2011.

[8] N. Courtois and W. Meier, "Algebraic attacks on stream ciphers with linear feedback," *Advances in Cryptology - EUROCRYPT 2003*, pp. 644–644, 2003.

[9] N. T. Courtois, "Fast algebraic attacks on stream ciphers with linear feedback," in *Advances in Cryptology-CRYPTO 2003*. Springer, 2003, pp. 176–194.

[10] D. K. Dalai, K. C. Gupta, and S. Maitra, "Results on algebraic immunity for cryptographically significant boolean functions," in *Progress in Cryptology-Indocrypt 2004*. Springer, 2005, pp. 92–106.

[11] D. K. Dalai, S. Maitra, and S. Sarkar, "Basic theory in construction of boolean functions with maximum possible annihilator immunity," *Designs, Codes and Cryptography*, vol. 40, no. 1, pp. 41–58, 2006.

[12] E. Dawson and C.-K. Wu, "Construction of correlation immune boolean functions," *Information and Communications Security*, pp. 170–180, 1997.

[13] F. Didier, "Using wiedemanns algorithm to compute the immunity against algebraic and fast algebraic attacks," in *Progress in cryptology-INDOCRYPT 2006*. Springer, 2006, pp. 236–250.

[14] F. Didier and J.-P. Tillich, "Computing the algebraic immunity efficiently," in *Fast Software Encryption*. Springer, 2006, pp. 359–374.

[15] C. Ding, G. Xiao, and W. Shan, *The stability theory of stream ciphers*. Springer, 1991, vol. 561.

[16] Y. Du and D. Pei, "Construction of boolean functions with maximum algebraic immunity and count of their annihilators at lowest degree," *Science China Information Sciences*, vol. 53, no. 4, pp. 780–787, 2010.

[17] K. Limniotis, N. Kolokotronis, and N. Kalouptsidis, "Modifying boolean functions to ensure maximum algebraic immunity." *IACR Cryptology ePrint Archive*, vol. 2012, p. 46, 2012.

[18] F. Liu and K. Feng, "Efficient computation of algebraic immunity of symmetric boolean functions," in *Theory and Applications of Models of Computation*. Springer, 2007, pp. 318–329.

[19] W. Meier, E. Pasalic, and C. Carlet, "Algebraic attacks and decomposition of boolean functions," in *Advances in Cryptology-EUROCRYPT 2004*. Springer, 2004, pp. 474–491.

[20] S.-S. Pan, X.-T. Fu, and W.-G. Zhang, "Construction of 1-resilient boolean functions with optimal algebraic immunity and good nonlin-

earity," *Journal of Computer Science and Technology*, vol. 26, no. 2, pp. 269–275, 2011.

[21] E. Pasalic, "A design of boolean functions resistant to (fast) algebraic cryptanalysis with efficient implementation," *Cryptography and Communications*, vol. 4, no. 1, pp. 25–45, 2012.

[22] H. K. Qichun Wang, Jie Peng and X. Xue, "Constructions fo cryptographically significant boolean functions using primitive polynomials." *IEEE TRANSACTIONS ON INFORMATION THEORY*, vol. 56, pp. 3048–3053, 2010.

[23] L. Qu, C. Li, and K. Feng, "A note on symmetric boolean functions with maximum algebraic immunity in odd number of variables," *Information Theory, IEEE Transactions on*, vol. 53, no. 8, pp. 2908–2910, 2007.

[24] S. Song, J. Zhang, J. Du, and Q. Wen, "On the construction of boolean functions with optimal algebraic immunity and good other properties by concatenation," in *Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on*, vol. 1. IEEE, 2010, pp. 417–422.

[25] S. Su and X. Tang, "Construction of rotation symmetric boolean functions with optimal algebraic immunity and high nonlinearity," *Designs, Codes and Cryptography*, vol. 71, no. 2, pp. 183–199, 2014.

[26] X. Zeng, C. Carlet, J. Shan, and L. Hu, "More balanced boolean functions with optimal algebraic immunity and good nonlinearity and resistance to fast algebraic attacks," *Information Theory, IEEE Transactions on*, vol. 57, no. 9, pp. 6310–6320, 2011.

Appendix A

Source Codes (C-Language)

A.1 Computing Non-Linearity and Correlation Immunity

```
//COMPUTING NON-LINEARITY FROM WALSH TRANSFORM
void Walsh_Transform(unsigned char *truth_table)
{
  unsigned char i=0,i0=0,i1=0,step=0,j=0;
  unsigned char temp=0,max_walsh_value=0;
  int sum=0,diff=0,ci_temp=0;
  char sequence_BFn[256]={0};    // Max256 for n=8 variables
  char correlationimmunity[256]={0};

  //Converting TruthTable of f into Sequence of f;
  //a sequence of f is same as its TruthTable except
  //all One's are replaced by -1 and all Zero's by +1
  for(i=0;i<size_TT;i++)
    if(truth_table[i])  sequence_BFn[i] = -1;
    else       sequence_BFn[i] = 1;
  for (step=1;step<size_TT;step<<=1)
  {
    for (i1=0;i1<size_TT;i1+=2*step)
    {
      for (i0=0;i0<step;i0++)
      {
        i=i1+i0;
        sum=sequence_BFn[i]+sequence_BFn[i+step];
        diff=sequence_BFn[i]-sequence_BFn[i+step];
        sequence_BFn[i]=sum;
        sequence_BFn[i+step]=diff;
      }
    }
  }
```

```
  for(i=0;i<size_TT;i++)
  {
    temp = abs(sequence_BFn[i]);
    if(temp>max_walsh_value)  max_walsh_value = temp;
    //COMPUTING CORRELATION IMMUNITY
    ci_temp = 0;
    for(j=0;j<size_n;j++) ci_temp =  ((i>>j)&1) + ci_temp;
    correlationimmunity[ci_temp] = temp + correlationimmunity[
   ci_temp];
  }
  non_linearity = (size_TT/2) - (max_walsh_value/2);
  for(i=0;i<size_n;i++)
        if(correlationimmunity[i] == 0) CI_order = i;
}
```

A.2 Computing Algebraic Immunity

```
unsigned char Algebraic_Immunity(unsigned char *truth_table)
{
  unsigned char deg_ann_max=0;
  unsigned char i=0,j=0,k=0,deg=0,temp=0,sum=0;
  unsigned char supp[BF_limit]={0},s_count=0;
  unsigned int monomials[BF_limit]={0},m_count=0;
  unsigned int A[BF_limit][100]={0};
  unsigned int row=0,col=0,rank_matrix=0;
  unsigned char deg_matrix_f=0,count_f=0;
//Calculating max degree of Annihilators upto which this program
//will run; which is one less than optimal AI, i.e. n/2 - 1.

  if(size_n%2) deg_ann_max = size_n/2;  // if n is ODD
  else deg_ann_max = (size_n/2) - 1;    // if n is EVEN
 for(count_f=0;count_f<2;count_f++)
 {
  if(count_f == 1)
    for(i=0;i<size_TT;i++)
       truth_table[i] = (~truth_table[i]) & 0x01;

//Calculating Supp{f} matrix, contains values in which f(x)=1
 for(i=0;i<size_TT;i++) if(truth_table[i])  supp[s_count++] = i;

//Calculating AI, starting from deg=1 of possible annihilator
 for(deg=1;deg<=deg_ann_max;deg++)
 {        //Calculate Monomials of Degree = deg
  m_count=0; // Store all possible monomials of weight = deg

```

```
for(i=0;i<size_TT;i++)
  {
  temp = i;
  while(temp)
          { sum = sum + (temp&1);     temp = temp>>1;  }
  if(sum <= deg) monomials[m_count++] = i;
  sum=0;
  }
//Calculate A[][], contains monomials at vectors of Supp(f)
 for(i=0;i<s_count;i++)
  {
   for(j=0;j<m_count;j++)
    { // This is the main part that checks all variables of
    monomials that are equal to One in Supp(f) then make them
    available in matrix A[][]
   if((monomials[j] & supp[i]) == monomials[j]) A[i][j] = 1;
   else    A[i][j] = 0;
     }
   }
 for(col=0;col<m_count;col++)
 {
  for(row=col;row<s_count;row++)
   {
    if(A[row][col] == 1) // finding a pivot
      {
       if(row == col)//checking pivot position
         {
              for(i=row+1;i<s_count;i++)
                 {
                   if(A[i][col])
                   for(j=col;j<m_count;j++)
                    A[i][j] = A[i][j] ^ A[row][j];
                  }
         }
         else //replacing the row value 1, with zero
            {
            for(i=0;i<m_count;i++)
           {
            temp         = A[row][i];
             A[row][i] = A[col][i];
            A[col][i]     = temp;
        }
  col--;     // so that next time col come again
  row = col;  // row will automatically get incremented in loop
     }
  col++;       // column will incremented here manually
  }//end if pivot found
 } //end for row
```

```
76 k++; // if No value ONE found at pivot position and all values
       // under it then increment k, so that Rank = monomials-k
78 }//end for col
   k--;
80 rank_matrix = m_count - k;
   if(count_f == 1)
82  {
     if(rank_matrix < m_count)
84    {
       if(deg_matrix_f < deg)
86      {
           for(i=0;i<size_TT;i++)
88         truth_table[i] = (~truth_table[i]) & 0x01;
            return(deg_matrix_f);
90            }
        else
92        {
               for(i=0;i<size_TT;i++)
94             truth_table[i] = (~truth_table[i]) & 0x01;
               return(deg);
96          }
        }
98   }
     if(rank_matrix < m_count)
100   {
        deg_matrix_f = deg;
102     deg = deg_ann_max;
      }
104  }// end for deg
   }// end for count_f
106 for(i=0;i<size_TT;i++)
       truth_table[i] = (~truth_table[i]) & 0x01;
108 return(deg_ann_max+1); // means Optimal AI
   }//End AI Function
```

A.3 Computing Higher Order Non-Linearity

```
1  // This Program will Calculate Higher Order Non-Linearity upto n
       =5 variables
   #include<stdio.h>
3  #include<stdlib.h>
   #include<conio.h>
5  #include<math.h>
   #include<time.h>
7
```

```
//For any n - value just change here:
  #define BF_limit 4294967295         //65536 for n=4
  #define n_limit 5

int main (void)
{
  //*****  VARIABLES
  FILE *fwrite;
  char file_name[25];
  unsigned int count_BF=0;
  unsigned int truthtable = 0x01;
  unsigned int d=0,i=0,j=0;
  unsigned int HD_1 = 0xFF,HD_2 = 0xFF,HD_3 = 0xFF,HD_4 = 0xFF,
   HD_5 = 0xFF,Hamming_Weight=0;
  unsigned int degree=0;
  unsigned int temp_tt=0,xor=0;
  unsigned int size_n=0;       //No of variables
  unsigned int size_TT=0;      //No of elements of Truth Table
  unsigned int size_BF=0;      //No of BFns
  unsigned int i0=0,i1=0,step=0;
  unsigned int x=0,y=0;
  clock_t begin, end;
  double time_spent;

  begin = clock();  // To computer Processing Time

  printf("\nThis Program will Calculate Higher Order Nf of a BFn
    of n=5 ");
  printf("\n\nEnter Boolean Function in Decimal Form (e.g.
   00100000 = 32) ==> ");
  scanf("%d",&truthtable);
  printf("\n\n\tComputing Please Wait . . . ");
  size_n = n_limit;

  if(size_n > 5)
  { printf("\n\n\n\tSorry! The Boolean Functions set will be too
    long, { 2<=n<=5 } ");
    printf("\n\n\n\n\t\t THE END \n\npress any key to continue
   ...");
    _getch(); exit(0);  }

  size_TT = pow(2,size_n);  //This will calculate no of values
   of TruthTable of a n-variable function f, i.e. 2^size_n
  size_BF = pow(2,size_TT); //This will calculate Total Number
   of BF of n variables i.e. 2^(2^size_n) for n=3, it is 2^2^3 =
    2^8 = 256

  sprintf(file_name,"Nf_n=%d.xls",size_n);
```

```
fwrite = fopen(file_name,"w"); //Writing results in an Excel File
fprintf(fwrite,"BFns-TT\t1st-Order\t2nd-Order\t3rd-Order\t4th-Order\t5th-Order\n");

//***** MAIN BODY (PROCESSING)
for(count_BF=1;count_BF<size_BF-1;count_BF++)
  {
    temp_tt = count_BF;
    xor = temp_tt ^ truthtable;
    Hamming_Weight = 0;
    // Counting No of 1's after XOR i.e. Hamming Distance
    for(i=0;i<size_TT;i++)  Hamming_Weight = Hamming_Weight + ((xor>>i)&1);

    //Moebius_Transform to compute Degree of each BFn
    for (step=1;step<size_TT;step<<=1)
    {
      for (i1=0;i1<size_TT;i1+=2*step)
      {
        for (i0=0;i0<step;i0++)
        {
          i=i1+i0;
          x = ((temp_tt>>(i+step))&1) ^ ((temp_tt>>i)&1);
          y = (temp_tt>>(i+step)) & 0xFFFFFFFE;
          y = y | x;
          y = y<<(i+step);
          temp_tt = temp_tt<<(32-i-step);
          temp_tt = temp_tt>>(32-i-step);
          temp_tt = temp_tt | y;
        }
      }
    }
    degree=0;
    for(i=0;i<size_TT;i++)
    {
      if((temp_tt>>i)&1)
      {
        for(j=0;j<size_n;j++)
        {
          if(i>>j & 0x01) d++;
        }
        if(d>degree) degree = d;
        d=0;
      }
    }

    if(degree == 1)
    {
```

```
            if(Hamming_Weight<HD_1) HD_1 = Hamming_Weight;
        }
        else if(degree == 2)
        {
            if(Hamming_Weight<HD_2) HD_2 = Hamming_Weight;
        }
        else if(degree == 3)
        {
            if(Hamming_Weight<HD_3) HD_3 = Hamming_Weight;
        }
        else if(degree == 4)
        {
            if(Hamming_Weight<HD_4) HD_4 = Hamming_Weight;
        }
        else if(degree == 5)
        {
            if(Hamming_Weight<HD_5) HD_5 = Hamming_Weight;
        }
      }// end count_BF for loop
    end = clock();
    time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
    printf("\n\n\n\tTOTAL TIME TAKE : %f seconds",time_spent);
    fprintf(fwrite,"%d\t%d\t%d\t%d\t%d\t%d\n",truthtable,HD_1,HD_2
      ,HD_3,HD_4,HD_5);
    printf("\n\n The values of Higher Order Nf of all BFns of n=%d
       are stored in <%s>",size_n,file_name);
    printf("\n\n\n\n\t\t THE END \n\npress any key to continue ...
      ");
    getch();
}
//******** The End Main ***************//
```