ON APPLICATIONS OF SIMULATED ANNEALING TO CRYPTOLOGY

WEN MING LIU

A THESIS

IN

THE CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE IN INFORMATION SYSTEMS

SECURITY

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

DECEMBER 2008

Your file Votre référence
ISBN: 978-0-494-63241-3
Our file Notre référence
ISBN: 978-0-494-63241-3

ABSTRACT

On Applications of Simulated Annealing to Cryptology

Wen Ming Liu

Boolean functions are critical building blocks of symmetric-key ciphers. In most cases, the security of a cipher against a particular kind of attacks can be explained by the existence of certain properties of its underpinning Boolean functions. Therefore, the design of appropriate functions has received significant attention from researchers for several decades. Heuristic methods have become very powerful tools for designing such functions.

In this thesis, we apply simulated annealing methods to construct Boolean functions with particular properties. Our results meet or exceed the best results of available theoretical constructions and/or heuristic searches in the literature, including a 10-variable balanced Boolean function with resiliency degree 2, algebraic degree 7, and nonlinearity 488 for the first time. This construction affirmatively answers the open problem about the existence of such functions.

This thesis also includes results of cryptanalysis for symmetric ciphers, such as Geffe cipher and TREYFER cipher.

Acknowledgments

I would like to appreciate lots of people who provide help for this thesis.

First of all, I would like to express my deepest gratitude to my supervisors, Dr. Amr Youssef and Dr. Lingyu Wang, for their constant support, heartily guidance and enduring patience during my graduate study. This thesis would not have been possible without their help. Their attitude and enthusiasm for scientific and academic research will always be my role model.

I also wish to express my appreciation to all the faculty and people at Concordia Institute for Information Systems Engineering for having such a warm and cosy working environment. To each of my professors, I owe a great debt of gratitude for their wonderful teaching, which has helped me in reaching this stage.

I thank my late parents for teaching me valuable lessons of life. I also thank my wife who always keeps me away from family burdens and allows me to concentrate on my study. Thanks (or rather apologies) also go to my little daughter who tolerated my frequent refusals to take her on outings.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Cryptology plays a central role in today's security mechanisms. The field of cryptology can be largely divided into cryptography and cryptanalysis [6]. Cryptography is the art of building cryptosystems and creating secret codes, while cryptanalysis is the study of finding weaknesses in the cryptosystems and breaking secret codes. The goals of cryptography include confidentiality, data integrity, authentication and nonrepudiation. Mainly three types of cryptographic primitives are used to achieve these goals: symmetric ciphers, public ciphers, and hash functions [14].

Boolean functions ($f : Z_2^n \to Z_2$) and s-boxes ($f : Z_2^n \to Z_2^m$) are the main building blocks for stream ciphers and block ciphers. The success of cryptanalytic attacks on these symmetric primitives and the cryptographic properties of Boolean functions are strongly connected. In fact, the security of a symmetric cipher against a particular category of attacks can be explained by the existence of certain properties of its building blocks. For example, the correlation attacks [14] are related to the properties of correlation immunity [46] [53] and resiliency [27]; the algebraic attacks [15] are related to the property of algebraic immunity [15] [14]; the linear cryptanalysis [33] is related to the property of nonlinearity [44] [35]. Therefore, the strength of its building blocks is a basic requirement of any secure cryptosystems. Note that, different cryptosystems for different purposes may have different requirements on their building blocks.

Due to their importance, Boolean functions have been studied for a long time. While many problems in this area have been solved over time, novel problems constantly arise due to ongoing developments in cryptanalysis.

To construct certain Boolean functions, we have to determine the value of each element, either 1 or 0, so that the function will satisfy required criteria. Clearly, this is a combinatorial optimization problem. Most existing methods for constructing Boolean functions that can satisfy multiple criteria are related to search techniques for solving problems of combinatorial optimization. Those methods can be categorized in three classes: exhaustive search, algebraic constructions, and heuristic techniques.

Theoretically, exhaustive search can always find the optimal functions. However, since the complexity of constructing Boolean functions is doubly exponential in variable n, when n is larger than 5, such method becomes computationally infeasible. For example, if we assume that a typical computer can search 2^{32} functions per second, then for $n = 10$ exhaustive search would require about 1.3×10^{291} years.

Algebraic constructions can achieve specific combinatorial properties to some extent. However, in most cases, they tend to lead to sub-optimal results, especially for the properties that have not been considered in devising the construction. Furthermore, even when algebraic constructions can achieve the optima, the inherent algebraic structure in the constructed Boolean function may make it comparatively vulnerable to algebraic attack.

Heuristic methods are based on enumerative methods but use extra knowledge to guide the search. The knowledge is usually derived from the simulation of natural processes and the understanding of the problem under consideration. Heuristics is commonly known as

Rules of Thumb, educated guesses, intuitive judgments, or just common sense. It is a suitable method when the problem is fuzzy, complex, or large. When a problem presents these characteristics, it is possible to just rely on suitable Rules of Thumb for a solution. The purpose of heuristic methods is to identify problem solutions where time is more important than solution quality or the knowledge of quality. They can produce good results in reasonable short runs for such problems.

In our cases, firstly, the search space is huge (2^{2^n}). This is far away from the ability of brute force. Secondly, the problem itself is uncertain. For example, researchers are trying to break the famous conjecture given by Dobbertin [16], but the conjecture cannot be proved or disproved so far. It is uncertain whether the examples for breaking the conjecture exist or not. In the literature of cryptography, many interesting results have already been obtained using heuristic methods.

In this thesis, we also address applications of heuristic methods to cryptanalysis of ciphers. Although the approaches used to carry out attacks against ciphers can vary considerably in different work, they are often making use of some properties of the cipher's internal components. It would be an attractive and revolutionary finding, if there exists a way for attackers to execute attacks while the corresponding ciphers are treated as black-boxes. Due to the nature of heuristic methods, it is possible to implement attacks based on such methods without analyzing, or having minimal analysis on, the internal components of the ciphers. If such attacks may succeed, it would be unnecessary for attackers to understand the internal details of ciphers before they can implement attacks on such ciphers.

1.2 Our Contributions

Through our research, we focus on the study of constructing special Boolean functions through heuristic methods. We also apply heuristic methods to break certain ciphers. More specifically,

- Based on the understanding of Boolean function and heuristic method, we study different approaches to the construction of Boolean functions. We conduct experiments with different cost functions, different search domains, and different neighbor policies to construct examples of Boolean functions with different cryptographic properties, such as $(10, 2, 7, 488)$, $(8, 116)$, $(10, 492)$, and $(12, 2010)$ functions [1].

- We attempt to apply simulated annealing methods and guided search techniques to break certain ciphers. Our objective is to find methods with which attacker can execute attacks without analyzing the internal details of ciphers being attacked.

1.3 Thesis Organization

The rest of this thesis is organized as follows.

- In Chapter 2, we introduce mathematic background and necessary definitions of heuristic methods and Boolean functions.

- In Chapter 3, we apply simulated annealing methods to construct several Boolean functions with different cryptographic properties (resilient, nonlinearity, balance).

- In Chapter 4, we apply simulated annealing methods and guided search techniques to attack some symmetric ciphers.

- Finally, in Chapter 5, we conclude the research and give future work.

[1]The notations for (n, m, d, NL), (n, NL) Boolean functions are defined in Section 3.1.

Chapter 2

Preliminaries

In this chapter, we first review heuristic methods and the simulated annealing method in Section 2.1. We then review the cryptology problems on which we shall apply the simulated annealing method in Section 2.2.

2.1 Heuristic Methods

Exhaustive search techniques can be used to solve search problems by trying all possible solutions and verifying the best solutions satisfying the search requirement. However, the time and space complexity of exhaustive searches are usually prohibitive. Therefore, such technique has limitations on even medium-sized problems.

In order to solve large-sized combinatorial search problems in reasonable time, there exist heuristic methods such as simulated annealing, genetic algorithm, tabu search, and so on. Such methods provide general ways to search for good, but not always optimal, solutions.

To use heuristic methods, the following four factors must be determined.

1. Formulate the problem as a guided search problem

   The problem must be tranformed into a representation of the solution space and corresponding cost function in order to measure how good a given solution is, in an appropriate and easily computable way.

2. Determine search space

   Search space refers to all the possible inputs. In some cases, in order to reduce the search space, a subset of all possible inputs may replace the whole set as the search space. For example, when we construct $(10, 2, 7, 488)$ Boolean functions, we use rotation symmetric functions (RSBF) [43] instead of all the Boolean functions as our search space.

3. Construct cost function

   Each candidate input has a cost value calculated by cost function. The cost value can be used to evaluate how well a candidate input matches the desired solution. The effective cost function must maximize or else minimize the cost value of the desired solution.

4. Define search strategy

   A simple transition mechanism should be defined to move from one candidate solution to the other by slightly modifying the current solution. Typical transition mechanisms for constructing Boolean functions include flipping the output value of one position, swapping the values of a pair of positions either in the truth table representation or the Walsh transform representation of the function.

The cooperative association of the above four factors offer an excellent ability to escape from local optima and finally reach the best solution.

Each of such techniques depends on a simple model of a real-world physical process. In the remainder of this section, we discuss the simulated annealing method in details and briefly introduce genetic algorithm, tabu search, and ant colony.

2.1.1 Simulated Annealing

Simulated annealing is inspired by the physical process of cooling molten materials down to the solid state. In this process, solid will be fully heated to high temperature and then slowly cooled down. During heating, the internal particles of solid are changed into states of disorder and its energy increases; during slowly cooling, particles gradually become orderly, and in each temperature achieves a balanced state, and finally, brings the material to a low-energy, optimal state. According to Metropolis criteria [36], for particles in temperature t, the probability of reaching a balanced state is $e^{-\Delta E/(kt)}$, wherein E is the energy under certain temperature t, ΔE is the energy difference between two temperatures, and k is a constant.

Through guided transitions generated based on the above probability distribution, the physics can be simulated to solve combinatorial optimization problems. By simulating Energy E to be the objective function value $cost()$, and temperature t to be control parameters T, the simulated annealing algorithm can be derived: Starting from initial possible state S and initial control parameter T, iterate on current state by the process of "generating new state $\rightarrow$ calculating the difference of objective function $\rightarrow$ accepting or abandoning this new state", and gradually decay T value. This is repeated until the system freezes into a steady state. At this point, the current state is the approximate optimal solution.

To use simulated annealing, the above four factors must first be determined. Furthermore, similar with physical annealing, simulated annealing process and its quality are controlled by the cooling schedule, which can be regulated by several parameters. These parameters are problem-sensitive and govern how likely a bad transition is accepted as a function of time [52]:

1. Initial value of control parameter T_0

   This value starts high enough and is then gradually lowered. Its selection is a key factor of the method. If it does not start high enough, the ending state will be very close to the starting state. However, if it starts too high, the search may be transformed into a random search. To compare with physical annealing, in the remainder of this section, this parameter is regarded as the temperature.

2. The number of iterations L at each temperature T

   At each temperature, a certain number of iterations L are attempted before lowering the temperature. One approach is to fix a constant number, and the other way is to dynamically change the number of iterations at runtime. At lower temperatures, a larger number of iterations must be done to completely explore the local optima, while at higher temperatures, the number of iterations can be less.

3. Acceptance criteria

   This is used to determine whether a transition from S_{cur} to S_{next} is accepted. If the S_{next} state has better cost value than S_{cur} has, then a move to that state S_{next} is taken; if not, then it is accepted with some probability. Similar with physical annealing, the worse a move is, the less likely it is to be accepted; the lower the temperature T, the less likely is a worsening move to be accepted. Initially, the temperature is high and almost any move is accepted. As the temperature is decreased, it becomes more difficult to allow worsening moves. Finally, only improving moves are accepted.

4. temperature decrement factor ΔT

   At the end of each inner loop, the temperature is lowered, the typical way of lowering the temperature is to multiply by a decrement factor ΔT in the range of $(0..1)$. i.e. $T_{next} = \Delta T \times T_{cur}$, wherein $0.0 < \Delta T < 1.0$. It is obvious that the temperature is an exponential decay instead of a linear decay.

Thermodynamic Simulation	**Combinatorial Optimization**
System States	Feasible Solutions
Energy	Cost
Change of State	Neighboring Solutions
Temperature	Control Parameter
Frozen State	Heuristic Solution

Table 1: Relationship between Physical Annealing and Simulated Annealing

5. Stopping criteria

   The algorithm terminates when the stopping criteria are met. There are many possible stopping criteria: A fixed number of inner loops have been executed; consecutive inner loops are executed without a single move being accepted; the cost function value satisfies certain requirements; the temperature is lowered enough to certain value; a combination of above conditions.

Table 1 shows how physical annealing can be mapped to simulated annealing.

The basic simulated annealing algorithm is shown in the following pseudo code [20]. The search starts at certain initial state S_0. At each temperature T_{cur}, an iteration number L of transition moves are tried. A candidate state S_{next} is randomly selected from the neighborhood $N(S)$ of current state S_{cur} which is formed based on the search strategy. The difference of objective function δ is calculated. Acceptance criterion is then used to determine whether to accept this transition or not. At last, the algorithm terminates when the stopping criteria are met.

Procedure: Pseudo Code for Simulated Annealing Algorithm (Minimization Cases):

```
1. $S_{cur} = S_0$;
2. $T_{cur} = T_0$;
3. calculate cost($S_{cur}$);
4. do
5. {
```

```
6.   for ($i$=0; $i < L$; $i++$)
7.   {
8.       select a random transition from $S_{cur}$ to $S_{next}$, where $S_{next} \in N(S_{cur})$;
9.       $\delta = cost(S_{next}) - cost(S_{cur})$;
10.      if ($\delta < 0$)
11.          $S_{cur} = S_{next}$);
12.      else if ($e^{(-\delta/T_{cur})} > random(0,1)$)
13.          $S_{cur} = S_{next}$;
14.  }
15.  $T_{cur} = T_{cur} \times \Delta T$;
16. }
17. while (stopping criterion is not met)
18. return ($S_{cur}$);
```

Simulated annealing has been used for real combinatorial search problems, such as traveling salesman problem, maximum cut problem, circuit board placement problem, scheduling problem. It has also been successfully applied to cryptological problems, such as component designing Boolean functions with desirable properties [20]. There exist modified versions of simulated annealing. For example, the best state (solution) so far can be recorded during the search process, so in the end the best state can be chosen from the recorded states as the output. To make good use of simulated annealing method, the construction of cost functions and the choice of cooling schedule are the key factors.

2.1.2 Other Methods

1. **Genetic Algorithms (GA) - Evolutionary**

   This technique has been initially developed by Holland (1975). The inspiration of genetic algorithms comes from evolution and natural selection. Genetic algorithms imitate the evolutionary process of species that sexually reproduce. Thus, genetic algorithms might be considered as the prototype of a population-based method. New candidates are generated with a mechanism, namely, $crossover(recombination)$. The newly created individual, called *child*, can then apply a random $mutation$, which means the elements are somewhat changed. If the new individual inherits good characteristics from his parents evaluated by the cost function, it will have a higher probability to survive [19]. The following pseudo code shows its procedure [18].

Procedure: Pseudo Code for Genetic Algorithm:

```
1.  generate initial population $P$ of solutions;
2.  while (stopping criterion is not met) do
3.  {
4.     select $P' \subset P$ (mating pool), initialize $P'' = \phi$ (set of children);
5.     for ($i$=1; $i < n$; $i++$)
6.     {
7.        select individuals $x_a$ and $x_b$ at random from $P'$;
8.        apply crossover to $x_a$ and $x_b$ to produce $x_{child}$;
9.        randomly mutate produced child $x_{child}$;
10.       $P'' = P'' \cup x_{child}$;
11.    }
12.    $P$=survive($P'$, $P''$);
13. }
```

Genetic algorithm is applied to construct Boolean functions for a long time. Recently, in [5], genetic algorithm combined with hill climbing are used to construct $(8, 114)$, $(10, 480)$ and $(12, 1970)$ Boolean functions.

2. **Tabu Search (TS) - Guided Local Search**

It is inspired by search principles from artificial intelligence or "human" behavior. This method implements the selection of the neighborhood solution in a way to avoid cycling, i.e., visiting the same solution more than once. This is achieved by employing a short term memory, known as the tabu list which contains the solutions that are most recently visited. The following pseudo code shows its procedure [18].

Procedure: Pseudo Code for Tabu Search:

```
1. generate initial solution: $S_{cur} = S_0$;
2. initialize tabu list: $T = \phi$;
3. while (stopping criterion is not met) do
4. {
5.     Compute: $V = \{S_{next} | S_{next} \in N(S_{cur})\} \backslash T$;
6.     select: $S_{next} = min(V)$;
7.     $S_{cur} = S_{next}$ and $T = T \cup S_{cur}$;
8.     update memory;
9. }
```

TS generates a random initial solution as current candidate and mantains a TABU list. Based on each current solution, TS generates its ordering set of neighbours, the best of which is chosen as the next candidate provided that it is not already on the TABU list(T), whilst the current candidate is appended to the TABU list. If the best neighbours of current candidate are already present on TABU list T, then the second best is chosen as the next candidate, and so on. A simple way to update memory is to use a queue, which always removes the eldest entry in the list when recording a new entry. The stopping criterion can be defined as a given number of total iterations or a given number of consecutive iterations without improvement for the current solution.

Tabu Search is usually combined with other heuristic methods to solve combinatorial problems due to its nature of local search [23].

3. **Ant Colony**

   This technique is first introduced by Colorni *et al.* in 1992. The Ant Colony optimization algorithm is a cooperative heuristic searching algorithm inspired by the ethological study on the behavior of ants. It imitates the way that ants search for food and find their optimal path between their colony and the food source. This is done by an indirect communication known as stigmergy via the chemical substance, or pheromone, left by the ants on the paths. The intensity of the pheromone traces depends on the quantity and quality of the food available at the source as well as from the distance between source and colony. As an ant traverses a path, it reinforces that path with its own pheromone. A collective autocatalytic behavior emerges as more ants will choose the shortest trails, which in turn creates an even larger amount of pheromone on those short trails, which makes those short trails more likely to be chosen by future ants. Pheromone trails evaporate and once a source of food is exhausted the trails will disappear and the ants will start to search for other sources. The following pseudo code shows its procedure [18].

   Procedure: Pseudo Code for Ant Colony:

```
1.  initialize pheromone trail;
2.  while (stopping criterion is not met) do
3.  {
4.    for (all ants)
5.    {
6.      while (solution incomplete)
7.        select next elment in solution randomly according to pheromone trail;
8.      evaluate objective function and update best solution;
9.    }
10.   for (all ants)
11.       update pheromone trail (more for better solutions)
12. }
```

Ant Colony has been applied to solve combinatorial problems for a long time. In [51], it is used to attack some simple substitution ciphers.

2.2 Boolean Functions

2.2.1 Introduction

Boolean functions form important components in various practical cryptographic applications. A proper choice of a Boolean function may significantly increase the resistance to different kind of attacks [34]. In the following subsections, we first describe the representation of Boolean functions, then introduce its fundamental definitions and its cryptographic properties, finally end with an overview of its two cryptographic rich sub classes.

2.2.2 Representation of Boolean Functions

A Boolean function of n variables f is a mapping $\{0,1\}^n \rightarrow \{0,1\}$. There are many means to represent a Boolean function. Here we introduce five methods related to our research.

1. **Binary Truth Table (TT):**

   The binary truth table of Boolean function $f(x)$, where $x = (x_1, x_2, \ldots, x_n)$, $x_i \in \{0,1\}$, $i = 1, \ldots, n$, is a 2^n-element binary sequence, $f(x) = [f(0,\ldots,0,0), f(0,\ldots,0,1), f(0,\ldots,1,0), \ldots, f(1,\ldots,1,1)]$. The truth table contains 2^n elements corresponding to all possible combinations of the n binary inputs.

2. **Polarity Truth Table (PT):**

   It is defined by $\hat{f}(x) = (-1)^{f(x)} = 1 - 2f(x)$, where $\hat{f}(x) \in \{-1, 1\}$.

3. **Algebraic Normal Form (ANF):**

   A Boolean function has a unique representation as a polynomial over field Z_2, called the algebraic normal form (ANF). This polynomial can be obtained by summing up distinct products terms of $x_1, x_2, \ldots, x_n$, which can be written as follows. The number of variables in the highest order product term with nonzero coefficient is called the algebraic degree and denoted by $deg(f)$.

$$f(x_1, \ldots, x_n) = a_0 \bigoplus_{i=1}^{n} a_i x_i \bigoplus_{1 \leq i < j \leq n} a_{ij} x_i x_j \bigoplus \ldots \bigoplus a_{123\ldots n} x_1 x_2 \ldots x_n,$$

   where $a_0, a_i, \ldots, a_{123\ldots n} \in Z_2$.

4. **Walsh Spectrum (WS):**

   Before this representation is described, we give some correlative fundamental definitions.

   Affine and Linear Boolean Functions:

   A Boolean function $f(x)$ having algebraic degree at most one is called an affine function of $x = (x_1, \ldots, x_n) \in \{0, 1\}^n$. Having selected $\omega \in Z_2^n$ and $c \in \{0, 1\}$, its ANF representation is denoted by

$$L_\omega(x) = c \oplus \omega x = c \oplus \omega_1 x_1 \oplus \omega_2 x_2 \oplus \ldots \oplus \omega_n x_n$$

   where $\omega_i x_i$ denotes the bitwise AND of the ith bit of ω and x, and $\oplus$ denotes bitwise XOR. An affine function with the constant term $c = 0$ is called linear function [27] [20], which is correspondingly denoted by $A_\omega(x)$.

Hamming Weight:

The Hamming weight of a Boolean function is the number of ones in its binary truth table or equivalently the number of -1's in the polarity truth table [35], and it is denoted by $wt(f)$.

Hamming Distance.

The Hamming distance between two Boolean functions f and g is the number of position in which their truth tables differ and it is denoted by

$d(f, g) = wt(f \oplus g) = \sum_{x \in Z_2^n}(f(x) \oplus g(x))$.

The Hamming distance to linear functions is an important cryptographic property, because ciphers that apply nearly linear Boolean functions are easily attacked by various methods.

Walsh Hadamard Transform (WHT).

For a Boolean function f, the Walsh Hadamard Transform $F(\omega)$ is defined by

$$F(\omega) = \sum_{x \in Z_2^n} (\hat{f}(x) \times \hat{L}_\omega(x)) = \sum_{x \in Z_2^n} (-1)^{f(x) \oplus x.\omega}. \qquad (1)$$

where $x.\omega$ denotes the dot product between ω and x, i.e.

$$x.\omega = \bigoplus_{i=1}^{n} x_i \times \omega_i.$$

From this definition, It is clear that the value of $F(\omega)$ shows how its corresponding Boolean function $f(x)$ is correlated with all linear functions.

There is also another way to define the Walsh transform $F'(\omega)$ of Boolean function f as follows.

$$F'(\omega) = \sum_{x \in Z_2^n} (f(x) \times (-1)^{x.\omega}). \qquad (2)$$

We call $F(\omega) = [F(0,\ldots,0,0),\ F(0,\ldots,0,1),\ F(0,\ldots,1,0),\ \ldots,\ F(1,\ldots,1,1)]$ the Walsh Spectrum, or simply the spectrum of $f(x)$. This is one of the most useful representations of Boolean function since several important cryptographic properties can be directly or easily checked by its corresponding Walsh Spectrum.

5. **Cayley Graph:**

We first present some definitions in terms of graph [24] and its associated Boolean functions.

Definition 1 *Cayley Graph and Cayley Set:*

Let Γ be a group with identity element e. Suppose C is a Cayley subset of G that is $e \notin C$ and whenever $g \in C$, then $g^{-1} \in C$. The Cayley graph $G = G(\Gamma, C)$ of Γ with respect to C is the graph whose vertex set is Γ, with two vertices g and h adjacent if $gh^{-1} \in C$.

(This definition is slightly modified by dropping the condition $e \notin C$. This generalization is equivalent to allowing the presence of self-loops in the graph.)

Definition 2 *The Spectrum of G_f:*

Given a graph G and its adjacency matrix A , the spectrum of G is the set of the eigenvalues of A, which are also called eigenvalues of G .

Definition 3 *Connected Graph:*

- ***Path** is a list of vertices of a graph where each vertex has an edge from it to the next vertex.*
- *A graph G in which any two vertices are connected by a path is called a **connected graph**.*

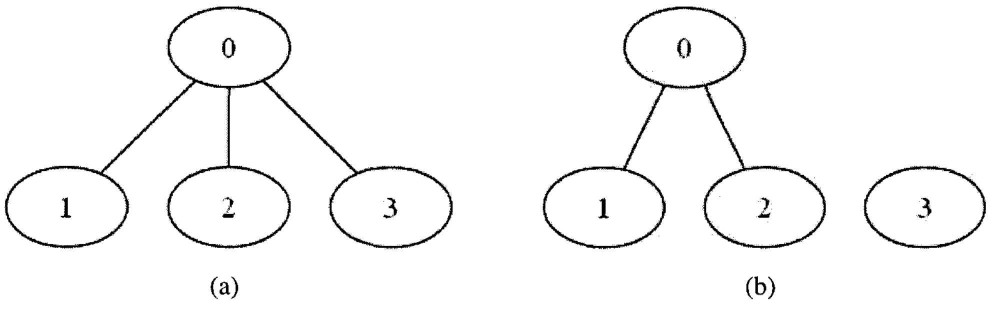

Figure 1: Examples of Connected Graph and Disconnected Graph

The left graph of Figure 1 is an example of Connected Graph, while the right graph is not a connected graph.

Definition 4 *Regular Graph:*

- *Two vertices μ and ν of a graph G are said to be* ***adjacent*** *if there is an edge joining them. The vertices μ and ν are then said to be* ***incident*** *to such an edge.* ***Degree*** *of a vertex μ of G is the number of edges incident to μ.*
- *A graph G in which every vertex has the same degree is called a* ***regular graph,*** *if every vertex has degree r, the graph is called regular of degree r.*

Figure 2.2(a) is a 3-regular graph, while Figure 2.2(b) is a 2-regular graph.

Definition 5 *Strongly Regular Graph:*

A regular graph G is strongly regular if there exist nonnegative integers e and d such that, for all vertices μ, ν, the number of vertices adjacent to both μ and ν, $\delta(\mu, \nu)$ is given by

$$\delta(\mu,\nu) = \begin{cases} e, & \text{if } \mu \text{ and } \nu \text{ are adjacent,} \\ d, & otherwise. \end{cases}$$

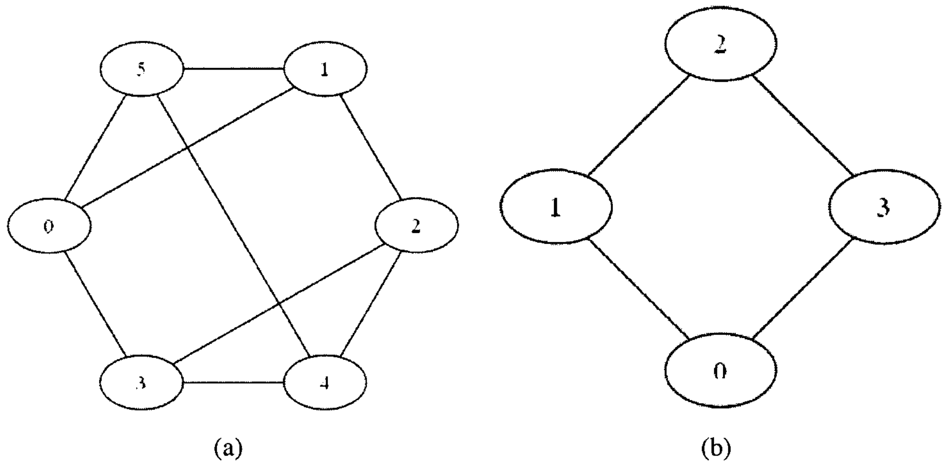

Figure 2: Examples of Regular Graph and Strongly Regular Graph

As showed in Figure 2.2(b), node 0 and 1 are adjacent and have 0 common neighbors $\Rightarrow e = 0$; node 0 and 2 are not adjacent and have 2 common neighbors $\Rightarrow d = 2$.

Definition 6 *the Graph G_f Associated to Boolean function f:*

Let $f : Z_2^n \longrightarrow Z_2$, we can associate f to the Cayley graph $G_f = G(Z_2^n, \Omega_f)$ of Z_2^n with respect to the set $\Omega_f = \{\omega \in Z_2^n | f(\omega) = 1\}$.

- *The vertex set $V(f)$ of G_f is equal to Z_2^n.*
- *The edge set E_f is defined as:*

$$\begin{aligned} E_f &= \{(\mu, \nu) \in Z_2^n \times Z_2^n | \mu \oplus \nu \in \Omega_f\} \\ &= \{(\mu, \nu) \in Z_2^n \times Z_2^n | f(\mu \oplus \nu) = 1\} \end{aligned}$$

- *The adjacency matrix A_f of such a graph is defined as $(A_f)_{i,j} = f(b(i) \oplus b(j))$ where $b(i) \in Z_2^n$ is the binary expansion of the integer i.*

In [9] [10] [50], it was shown that the spectrum of the Cayley Graph G_f coincides with the Walsh spectrum $F'(\omega)$ of its associated Boolean function f. It was also proved that the Boolean bent functions can be exactly distinguished by a special class of strongly regular graphs. However, in the literature, only bent functions are identified by special Cayley graph.

An example of $n = 3$ Boolean function is showed in different representations as follows.

binary truth table:	$f(x)$	=	[0 1 1 1 0 0 1 0]
polarity truth table:	$\hat{f}(x)$	=	[1 -1 -1 -1 1 1 -1 1]
Algebraic Normal Form:	$f(x_1, x_2, x_3)$	=	$x_1 \oplus x_2 \oplus x_1x_2 \oplus x_1x_3$
Walsh Spectrum 1:	$F(\omega)$	=	[0 0 4 4 -4 4 0 0]
Walsh Spectrum 2:	$F'(\omega)$	=	[4 0 -2 -2 2 -2 0 0]
its Cayley Graph:			is shown in Figure 3

Table 2: An Example for Boolean Function Representations

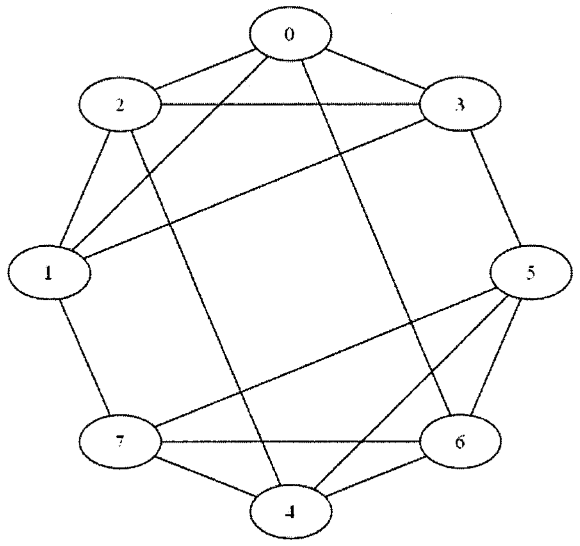

Figure 3: An Example of Cayley Graph Representation of Boolean Function

2.2.3 Cryptographic Properties of Boolean Functions

When used in cryptographic systems, Boolean functions should satisfy several cryptographic properties such as balance, high nonlinearity, resiliency, and high algebraic degree.

In this subsection, we introduce those properties.

Balance When a Boolean function has the same number of zeros and ones in its truth table, this function is called balanced. A function is balanced if and only if its Walsh transform satisfies $F(0) = 0$.

Correlation Immunity (CI) The correlation immunity of a Boolean function is to measure the degree of which its outputs are uncorrelated with some subset of its inputs. Specifically, a Boolean function is said to be correlation immune of order m if the distribution probability of its output is unaltered when any m bits of its input are fixed [46]. A function is m-th order correlation immune if and only if its Walsh transform satisfies $F(\omega) = 0$; for all ω with $1 \leq wt(\omega) \leq m$ [46] [53].

Resiliency A Boolean function is said to be resilient of order m if it is correlation immune of order m and it is balanced. Let $res(f)$ denote the resiliency degree of $f(x)$. Then

$$res(f) = m \Leftrightarrow F(w) = 0, \text{ for } 0 \leq wt(w) \leq m$$

Nonlinearity(NL) The nonlinearity of an $n - variable$ Boolean function $f(x)$ is defined as the minimum hamming distance between $f(x)$ and the set of all $n - variable$ affine functions [44] [27]. i.e.

$$nl(f) = \min_{g \in A(n)} (d(f, g)),$$

Complementing the binary truth table of a Boolean function will not change its nonlinearity, so only the 2^n number of linear functions instead of 2^{n+1} affine functions are to be considered. In terms of Walsh spectrum, the nonlinearirty of function f is given as follows:

$$nl(f) = 2^{n-1} - \frac{1}{2} \max_{\omega \in Z_2^n} |F(\omega)|.$$

Autocorrelation(AC) The autocorrelation transformation of a Boolean function *f* is given by

$$r_f(s) = \sum_x \hat{f}(x)\hat{f}(x \oplus s) = \sum_x (-1)^{f(x) \oplus f(x \oplus s)}.$$

where $s \in Z_2^n$. The maximum absolute value excluding the value at the origin (equal to 2^n) in the autocorrelation spectra of *f* is also known as the absolute indicator [54] and denoted as

$$C_{AC} = \max_{s \in Z_2^n \wedge s \neq (0,0,\ldots,0)} |r_f(s)|.$$

The lower value of C_{AC}, the better. Maximal values are serious weakness called the linear structure. Bent functions have the minimal aucorrelation, therefore, they optimize this property [38].

There are some other cryptographic criteria for Boolean functions, such as Completeness, Output Bit Independence Criterion (BIC), Strict Avalanche Criterion (SAC), Higher Order SAC, Propagation Criterion (PC), and so on. Since these criteria are not considered in our research, their discussions are omitted here.

Balanced functions, with high nonlinearity, high algebraic degree, high order of correlation immunity, and low autocorrelation, are typically preferred in the cryptographic literature. However, from the definitions above, some of these properties are in conflict. For example, bent functions (will be introduced shortly) achieve the maximum possible nonlinearity (such functions minimize the maximum magnitude of Walsh values) but are unbalanced. If we require a function to be balanced ($F(0) = 0$), then some other $F(\omega)$ must have absolute values greater than $2^{\frac{n}{2}}$ based on Parseval's theorem given in Theorem 2. This will respectively decrease its nonlinearity. The other example is that increasing order of correlation immunity can never result in an increase in achievable nonlinearity. The conflict means that tradeoffs have to be made when we construct Boolean functions [20].

In the remainder of this subsection, bent functions will be elaborated. Bent function is an important class of Boolean functions. It was defined and first analyzed by Rothaus [41]. He showed that binary bent functions exist only when the dimension n of the vector space Z_2^n is even. Several properties of bent functions were noted by Rothaus and two large classes of bent functions were also presented in his paper. Other properties, constructions, and equivalence bounds for bent functions can be found in [4, 11, 21, 42]. Kumer, Scholtz and Welch [39] defined and studied bent functions over $GF(p)$. Bent functions have been the subject of great interest in several areas including cryptography. In fact, the Canadian government block cipher standard (CAST [3]) is designed based on these functions. A Boolean function f is called bent if all the Walsh transform coefficients have the same absolute value, i.e., $|F(\omega)|$ is constant for all $\omega \in Z_2^n$. Based on Parseval's theorem (2), f is a bent function if and only if $|F(\omega)| = 2^{\frac{n}{2}}$ for all ω, to satisfy $|F(\omega)|$ to be an integer, n should be even. The bent function holds the following properties.

- Bent function achieves the maximum possible nonlinearity. The nonlinearity of any bent function is given by

$$NL = (2^{n-1} - 2^{\frac{n}{2}-1})$$

- Bent function is never balanced. However, when n is large enough, it becomes statistically indistinguishable from balanced functions.

- The order (algebraic degree) of bent function is at least 2 and not more than $\frac{n}{2}$. Bent functions of higher algebraic degree are preferred for cryptographic purposes.

- All the bent functions have zero autocorrelation for all non-zero s in Z_2^n.

Bent function can be constructed in mathematic ways. These include, but are not limited to, Rothaus' construction [41], Maiorana-McFarland's construction [40], Yarlagadda and Hershey's construction [42], Dillon's construction [22]. For example, Maiorana-McFarland constructed bent functions by concatenating affine functions as follows.

$f(x,y) = x \cdot \phi(y) + g(y)$,

where $x, y \in Z_2^{\frac{n}{2}}, \phi : Z_2^{\frac{n}{2}} \to Z_2^{\frac{n}{2}}, g : Z_2^{\frac{n}{2}} \to Z_2$,

Then, f is bent $\Leftrightarrow$ ϕ is a permutation.

Even though there are many algebraic ways to construct bent functions, it cannot be exhausted so far. There are some constructions based on heuristic methods in the literature. In our experiments, $10 - variable$ bent functions are also successfully constructed by simulated annealing.

2.2.4 Sub Classes of Boolean Functions

A variety of desirable criteria for Boolean functions with cryptographic application have been identified: balancedness, high nonlinearity, correlation immunity, high algebraic degree, and so on. It is difficult to construct an appropriate Boolean function which satisfies tradeoff requirement between these criteria from the whole set of possible Boolean functions, since the search space is very huge. Thus a natural idea is to decrease the search space by considering certain sub classes. Certain sub classes with high density of good properties have received a lot of attention in Boolean function literature [12,26,47]. Here two such sub classes of functions (RSBF, DSBF) are mentioned.

1. Rotation Symmetric Boolean Functions (RSBF) Let $x_i \in Z_2$, $1 \le i \le n$. For $1 \le k \le n$, we define

$$\rho_n^k(x_i) = \begin{cases} x_{i+k} & \text{if } i+k \le n, \\ x_{i+k-n} & \text{if } i+k > n. \end{cases}$$

The definition of ρ_n^k can be extended to n-tuples as

$$\rho_n^k(x_1, x_2, \cdots, x_n) = (\rho_n^k(x_1), \rho_n^k(x_2), \cdots, \rho_n^k(x_n)).$$

Definition 7 *A Boolean function f is called rotation symmetric (RSBF) if for each input $(x_1, \cdots, x_n) \in Z_2^n$, $f(\rho_n^k(x_1, x_2, \cdots, x_n)) = f(x_1, \cdots, x_n)$ for $1 \leq k \leq n$.*

RSBFs were first introduced in cryptography by Pieprzyk and Qu [37] in the context of efficient hash function design. The number of n-bits RSBFs is given by 2^{g_n} where

$$g_n = \frac{1}{n} \sum_{k|n} \phi(k) 2^{\frac{n}{k}},$$

where $\phi(\cdot)$ denotes the $Euler'sphi-function$ [48]. It can easily be checked that $g_n \approx 2^{\frac{2^n}{n}}$ [47]. Since $2^{g_n} << 2^{2^n}$, the number of n-variable RSBFs is much smaller than the total space of Boolean functions. On the other hand, the set of RSBFs proved to be a very rich structure full of functions with several interesting cryptographic properties [26, 49].

Note that for n-variable Boolean functions, there are 2^n different possible inputs. From the above definition, it is obvious that for RSBFs, the function f has same value for each of the classes generated from the rotational symmetry. An orbit is completely determined by its representative element $\Lambda_{n,i}$, which is lexicographically the first element belonging to the i-th orbit. The rotation-symmetric truth table (RSTT) is defined as the g_n-bit string $[f(\Lambda_{n,0}), f(\Lambda_{n,1}), \ldots, f(\Lambda_{n,g_n-1})]$ [26]. For example, for $n = 4$, the classes information is shown in Table 3. There are 6 different classes which partition the $2^4 = 16$ input patterns. So there are 2^6 RSBFs on 4 variables.

In [26], it has been shown that many functions in this class are rich in terms of good cryptographic properties. Furthermore, the RSBF class is much smaller ($g_n \approx 2^{\frac{2^n}{n}}$) comparing to the space of n-variable Boolean functions (2^{2^n}) and, hence search techniques can be more efficient. Table 4 shows the RSBF class size for certain ns. $n = 9$ RSBFs are used

class no.	inputs
1	{0,0,0,0},
2	{0,0,0,1},{0,0,1,0},{0,1,0,0},{1,0,0,0},
3	{0,0,1,1},{0,1,1,0},{1,1,0,0},{1,0,0,1},
4	{0,1,0,1},{1,0,1,0},
5	{0,1,1,1},{1,1,1,0},{1,1,0,1},{1,1,0,1},
6	{1,1,1,1}.

Table 3: Classes Information for 4-variable RSBFs

to concatenate to construct $(10, 2, 7, 488)$ Boolean functions in our experiments.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
g_n	2	3	4	6	8	14	20	36	60	108	188	352	632	1182	2192	4116

Table 4: Classes Size of RSBF

2. Dihedral Symmetric Boolean Functions (DSBF) Rotational symmetric class contains many good functions, but it is infeasible to search if $n \geq 10$ due to its space complexity. The literature tries to study some other classes with smaller size and denser functions with good properties. The class of Dihedral Symmetric Boolean functions(DSBFs) is a subclass of RSBFs [25, 31].

Before addressing DSBFs, some group theory concepts are needed.

Definition 8 *Symmetric group, Rotation (cyclic) group, Dihedral group*

Symmetric group *is a group of all permutations and denoted as S_n where n is the number of elements. This concept is used to construct Symmetric functions.*

Rotation (cyclic) group *is a group of all cyclic shift permutations and denoted as C_n. This permutation is elaborated above, and the concept is used to construct RSBFs.*

Dihedral group *is a Group of cyclic shift and reflection permutations and denoted as D_n which, besides the cyclic shift C_n, includes a reflection operator $\tau_n(x_1, x_2, \cdots, x_n) = (x_n, \cdots, x_2, x_1)$. This concept is used to construct DSBFs.*

Obviously, $S_n \subseteq D_n \subseteq C_n$. In addition,

Definition 9 *Group action, Boolean function invariant*

Group action *The group action of a group G on a set X is a mapping $\psi : G \times X \rightarrow X$ denoted as $g.x$, which satisfies the following two actions.*

$(gh) \cdot x = g \cdot (h \cdot x)$, for all $g, h \in G$ and for all $x \in X$.

$e \cdot x = x$, for every $x \in X$, e is the identity element of G.

Boolean function invariant under Group Action *Let G acts on X. A Boolean function f is said to be invariant under the action of G, if $f(g \cdot x) = f(x)$, for all $g \in G$ and for all $x \in X$. That is, $f(x)$ is same for all x in each class.*

Based on above concepts, correspondingly, Boolean functions invariant under the action of S_n is called Symmetric Boolean function; Boolean functions invariant under the action of C_n is called Rotational Symmetric Boolean function(RSBF); and Boolean functions invariant under the action of D_n is called Dihedral Symmetric Boolean function(DSBF).

Table 5 shows the DSBF class size for certain ns. $N = 10$ DSBFs will be the search space during constructing $(10, 492)$ Boolean functions in our experiments. The larger the variable n is, the bigger the difference of the number of classes between RSBF and DSBF is. The example in Table 3 also shows the class information of the $n = 4$ DSBFs, since its reflection permutations is a subset of its cyclic shift permutations. Another example, for $n = 5$, the classes information is shown in Table 6.

There are 13 different classes partitioning the $2^6 = 64$ input patterns, which have $2^{2^6} = 2^{64}$ different functions, so there are 2^{13} DSBFs on 6 variables. Note that, 6 variable RSBFs

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
d_n	2	3	4	6	8	13	18	30	46	78	126	224	380	687	1224	2250

Table 5: Classes Size of DSBF

class no.	inputs
1	{0,0,0,0,0,0},
2	{0,0,0,0,0,1},{0,0,0,0,1,0},{0,0,0,1,0,0},{0,0,1,0,0,0},{0,1,0,0,0,0},{1,0,0,0,0,0},
3	{0,0,0,0,1,1},{0,0,0,1,1,0},{0,0,1,1,0,0},{0,1,1,0,0,0},{1,1,0,0,0,0},{1,0,0,0,0,1},
4	{0,0,0,1,0,1},{0,0,1,0,1,0},{0,1,0,1,0,0},{1,0,1,0,0,0},{0,1,0,0,0,1},{1,0,0,0,1,0},
5	{0,0,0,1,1,1},{0,0,1,1,1,0},{0,1,1,1,0,0},{1,1,1,0,0,0},{1,1,0,0,0,1},{1,0,0,0,1,1},
6	{0,0,1,0,0,1},{0,1,0,0,1,0},{1,0,0,1,0,0},
7	{0,0,1,0,1,1},{0,1,0,1,1,0},{1,0,1,1,0,0},{0,1,1,0,0,1},{1,1,0,0,1,0},{1,0,0,1,0,1},
	{1,1,0,1,0,0},{0,1,1,0,1,0},{0,0,1,1,0,1},{1,0,0,1,1,0},{0,1,0,0,1,1},{1,0,1,0,0,1},
8	{0,0,1,1,1,1},{0,1,1,1,1,0},{1,1,1,1,0,0},{1,1,1,0,0,1},{1,1,0,0,1,1},{1,0,0,1,1,1},
9	{0,1,0,1,0,1},{1,0,1,0,1,0},
10	{0,1,0,1,1,1},{1,0,1,1,1,0},{0,1,1,1,0,1},{1,1,1,0,1,0},{1,1,0,1,0,1},{1,0,1,0,1,1},
11	{0,1,1,0,1,1},{1,1,0,1,1,0},{1,0,1,1,0,1},
12	{0,1,1,1,1,1},{1,1,1,1,1,0},{1,1,1,1,0,1},{1,1,1,0,1,1},{1,1,0,1,1,1},{1,0,1,1,1,1},
13	{1,1,1,1,1,1}.

Table 6: Classes Information for 6-variable DSBFs

have one more class than 6 variable DSBFs. The only difference is that the class 7 in DSBFs is split into two classes in RSBFs. In Table 6, first line of class 7 is one class in RSBF, while its reflector (second line of class 7) is another class in RSBF.

Chapter 3

Construction of Boolean Functions

There are mainly two approaches for constructing Boolean functions. Algebraic techniques [17, 28, 32] construct functions based on certain mathematical results. On the other hand, heuristic methods search for local optimal solutions within a prespecified search space. In this thesis, a combination of the above two techniques is used to reduce the search space of heuristic techniques which allow us to achieve some results that have not been achieved previously by any of the above techniques when used separately.

3.1 Motivation

Boolean functions are among the most important elements of various cryptographic algorithms. Many work exist on constructing Boolean functions with special properties. In particular, resilient functions [45] are an important class of Boolean functions. These functions play a central role in several cryptographic applications, especially stream cipher design. When used in a stream cipher as a combining function for linear feedback shift registers, a Boolean function with low-order resiliency is more susceptible to a correlation attack than a function with resiliency of high order. Let (n, m, d, NL) denote an n-variable, m-resilient Boolean function with algebraic normal form degree d and nonlinearity NL. Further, by

$[n, m, d, NL]$, we denote unbalanced n-variable, m-th order correlation immune function with algebraic normal form degree d and nonlinearity NL. Any component is replaced by '-' if we do not specify it. e.g., $(n, -, -, NL)$ if we do not wish to specify resiliency order and the algebraic degree. For simplicity, we use (n, NL) to present the above function if no ambiguity is possible. The existence of $(10, 2, 7, 488)$ functions had been an open problem [45] until this work.

A basic criterion for the construction of Boolean functions is nonlinearity. The significance of this criterion has always been emphasized due to the development of linear cryptanalysis. Dobbertin conjectured in [16] that the nonlinearity of balanced Boolean functions defined on $GF(2)^n$ cannot exceed $2^{n-1} - 2^{\frac{n}{2}} + N_\theta$, where N_θ denotes the maximum achievable nonlinearity of a balanced Boolean function θ defined on $GF(2)^{\frac{n}{2}}$. Based on his conjecture, the upper bound nonlinearity of balanced Boolean functions for $N = 8$, $N = 10$ and $N = 12$ are 116,492, and 2010, respectively. Some work exist on the constructions of such Boolean functions by arithmetic ways. However, to our best knowledge there is no work on constructions by heuristic methods. Furthermore, there is no known result on breaking this conjecture. In this thesis, we construct examples of $(8, 116)$,$(10, 492)$,$(12, 2010)$ Boolean functions by heuristic methods. Our onging work apply other methods to construct examples of Boolean functions as an attempt to break the Dobberin's conjecture.

3.2 Theorems and Lemmas

The following theorems and lemmata will be used in our construction.

Theorem 1 (Walsh Summation [20]) *This states that the absolute value of the sum of the Walsh-Hadamard Transform (WHT) values is the same constant for every Boolean function:*

$$\sum_{\omega \in Z_2^n}(F(\omega)) = 2^n \times \hat{f}(0) = \pm 2^n$$

Theorem 2 (Parseval's Theorem [29]) *This states that the sum of the squares of the Walsh-Hadamard Transform (WHT) values is the same constant for every Boolean function:*

$$\sum_{\omega \in Z_2^n}(F(\omega))^2 = 2^{2n}$$

From this theorem, we can know that a tradeoff exists in minimizing correlation to affine functions. When a Boolean function is altered to reduce its correlation to some affine functions, the correlation to some other affine functions will be increased.

Lemma 1 *If $n \geq 3$ and $m \leq n-3$, then the Walsh values of an m-th order resilient function $f(x)$ on n variables must satisfy [30]:*

$$res(f) = m \Rightarrow |F(\omega)| \equiv 0 \, mod \, 2^{m+2}.$$

Using Lemma 1, we can obtain the upper bound nonlinearity of a n-variable, m-resilient function represented by the following Theorem 3.

Theorem 3 (Nonlinearity [30]) *Upper bound nonlinearity of n-variable m-resilient Boolean function:*

1) If n is even and $m+1 > \frac{n}{2}-1$,then $NL(n,m) \leq 2^{n-1} - 2^{m+1}$.
2) If n is even and $m+1 \leq \frac{n}{2}-1$,then $NL(n,m) \leq 2^{n-1} - 2^{\frac{n}{2}-1} - 2^{m+1}$.
3) If n is odd and $2^{m+1} > 2^{n-1} - \widehat{NL}(n)$,then $NL(n,m) \leq 2^{n-1} - 2^{m+1}$.
4) If n is odd and $2^{m+1} \leq 2^{n-1} - \widehat{NL}(n)$,then $NL(n,m)$ is the highest multiple of 2^{n-1} which is $\leq \widehat{NL}(n)$.
where, $\widehat{NL}(n)$ is the maximum possible nonlinearity of an n-variable function.

From Theorem 3, we can get the corresponding upbound nonlinearity as in Table 7.

By Lemma 1, together with the condition of nonlinearity (see Section 2.2.3), possible Walsh values can be determined for resilient functions. For example, the Walsh values for

$n\backslash m$	0	1	2	3	4	5	6	7	8
5	12	12	8	0					
6	26	24	24	16	0				
7	56	56	56	48	32	0			
8	118	116	112	112	96	64	0		
9	244	244	240	240	224	192	128	0	
10	494	492	488	480	480	448	384	256	0

Table 7: Upper Bound Nonlinearity of n-variable m-resilient Boolean Function

$(7, 2, 4, 56)$ must be 0, $+16$ or -16, since a Walsh value of 32 or above will reduce the nonlinearity to $48 = \frac{1}{2}(2^7 - 32)$ or less; similarly, the Walsh values for $(10, 2, 7, 488)$, if such functions exist, must be $0, \pm16, \pm32$ or ±48. Furthermore, considering Theorem 1, when $n = 7$, the distribution of these Walsh values can be determined. For example, for $(7, 2, 4, 56)$, the Walsh spectrum must contain 36 many $+16$s, 28 many -16s and 64 many 0s or contain 36 many -16s, 28 many $+16$s and 64 many 0s. Unfortunately, when n is larger than 7, the available Walsh spectrum results do not allow us to specify the distribution of the Walsh spectrum for function f.

Lemma 2 *Let $f : Z_2^n \longrightarrow Z_2$ be the function obtained from the concatenation of f_1 and f_2, $f_i : Z_2^{n-1} \longrightarrow Z_2$ and their corresponding Walsh transforms are F_1,F_2, i.e., $f = [f_1|f_2]$. Then the Walsh transform F of f is given by*

$$F = [F_1 + F_2|F_1 - F_2].$$

Lemma 3 *Let $f : Z_2^n \longrightarrow Z_2$ be the function obtained from the concatenation of f_1 and f_2, $f_i : Z_2^{n-1} \longrightarrow Z_2$, i.e., $f = [f_1|f_2]$. Then*

$$res(f) = m \Rightarrow res(f_i) \geq m - 1,\ i = 1, 2.$$

Note that Lemma 3 is a sufficient condition instead of necessary and sufficient condition. It means that, if $res(f_i) = m - 1$ ($i = 1, 2$) is the only condition we have, we can just ascertain that $res(f) \geq m-1$, but we cannot determine whether $res(f) = m$. However, based on Lemma 2, an additional limitation can be applied on $m-1$-resilient functions f_1 and f_2 in order to achieve their concatenation to be m-resilient.

Definition 10 *Let $f : Z_2^n \longrightarrow Z_2$ be a $n-variable$ Boolean function and its corresponding Walsh transform is F. Then we denote $\bar{f} = 1 \oplus f$, and its Walsh transform is $\bar{F}$. Obviously, $\bar{F} = -F$.*

Definition 11 *Since the relationship between nonlinearity and the maximum absolute Walsh transform value is fixed, we denote the latter as $WH_{max}(f)$ for Boolean function f.*

Lemma 4 *Let $f : Z_2^n \longrightarrow Z_2$ be the function obtained from the concatenation of f_1, f_2, f_3 and f_4, $f_i : Z_2^{n-2} \longrightarrow Z_2$ and their corresponding Walsh transforms are F_1, F_2, F_3 and F_4, i.e., $f = [f_1|f_2|f_3|f_4]$. Then the Walsh transform F of f is given by*

$$F = [F_1 + F_2 + F_3 + F_4 | F_1 - F_2 + F_3 - F_4 | F_1 + F_2 - F_3 - F_4 | F_1 - F_2 - F_3 + F_4].$$

Lemma 5 *Based on [35], small changes to a truth table result in small-magnitude change to its Walsh transform values. In particular, each $F(\omega)$ will be altered by ± 2 by flipping a single bit in the truth table, while each $F(\omega)$ will be altered by 4, 0 or -4 by flipping two bits in the truth table.*

3.3 Construction of $(10, 2, 7, 488)$ Boolean Functions

3.3.1 Search Algorithm

Different optimization heuristics have been used to construct examples for Boolean functions with desirable cryptographic properties (e.g., [20,26,43]).

Before starting the search, one has to decide whether the search is performed in the Walsh spectrum domain (frequency domain) using the spectral inversion technique [20,43] or in the truth table domain (time domain). In our case, using spectral inversion does not present an attractive option. In particular, while we know that for a $(10, 2, 7, 488)$ function, $F(w)$ satisfies

$$|F(\omega)| = \begin{cases} 0 & \text{if } wt(\omega) \leq 2, \\ \leq 48 & \text{if } wt(\omega) > 2, \\ 0 \bmod 16 & \text{for all } \omega, \end{cases} \tag{3}$$

these constraints do not allow us to specify the possible distributions of F.

On the other hand, using the truth table domain, for $n \geq 9$, direct application of these heuristic techniques becomes ineffective because of the super-exponential increase (2^{2^n}) in the search space. Even if the search space is constrained to the set of RSBFs, the search space for $n = 10$ is still relatively large ($2^{g_{10}} = 2^{108}$). Our direct search for $(10, 2, 7, 488)$ Boolean function (or RSBF) proved to be not successful because of the huge search space.

Our main observation is that the search space can be reduced dramatically by noting that a $(10, 2, 7, 488)$ function, f, may be constructed by concatenating two RSBFs $f_1 : Z_2^9 \longrightarrow Z_2$ and $f_2 : Z_2^9 \longrightarrow Z_2$ that satisfy the following constraints:

$$|F_i(\omega)| = \begin{cases} 0 & \text{if } wt(\omega) \leq 1, \\ \leq 24 & \text{if } wt(\omega) = 2, \\ \leq 48 & \text{if } wt(\omega) > 2, \end{cases} \tag{4}$$

where $i = 1, 2$. The first constraint in Equation 4 follows from Lemma 3 which specifies that $res(f_i) \geq 1$. The second constraint follows from Lemma 2 and the nonlinearity of f. The third constraint also follows from the nonlinearity of f. This observation reduced the search space roughly from $2^{g_{10}} = 2^{108}$ to $2^{g_9} = 2^{60}$. It is worth noting that our search with the restriction that $res(f_i) = 2$ for $i = 1, 2$, while theoretically possible, did not yield any useful results. Let (T_0, α, MIL) denote the (initial temperature, cooling factor, maximum number of internal iterations) parameters of the SA algorithm [20]. Throughout our search for $(10, 2, 7, 488)$ Boolean function, we set $T_0 = 10,000$, $\alpha = 0.98$, and $MIL = 1000$. The SA search terminates when $T \leq 1$ or a Boolean function that satisfies certain constraint is constructed. The search procedure can be summarized as follows.

1. Construction of the first half ($f_1 : Z_2^9 \rightarrow Z_2$) :

To construct f_1, we represent the Boolean function in Polarity Truth Table (PT), and obtain neighbors in the search space by swapping two different RSBF classes and their corresponding two groups of bits. We calculate the related cost function of each Boolean function in search space by transforming it to frequency domain, i.e. Walsh Spectrum (WS) and penalizing bit by bit.

Search Space 9-bit RSBF function f_1 that satisfies the following conditions:

- It meets the constraints in Equation 4
- It is balanced, i.e. $F_1(0) = 0$

Cost Function During this stage, the following cost function is used:

$$\begin{aligned} cost(f_1) = & \sum_{\omega | wt(\omega) \leq 1} |F_1(\omega)|^2 \\ & + \sum_{\substack{\omega | wt(\omega)=2, \\ |F_1(\omega)| \notin \{8,16,24\}}} |F_1(\omega)|^2 \\ & + \max_{\omega} |F_1(\omega) - 32|^2, \end{aligned} \tag{5}$$

where $\omega \in Z_2^9$.

Note that, for the second term of the cost function above, we do not penalize the Walsh coefficients that confirm to the divisibility requirements (see Lemma 1). The reason we do penalizing in this way is based on the following fact.

From Lemma 2, we have $F = [F_1 + F_2 | F_1 - F_2]$. We can thus conclude that if both F_1 and F_2 are m-resilient Boolean functions, then their concatenation BF is m-resilient. Furthermore, $F(\omega) \equiv 0$ holds, when $2^{n-1} \leq \omega < 2^n$ and $wt(\omega) = m + 1$. This implies if we force both f_1 and f_2 to be m-resilient, their concatenation f will always have some bits satisfying $F(\omega) = 0$, which is not necessory for achieving f to be m-resilient. Based on Theorem 2, this will lead to some other bits with higher WHT value, thus decrease the nonlinearity. Hence, we relax the constraint on these bits by allowing their WHT values to be $\pm 8, \pm 16, \pm 24$. Moreover, when constructing f_2, we limit the WHT value of the corresponding bit to be its opposite. Our experimental results show that this strategy performs better than the case where this term of the cost function penalizes the Walsh coefficients. In summary, for all elements having the hamming-distance equal to 2, the limit for the corresponding Walsh value can be relaxed but should satisfy the condition for concatenation in further step.

Search Strategy In this stage, we limit the initial state to be balanced. Based on the RSBF classes information for $n = 9$, there are totally 60 classes in which two of them have one element; two have three elements; the other 56 classes have nine elements. For such a distribution, we can easily assign values to each class to keep the corresponding Boolean function balanced. Also, during transitions among search space, we keep the balanced property of Boolean functions by swapping two classes with the same number of elements and with opposite values. In this way, we can further reduce the search space from $2^{g_9} = 2^{60}$ to $\binom{2}{1} \times \binom{2}{1} \times \binom{56}{28} \approx 2^{54.8}$.

From our experiments, we succeed in constructing such Boolean functions in 4096 out of 4096 runs of simulated annealing.

trial no.	found-at-itt
1	259
2	120
3	1968
4	not found
5	934
6	691
7	82,86,306
8	not found
9	not found
10	1784

Table 8: Success Rate of Constructing f_2 based on f_1

2. Construction of the second half ($f_2 : Z_2^9 \rightarrow Z_2$) :

Once f_1 is found, we use simulated annealing methods to find another RSBF ($N = 9$), which will satisfy the requirement of Boolean function $(10, 2, -, 488)$ when concatenating with the RSBF obtained from the first step. The search space and search strategy remain the same as the above. The main difference is to minimize its cost function in the following way.

$$\begin{aligned} cost(f_2) = & \sum_{\omega | wt(\omega) \leq 1} |F_2(\omega)|^2 \\ & + \sum_{\omega | wt(\omega) = 2,} |F_1(\omega) + F_2(\omega)|^2 \\ & + (\max_{\omega}(|F_1(\omega)| + |F_2(\omega)|) - 32)^2, \end{aligned} \tag{6}$$

where $w \in Z_2^9$.

Note that, for the second term of the cost function above, we only penalize $|F_1(\omega) + F_2(\omega)|$ and but do not penalize $|F_1(\omega) - F_2(\omega)|$. This is because by Lemma 2, $F_1(\omega) - F_2(\omega)$ is for the bit whose hamming weight equals to 3.

162D	5CB7	62E5	8B7A	2D4C	AC26	C1CF	3A89
5DB2	75B0	CCF5	0D2C	B102	F4AE	1AD9	D583
63E7	8B1C	3A22	9B41	F4F4	AF32	50E2	4DF1
9B16	4059	EE34	C8BD	568D	A687	E323	915F
125C	62A0	7C49	D955	6FA5	7587	B6C6	2376
38EB	DD73	2E62	816A	CB29	B03C	1D5A	6A69
4E80	E9DF	B3F3	6A1E	49FD	2818	9146	7C89
A09F	5D86	9F04	5AE1	07F2	72C9	3CC9	6997
68C5	E023	E914	184E	E993	1774	1785	74FD
B893	875F	077E	6E70	067B	D572	2E64	BAA2
8B81	964A	957E	73EF	453F	3EAC	2DFC	2A40
1168	7EDB	B762	3E08	58ED	7860	CBC8	9849
EDB3	DA1E	F688	47F9	AB28	D084	642E	AE97
C9CE	09D5	B315	8434	7921	1DB8	98ED	872B
B5C6	E0BD	10C2	A227	9A1F	5726	D174	4B31
7A87	4803	47F2	DE95	C381	BCE7	916B	59DE

Table 9: Two Examples for (10,2,7,488) Functions in Hexadecimal Notation

Based on ten many f_1s, we test how often and how many f_2s can be constructed. The success rate to construct f_2 within 4096 rounds of simulated annealing is around 70% (see Table 8). From the table, we can know that not all Boolean functions f_1s have their corresponding f_2s satisfying that the concatenation is m-resilient, and at the same time some may have more than one such candidates.

3. Test if $f = [f_1|f_2]$ is a $(10, 2, 7, 488)$ function. This is necessary since it is not guaranteed that a solution exists for every f_1 with the above constraints. If the SA search for f_2 fails for a predetermined number of steps, then we go to step 1 and find another f_1.

3.3.2 Experimental Results

Table 9 shows, in hexadecimal notation, two examples for $(10, 2, 7, 488)$ functions constructed by our search. When terminated successfully, the search process required about 1 hour on a Dell Dimension XPS Gen 4 PC with 3.4 GHz Pentium 4 CPU and 1 GB RAM.

3.4 Construction of $(8, -, -, 116)$ Boolean Functions

Search space By lemma 1 together with the condition of nonlinearity (see 2.2.3), the possible Walsh values for $(8, 116)$ Boolean functions are $\{0, \pm 4, \pm 8, \pm 12, \pm 16, \pm 20\}$. This result does not allow us to specify the distribution of its spectrum. Furthermore, the number of classes for $N = 8$ RSBFs and DSBFs is respectively 36 and 30. The problems in search space with such complexity are easier to solve with exhaustive search and mostly likely these have already been addressed in the literature. Hence, the search space is relaxed to any $N = 8$ Boolean functions in our experiment. We also allow the initial state to be any random functions.

Neighbour policy The neighbor of current state is determined by flipping any two of its output bits. The balanced property is ignored during the neighbor selection, and instead it is determined by checking whether the solutions have element ω with $F(\omega) = 0$. This allows the non-balanced candidates to be transition state.

Cost function The following cost function is used:

$$cost(f) = \sum_{|F(\omega)|>24} ((F(\omega))^2 - 20^2)^3 \tag{7}$$

where $\omega \in Z_2^8$.

Since the only conditions for (n, NL) are nonlinearity and balanced, the cost function only penalizes the elements whose Walsh transform values exceed certain value, and the balance property is not penalized here. The only additional condition for accepting the current local optimization is that the summation of maximum and minimum absolute Walsh values cannot exceed the target $WH_{max}(f)$ (here is 24). i.e. $\max_{\omega_1} |F(\omega_1)| + \min_{\omega_2} |F(\omega_2)| \leq 24$. The reason is that whenever the above condition is satisfied, the local solution can always be transformed to be balanced with

EB85	6336	465E	B226	3F80	5FA3	E343	40B1
F35B	F7B9	4A96	B661	C92D	44D5	305C	9F57
CB8F	BC20	2628	3DCA	0751	71B4	17A3	2F80
1224	1DE5	5822	650C	7A55	94B6	AEED	7441

Table 10: Two Examples for (8,116) Functions in Hexadecimal Notation

$WH_{max}(f) \leq 24$.

Related parameters Here we set parameters (T_0, α, MIL) of the SA algorithm to be $(20,000, 0.98, 8192)$.

Experimental results From our experiments, we succeed in constructing $(8, 116)$ Boolean functions in 1000 out of 1000 runs of simulated annealing. All of them are balanced or can be transformed to be balanced by linear transformation. Note that if the neighbor were obtained by flipping one of its output bit, many of such Boolean functions would also be found, although the success rate would not be 100%. Table 10 shows, in hexadecimal notation, two examples for $(8, 116)$ functions constructed by our search.

3.5 Construction of $(10, -, -, 492)$ Boolean Functions

Search space By lemma 1 together with the condition of nonlinearity (see 2.2.3), the possible Walsh values for $(10, 492)$ Boolean functions have 21 many different choices. This result does not allow us to specify the distribution of its spectrum. Furthermore, the search space for $N = 10$ general functions and RSBFs is respectively 2^{1024} and 2^{108}. Heuristic methods cannot work efficiently on such large size problems. Hence, the search space in our experiments is determined to be $N = 10$ DSBFs.

Based on the DSBF classes information for $n = 10$, there are totally 78 classes in which two of them have one element, one have two elements, six have five elements, 39 have ten elements and the other 30 classes have 20 elements. For such a distribution, we can assign values to each class to keep the corresponding Boolean function be balanced. Once there exists $F(\omega) = 0$ in the Walsh spectrum, we can transform it to be balanced by linear transformation. Therefore, to be more generalized, we only limit the initial state to be balanced, while we ignore the balance property during transition among search space.

Neighbor policy The neighbor of current state is determined by flipping any one of its classes and its corresponding bits. The balanced property is ignored during the neighbor selection, and instead it is determined by checking whether the solutions have element ω with $F(\omega) = 0$.

Cost function The following cost function is used:

$$\begin{aligned} cost(f) = &\sum_{|F(\omega)|>TAR\_MAX} ||F(\omega)| - CACU\_COST|^3 \\ &+ \sum_{\omega} T_1(\omega) \\ &+ \sum_{0<|(F(\omega)|<AT\_LEAST} (AT\_LEAST - |F(\omega)|)^2 \\ &+ T_2, \end{aligned} \tag{8}$$

where $\omega \in Z_2^{10}$, and

$$T_1(\omega) = \begin{cases} 0 & \text{if } (F(\omega) = 0) \wedge (\#\{\omega_1 | (\omega_1 < \omega) \wedge (F(\omega_1) = 0)\} \leq 32), \\ (CACU\_COST)^3 & \text{if } (F(\omega) = 0) \wedge (\#\{\omega_1 | (\omega_1 < \omega) \wedge (F(\omega_1) = 0)\} > 32). \end{cases}$$

and

$$T_2 = \begin{cases} 0 & \text{if } \#\{\omega|(F(\omega) = 0)\} > 0, \\ 12.0 \times (TAR\_MAX)^2 & \text{if } \#\{\omega|(F(\omega) = 0)\} = 0. \end{cases}$$

The parameters in the cost function have to be tuned for optimization, such as, TAG_-MAX, AT_LEAST, and CACU_COST . From our experiments, they are set to be 36, 12, 28 respectively.

Since the only conditions for (n, NL) are nonlinearity and balanced, theoretically the first term and the last term of the cost function may be enough. The first term penalizes the elements whose Walsh transform values exceed certain value, while the last term checks whether the solution can be transformed to be balanced. However, from our experiments, if the cost function is only defined to satisfy these conditions, the result will be far away from the solution. So we penalize it for some additional states, such as, the number of 0s and the minimum Walsh values. Note that these additional penalizations are determined by our observation and experiments; there may have other ways to define the cost function.

Related parameters Here we set parameters of the SA algorithm to be $T_0 = 20,000$, $\alpha = 0.98$, and $MIL = 8192$.

Experimental results We succeed in constructing $(10, 492)$ Boolean functions in 144 out of 1024 runs of simulated annealing. All of them are balanced or can be transferred to be balanced by linear transformation. Note that if the neighbor obtained by flipping any one of the DSBF classes and its corresponding bits, many of such Boolean functions can also be found, although the success rate is far lower than that from flipping two classes. Also, the examples constructed in this way always fall into a pattern, that is, the locations of $F(\omega) = 0$ are the same no matter how we tune the parameters. We find this is because of the extra penalizations in the cost function. Table 11

E983	904A	D315	74CC	B34F	0376	7A25	E4F1
9B5B	74EB	044F	2E39	7AC8	4826	E920	EE47
829F	339E	2E74	A98F	1171	25AE	48F9	4ED3
2E99	E180	74C5	493D	AC82	5C51	FCBD	217F
9148	D7EA	5B1E	C3AD	5DE9	2B25	C882	90AB
0756	2E13	5D32	8DAD	2590	EB97	20ED	E71F
58F8	D693	F913	8041	3E25	F433	3497	0BB7
CCE5	D509	63E5	6717	ABF1	9BB7	1D57	7FFF
ECF4	FF65	FAAE	6823	FBD8	99BD	7CD5	4D1E
FACE	E394	9682	DAF2	6EB5	B277	21E3	13B9
FF98	A1FD	B94A	D320	C328	8018	B28D	EA19
68FD	9B73	DA19	7E2E	4D43	F84A	131E	DA86
AEFB	C281	DC57	EFB6	CA96	60CC	A74F	1955
F41F	58D0	9004	4385	8F5D	C5E2	B9C8	56C2
2D81	EFB6	868A	3B5F	E7CC	4293	3FB8	58E8
31B6	205F	BA85	74C8	560F	43E8	E398	9469

Table 11: Two Examples for (10,492) Functions in Hexadecimal Notation

shows, in hexadecimal notation, two examples for $(10, 492)$ functions constructed by our search.

3.6 Construction of $(12, -, -, 2010)$ Boolean Functions

In this section, we first introduce a special method for constructing $(12, 2008)$ Boolean functions by concatenating $(10, 492)$ Boolean functions. Then, with another concatenation method, simulated annealing method is used with a $(10, 492)$ function as initial state to construct $(12, 2010)$ Boolean function.

3.6.1 Construction of $(12-, -, 2008)$ Boolean Functions

As mentioned above, we can construct $(10, 492)$ Boolean function in many different ways. For $N = 10$ and $NL = 492$, we can conclude that the maximum absolute value of Walsh transform of such Boolean functions is $WH_{max}(f) = 40$.

Let f_{10} be any one of $N = 10$ and $NL = 492$ Boolean functions with Walsh transform F_{10} and $Zero_{N10}$ number of zeros in its Walsh transform, and let $f_{12} = [f_{10}|f_{10}|f_{10}|\bar{f_{10}}]$. Based on Lemma 4, the Walsh transform F_{12} of f_{12} is derived as follows,

$$\begin{aligned}F_{12} &= [F_{10} + F_{10} + F_{10} + \bar{F_{10}}|F_{10} - F_{10} + F_{10} - \bar{F_{10}}|F_{10} + F_{10} - F_{10} - \bar{F_{10}}|F_{10} - F_{10} - F_{10} + \bar{F_{10}}] \\ &= [F_{10} + F_{10} + F_{10} - F_{10}|F_{10} - F_{10} + F_{10} + F_{10}|F_{10} + F_{10} - F_{10} + F_{10}|F_{10} - F_{10} - F_{10} - F_{10}] \\ &= [2 \times F_{10}|2 \times F_{10}|2 \times F_{10}| - 2 \times F_{10}]\end{aligned} \tag{9}$$

From the above equation, the maximum absolute value of Walsh transform of F_{12} is $2 \times WH_{max}(f_(N10)$ = $2 \times 40 = 80$. Furthermore, there are $4 \times Zero_{N10}$ many zeros in the frequency domain of F_{12}. Hence, we know F_{12} can be transformed to a balanced Boolean function and has nonlinearity equals to $2^{12-1} - \frac{1}{2} \times 80$ = 2008. Similarly, the nonlinearity cannot be improved by respectively flipping one, two or three bits from the results.

3.6.2 Construction of $(12, -, -, 2010)$ Boolean Functions

Our observation Let $f = [f_1|f_2|f_3|\bar{f_3}]$, where f is $N = 12$ function. f_1 is a $(10, 492)$ Boolean function obtained above, and f_3 is any $N = 10$ bent function. Correspondingly, their Walsh transform are denoted as F, F_1, F_2, F_3 and $\bar{F_3}$. We have:

$$\begin{aligned} F &= [F_1 + F_2 + F_3 + \bar{F_3} | F_1 - F_2 + F_3 - \bar{F_3} | F_1 + F_2 - F_3 - \bar{F_3} | F_1 - F_2 - F_3 + \bar{F_3}] \\ &= [F_1 + F_2 + F_3 - F_3 | F_1 - F_2 + F_3 + F_3 | F_1 + F_2 - F_3 + F_3 | F_1 - F_2 - F_3 - F_3] \\ &= [F_1 + F_2 | F_1 - F_2 + 2 \times F_3 | F_1 + F_2 | F_1 - F_2 - 2 \times F_3] \\ &= [F_1 + F_2 | F_1 - F_2 \pm 64 | F_1 + F_2 | F_1 - F_2 \mp 64] \end{aligned} \tag{10}$$

From the above equation, for such construction, the $(12, 2010)$ Boolean function can be obtained, if and only if the following conditions are satisfied.

$$\begin{cases} (\forall \omega \in Z_2^{10}), & |F_1(\omega) + F_2(\omega)| \leq 76; \\ (\forall \omega \in Z_2^{10}), & |F_1(\omega) - F_2(\omega)| \leq 12; \\ (\exists \omega \in Z_2^{10}), & F_1(\omega) + F_2(\omega) = 0. \end{cases} \tag{11}$$

By limiting the f_1 to be $(10, 492)$, we try to relax the constraint of f_2.

From these conditions, one intuitionistic idea is to obtain f_2 by flipping a small number of output bits from f_1. To satisfy the second condition and keep flipped bits as few as possible, we may limit the number of flipped bits to be less than 6. Unfortunately, the search space is still huge by flipping less than six bits. It is equal to $\sum_{i=1}^{6} \binom{1024}{i} \approx 1.6 \times 10^{15} \approx 2^{51}$. This space still cannot be searched by brute force, and heuristic method is more feasible.

Furthermore, based on the conditions, we can get the following fact,

$F_2(\omega) \in [28, 36], \forall \omega \in \{\omega | F_1(\omega) = 40\}$

and

$F_2(\omega) \in [-36, -28], \forall \omega \in \{\omega | F_1(\omega) = -40\}$

This fact means the Walsh transform values in f_2 for all the positions with maximum absolute Walsh transform values in f_1 should be improved. This has quite low possibility.

Search space The search space is regular 10-bit Boolean function and the initial state is limited to be f_1.

Neighbor policy The neighbor of current state is determined by flipping any two bits of its output.

Cost function The following cost function is used:

$$\begin{aligned} cost(f_2) = & \sum_{|F_2(\omega)|>TAR\_MAX} 2.0 \times ||F_2(\omega)| - CACU\_COST|^3 \\ & + \sum_{\omega} T_1(\omega) \\ & + \sum_{0<|(F_2(\omega)|<AT\_LEAST} (AT\_LEAST - |F_2(\omega)|)^2 \\ & + \sum_{|F_1(\omega)+F_2(\omega)|>76} (|F_1(\omega) + F_2(\omega)| - 68)^2 \\ & + \sum_{|F_1(\omega)-F_2(\omega)|>12} (|F_1(\omega) - F_2(\omega)|)^2 \\ & + T_2, \end{aligned} \tag{12}$$

where $\omega \in Z_2^{10}$, and

$$T_1(\omega) = \begin{cases} 0 & \text{if } (F(\omega) = 0) \wedge (\#\{\omega_1 | (\omega_1 < \omega) \wedge (F_2(\omega_1) = 0)\} \leq 32), \\ (CACU\_COST)^3 & \text{if } (F(\omega) = 0) \wedge (\#\{\omega_1 | (\omega_1 < \omega) \wedge (F_2(\omega_1) = 0)\} > 32). \end{cases}$$

and

$$T_2 = \begin{cases} 0 & \text{if } \#\{\omega | (F_2(\omega) = 0)\} > 0, \\ (TAR\_MAX)^5 & \text{if } \#\{\omega | (F_2(\omega) = 0)\} = 0. \end{cases}$$

The parameters in the cost function have to be tuned for optimization, such as, TAG_-MAX, AT_LEAST , and CACU_COST . From our experiments, they are set to be 36, 16, 28 respectively.

From the above function, the constraints from term 1 to 3 and 6 is similiar with the cost function for constructing $(10, 492)$ Boolean function. The purpose is to force f_2 to be around $(10, 492)$. Although the constraint seems to be unnecessary, our experiments show that it helps to obtain the solution.

The constraint term 4 is related to the condition $|F_1 + F_2| \leq 76$, while the constraint term 5 is related to the condition $|F_1 - F_2| \leq 12$.

Additionally, $WH_{max}(f) > 44$ is checked. The reason we check it is as follows. If $WH_{max}(f) > 44$ then we can know that no matter what value its corresponding F_1 is, the combination between F_1 and F_2 cannot satisfy the first two conditions of Equation 11 at the same time.

Although the only conditions that F_2 should satisfy are Conditions 11, from our experiments, if the cost function is only defined to satisfy these, the results will be far away from the solution.

Related parameters The parameters of the SA algorithm are set to $T_0 = 100$, $\alpha = 0.99$, and $MIL = 4096$.

Experimental results Based on 50 many f_1s, we test how often and how many f_2s can be constructed in 32 runs of simulated annealing. From the samples, not all Boolean functions f_1 has its corresponding f_2 , while some of them can find as many as 20

f_1:

```
1001001000001001000000011000011001010100010001101100000100111000
0011011001110000011101010011110111110101010101100001111110000100
0000011001101100011010100000000101111111111001100100101110000011
1010111100100011001001110011100100010110101111101001000011110100
0000000101111000011011001111101011111000110010000100010001010011
0010111110101110010110100101100000110101100111101101000100001011
1000100111111110000011000001111101011001001111100101111111010011
0101011100111110110010111011110110000010000000010111101101110100
0100000100010011011010101001010001101100111101011111111001110010
0010101010000001111101001001000100110001001101010111011001001110
0001110010101111110010001010110100110010111011000111011011010001
0001111000100111110101001111110111110110000101110000000111011110
1001010010000110101011111110110000000001111001010101011111101010
0110001110000011010010101110110101100011111011111110011100011110
0110001000111010000110111111000010010001100010111001101110110110
1000010001011100010100010101011010101110110111100010111000100001
```

f_2:

```
1001001000001001000000011000011001010100010001101100000100111000
0011011001110000011101010011100111110101010101100001111110000100
0000111001101100011010100001000101111111111001100100101110000011
1010111100100011001001110011100100010111101111101001000011110100
0000000101111000011011001111101011111000110010000100010001010011
0010111110101110010110100101100000110101100011101101000100001011
1000100111111110000011000001111101011001001110100101111111010011
0101011100111100100010111011110110000010000000010111101101110100
0100000100010011011010101001010001101100111101011111111001110010
0010101010000001111101001001000100110001001111010110011001001110
0001110010101111110010001010110100110010111011000111010011010001
0001111001100111110101101111110111110110000101110000000111011110
1001010010000110101011111110110000000011111001010101011111101010
0110001110000011010010101100110101100011111011111110011100011110
0110001000111010000110111111100010010001100010111001101110110110
1000010001011100010100010101011010101110110111100010111000100001
```

Table 12: F_1, F_2 Example for $(12, 2010)$ Functions (Difference Showed in Bold and Underline)

candidates. Hence, the selection of f_1 is critical to the search procedure.

Table 12 shows an example constructed in this way, in which F_1 is a $(10, 492)$ Boolean function, and F_2 is obtained by simulated annealing with F_1 as initial state. From this table, the hamming distance between these two functions is 16 instead of what we expected, i.e., 6. As mentioned above, these two functions concatenate with any $N = 10$ bent function F_3 and $\bar{F}_3$ will lead to the outcome's nonlinearity being 2010. Furthermore, $F_1(0) = F_2(0) = 0$, which implies the outcome satisfies $F(0) = 0$ which means it is balanced.

3.7 Open Problem

3.7.1 Attempts on Constructing $(8, -, -, 118)$ Boolean Functions

Our ultimate goal is to construct $(8, 118)$ Boolean functions. Some attempts are applied as follows.

Attempt 1: Adjust the parameters in cost function with constructing $(8, 116)$ Inspired by the results in Section 3.4, the same idea as in constructing $(8, 116)$ Boolean functions is applied to construct $(8, 118)$.

Now the cost function is adjusted to be:

$$cost(f) = \sum_{|F(\omega)|>20} ((F(\omega))^2 - 16^2)^3 \tag{13}$$

where $\omega \in Z_2^8$. The additional condition is also changed to $\exists \omega \in Z_2^8, F(\omega) = 0$ and the maximum absolute Walsh value cannot exceed the target $WH_{max}(f)$ (which is equal to 20), that is, $\max_\omega |F(\omega)| \leq 20$. Here, we do not check the summation of maximum and minimum, because there is no such result.

We did not succeed in constructing the Boolean functions with this method.

Attempt 2: Concatenate two $(7, 55)$ **Boolean functions** It is mathematically proved in [44] that, if it is impossible to construct $(8, 0, 7, 118)$ function by concatenating two 7-variable, degree 7, nonlinearity 55 functions, then the maximum nonlinearity of balanced 8-variable functions is 116.

Based on the above conclusion, concatenating two $(7, 55)$ Boolean functions is attempted. The search procedure can be summarized as follows.

- Obtain first 7-bit function f_1 with cost function:

$$cost(f_1) = \begin{cases} 0 & \text{if } WH_{max}(f) = 18, \\ (WH_{max}(f) - 18)^2 & \text{if } WH_{max}(f) \neq 18 \end{cases}$$

  Then the algebraic degree and the balance of the local solution is checked.

- Obtain second 7-bit function f_2 with cost function:

$$cost(f_2) = \sum_{|F_1(\omega)|+|F_2(\omega)|>20} (|F_1(\omega)| + |F_2(\omega)| - 16)^2$$

Then the balance of the local solution is checked. We did not succeed in constructing the Boolean function with this method.

3.7.2 Attempts on Constructing $(10, -, -, 494)$ Boolean Functions

Our ultimate goal is to construct $(10, 494)$ Boolean functions as the counter example for breaking H.Dobbertin's conjecture. Some attempts are described as follows.

Attempt 1: Adjust the parameters of cost function in constructing $(10, 492)$ Inspired by the results in Section 3.5, the same idea as in constructing $(10, 492)$ Boolean functions is applied to construct $(10, 494)$. The program continues to search for $(10, 494)$ functions instead of exiting after $(10, 492)$ function is found. The parameters in cost functions are adjusted to different values. However, after around two months' running, no $(10, 494)$ function was found.

Attempt 2: Concatenate two $N = 9$ Boolean functions In this attempt, two $N = 9$ Boolean functions are constructed and then concatenated to yield $N = 10$s based on a similar idea as in Section 3.3. However, this problem is different from the $(10, 2, 7, 488)$

problem. In constructing $(10, 2, 7, 488)$, there are some guidable, essential constraints, such as $F(\omega) = 0$ for all $wt(\omega) \leq 1$ and $F(\omega) mod 4 = 0$ for all ω. For $(9, -)$ to be concatenated with $(10, 494)$, the only necessary condition for constructing F_1 is as follows,

$$(\forall \omega \in Z_2^9), |F_1(\omega)| \leq 36; \tag{14}$$

and for f_2,

$$\begin{cases} (\forall \omega \in Z_2^9), & |F_2(\omega)| \leq 36, \\ (\forall \omega \in Z_2^9), & (|F_1(\omega) + F_2(\omega)| \leq 36) \wedge (|F_1(\omega) - F_2(\omega)| \leq 36) \\ (\exists \omega \in Z_2^9), & (F_1(\omega) + F_2(\omega) = 0) \vee (F_1(\omega) - F_2(\omega) = 0). \end{cases} \tag{15}$$

Experimental results should be analyzed based on the tradeoff in above conditions. For example, from Condition 14, when $WH_{max}(f_1)$ is limited to be 36, f_1 can be easily found. However, the weaker the conditions are, the more random the results are. It is possible that a large number of f_1s will never be used to find corresponding f_2s. In the mean time, if we penalize this by extra constraints, it is possible that such extra constraints may prevent us from finding any solution.

Different extra constraints are tested during the experiments. One of them is drafted as follows.

Search space The search spaces for f_1 and f_2 are limited to $N = 9$ RSBFs.

Neighbor policy Flipping one class and its corresponding bits.

Cost function

$$cost(f_1) = \sum_{\substack{\omega|\omega>0,\\ |F_1(\omega)|>36}} |F_1(\omega) - 24|^2 \\ + T_1, \tag{16}$$

$$T_1 = \begin{cases} 0 & \text{if } (|F_1(0)| \leq 18, \\ |F_1(0)|^2 & \text{if } (|F_1(0)| > 18. \end{cases}$$

$$cost(f_2) = \sum_{\substack{\omega|\omega>0,\\ |F_2(\omega)|>36}} |F_2(\omega)|^2 \\ + \sum_{\substack{\omega|\omega>0,\\ |F_1(\omega)|+|F_2(\omega)|>36}} (|F_1(\omega)| + |F_2(\omega)| - 24)^2 \\ + T_2, \tag{17}$$

$$T_2 = \begin{cases} |F_1(0) + F_2(0)|^3 & \text{if } (|F_1(0) + F_2(0)| \neq 0, \\ (|F_1(0) - F_2(0)| - 24)^2 & \text{if } (|F_1(0) - F_2(0)| > 36. \end{cases}$$

In this way, we expect the results of two concatenations to be balanced so the extra condition would be unecessary.

Attempt 3: Brute Force - flipping two bits of $(10, 492)$ **functions** Based on Lemma 5, it is possible that, after flipping certain two bits of the $(10, 492)$ Boolean functions, the Walsh value reduce from 40 to 36, and increase from -40 to -36, with some 0's in the derived Boolean function. In our experiments, around 200 many $(10, 492)$ Boolean functions are verified in this way, but none of them falls into this class.

Attempt 4: Hill Climbing - SA applied on $(10, 492)$ functions Based on the result of constructing f_2 in Section 3.6.2, if this function exists then the hamming distance between the $(10, 494)$ function and its closest $(10, 492)$ function will not necessary be equal to 2, but instead may be a little greater than 2. However, it is impractical to verify all the possibilities by flipping more than 2 bits for $N = 12$, so simulated annealing is used and the details are shown in the following.

Search space The search space is regular 10-bit Boolean function, and the initial state is limited to be $(10, 492)$ functions which is already found.

Neighbor policy The neighbor of current state is determined by flipping any two output bits. The balanced property is ignored during the neighbor selection, and instead it is determined by checking whether the solutions have element ω satisfying $F(\omega) = 0$.

Cost function The following cost function pattern is used:

$$\begin{aligned} cost(f) = & \sum_{|F(\omega)|>TAR\_MAX} 2.0 \times ||F(\omega)| - CACU\_COST|^3 \\ & + \sum_{\omega} T_1(\omega) \\ & + \sum_{0<|(F(\omega)|<AT\_LEAST} (AT\_LEAST - |F(\omega)|)^2 \\ & + T_2, \end{aligned} \tag{18}$$

where $\omega \in Z_2^{10}$, and

$$T_1(\omega) = \begin{cases} 0 & \text{if } (F(\omega) = 0) \wedge (\#\{\omega_1 | (\omega_1 < \omega) \wedge (F(\omega_1) = 0)\} \leq 32), \\ (CACU\_COST)^3 & \text{if } (F(\omega) = 0) \wedge (\#\{\omega_1 | (\omega_1 < \omega) \wedge (F(\omega_1) = 0)\} > 32). \end{cases}$$

and

$$T_2 = \begin{cases} 0 & \text{if } \#\{\omega | (F(\omega) = 0)\} > 0, \\ (TAR\_MAX)^5 & \text{if } \#\{\omega | (F(\omega) = 0)\} = 0. \end{cases}$$

The cost function is similar to the one used in constructing $(10, 492)$ functions in Section 3.5. The main difference is that here the initial state is one of the $(10, 492)$ functions obtained in Section 3.5. The main idea is that the hamming distance between $(10, 494)$ functions and $(10, 492)$ functions is potentially not large.

Moreover, many other cost functions have been designed and their parameters adjusted, but no better results are obtained.

Related parameters Here we set parameters of the SA algorithm to be $T_0 = 100$, $\alpha = 0.99$, and $MIL = 4096$. Note that when T_0 is increased to around $1,000$, we cannot even construct $(10, 492)$ functions.

Experimental results It took 6 months to apply this method on 30 many $(10, 492)$ Boolean functions, with 1024 iterations each. The $(10, 494)$ functions are not found in this way. However, many of other $(10, 492)$ functions are constructed under each seed. Moreover, some examples do not fall into the same pattern. Table 13 shows the number of $(10, 492)$ functions found with ten of the seeds.

Others Other attempts are also made including, but not limited to,

- Replace the search space from DSBFs to RSBFs;
- Constraint the neighbor to be balanced;
- Fix some classes when moving to next state;
- Use new random function instead of the one in standard C library.

These attempts, while also theoretically possible, do not yield any better results.

seed#	the number of (10,492) functions found
1	851
2	698
3	631
4	602
5	575
6	609
7	549
8	527
9	688
10	709

Table 13: The Number of (10,492) Boolean Functions Found by 1024-time SAs on Seed Functions

3.7.3 Attempts on Constructing $(12-,-,2012)$ Boolean Functions

The ultimate goal is to construct $(12, 2012)$ Boolean function. Some attempts are made through revising methods in Section 3.6.2.

Attempt 1: Adjust the parameters in the cost function for constructing $(12, 2010)$ We apply simliar idea as in constructing (12,2010) Boolean functions to construct $(12, 2012)$. That is, let $f = [f_1|f_2|f_3|\bar{f}_3]$ in which f_1 is a $(10, 492)$ Boolean function obtained as in Section 3.5, and f_3 and $\bar{f}_3$ are bent functions. Then simulated annealing method is applied to search function f_2.

Now the conditions are adjusted to be:

$$\begin{cases} (\forall\omega \in Z_2^{10}), & |F_1(\omega) + F_2(\omega)| \le 72; \\ (\forall\omega \in Z_2^{10}), & |F_1(\omega) - F_2(\omega)| \le 8; \\ (\exists\omega \in Z_2^{10}), & F_1(\omega) + F_2(\omega) = 0. \end{cases} \tag{19}$$

With f_1 also limited to be $(10, 492)$, $F_1(\omega) = 40$ implies $F_2(\omega)$ must be 32, and

$F_1(\omega) = -40$ implies $F_2(\omega)$ must be -32.

We did not succeed in constructing the Boolean function using this method after applying it to one f_1 in 400 iterations (which consumes 18 days).

Attempt 2: Relax the constraint of f_1 to lower nonlinearity The f_2 in attempt 1 has strict constraint on the elements with $F_1(\omega) = \pm 40$, and f_2 must have nonlinearity no smaller than that of f_1. That is, both f_1 and f_2 are $(10, 492)$ Boolean functions. This result reaches the upper bound of Dobberin's conjecture, with additional constraints on those functions.

In attempt 2, the nonlinearity of f_1 and f_2 is relaxed to be certain lower values (e.g. 44 and 48), so that there would be more candidates of f_1 and f_2.

However, when only relaxing f_1 and f_2 to lower nonlinearity and keeping f_3 and $\bar{f}_3$ unchanged, there is no solution available. For example, suppose $F_1(\omega) = 44$, in order to satisfy $|F_1(\omega) + F_2(\omega)| \leq 72$, $F_2(\omega)$ must be no greater than 28, which breaks the $2nd$ condition $|F_1(\omega) - F_2(\omega)| \leq 8$. It is easy to prove that if $F_1(\omega) > 40$ under such a concatenation, then no matter what value its corresponding $F_2(\omega)$ is, the combination of F_1 and F_2 cannot simultaneously satisfy the first two of Conditions 19.

Attempt 3: Replace f_3 with its 1 or 2 bit flipping, and respectively obtaining $\bar{f}_3$ Based on the observation of Lemma 5, after flipping bits of $N = 10$ bent function, the Walsh transform values will belong to: $\{\pm 30, \pm 34\}$ after 1-bit flipping, and $\{\pm 28, \pm 32, \pm 36\}$ after 2-bit flipping. Therefore,

$$\begin{cases} |F_3(\omega) + \bar{F}_3(\omega)| = 0; \\ |F_3(\omega) - \bar{F}_3(\omega)| \in \{56,60,64,68,72\} \end{cases}$$

Now the conditions are adjusted to be:

$$\begin{cases} (\forall\omega \in Z_2^{10}), & |F_1(\omega) + F_2(\omega)| \leq 72; \\ (\forall\omega \in Z_2^{10}), & |F_1(\omega) - F_2(\omega)| \leq \delta; \\ (\exists\omega \in Z_2^{10}), & F_1(\omega) + F_2(\omega) = 0. \end{cases} \tag{20}$$

wherein,

$$\delta = \begin{cases} 12, & F_3(\omega) = \pm 30 \\ 4, & F_3(\omega) = \pm 34 \end{cases}$$

when flipping one bit, or

$$\delta = \begin{cases} 16, & F_3(\omega) = \pm 28 \\ 8, & F_3(\omega) = \pm 32 \\ 0, & F_3(\omega) = \pm 36 \end{cases}$$

when flipping two bits.

From Conditions 20, all conditions remain the same except the second condition. The difference between $F_1(\omega)$ and $F_2(\omega)$ is changed from only allowing ≤ 8 to having different possibility element by element (from 0 to 16). When f_1 is limited to $(10, 492)$ Boolean functions, this allows f_2 to have lower nonlinearity. For example, for certain ω, $F_1(\omega) = 28$, $F_2(\omega) = 44$, $F_3(\omega) = 28$, and $F_4(\omega) = \bar{F}_3(\omega) = -28$ may be acceptable. However, this revision only makes possible Walsh transform values to be as high as 44. It does not relax the constraints to generate more relaxed candidates. That is, if there is a component satisfying $F_i(\omega) = 44$, then the Walsh value for the other threes can only be 28, 28 and -28.

The procedures used in our experiment are as follows.

- Construct one $N = 10$ bent function f_{bent};
- Flip f_{bent} 1 or 2 bit(s), get f_3 and calculate f_4=$\bar{f}_3$;
- Check whether $(10, 492) f_1$ satisfies $|F_1(\omega)| + |F_3(\omega)| \leq 72$, and if no, check next f_1
- Use simulated annealing method to search for f_2 with f_1 as the initial state and bit complement in truth table as neighbor.
- Repeat above 4 steps.

Attempt 4: The $4th$ component relaxed from $\bar{f}_3$ bent function to any bent functions By observing the experiment of constructing $(12, 2010)$, we have that although $F_1 - F_2 \leq 12$, the f_2 is not derived by flipping 6 bits of f_1. This implies that there are few possible solutions by requiring $F_1 - F_2 \leq 8$. Relaxing $4th$ component will allow f_1 and f_2 to have a large-magnitude difference.

We adjusted the parameters to a large extent under the same search space, neighbor policy, and cost function, but the results could not be improved.

3.7.4 Other Attempts on Constructing $(12, -, -, 2012)$ Boolean Functions

Attempt 1: Construct through concatenation of $(10, 494)$ functions Based on the idea described in Section 3.6.1, if $(10, 494)$ functions can be constructed, we can obtain $(12, 2012)$ function by concatenating them as $f_{12} = [f_{10}|f_{10}|f_{10}|\bar{f}_{10}]$. The maximum absolute value of Walsh transform of F_{12} is $2 \times WH_{max}(f_{N10})$ = 2×36 = 72. Furthermore, there are $4 \times Zero_{N10}$ many zeros in frequency domain of F_{12} with $F(0) = 0$. Hence, we know F_{12} is balanced and has nonlinearity equals to $2^{12-1} - \frac{1}{2} \times 72 = 2012$. Obviously, this problem is equivalent to constructing $(10, 494)$ Boolean functions, which is also an open problem based on the conjecture made by Dobbertin.

Attempt 2: Directly apply simulated annealing method to $N = 12$ DSBFs In this way, the best solution we have achieved is to construct $(12, -, -, 2000)$ Boolean functions. The detailed procedure is as follows.

Search space The search space is limited to $N = 12$ DSBF functions. Based on the class information for $n = 12$, there are totally 224 classes in which two of them have one element, one has two elements, two have three elements, three have four elements, seven have six elements, 82 have 12 elements and the others (127s) have 24 elements. We allow the initial state to be any random DSBF functions.

Neighbor policy The neighbor of current state is determined by flipping any one of its classes and its corresponding bits.

Cost function The following cost function is used:

$$cost(f) = \sum_{|F(\omega)|>96} |F(\omega) - 68|^3 + \sum_{\omega} T(\omega), \tag{21}$$

where $\omega \in \Lambda_{12,i}(i \in Z_{g_n})$, and

$$T(\omega) = \begin{cases} 0 & \text{if } (F(\omega) \leq 20) \wedge (\#\{\omega_1 | (\omega_1 < \omega) \wedge (|F(\omega_1)| \leq 20)\} \leq 60), \\ 2^{(20-|F(\omega)|)} & \text{if } (F(\omega) \leq 20) \wedge (\#\{\omega_1 | (\omega_1 < \omega) \wedge (|F(\omega_1)| \leq 20)\} > 60). \end{cases}$$

Note that, the only conditions for (n, NL) are nonlinearity and balanced. The cost function theoretically can only penalize the elements whose Walsh transform values exceed certain value. However, since this condition is too relaxed to guide the SA in finding the solution in the right direction, we also penalize certain low values of Walsh transform in the second term of the above cost function.

Related parameters The parameters of the SA algorithm are set to be $T_0 = 10,000$, $\alpha = 0.96$, and $MIL = 8192$.

Experimental results 170 out of 185 trials succeed. We adjust the parameters to a large extent under the same search space, neighbor policy, and cost function, but results cannot be improved. Also, our experiments cannot improve the nonlinearity by respectively flipping one, two or three bits from 100 examples of the results.

Attempt 3: Relax f_3 and $\bar{f}_3$ to be any Boolean functions instead of bent functions

Our observation Let $f = [f_1|f_2|f_3|f_4]$, where f is $N = 12$ function. f_i $(i = 1, 2, 3, 4)$ represents four $N = 10$ Boolean functions, and their Walsh transform spectrum are denoted as F, F_1, F_2, F_3, and F_4. Based on Lemma 4, the Walsh transfrom spectrum F derived from $F_i(i = 1, 2, 3, 4)$ can be formulated as follows, when ignoring the order of concatenation.

$$F = [(F_i + F_j) \pm (F_k + F_l)|(F_i - F_j) \pm (F_k - F_l)] \tag{22}$$

where tuple (i, j, k, l) can be any permutation of $(1, 2, 3, 4)$.

From Equation 22, we have that to obtain certain nonlinearity Boolean functions, the following conditions must be satisfied.

$$|F_i| + |F_j| \leq WH_{max}(f), ((i, j \in \{1, 2, 3, 4\}) \wedge (i \neq j)) \tag{23}$$

Condition 23 can be used to define the cost functions for f_2, f_3 and f_4.

However, such constraint is not sufficient to guide heuristic methods to search for the solution in the right direction. There are still many other factors that should be considered during the construction, such as the search space, the goal for each component, the definition of neighborhood, the practical cost functions, and so on.

There are other tradeoffs that may be made. For example, if the constraint for searching f_1 is relaxed too much to accept very low nonlinearity, then f_1 can be easily found. However, it is possible that the f_1 found will never have remainder components (f_2, f_3 and f_4) for concatenation. This is similar to randomly picking up any one f_1. Unfortunately, since the (n, NL) functions are only relevant to two conditions, that is, balanced and nonlinearity, we cannot apply further conditions to it. We only know that we cannot concatenate two $(7, 56)$ Boolean functions to form $(8, 118)$ functions. If $(8, 118)$ function exists through concatenation, then it should be the concatenation of two $(7, 55)$ functions. So far there is no known result on what kind of f_i can or cannot be concatenated to construct (n, NL) Boolean functions.

We conduct experiments with different combinations, but $(12, 2012)$ Boolean functions cannot be constructed.

Search space The search space can be either $N = 10$ RSBF or DRSBF Boolean functions, whose sizes are 2^{108} or 2^{78}, respectively. The regular Boolean functions are not tried since the search space size $(2^{2^{10}} = 2^{1024})$ is prohibitive. We allow the initial state to be any random RSBF or DRSBF functions.

Neighbor policy The neighbor of current state is determined by flipping any one or two class(es) and its corresponding bits.

Cost function the basic idea of cost function is to satisfy $|F_i| + |F_j| \leq 72$ where $i, j \in \{1, 2, 3, 4\} \wedge i \neq j$.

	The number of derived functions			
Seed	$WH_{max}(f)$=72	$WH_{max}(f)$=76	$WH_{max}(f)$=80	$WH_{max}(f)$=84
1	0	44	2064876	31208
2	0	28	2064876	31224
3	0	16	2064876	31236
4	0	12	2064876	31240
5	0	28	2064876	31224

Table 14: Nonlinearity Distribution of Derived Boolean Functions by Flipping Five (12,2010) Functions

Related parameters The parameters (T_0, α, MIL) of the SA algorithm for four components are respectively set to $(10000, 0.98, 1000)$, $(10000, 0.98, 2048)$, $(10000, 0.98, 1024)$, $(10000, 0.98, 4096)$.

Attempt 3: Brute Force - flipping two bits of $(12, 2010)$ **functions** Based on Lemma 5, it is possible that, after flipping certain two bits of the $(12, 2010)$ Boolean functions, the Walsh value reduce from 76 to 72 and increase from -76 to -72, with some 0s kept in the derived Boolean function. In our experiments, around 100 many $(12, 2010)$ Boolean functions are verified in this way, but no one falls into this class. Furthermore, as shown in Table 14 (only five examples are listed here), there are only very few derived functions that can keep the same nonlinearity while most of them have reduced nonlinearity.

Chapter 4

Other Applications: Cryptanalysis of Symmetric Ciphers

In this chapter, we describe our research on using heuristic methods to attack symmetric ciphers. Two example attacks are given in this chapter. We use simulated annealing to reconstruct the initial state of the LFSRs for Geffe cipher (stream cipher); in the mean time, we use guided search techniques to perform experiments on key distinguishing attack [2] on TREYFERR cipher (block cipher). The most attractive aspect of such cryptanalysis approach is that the ciphers being attacked can be treated as black-boxes by the attackers. If such attack succeeds, it would be unnecessary for attackers to understand internal details of the ciphers.

4.1 Attack on Geffe Cipher

One of the main objectives in attacking a LFSR-based stream cipher is to reconstruct the initial state of all the LFSRs, which are the key components of stream ciphers. In our attack on Geffe cipher, we need not to analyze the internal details of the cipher, that is, we regard the cipher as a black-box. We feed the cipher with different LFSR register initial states and try to analyze the difference between keystream outputs (generated sequences) under different initial states, and output sequences we observe (observed sequences).

4.1.1 Stream Ciphers

Stream ciphers are important primitives for ensuring privacy in communication. Stream ciphers have good properties, such as being secure, efficient, and small in terms of implementation. Stream cipher algorithms are usually faster than block ciphers, such as DES. Stream ciphers are often used in mobile devices, such as A5 in GSM cell phone system. Performance benefits may lead to their application to videoconferencing and other multimedia applications [7]. A stream cipher produces a pseudo-random sequence of bits which are exclusive-OR'ed with the plaintext to produce the ciphertext. It is sometimes also called state cipher since encryption depends on not only the key and plaintext, but also on the current state.

Linear Feedback Shift Registers (LFSR) Many stream ciphers make use of the linear feedback shift register (LFSR), since:

1. LFSR is well-suited for hardware implementation;
2. LFSR can produce sequences of large period;
3. LFSR can produce sequences with good statistical properties;
4. LFSR can be readily analyzed using algebraic techniques because of its structure.

Figure 4 illustrates a LFSR defined by the primitive polynomial $x^{10} \oplus x^3 \oplus 1 = 0$.

An LFSR is a finite state machine and consists of L memory cells (stages) $r_0, r_1, \ldots, r_{l-1}$. Each cell contains one value from Z_2. LFSR has one input and one output, and a clock which is used to control the movement of data. At any time t, the content of the register is called the state of the LFSR at time t, and denoted as $S_t = (s_{t+l-1}, s_{t+l-2}, \ldots, s_t)$. The

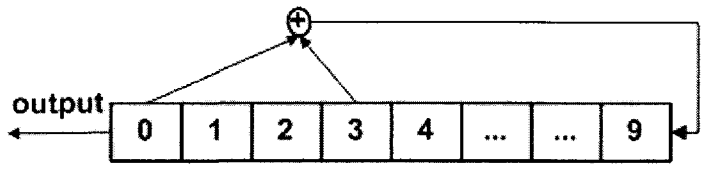

Figure 4: An Example of Linear Feedback Shift Register (LFSR)

state at time zero, S_0, is called the initial state of the LFSR.

During each unit of time, the following operations are performed. When the control unit of the finite state machine is clocked,

1. The value of the cell r_0 goes to the output and forms part of the *output sequence*;

2. The remaining cells are shifted as $r_i = r_{i+1}, i = 0, 1, \ldots, l-2$; and

3. The last cell r_{l-1} is the*feedback bit* which is loaded with a new value s_{t+l} calculated through the corresponding primitive polynomial.

Note that the new value of the last cell can either be the output.

A periodic LFSR is defined by a (primitive) feedback polynomial of degree L, which is the size of the LFSR. When the feedback polynomial is primitive and of degree L, the output sequence of a maximum length LFSR is periodic with period $m = 2^L - 1$ and is called an m-sequence. Note that m-sequences have good statistical properties but they are predictable. If a stream cipher has linear complexity n, we can find its initial state using $2 \times n$ consecutive bits using Massey-Berlekamp algorithm. Hence, we need to increase the linearity complexity, before the sequence can be used. There are several methods for achieving this. One is to use several LFSRs and combine the output from each of them like in Geffe cipher; the other is to apply nonlinear filter function on one single LFSR like in Shrinking Generator. This approach is efficient for building stream ciphers, especially in software, because bit-wise shifting LFSR is rather costly operation in software.

Geffe Cipher Geffe system is used as a key generator with LFSR's of length (17, 11, and 13), respectively, with tapping as shown in Figure 5 [8]. It belongs to nonlinear combiner generator: $F(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3$.

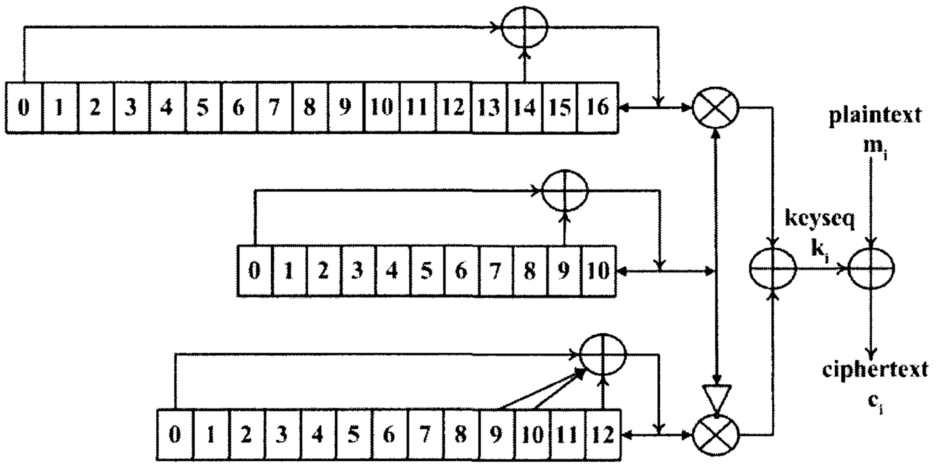

Figure 5: Structure of Geffe Stream Cipher

4.1.2 Algorithm for Attacking on Geffe Cipher

Search space The search space includes all possible initial states (IVs) $\in Z_2^{41}$, where $41 = 17 + 11 + 13$ is the sum of the length of all three LFSRs.

Cost function Since Geffe cipher is clocked in regular way, we consider using the hamming distance between observed sequence and generated sequence as the cost function.

$$\begin{aligned} cost(IV_i) &= d(gen\_seq_i, obs\_seq) \\ &= wt(gen\_seq_i \oplus obs\_seq) \\ &= \sum_{x \in Z_2^{observed\_size}} (gen\_seq_i(x) \oplus obs\_seq(x)) \end{aligned} \tag{24}$$

where gen_seq is the keystream output sequence generated by current initial state IV_i, and obs_seq is the keystream output sequence we observed. They are both in the size of $observed\_size$.

Search strategy We start to evaluate by randomly selecting one possible initial state from the search space. The neighborhood of the current state is defined by the subset of all initial states that are flipped one bit from the current state.

Experimental results Here we set parameters of the SA algorithm to be $T_0 = 10,000$, $\alpha = 0.99$, and $MIL = 1000$ for observing 1000 bits and 2000 bits of the keystream output, and to be $T_0 = 15,000$, $\alpha = 0.99$, and $MIL = 1500$ for observing 3000 bits of the keystream output. The SA search terminates when $T \leq 1$ or $cost = 0$.

From Table 15, we can conclude the following. The longer the observed keystream size is, the more efficiently we can reconstruct the initial state of LFSRs and hence to break the cipher. However, note that the longer the keystream is, the more time is needed on internal calculation.

4.2 Attack on TREYFER Cipher

We perform a series of attacking schemes to verify the capability of block ciphers, such as TREYFER and AES, against key distinguishability attack. For a cipher algorithm, if

tri	keystream size:1000bits			keystream size:2000bits			keystream size:3000bits		
val	iterations	2.20E+12	2^{41}	iterations	2.20E+12	2^{41}	iterations	2.20E+12	2^{41}
1	01709068	1.71E+06	20.7	01648735	1.65E+06	20.7	06640740	6.64E+06	22.7
2	10871709	1.09E+07	23.4	23530999	2.35E+07	24.5	11150336	1.12E+07	23.4
3	08049188	8.05E+06	22.9	07990605	7.99E+06	22.9	14057378	1.41E+07	23.7
4	58448768	5.84E+07	25.8	14362750	1.44E+07	23.8	00991414	9.91E+05	19.9
5	23631731	2.36E+07	24.5	05233592	5.23E+06	22.3	03798894	3.80E+06	21.9
6	29110129	2.91E+07	24.8	06105174	6.11E+06	22.5	02353033	2.35E+06	21.2
7	22666798	2.27E+07	24.4	08883927	8.88E+06	23.1	05175545	5.18E+06	22.3
8	34574549	3.46E+07	25.0	03387418	3.39E+06	21.7	02499694	2.50E+06	21.3
9	42006563	4.20E+07	25.3	07152750	7.15E+06	22.8	05259783	5.26E+06	22.3
10	05241524	5.24E+06	22.3	02486573	2.49E+06	21.2	00891773	8.92E+05	19.8
11	08013216	8.01E+06	22.9	04294451	4.29E+06	22.0	06722778	6.72E+06	22.7
avg	22211204	2.22E+07	24.4	07734270	7.73E+06	22.9	05412852	5.41E+06	22.4

Table 15: Efficiency of Attacking on Geffe Cipher

the local optima generated by consecutive searches highly depend on the genuine key and form patterns that are equivocally connected with each particular key, then this cipher is key distinguished. When a cipher suffers from such a weakness, we can distinguish ciphertexts generated by certain key from other keys or random process, with the cipher itself regarded as a black-box. In our research, we apply the methods to different block ciphers with different rounds, random functions, and s-boxes.

In our experiments, we firstly apply a simple local optima search strategy to recover the key, which is used to generate a set of (plaintexts, ciphertexts) pairs. Then the local optima are summed up into profiles. Finally, we attempt to measure the distance between profiles to observe the relationship between a large number of local optima and the genuine key. This main idea is originally proposed by Clark *et al.* [2]. From our results, the local optima of TREYFER with one round are highly dependent on the corresponding key, those with two rounds are to some extent dependent on the key, while there is no apparent trend with those of TREYFER with more than two rounds.

4.2.1 TREYFER Block Cipher

TREYFER was designed for environments with limited resources. The procedure of this cipher with 8-byte key and 8-byte plaintext under C is given by following pseudo code [55].

Procedure: Pseudo Code for TREYFER Implementation:

```
1. for (r=0;$r < NumRounds$; r++)
2. {
3.    text[8]=text[1];
4.    for (i=0;$i < 8$;i++)
5.       text[i+1]=rotate_1_left(text[i+1]+S-box[(key[i]+text[i])%256]);
6. }
```

In our experiments, we use two different S-boxes. One uses the first 256 primes (all modulo 256), starting with 2; the other uses the S-box used within the Advanced Encryption Standard [13] [1].

4.2.2 Algorithm for Attacking on TREYFER Cipher

In this section, we first describe the local optima search strategy and then propose a method to sum up these local optima into profiles.

Local optima search scheme Based on a set of (plaintexts, ciphertexts) pairs $\{(p_1, c_1),(p_2, c_2) \ldots (p_n, c_n)\}$ under certain key $k_{genuine}$, an all-bit-zero key k_{opt} is initialized, and the corresponding cost is calculated using following cost function:

$$cost(k_{cur}) = \sum_{i=1}^{n} d(c_i, E_{k_{cur}}(p_i)) \qquad (25)$$

where $d(c_i, c_j)$ is the Hamming distance between the ciphertexts.

The search examines each bit in the current optimum key k_{opt} from left to right. At each position bit b, a new candidate k_{cur} is obtained by flipping bit b in k_{opt}. The cost $cost(k_{cur})$ is calculated in the same way. If $cost(k_{cur}) < cost(k_{opt})$, k_{opt} is replaced by k_{cur} as best solution found so far. Otherwise, current bit flipping is resumed and next bit flipping will be tried. The search procedure is continued until $|the\_number\_of\_key\_bits| - 1$ successive non-improving bit moves under certain k_{opt}. The description of this search algorithm is shown as follows.

Procedure: Pseudo Code for Local Optima Search Scheme:

```
1. $k_{opt} \leftarrow 00\ldots0$
2. $cost(k_{opt}) \leftarrow \sum_{i=1}^{n} d(c_i, E_{k_{cur}}(p_i))$
3. $nim \leftarrow 0$
4. $b \leftarrow 0$
5. while $(nim < MAXNIM)$
6. {
7.    $k_{cur} \leftarrow flip\_bit\_b\_in\_k_{opt}$;
8.    $cost(k_{cur}) \leftarrow \sum_{i=1}^{n} d(E_{k_{cur}}(p_i), c_i)$;
9.    if $(cost(k_{cur}) < cost(k_{opt}))$
10.   {
11.      $k_{opt} \leftarrow k_{cur}$;
12.      $cost(k_{opt}) \leftarrow cost(k_{cur})$;
13.      $nim \leftarrow 0$;
14.   }
15.   else
16.      $nim \leftarrow nim + 1$;
17.   $b \leftarrow (b+1)\%|k_{opt}|$;
18. }
19. return $k_{opt}$;
```

Local optima profile scheme After the above process finishes, for certain key k, a local optima set $\{o_1, \ldots, o_T\}$ for T iterations can be obtained. To profile these local optima, correlations between every two bits are considered. The profile is given by a $|k| * |k|$ matrix:

$$P(k) = [c_{ij}]; c_{ij} = \sum_{t=1}^{T} (-1)^{o_t(i) \oplus o_t(j)} \tag{26}$$

wherein each element c_{ij} scales the level of correlation between bit i and j for the set of local optima, and $o_t(i)$, $t = 1, \ldots, T$ and $i = 1, \ldots, |k|$ denotes the ith-bit value in the local optimum o_t.

Profiles similarity scheme Given two key profiles, the similarity between each profile in the first set and any one in the second set can be measured by summing up the absolute values between elements. More formally, if $P_0(k_0) = [c_{ij}]$ and $P_1(k_1) = [c_{ij}^{'}]$ are two profiles, their distance is given by:

$$dcorr(P_0(k_0), P_1(k_1)) = \sum_{i=1}^{|k|} \sum_{j=i}^{|k|} |c_{ij} - c_{ij}^{'}| \tag{27}$$

The technique described above has been applied to TREYFER with different number of rounds and different s-boxes, and AES with different number of rounds.

4.2.3 Implementation on Attacking TREYFER

Implementation details In the case of the local search algorithm, we used a number of $|k| - 1$ successive non-improving moves (MAXNIM), and the full $|k|$ bits of the key are examined. In order to rate the distinguishing abilities of the proposed techniques on this cipher, firstly N_K different keys are randomly generated, Next, in each iteration, two sets of profiles $\{P_0(k_1), \ldots., P_0(k_{N_K})\}$ and $\{P_1(k_1), \ldots., P_1(k_{N_K})\}$ are accumulated. Then, distances between any key pairs in two profiles are calculated for every certain number of iterations. Finally, based on the distances obtained above, $rank(P(k_i))$ for each key k_i are calculated, wherein $rank(P_0(k_i))$ denotes the number of incorrect profiles in the second set are closer to $P_0(k_i)$ than the correct one $P_1(k_i)$.

We give the procedures in pseudo code as follows.

Procedure: Pseudo Code for Implementation:

```
1.  randomly selected $N_K$ different keys;
2.  for (i=0;$i < T$;i++)
3.  {
4.     for (profileno=0;$profileno < N_{PROFILE}$;profileno++)
5.     {
6.        select randomly a number $N_P$ of plaintexts;
7.        encrypt the plaintexts using each key to obtain $N_K$ sets of p/c pairs;
8.        for (j=0;$j < N_K$;j++)
9.        {
10.          search for local optimum $o_i(j)$ for key $j$;
11.          accumulate current local optimum contribution to profiles via equation 26;
12.       }
13.       if (need to record the intermediate result)
14.       {
15.          for each key pairs ($k_i$, $k_j$), calculate the $dcorr(P_0(k_i), P_1(k_j))$ based on two the
profiles attained so far;
16.          for each key $k_i$ in profile 0, calculate $rank(P_0(k_i))$;
17.       }
18.    }
19. }
```

Results for 1,2,3 and more rounds TREYFER Distances obtained in Equation 27 can be grouped into a $N_K \times N_K$ matrix, $D = [d_{ij}]$, where d_{ij} measures the distance between profiles $P_0(k_i)$ and $P_1(k_j)$. One simple way to measure the distinguishability is the following. Given a profile $P_0(k_i)$, counting the number of incorrect profiles in the second set are closer to $P_0(k_i)$ than to the correct one, that is, the rank of a profile is given by,

$$Rank(P_0(k_i)) = \#\{P_1(k_j)|(1 \leq j \leq |N_K|) \wedge (i \neq j) \wedge (d_{ij} \leq d_{ii})\} \tag{28}$$

Generally, it can be deemed that the distinguishability in statistical sense is achieved if all the ranks are less than $\frac{N_K}{2}$. From our results, the local optima of TREYFER with one round are highly dependent on the corresponding key, those with two rounds are to some extent dependent on key, while there is no apparent trend with those of TREYFER with more than two rounds. Table 16,17,18 show the results for one, two and three rounds TREYFER with two different S_Boxes. Throughout our experiments, we set $N_K = 10$, $N_P = 20$, and $MAXNIM = 63$.

From our experimentation, only TREYFER with one round apparently suffers from the key distinguishable weakness. However, it does not mean that these block ciphers are absolutely immune to such kind of attacks. Actually, there exists much potential for improvements in our experiments.

First, by using current local optima search scheme, it is obvious that the probability of certain bits in the candidate key being 1 is sharply decreased in the order of being flipped. We found in our experimentation that the bit number of 1's in the local optima is just around 10, when the key size is 128 bits in AES. The correlation between candidate solution and the genuine key is restricted by this property.

Second, current local optima profile scheme only analyzes the correlations between

sbox1	k00	k01	k02	k03	k04	k05	k06	k07	k08	k09	rank
k00	594136	4717580	5830412	5247780	4764340	4994408	4505260	5220788	5478464	4649112	0
k01	4799872	595596	3836268	5209148	2295060	3453280	1764236	2096804	2526000	2432912	0
k02	5729528	3906076	568028	6938692	3499980	3149736	4328628	4302580	3927616	2322976	0
k03	5199504	5158668	6988692	601316	5956284	5432288	5672964	5642692	4198248	6280448	0
k04	4857816	2396636	3279532	6085172	619316	2879280	2413316	2804484	3290200	2005744	0
k05	5119120	3364540	2986244	5503780	2714932	504928	3518900	3871372	2954984	2608400	0
k06	4394008	1701580	4248436	5611956	2125388	3563976	605452	1910356	3015592	2656368	0
k07	4981748	1961080	4254144	5596688	2640928	3957980	1860960	593720	2598732	2601100	0
k08	5400720	2209548	3887228	4075564	3068508	2866152	2972380	2440660	489648	3019496	0
k09	4578064	2471748	2202420	6319748	2097684	2673048	2697940	2624260	3102904	508352	0
sbox2	k00	k01	k02	k03	k04	k05	k06	k07	k08	k09	rank
k00	563240	2294452	2479024	1853132	2370504	2069192	1988704	2164228	2048156	2318232	0
k01	2227036	556272	1180028	1320256	1450556	1308588	944252	882928	1042424	1351116	0
k02	2595344	1106364	640352	1746188	1390016	1735080	1253168	1100188	1242660	1143280	0
k03	1781380	1349336	1810684	611616	2023452	1250732	1346764	1558544	1385456	1909828	0
k04	2473220	1364288	1268044	2114888	598028	1954380	1477740	1256232	1546848	1220812	0
k05	2028508	1317856	1737284	1231536	1910428	573964	1390100	1243944	1316016	2070964	0
k06	2257480	870228	1133904	1382980	1489136	1466352	667576	954468	897396	1008776	0
k07	2287332	818000	1111332	1540760	1411340	1148668	1011948	535512	1080216	1324572	0
k08	2138560	1079500	1304200	1421548	1635616	1396776	967712	1163788	480780	1242136	0
k09	2416668	1277984	1064980	1850808	1425764	2125964	1287308	1342264	1273328	578356	0

Table 16: Distances and Ranks among Key Profiles for 1-round TREYFER(30K Local Optima Profiled)

sbox1	k00	k01	k02	k03	k04	k05	k06	k07	k08	k09	rank
k00	455856	2098576	2145876	4262064	1783064	2428188	2166356	2971232	3645384	2179064	0
k01	2085500	482700	2223408	3232044	1053396	1987328	690120	1560468	2197204	1882476	0
k02	2163772	2235772	440056	2925180	1557804	940592	1897544	2310292	2578972	782500	0
k03	4386352	3378632	3038028	513464	3383424	2642636	3184332	2323872	1494112	2710928	0
k04	1837092	1148468	1511856	3317028	480772	1418896	985200	1744796	2364508	1264948	0
k05	2451352	2163856	1016172	2577560	1469304	448404	1776580	1768144	2195664	702576	0
k06	2238892	722308	2051240	3146084	1002036	1860152	453040	1415732	2199196	1658940	0
k07	3122928	1534864	2339876	2358008	1739552	1790900	1307980	420528	1249824	1834640	0
k08	3556852	2215908	2675064	1517356	2437228	2152280	2003696	1081300	554748	2230036	0
k09	2078760	1810560	948156	2969584	1145240	812852	1489292	1840624	2312360	533456	0
sbox2	k00	k01	k02	k03	k04	k05	k06	k07	k08	k09	rank
k00	378360	390224	407492	410336	484484	378872	481632	437464	440984	464132	0
k01	469856	548288	524252	465272	450668	508496	462680	508608	437640	541956	9
k02	468800	435480	466620	404560	353924	432800	393728	444000	423016	439316	8
k03	443456	473344	485852	435592	493812	460096	481816	489944	460528	508780	0
k04	423336	430688	437460	435144	475948	416112	456008	421560	420200	566204	8
k05	455984	519280	482668	437920	544492	499368	490256	471232	472552	612668	6
k06	474200	520528	555620	526712	516564	507016	577032	517080	549112	560468	9
k07	454748	467940	506504	466508	522632	476124	450604	450596	494764	499664	0
k08	484332	452476	514336	466428	498192	468500	474388	485828	442956	486688	0
k09	456264	421584	425076	376448	439388	439312	419640	469352	447800	475236	9

Table 17: Distances and Ranks among Key Profiles for 2-round TREYFER(30K Local Optima Profiled)

sbox1	k00	k01	k02	k03	k04	k05	k06	k07	k08	k09	rank
k00	4167576	5210920	5169812	5512496	5449888	5889280	5323720	4883940	5096752	5608548	0
k01	3913788	4338404	4240504	4369532	4424276	4456228	3868852	3953008	5087660	4668984	4
k02	4613796	4826524	4031016	5240996	3852588	4747660	4620260	4494440	5234220	5051096	1
k03	4797116	5396500	4885048	5651316	4975252	5086540	4946292	4890960	5555436	5566264	9
k04	4234448	4214536	4502452	5592576	4124864	4669720	4269512	4609116	4842736	4685556	0
k05	4290460	4277692	4495584	5138796	4352132	5380372	5290828	5023512	4220988	4954752	9
k06	4235664	4556584	4181684	5654520	4312640	5483968	4481024	4424516	5243840	5052164	4
k07	4023380	4556164	3880184	4958748	3624964	4945564	3920052	4205224	4578908	4292680	4
k08	4388860	4179060	4076752	4926692	4292948	4576228	4643204	4589496	4590316	5200520	6
k09	3652192	4761112	4324476	4771440	4836768	5417424	5158624	4584188	4964904	4936772	6

sbox2	k00	k01	k02	k03	k04	k05	k06	k07	k08	k09	rank
k00	4684424	4625368	4740476	5018168	4966564	4283620	4539668	4616684	3813752	4416132	6
k01	4753796	5406556	4613552	4754100	4394576	3607416	4316584	3998032	4756508	4189344	9
k02	4507156	4928116	5000824	4190852	4863616	4987832	4579568	4826424	5644580	4455936	8
k03	4254840	4269920	4651004	4752392	4637548	4188556	4342668	3870748	4207848	3955876	9
k04	4731732	4694340	4983744	4751676	4390344	4507488	4328760	4365184	4215756	4269224	4
k05	4669140	4766700	4782000	4757612	4603496	3778208	4440376	4532552	4773724	4365456	0
k06	4658408	4144280	4578132	4261576	4520012	4466740	3944292	4602172	4523976	3704772	1
k07	4724612	4588052	4995736	4719812	4572800	4224304	4805416	4979968	4829724	4942600	8
k08	4716968	4459904	4866236	5011200	4702028	4139244	4849028	4155884	5056808	4396772	9
k09	4702880	5213288	4886708	4748184	4516292	4401076	3986228	4458932	4795488	4482436	3

Table 18: Distances and Ranks among Key Profiles for 3-round TREYFER(3.0M Local Optima Profiled)

each two bits. It is necessary to develop a more sophisticated scheme which can more accurately present the correlations between keys.

Third, current profiles similarity scheme only considers the 1-norm distance measures.

Chapter 5

Conclusion and Future Work

In this thesis, we concentrated on the study of construction of special Boolean functions using heuristic methods. In the mean time, we also applied heuristic methods to break existing ciphers. Throughout this thesis,

- We have successfully constructed several examples for $(10, 2, 7, 488)$ Boolean functions. This result affirmatively answered the open problem about the existence of such functions.

- We constructed several examples for $(8, 116)$, $(10, 492)$, $(12, 2010)$ Boolean functions by simulated annealing method. These results hit the upper bound of the nonlinearity of balanced Boolean functions based on Dobbertin's conjecture in [16]. Although such functions were already constructed in the literature, they are all constructed in arithmetic ways.

- We proposed methods for attempting to construct $(8, 118)$, $(10, 494)$, and $(12, 2012)$ Boolean functions, which decreases the complexity of the search procedure. We also provided mathematic formula to construct $(12, 2012)$ functions from $(10, 494)$ function (if exists).

- Some attempts were made to break symmetric ciphers including Geffe Cipher, TREYFER cipher, and so on.

Based on the research elaborated in this thesis, further studies can be conducted in the following directions.

- Construction of Boolean functions
  - Apply different heuristic methods to attempt on constructing examples of Boolean functions to break Dobbertin's conjecture, for example, $(12, 2012)$, $(14, 8122)$ Boolean functions.
  - Explore more on the mathematic background of Boolean functions. With the increase in variable N, using pure heuristic methods to construct becomes more and more difficult. Sound mathematic background knowledge will help to design a better cost function. We can also construct Boolean functions first by arithmetic construction, and then use it as the initial state of heuristic method to achieve better solution.
  - Attempt to design better cost functions.
  - Experiment with larger N, for example, $N = 14$, $N = 16$.
- Attack on ciphers

  Although the methods we applied could not break strong ciphers, we may attempt to use different and more sophisticated local optima search scheme, local optima profile scheme, and profiles similarity scheme to employ key distinguishability and to refine the cost functions for reconstructing the initial state of stream ciphers.

Bibliography

[1] Federal Information Processing Standards Publication (FIPS 197). Advanced encryption standard (aes). Nov.26,2001.

[2] John A.Clark and Juan M.E. Tapiador. Analysis of local optima in block ciphers. *http://eprint.iacr.org/2007/.*

[3] C. Adams. Constructing symmetric ciphers using the cast design procedure. *Designs, Codes and Cryptography*, 12(3):283–316, 1997.

[4] C. Adams and S. Tavares. Generating and counting binary bent sequences. *IEEE Transactions on Information Theory*, 36(5):1170–1173, 1999.

[5] A.Dimovski and D.Gligoroski. Generating highly nonlinear boolean functions using a genetic algorithm. *Proc. IEEE 6th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Service, 2003. TELSIKS 2003*, pages 604–607, 2003.

[6] Paul C. van Oorschot Alfred J. Menezes and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[7] R. J. Anderson. Faster attack on certain stream ciphers. *ELECTRONICS LETTERS*, 29(15):1322-1323, 1993.

[8] Musbah J. Aqel, Ziad A. Alqadi, and Ibraheim M. El Emary. Analysis of stream cipher security algorithm. *Journal of Information and Computing Science,England, UK*, 2(4):288-298, 2007.

[9] Anna Bernasconi and Bruno Codenotti. Spectral analysis of boolean functions as a graph eigenvalue problem. *IEEE Trans. Computers*, 48(3):345–351, 1997.

[10] Anna Bernasconi and Bruno Codenotti. A characterization of bent functions in thers of strongly regular graphs. *IEEE Trans. Computers*, 50(9):984–985, 1999.

[11] C. Carlet. A construction of bent functions. *Finite Fields and Applications, London Mathmatical Society ,Lecture Series 233,Cambridge University Press.*

[12] J. Clark, J. Jacob, S. Stepney, S. Maitra, and W. Millan. Evolving boolean functions satisfying multiple criteria. *In INDOCRYPT 2002 in Lecture Notes in Computer Science Springer-Verlag*, 2551:246–259, 2002.

[13] Joan Daemen and Vincent Rijmen. The block cipher rijndael. In *CARDIS '98: Proceedings of the The International Conference on Smart Card Research and Applications*, pages 277–284. Springer-Verlag, 2000.

[14] Deepak Dalai. Cryptographic properties of boolean functions and s-boxes. *PHD thesis , Katholieke Universiteit Leuven (Belgium)*, 2006.

[15] Deepak Dalai. On some necessary conditions of boolean functions to resist algebraic attacks. *PHD thesis , Indian Statistical Institute (Kolkata, India)*, 2006.

[16] H. Dobbertin. Construction of bent functions and balanced boolean functions with high nonlinearity. *In Fast Software Encryption (Workshop on Cryptographic Algorithms, Leuven 1994 (1995), no. 1008 in Lecture Notes in Computer Science, Springer-Verlag.*

[17] Ali Doğanoksoy et al. Constructions of highly nonlinear balanced boolean functions. *I. Ulusal Kriptoloji Sempozyumu*, 2005.

[18] M. Gilli and P. Winker. *Heuristic Optimization Methods in Econometrics.* 2007.

[19] Fred Glover and Gary A. Kochenberger. *Handbook of Metaheuristics.* Springer, 2003.

[20] J.Clark, J.Jacob, S.Maitra, and P.Stanica. Almost boolean functions: The design of boolean functions by spectral inversion. *Computational Intelligence*, 20:450-462, 2004.

[21] R.A.Scholtz J.D.Olsen and L.R.Welch. Bent-function sequences. *IEEE Transactions on Information Theory*, IT-28(6).

[22] J.F.Dillon. Elementary hadamard difference sets. *Proceedings of the Sixth South-eastern Conference on Combinatorics , Graph Theory and Computing, F. Hoffman et al.(Eds), Utilitas Math.*

[23] Peter Thompson Julian F. Miller. Restricted evaluation genetic algorithms with tabu search for optimising boolean functions as multi-level and-exor networks. *In Proceedings of Evolutionary Computing, AISB Workshop'1996*, pages 85–101, 1996.

[24] Robin J.Wilson. *introduction to Graph Theory (second Edition).* Academic Press, 1979.

[25] Seçlk Kavut and Melek D. Yucel. Generalized rotation symmetric and dihedral symmetric boolean functions - 9 variable boolean functions with nonlinearity 242. *eprint.iacr.org/2007/308*, 2007.

[26] Selçuk Kavut, Subhamoy Maitra, and Melek D. Yucel. Search for boolean functions with excellent profiles in the rotation symmetric class. *IEEE Transactions on Information Theory*, 53(5), 2007.

[27] Selçuk Kavut, Melek D. Yucel, and Subhamoy Maitra. Construction of resilient functions by the concatenation of boolean functions having nonintersecting walsh spectra. *BFCA'07*, 2007.

[28] Khoongming KHOO and Guang GONG. New construction for balanced boolean functions with very high nonlinearity. *IEICE TRANS. FUNDAMENTALS*, E90-A:29–35, 2007.

[29] F.J. MacWilliams and N.J.A Sloane. The theory of error correcting codes. *North-Holland Publishing Company, Amsterdam*, 1978.

[30] S. Maitra and P. Sarkar. New directions in design of resilient boolean functions. *Advances in Cryptology - CRYPTOŠ00. , number 1880 in Lecture Notes in Lecture Notes in Computer Science*, pages 515–532, 2000.

[31] Subhamoy Maitra, Sumanta Sarkar, and Deepak K. Dalai. On dihedral group invariant boolean functions. *International Workshop on Boolean Functions: Cryptography and Applications*, 2007.

[32] Soumen Maity and Subhamoy Maitra. Minimum distance between bent and 1-resilient boolean functions. *Workshop on Fast Software Encryption, FSE 2004, Lecture Notes in Computer Science, Springer, Berlin*, 3017:143–160, 2004.

[33] M. Matsui. Linear cryptanalysis method for des cipher. *Advances in Cryptology - EUROCRYPT'93 of Lecture Notes in Computer Science,Springer-Verlag*, 765:386–397, 1993.

[34] A. Maximov. Some words on cryptanalysis of stream ciphers. *PhD thesis, Lund University, Lund, Sweden*, 2006.

[35] W. Millan, A. Clark, and E Dawson. Smart hill climbing finds better boolean functions. *En Workshop on Selected Areas in Cryptology (SAC97)*, pages 50–63, 1997.

[36] M. R. A. H. Teller N. C. Metropolis, A. W. Rosenbluth and E. Teller. Equation of state calculation by fast computing machines. *J. Chem. Phys.*, 21:1087–1092, 1953.

[37] J. Pieprzyk and C. X. Qu. Fast hashing and rotation-symmetric functions. *Journal of Universal Computer Science*, 5(1):20-31, 1999.

[38] B. Preneel. Analysis and design of cryptographic hash functions. *PHD thesis , University of Leuven*, 1994.

[39] R.A.Scholtz P.V.Kumar and L.R.Welch. Generalized bent functions and their properties. *J. Combinatorial theory*, 40(A):90–107, 1985.

[40] R.L.McFarland. A family of difference sets in non-cyclic groups. *J. Combinatorial Theory*, A 15:1–10, 1973.

[41] O. S . Rothaus. On bent functions. *J. Combinatorial theory*, 20(A):300–305, 1976.

[42] R.Yarlagadda and J.E.Hershey. Analysis and synthesis of bent sequences. *Computers and Digital Techniques, IEE Proceedings E*, 136(2):112–123, 1989.

[43] Z. Saber, M. F. Uddin, and Amr Youssef. On the existence of $(9, 3, 5, 240)$ resilient functions. *IEEE Transactions on Information Theory*, 52:2269-2270, 2006.

[44] P. Sarkar and S. Maitra. Nonlinearity bounds and constructions of resilient boolean functions. *Proc. of Cypto 2000, LNCS 1880, Springer-Verlag*, pages 516–533, 2000.

[45] P. Sarkar and S Maitra. Construction of nonlinear resilient boolean functions using “small” affine functions. *IEEE Transactions on Information Theory*, 50(9), 2004.

[46] T. Siegenthaler. Correlation immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, IT-30:776-780, 1984.

[47] P. Stănică and S. Maitra. Rotation symmetric boolean functions Ű count and cryptographic properties. *R.C. Bose Centenary Symposium on Discrete Mathematics and Applications, Electronic Notes in Discrete Mathematics, Elsevier*, 15:139-145, 2002.

[48] P. Stănică and S. Maitra. A constructive count of rotation symmetric functions. *Inf. Process. Lett.*, 88:299-304, 2003.

[49] P. Stănică, S. Maitra, and J. Clark. Results on rotation symmetric bent and correlation immune boolean functions. *in Proc. Fast Software Encryption Workshop (FSE 2004), New Delhi, India (Lecture Notes in Computer Science)*, 3017:161-177, 2004.

[50] Pantelimon Stănică. Graph eigenvalues and walsh spectrum of boolean functions. *Electronic Journal of Combinatorial Number Theory*, 7(2), 2007.

[51] Mohammad Faisal Uddin. Artificial life techniques for cryptology. *M.ASc thesis , Concordia University*, 2006.

[52] P.J. van Laarhoven and E.H. Aarts. *Simulated Annealing: Theory and Applications (Mathematics and Its Applications)*. Springer-Verlag, 1987.

[53] G. Xiao and J. L. Massey. A spectral characterization of correlation-immune combining functions. *IEEE Transactions on Information Theory*, IT-34(3):569-571, 1988.

[54] X.M.Zhang and Y.Zheng. Gac - the criterion for global avalanche characteristics of cryptographic functions. *Journal of Universal Computer Science*, 1(5):316-333, 1995.

[55] G. Yuval. Reinventing the travois: Encryption/mac in 30 rom bytes. *Proc. of the 4th International Workshop on Fast Software Encryption (FSE'97)*, 2(4):288-298:205–209, 1997.