Artificial Life Techniques for Cryptology

Mohammad Faisal Uddin

A Thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science
Concordia University
Montreal, Quebec, Canada

July 2006

Your file Votre référence
ISBN: 978-0-494-28930-3
Our file Notre référence
ISBN: 978-0-494-28930-3

Abstract

Artificial Life Techniques for Cryptology

Mohammad Faisal Uddin

In this thesis, we investigate the applications of two swarm-inspired artificial life optimization techniques in cryptology. In particular, we investigate the use of both Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) for automated cryptanalysis of simple classical substitution ciphers. We also use PSO to construct Boolean functions with some desirable cryptographic properties.

Both ACO and PSO based attacks proved to be effective for the cryptanalysis of simple substitution ciphers encoded with various sets of encoding keys. Purely uni-gram and bi-gram statistics are used for solving this problem.

Boolean functions are vital components of symmetric-key ciphers such as block ciphers, stream ciphers and hash functions. When used in cipher systems, Boolean functions should satisfy several cryptographic properties such as balance, high nonlinearity, resiliency and high algebraic degree. Using PSO, with an unorthodox approach of spectral inversion, we are able to construct Boolean functions that achieve the maximum possible nonlinearity (Bent function) and several other important resilient functions. In fact, we were able to construct, for the first time, a 9-variable Boolean function with nonlinearity 240, algebraic degree 5, and resiliency degree 3. This construction affirmatively answers the open problem about the existence of such functions.

Acknowledgments

First of all, I would like to express my earnest gratitude to my supervisor Dr. Amr M. Youssef for his constant support, guidance and enduring patience.

I'd also like to express my appreciation to all the faculty and people at Concordia University, especially those at Concordia Institute for Information Systems Engineering, who contributed to my success one way or the other.

Special thanks goes to Mr. Ziad Saber, my research partner, whose help and patience made my work a lot easier.

I would like to thank my parents for their unconditional love, trust and encouragements. It would have been impossible for me to accomplish this endeavor without their support.

Finally, thanks to my loving wife for her constant love and care.

Table of Contents

List of Figures

List of Tables

List of Notations

Z_2	The set of binary numbers
Z_2^n	The set of binary n tuples
$\oplus$	XOR operation
x	A variable in Z_2^n or GF(2)
$wt(f)$	The Hamming weight of f
$d(f,g)$	The Hamming distance between two Boolean function f and g
CI	Correlation immunity
m	Order of resiliency
$ANFD$	Algebraic normal form degree
NL	The nonlinearity of f
$F(\omega)$	Walsh Hadamard Transform
ρ	Pheromone evaporation rate
α, β	Weighting factor on pheromone trail and a priori distribution of choice for ant algorithm
c_1, c_2	Learning factors for PSO algorithm
$Ppbest$	Particle best position for PSO algorithm
$Pgbest$	Global best position for PSO algorithm

Chapter 1

Introduction

1.1 Cryptography and Cryptanalysis

The prosperity of mankind owes a great deal to the skill of communications, and a major aspect in this skill is the capability of communicating in writing. From the most primitive days of writing, individuals felt the need to limit their information to a restricted group of people. So the individuals developed ideas by means of which their communications could be made unintelligible to those who had not been provided with the special information needed to decode that message. The technique of hiding the meaning of the message started a new area of study known as *cryptography.* As people learnt to hide messages, it didn't take others a long time to realize the advantages to be gained from intercepting that secret information, and that led to a continuous battle between the code makers and code breakers. *Cryptanalysis* is the study of breaking the coded message, and *cryptology* which was for a long time used as a synonym of cryptography has recently been changed. In the last half century, the modern definition of cryptology became widely adopted. This modern definition classifies cryptology as a general term that encompasses cryptography and cryptanalysis as its main component classes.

Traditionally, cryptography only deals with the encryption process of the original messages, and cryptanalysis deals with the decryption of that encrypted messages.

However, both cryptography and cryptanalysis are totally related. The developments in one field always reflect on the other field because when the methods of cryptography improve, the need for better methods of cryptanalysis grows. Conversely, as cryptanalysts become more skillful in breaking messages, cryptographers feel the need for better ways to encipher them.

1.2 A Brief History

The word cryptography came from the Greek *kryptós*, 'hidden', and *graphein* 'write'; it is the art and science of making communications unintelligible to all except the intended recipient. Again, the word Cryptanalysis, from the Greek *kryptós*, 'hidden', and *analýein*, 'to loosen', is the art and science of breaking the secrets hidden by cryptography.

Cryptography is one of the oldest fields of technical study we can find records of. The records go back at least 4,000 years [1]. Since writing was developed and examples survive in stone inscriptions and papyruses showing that many ancient civilisations, including the Egyptians, Hebrews and Assyrians, developed cryptographic systems.

The first recorded use of cryptography was done by the Spartans as early as 400 B.C. They employed a cipher device called a "scytale" to send secret communications between military commanders. The scytale consisted of a tapered baton around which was wrapped a piece of parchment inscribed with the message. Once unwrapped the parchment appeared to contain an incomprehensible set of letters, however when wrapped around another baton of identical size the original text appears [2].

Julius Caesar used a system of cryptography (the 'Caesar Cipher') which shifted each letter 3 places further through the alphabet (e.g. *CIPHER* to *FLSKHU*). This technique worked for quite long before people learned how to read it.

In 1379 Gabriel de Lavinde made cryptology a more formally understood science when he published '*Circa*', the first European manual on cryptology.

In the 1600's Cardinal Richelieu created a card with holes in it and used it to write a secret message on a paper through the holes. After finishing writing the secret message, he used to remove the card and fill up the blanks of the paper by writing letter in a way that the whole letter seems like a normal letter. So people who don't have the card with the holes can not read the secret message [3].

Early in the 15th century, an Arabic author, Qalqashandi, wrote down a technique for solving ciphers using the average frequency of each letter of the language [1] [4].

In 1948, Shannon published "A Communications Theory of Secrecy Systems" [5]. Shannon was one of the first modern cryptographers to attribute advanced mathematical techniques to the science of ciphers. Shannon's analysis demonstrates several important features of the statistical nature of language that makes the solution to nearly all previous ciphers very straightforward. Perhaps the most important result of Shannon's famous paper is the development of a measure of cryptographic strength called the 'unicity distance'. The unicity distance is a number that indicates the amount of ciphertext required in order to uniquely determine the plaintext of a message. It is a function of the length of the key used to encipher the message and the statistical nature of the plaintext language.

For further information about the history of cryptology, the reader is referred to [1].

1.3 Cryptographic Goals

Of all the information security objectives, the following four form a framework upon which the others will be derived: (1) privacy or confidentiality, (2) data integrity, (3) authentication and (4) non-repudiation [6].

1.3.1 Privacy or Confidentiality

The service used to keep the content of information from all but those authorized to have it. Secrecy is a term synonymous with confidentiality and privacy. There are numerous approaches to providing confidentiality, ranging from physical protection to mathematical algorithms which render data unintelligible.

1.3.2 Data integrity

Data integrity addresses the unauthorized alteration of data. To assure data integrity, one should have the ability to detect data manipulation by unauthorized parties. Data manipulation includes insertion, deletion, and substitution.

1.3.3 Authentication

This service relates to identification. This function applies to both entities and information itself. Two parties entering into communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons this aspect of cryptography is usually subdivided into two major classes: entity authentication and data origin authentication.

Data origin authentication implicitly provides data integrity (for if a message is modified, the source has changed).

1.3.4 Non-repudiation

This service prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. For example, one entity may authorize the purchase of property by another entity and later deny such authorization was granted. A procedure involving a trusted third party is needed to resolve the dispute.

A fundamental goal of cryptography is to adequately address these four areas in both theory and practice.

1.4 A Secure Communications System

In this section, we introduce some of the basic cryptographic terminologies. A message is called plaintext. Encryption is the process of disguising a message in such a way as to hide its substance. An encrypted message is called ciphertext; and the process of turning ciphertext back into plaintext is called decryption. As shown in Figure 1.1, plaintext is encrypted with the help of encryption algorithm by the sender and going to the receiving end through an insecure channel and the receiver is turning back the ciphertext back into the original plaintext by decrypting the ciphertext using decryption algorithm. But while ciphertext is going through the insecure channel, cryptanalyst might intercept the ciphertext and if the encryption algorithm is not that strong he might be able to break the ciphertext and he can listen, insert, delete or modify the message. So a good algorithm is necessary for a secure communications system.

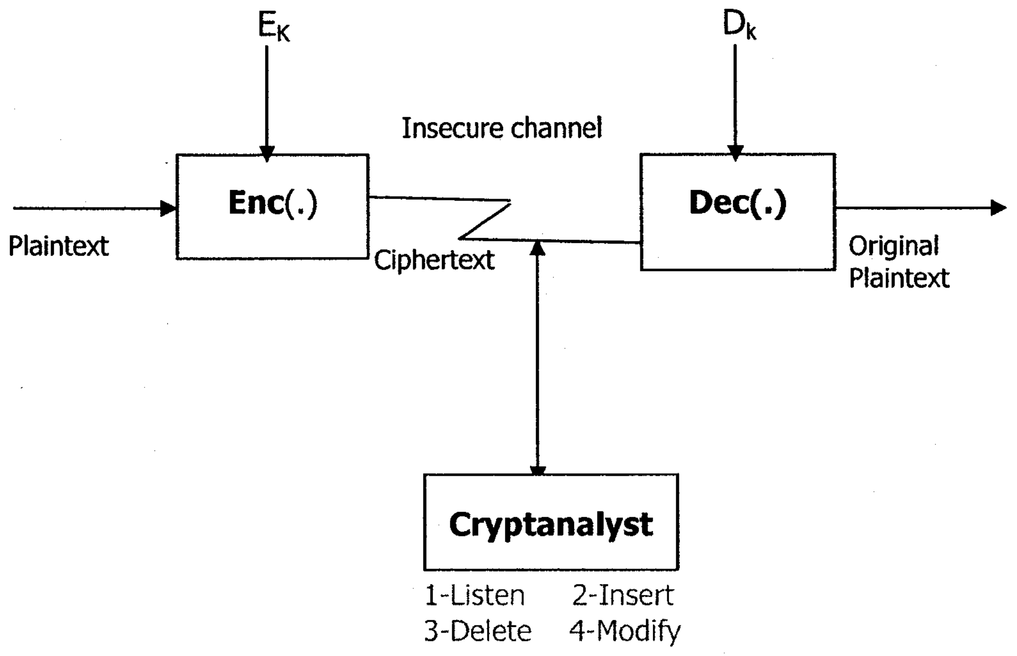

Figure 1.1: A typical secure communications system

1.4.1 Algorithm and Keys

A cryptographic algorithm is called a cipher, which is the mathematical function used for encryption and decryption.

Sometimes the security of an algorithm is based on keeping the algorithm works a secret. This type of algorithm is called restricted algorithm. But these types of algorithms are inadequate by today's standard because of the fact that if a user leaves the group of users, other must switch to a new algorithm. These types of algorithms allow no quality control or standardization. Every group of users must have their own unique algorithm. They can not buy or use off-the-shelf hardware or software products because others can buy the product and learn the algorithm. They need to write their own algorithms and implementations.

To overcome these problems modern cryptography uses a key, denoted by K. This key can be any one of a large number of values. The range of possible values of the key is called the keyspace. Both encryption and decryption operation use this key.

The plaintext is denoted by M and ciphertext is denoted by C. The encryption function dependent on the key K is E_K operates on M to produce C, i.e. $E_K(M) = C$.

In the reverse process, the decryption function dependent on the same key K is D_K operates on C to produce M, i.e. $D_K(C) = M$.

Since the whole point of encrypting and decrypting a message is to recover the original plaintext, the following identity must hold,

$$D_K(E_K(M)) = M.$$

The best thing about these algorithms is that all of the security in these algorithms is based on the keys; none is based on the details of the algorithm. So it doesn't matter if anybody knows the algorithm, as long the particular key used by the user is hidden the message is safe.

As the keys play such an important role in cryptosystem, a necessary condition for an encryption scheme is to secure the key. So, the keyspace must be large enough to preclude exhaustive search. It should be noted here that securing key by increasing keyspace is a necessary step but sometimes this is not enough.

1.5 Cryptographic Techniques

Cryptographic techniques are typically divided into two types: symmetric-key and public-key (Figure 1.2).

1.5.1 Symmetric-Key Algorithms

A cryptographic system is said to be a symmetric-key system when encryption key can be calculated from decryption key and vice-versa. In most of the cases the encryption key and the decryption key are the same [7]. The main drawback is that the sender and the receiver should somehow exchange the key in a secure way.

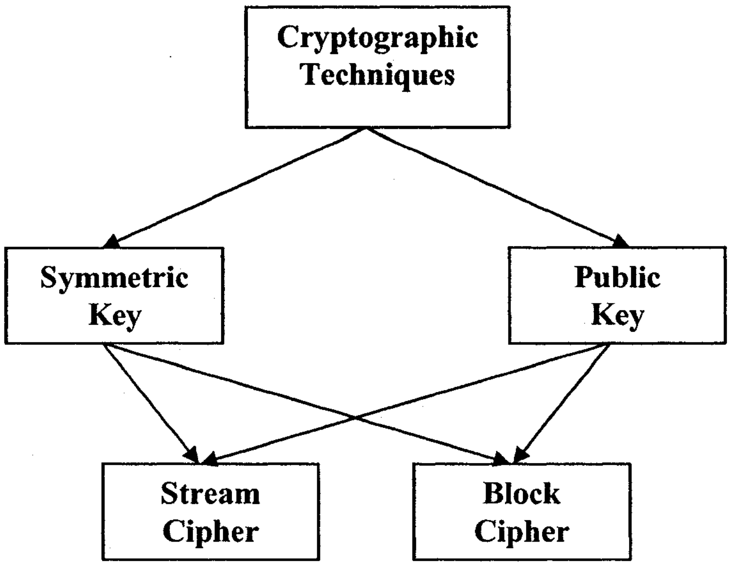

Figure 1.2: General classification of encryption systems.

1.5.2 Public-Key Algorithms

A cryptographic system is said to be a public-key system if the key used for encryption is different from the key used for decryption. Furthermore, the decryption key can not be calculated from the encryption key [7]. A public key algorithm uses two keys: a public key and a private key. The public key is known to everyone and the private or secret key is known only to the recipient of the message. An important element to the public key system is that the public and private keys are related in such a way that only the public key can be used to encrypt messages and only the corresponding private key can be used to decrypt them.

Both symmetric-key and public-key algorithms can be divided into two categories: stream cipher and block cipher.

1.5.3 Stream Cipher

A stream cipher produces a pseudo-random sequence of bits which are exclusive-OR'ed with the plaintext to produce the ciphertext. In stream cipher, plaintext digits are encrypted one at a time. Many stream ciphers make use of the linear feedback shift register (LFSR). Figure 1.3 illustrates a linear feedback shift register defined by the primitive polynomial $1 + x^3 + x^5 + x^6$.

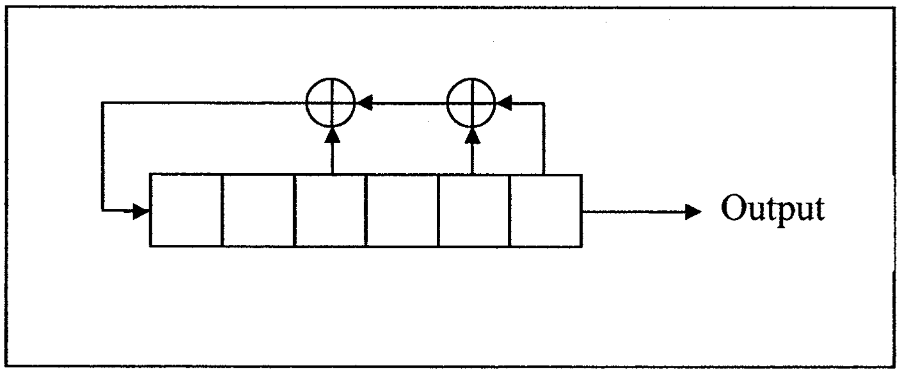

Figure 1.3: An example for a 6-bit linear feedback shift register

A periodic LFSR is defined by a feedback polynomial of degree L, called the length 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 2^L-1 and is called an *m*-sequence. One particular example of LFSR-based stream ciphers is the nonlinear combiner (Figure 1.4) which combines the output of k LFSR's using a nonlinear

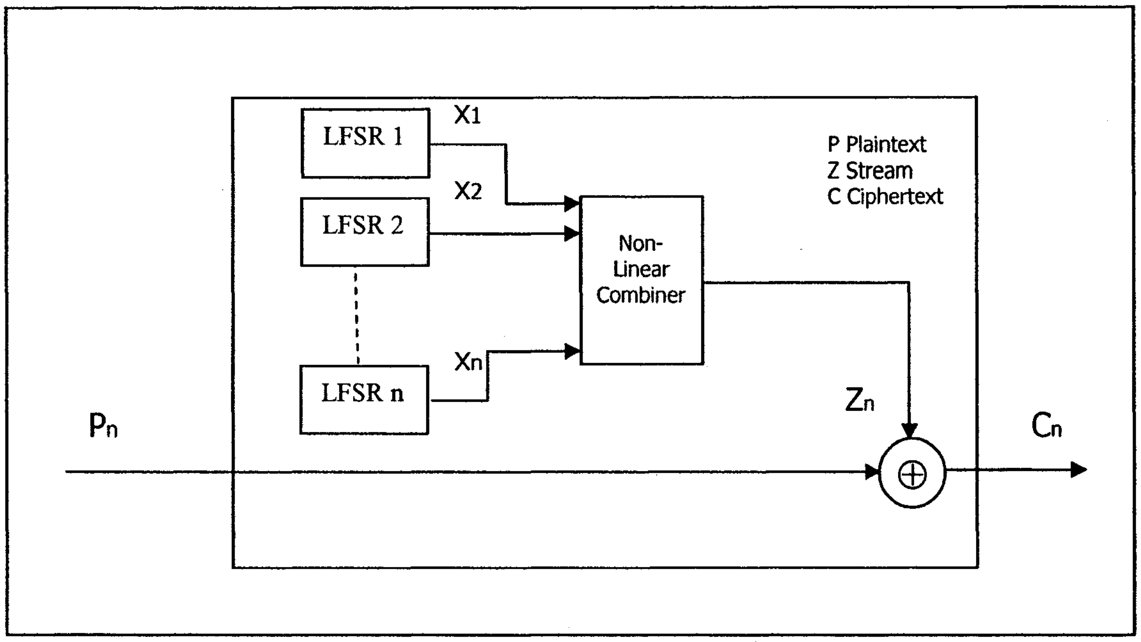

Figure 1.4: LFSR-based stream cipher

Boolean function to obtain the keystream. Thus, the combiner requires a nonlinear Boolean function of *k* inputs. The keying material for this cipher is generally the initial contents of the LFSR's. In some cases the feedback polynomials are assumed to be public knowledge, along with the combining function. A cryptographic weakness with some LFSR-based stream ciphers is that the output sequence from the LFSR is correlated to the output keystream sequence of the generator. Hence, the design of these non-linear combining Boolean functions is of great importance in stream cipher in order to stand against different cryptanalytic attacks. In other words, if the nonlinear combining function is not properly designed, then an attacker may be able reconstruct the keystream sequence.

1.5.4 Block Ciphers

A block cipher is an encryption scheme which breaks up the plaintext messages to be transmitted into strings called blocks of a fixed length L, then encrypts one block at a time. When encrypting, a block cipher takes L-bit block of plaintext as input, and output a corresponding L-bit block of ciphertext. The exact transformation is controlled by the secret key. Decryption is similar; the decryption algorithm takes an L-bit block of ciphertext together with the secret key, and yields the original L-bit block of plaintext.

Block ciphers can be contrasted with stream ciphers; a stream cipher operates on individual digits one at a time. The distinction between the two types is not always clear-cut; a block cipher, when used in certain modes of operation, acts effectively as a stream cipher (for example, output feedback mode (*OFB*)).

1.6 Outline of the Thesis

Chapter two gives some of the mathematical background and the necessary definitions required for the Boolean functions construction.

In chapter three, two swarm inspired optimization techniques, Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) are described in details.

In chapter four, we show how both PSO and ACO techniques are successfully applied for the cryptanalysis of simple substitution ciphers.

In chapter five, we use spectral inversion technique with swarm inspired optimizer PSO to construct several Boolean functions of different cryptographic properties (bent and resilient functions).

Finally, in chapter six, a summary of the results and directions for future works are given.

The results presented in chapter four appeared in [8][9]. A part of the results presented in chapter five was published in [10][11].

Chapter 2

Boolean Functions

2.1 Introduction

A Boolean function is a $\{0,1\}$-valued function defined on the set Z_2^n of all binary words of a given length *n*. Boolean functions are used in several different types of cryptographic applications, including the design of block ciphers, stream ciphers, and hash function [6].

The most basic representation of a Boolean function is by its binary truth table. The binary truth table of a Boolean function of n variables is denoted $f(x)$, where $f(x) \in \{0,1\}$ and $x = \{x_1, x_2, ..., x_n\}$, $x_i \in \{0,1\}$, $i = 1,....,n$. The truth table contains 2^n elements corresponding to all possible combinations of the n binary inputs.

Another representation of a Boolean function is over the set $\{1,-1\}$. The polarity truth table of a Boolean function is denoted $\hat{f}(x)$, where $f(x) \in \{0,1\}$ and $\hat{f}(x) = (-1)^{f(x)} = 1 - 2f(x)$. It is also important to note that XOR over $\{0,1\}$ is equivalent to real multiplication over $\{1,-1\}$. Thus if, $h(x) = f(x) \oplus g(x)$ then $\hat{h}(x) = \hat{f}(x).\hat{g}(x)$.

Example: Let $f(x_1, x_2) = x_2 \oplus x_1 x_2$, then it can be represented by the binary truth table as $[f(0,0)\, f(1,0)\, f(0,1)\, f(1,1)]$=[0 0 1 0] and by the polarity truth table as [1 1 -1 1].

Two fundamental properties of Boolean functions are Hamming weight and Hamming distance.

Hamming weight

The Hamming weight of a Boolean function is the number of ones in the binary truth table or equivalently the number of -1 in the polarity truth table [12]. So the Hamming weight of a Boolean function $wt(f)$ is given by

$$wt(f) = \sum_{x \in Z_2^n} f(x) = \frac{1}{2}(2^n - \sum_{x \in Z_2^n} \hat{f}(x))$$

Hamming distance

The Hamming distance between two Boolean functions $d(f,g)$ is the number of positions in which their truth tables differ. It can be calculated from either the binary truth table or the polarity truth table as follows:

$$d(f,g) = \sum_{x \in Z_2^n} (f(x) \oplus g(x)) = \frac{1}{2}(2^n - \sum_{x \in Z_2^n} \hat{f}(x).\hat{g}(x))$$

A linear function, $L_\omega(x)$, selected by $\omega \in Z_2^n$ is defined as

$$L_\omega(x) = \omega \cdot x = \omega_1 x_1 \oplus \omega_2 x_2 \oplus .....\omega_n x_n .$$

An affine function is one of the form:

$$A_\omega(x) = \omega \cdot x \oplus c ,$$

where $c \in Z_2$.

The Hamming distance to linear functions is an important cryptographic property. Ciphers that employ nearly linear functions can be broken easily by a variety of methods such as linear cryptanalysis [13]. Thus the minimum distance to the set of affine functions is an important indicator of the cryptographic strength of Boolean functions.

2.2 Algebraic Normal Form

The Algebraic Normal Form (*ANF*) describes a Boolean function in terms of an XOR sum of logical AND products of sub-sets of input variables. Any Boolean function $f(x)$ of n variables admits a unique (*ANF*) and can be written as:

$$f(x) = a_0 \bigoplus_{i=1}^{i=n} a_i x_i \bigoplus_{1 \le i \ne j \le n} a_{ij} x_i x_j \oplus \ldots \oplus a_{12\ldots n} x_1 x_2 \ldots x_n .$$

The *ANF* can be derived from the binary truth table in a binary matrix transformation, the Algebraic Normal Form Transformation (*ANFT*). The *ANFT* matrix is its own inverse, so the binary truth table may also be obtained from the *ANF* using the same *ANFT* operation. The most important cryptographic property related to the *ANF* is the algebraic normal form degree of a Boolean function, which is equal to the number of variables in the highest order product term with nonzero coefficient (*ANFD*). We refer to functions of degree two as quadratic, and function of order three as cubic. Affine functions are those Boolean functions of degree at most one.

Example: Let $f(x) =$ [0101101010100110], then *f(x)* expressed in its algebraic normal form is given by:

$$f(x) = x_1 \oplus x_2 \oplus x_1 x_2 x_3 \oplus x_4$$

And its algebraic normal form degree is 3.

Example: Let $f(x) =$ [0000111111110000], then *f(x)* expressed in its algebraic normal form is given by:

$$f(x) = x_3 \oplus x_4$$

And its algebraic normal form degree is 1. Thus f is a linear function.

2.3 Walsh-Hadamard Transform (WHT)

The Walsh-Hadamard Transform expresses a Boolean function in terms of its correlation with all linear functions. Several important cryptographic properties are expressed directly in terms of Walsh transform values.

The *WHT* of a Boolean function is calculated from the polarity truth table as:

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

The Walsh transform is sometimes called the spectral distribution or the spectrum of a Boolean function.

Example: Let $f(x) = [0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1]$, then the spectrum of the Boolean function f is given by

$$F(\omega) = [0\ 0\ 0\ 0\ 4\ \text{-}4\ 4\ 12\ 0\ 0\ 0\ 0\ \text{-}4\ 4\ \text{-}4\ 4].$$

2.4 Cryptographic Criterions for Boolean Functions

2.4.1 Balancedness

For cryptographic Boolean functions, it is usually desired that there are equal number of 0's and 1's in the binary truth table. When this is the case, the function is said to be balanced.

Example: Let $f(x) = [00011110]$. Since the number of 0's in f is equal to the number of 1's, *then* $f(x)$ is said to be balanced.

The balancedness of Boolean function can be defined from Walsh transform of f. *If* $F(0) = 0$ then the function is balanced.

Example: The Walsh transform of the function $f(x) = [0110100100101101]$, is $F(\omega) = [0\ 0\ 0\ 0\ 4\ \text{-}4\ 4\ 12\ 0\ 0\ 0\ 0\ \text{-}4\ 4\ \text{-}4\ 4]$, and as $F(0) = 0$ the function is balanced.

2.4.2 Nonlinearity (*NL*)

The nonlinearity (*NL*) of a Boolean function is defined as the minimum Hamming distance between f and the set of affine functions [14]. Complementing a Boolean function's binary truth table does not change the nonlinearity, so the magnitude of the correlation to all linear functions, of which there are 2^n, is to be considered. The Hamming distance between a pair of functions can be determined by evaluating both functions for all inputs and counting the disagreements. This process has complexity $O(2^{2n})$. It follows that determining the nonlinearity in this naive fashion will require $O(2^{2n})$ function evaluations, which is infeasible even for small n.

Example: Let n=2, $f(x) = x_1 \oplus x_2 \oplus x_1x_2$ and $a_i \in Z_2$. Then any affine function can be expressed as

$$A_i(x) = a_0 \oplus a_1x_1 \oplus a_2x_2 .$$

By taking all the combinations of $a_i's$, we can generate all the affine functions for $n = 2$ and they are represented in Table 2.1.

To find the nonlinearity of f we calculate the distance between f and all affine functions that are presented in Table 2.1. The minimum Hamming distance is the *NL* of f.

	Affine functions							
f	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
0	0	0	0	0	1	1	1	1
1	0	1	0	1	1	0	1	0
1	0	0	1	1	1	1	0	0
1	0	1	1	0	1	0	0	1
d(f,Ai)	3	1	1	1	1	3	3	3

Table 2.1: Distance between f and all affine functions.

$$d_{min}=1 \Rightarrow NL=1.$$

The nonlinearity of f can be obtained from the Walsh transform of f as follows:

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

Using the fast Walsh transform the complexity of calculating the nonlinearity is reduced to $O(n2^n)$. Clearly in order to increase the nonlinearity, $\max_{\omega \in Z_2^n} F(\omega)$ should be decreased. Note that a function is uncorrelated with linear function $L_\omega(x)$ when $F(\omega) = 0$. For cryptography, it would be desirable to find Boolean functions which have all *WHT* values equal to zero, since such functions have no correlation to any affine functions. However, it is known [15] that such functions do not exist. A well known theorem, widely attributed to Parseval [16], states that the sum of the squares of the *WHT* values is the same constant for every Boolean function,

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

Thus a tradeoff exists in minimizing affine correlation. When a function is altered so that its correlation to some affine function is reduced, the correlation to some other affine function is increased.

Example: Let $f(x) = [0111]$. Then the spectrum of the Boolean function is given by [-2 2 2 2]. Thus $\max_{\omega \in Z_2^n} F(\omega) = 2$ and hence the *NL* is equal to 1.

Example: Let $f = [1100]$. Then the spectrum of the Boolean function is given by [0 0 -4 0]. Thus $\max_{\omega \in Z_2^n} |F(\omega)| = 4$ and hence the *NL* is equal to 0.

2.4.3 Correlation Immunity (*CI*)

A Boolean function is said to be correlation immune of order *m* if the distribution probability of its output is unaltered when any *m* of its input are fixed [17].

So the function $f(x_1, x_2, ...x_n)$ is degree m correlation immune if,

$$\Pr(f(x) = 1 \mid x_{i_1} = a_{i_1}, x_{i_2} = a_{i_2}, ..., x_{i_m} = a_{i_m}) = \Pr(f(x) = 1)$$

where, $i_j \in \{1,2,....n\}$ and $j \in \{1,2,....m\}$

Example: Consider the linear function $f(x) = x_1 \oplus x_2 \oplus x_3$

x_1	x_2	x_3	*f(x)*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Table 2.2: Truth table for function $f(x) = x_1 \oplus x_2 \oplus x_3$

If we fix x_1 to 0, then we have:

$$\Pr(Z = 1 \mid x_1 = 0) = \Pr(f(x) = 1) = \frac{1}{2} .$$

Similarly if we fix x_1 to 1, then we have:

$$\Pr(Z = 1 \mid x_1 = 1) = \Pr(f(x) = 1) = \frac{1}{2}\ .$$

Similarly, fixing x_2 and x_3 individually with 0 and 1 the distribution probability of the output remains unaltered. Fixing $x_1\, x_2$, $x_2\, x_3$ and $x_3\, x_1$ with 0 and 1 also do not change the distribution probability of the output. But if $x_1\, x_2\, x_{31}$ all are fixed to 0 or 1 the distribution probability of the output changes. So the function is a degree 2 correlation immune function.

2.4.4 Resiliency

A Boolean function is said to be resilient of order *m* [16] if it is correlation immune of order *m* and it is also balanced.

There are some other cryptographic criterions for Boolean functions, such as Autocorrelation, completeness, output Bit Independence Criterion (*BIC*), Strict Avalanche Criterion (*SAC*), higher Order SAC and Propagation Criterion (*PC*). Discussions of these criterions are not relevant for the work reported in this thesis.

2.5 Important Classes of Boolean Functions

In this section we detail some properties of two important classes of cryptographic functions: bent functions and resilient functions.

2.5.1 Bent Functions

Bent functions, is an important class of cryptographic Boolean functions. It was defined and first analyzed by Rothaus [18] who 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, including a characterization in terms of Hadamard matrices. Two large classes of bent functions were also presented in his paper. Further properties and constructions and equivalence bounds for bent functions can be found in [19], [20], [21], [22]. Kumer, Scholtz and Welch [23] defined and studied bent functions from Z_q^n to Z_q. Bent functions have been the subject of great interest in several areas including cryptography. In fact, the Canadian government block cipher standard (CAST [24]) is designed using these functions.

Properties of Bent Functions

1. 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 ω. By using Parseval's theorem, f is a bent function if and only if $|F(\omega)| = 2^{\frac{n}{2}}$ for all ω and since $|F(\omega)|$ is an integer then n should be even.

2. Bent functions achieve the maximum possible nonlinearity. The nonlinearity of any bent function is given by

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

This means that the Hamming distance of f to every affine function is maximum and equal to $(2^{n-1} \pm 2^{\frac{n}{2}-1})$.

3. Bent functions are never balanced. However, for very large n, they become statistically indistinguishable from balanced functions.

4. The order (algebraic degree) of bent functions is at least 2 and not more than $\frac{n}{2}$, i.e. $2 \le ANFD \le \frac{n}{2}$.

Bent functions of higher algebraic degree are preferred from cryptographic point of view

5. f is bent if all its derivatives

$$D_s f(x) = f(x) \oplus f(x+s)$$

are balanced, where s is any non zero vector in Z_2^n [18].

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

$$\hat{r}_f(s) = 0\,,$$

where, $\hat{r}_f(s) = \sum_x \hat{f}(x)\hat{f}(x \oplus s)$.

2.5.2 Resilient Functions

Another important class of Boolean functions for cryptography is that of resilient functions. These functions play a central role in stream cipher design. In the standard model of these ciphers the output of several independent Linear Feedback Register (LFSR) sequences are combined using a nonlinear Boolean function to produce the keystream. This keystream is bitwise XORed with the message bitstream to produce the cipher. In 1984 Siegenthaler [17] pointed out that if the combining function is not chosen properly, then the whole system is susceptible to a divide-and-conquer attack. He introduced the concept of m-th order correlation immunity for combining functions as a measure of their resistance against such correlation attacks. He also showed that for an n-

variable function, of degree *ANFD* and order of correlation immunity *m*, the following holds:

$$m + ANFD \leq n$$

Further if the function is balanced then

$$m + ANFD \leq n - 1$$

Later on, Guo-Zehn and Massey [25] introduced an equivalent definition of resilient functions using the Walsh transform of the Boolean function by the following equation:

$$F(\omega) = 0 \quad 0 \leq wt(\omega) \leq m$$

Example: Let $f = [01011010]$

ω	0	1	2	3	4	5	6	7
$wt(f)$	0	1	1	2	1	2	2	3
$F(\omega)$	0	0	0	0	0	8	0	0

Table 2.3: Walsh Transform of f

From Table 2.3, we can see that $F(\omega)$ of Hamming weight 1 is zero and $F(0)$ is also zero, and then the function is resilient with resiliency degree 1.

Nonlinearity of Resilient Functions

The maximum possible nonlinearity of *m*-resilient Boolean functions was discussed in many papers. It is well-known that the nonlinearity of a Boolean function doesn't exceed $2^{n-1} - 2^{\frac{n}{2}-1}$ [18]. Let $\hat{NL}(n)$ denote the maximum possible nonlinearity of an *n*-variable function. Also denote the nonlinearity of an *n*-variable, *m*-resilient function by $NL(n,m)$. Then for even n bent functions achieve the maximum nonlinearity $(2^{n-1}-2^{\frac{n}{2}-1})$. For odd n, and $n \le 7$, $\hat{NL}(n) = 2^{n-1}-2^{\frac{(n-1)}{2}}$. For some small values of m and n the exact values of maximal nonlinearity are known. For higher values of n, Sarkar and Maitra [14] found a non trivial upper bound on the nonlinearity of resilient functions. Their work is presented by the following theorem:

Lemma 1: If $n \geq 3$ and $m \leq n-3$ then the Walsh values of m-th order resilient function on n variables satisfy $|F(\omega)| \equiv 0 \bmod 2^{m+2}$ [14].

Using lemma 1, it is possible to obtain an upper bound on the nonlinearity of an n-variable, m-resilient function represented by the following theorem.

Theorem 1 [14].

1) If n is even and $m+1 > \frac{n}{2}-1$, then $NL(n,m) \le 2^{n-1}-2^{m+1}$.

2) If n is even and $m+1 \le \frac{n}{2}-1$, then $NL(n,m) \le 2^{n-1}-2^{\frac{n}{2}-1}-2^{m+1}$.

3) If n is odd and $2^{m+1} > 2^{n-1}-\hat{NL}(n)$, then $NL(n,m) \le 2^{n-1}-2^{m+1}$.

4) If n is odd and $2^{m+1} \le 2^{n-1}-\hat{NL}(n)$, then $NL(n,m)$ is the highest multiple of 2^{m+1} which is $\le \hat{NL}(n)$.

Before we proceed, we would like to introduce few notations for future convenience. By an (n, m, $ANFD$, NL) function we mean an n-variable, m-resilient Boolean function with algebraic normal form degree $ANFD$ and nonlinearity NL. By [n, m, $ANFD$, NL] we denote an unbalanced function with the same notation as above. Any component is replaced by '-' if we do not specify it, e.g., (n, m,-, NL) if we do not wish to specify the algebraic degree.

Table 2.4 represents the upper bound on $NL(n,m)$ given by theorem 1.

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

Table 2.4: The upper bound on nonlinearity given by theorem 1.

(1)An algorithm to construct (7, 2, 2, 56) and (10, 4, 5, 480) functions has been presented in [26].

(2) This function is not achieved yet: (8, 0, 7, 118).

(3) Computer search has yielded to (8, 1, 6, 116) [17].

(4) Computer search has yielded to (8, 3, 4, 112), (9,4,4,224), (10,3,6,480) [14].

(5) The existence of (9, 0, -, 244), (9, 0, -, 242), (9, 1, -, 244) functions were linked to the question of whether nonlinearity of more than 240 for $n = 9$ is exist or not [27]. Very recently, unbalanced functions with nonlinearity 241 for $n = 9$ are constructed [28].

(6) Computer search has yielded to [9, 3, 5, 240] [29].

(7) An algorithm to construct (10, 0, -, 492) functions has been presented in [30].

(8)Computer search has yielded (10, 1, 8, 488) [31]. Using the weight divisibility results of resilient function involving the algebraic degree, it can be shown that the functions

(10, 1, -, 492), (10, 2, -, 488) if all exist are in the form of (10, 1, 8, 492), (10, 2, 7, 488) [32], [33].

(9)In our work we constructed several examples of (6,2,3,24), (6,3,2,16), (7,2,4,56), (7,3,3,48) (7,4,2,32) , (8,3,4,112) , (8,4,3,96), (9,3,5,240), (9,4,4,224), (10,4,5,480). The constructions with examples of these functions are described in chapter five. Note that the existence of (9,3,5,240) was a open problem and we are the first to show some examples for this function.

CHAPTER 3

Swarm Inspired Artificial Life Techniques

3.1 Introduction

The human made system, that posses some essential properties of life or living beings are called artificial life (ALife). Study of artificial life demonstrates how biological trends can help out with computational problems. These techniques, inspired from biological phenomenon are known as artificial life techniques. There are lots of computational techniques inspired by biological systems. For example: artificial neural network [34] and genetic algorithm [35]. The first one is a simplified model of human brain and the second one is inspired by the human evolution. However, apart of these biological systems, there are some other types of biological systems too. Swarm intelligence is one of them. Swarm intelligence is more of a social system. More specifically, the collective behaviors of simple individuals interacting with their environment and each other.

There are two popular swarm inspired methods in computational intelligence areas: Ant colony optimization (ACO) [36] and particle swarm optimization (PSO) [37] [38].

While ACO is inspired by the behaviors of ants PSO is inspired by the social behavior of bird flocking or fish schooling. Both techniques have successful application in discrete optimization problems.

3.2 Ant Colony Optimization

Ant Colony Optimization [36] is a heuristic optimization method for solving different combinatorial optimization problems. It is a population based approach which borrows ideas from biological ants. The social behaviors of ants have been much studied by the scientists and from their behavior the computer scientists came up with the idea of this optimization.

Experiments with real ants showed that ants go from the nest to the food source and backwards, then after a while, the ants prefer the shortest path from the nest to the food source. Real ants are capable of finding the shortest path from their nest to a food source without using visual cue [39]. Ants have a special way of communicating information concerning food sources. While walking, ants secrete an aromatic essence on the ground, called pheromone. The other ants will follow the path of greater pheromone trail with higher probability and as they follow the path, they as well will secrete pheromone there. So the pheromone of that path which greater number of ants are following will increase and as the pheromone trail of that path increases more ants will follow that path. Since ants passing through food source by shorter path will come back to the nest sooner than ants passing through longer paths, the shorter path will have a higher traffic density than that of the longer one. Thus a single ant will follow the shortest path with higher probability [36].

For example an experimental setting is shown in Fig. 3.1. Consider a path along which ants are walking from nest to food and vice versa (Fig.3.1.(a)). Suddenly an

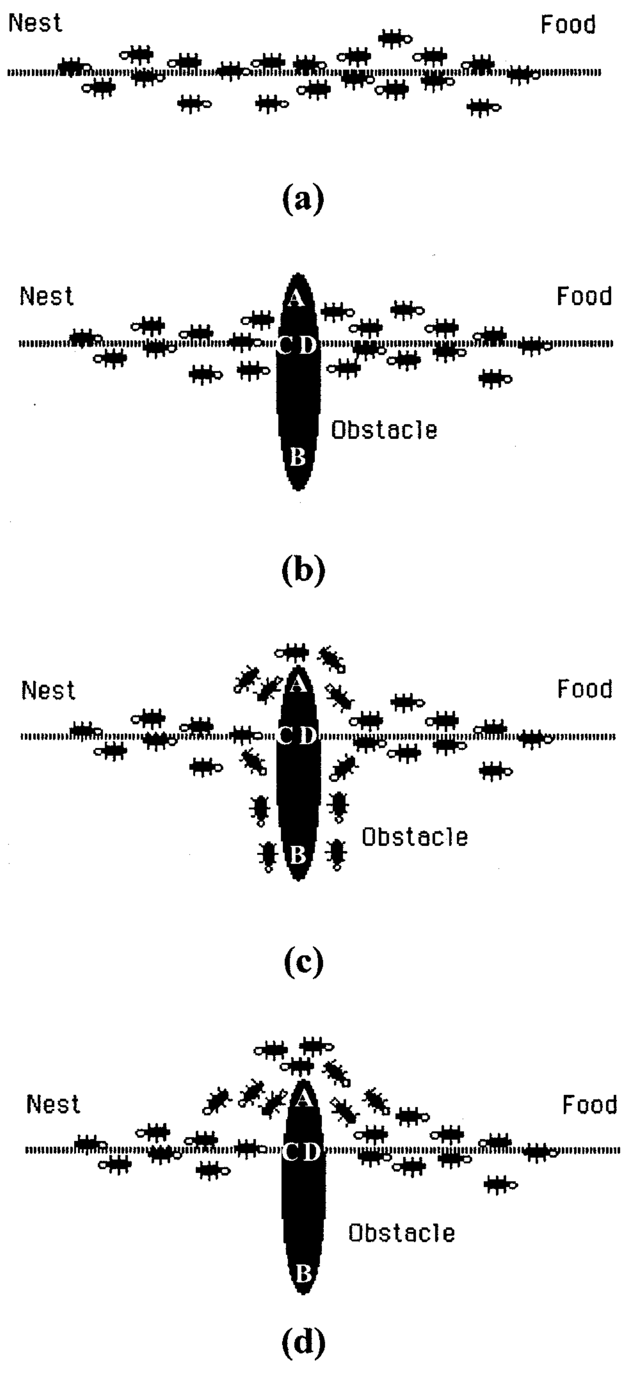

Figure 3.1: An example with real ants. (a) Ants follow a path between nest and food. (b) An obstacle is placed on the path between nest to food. (c) Ants can choose to go around the obstacle following one of the two paths with equal probability. (d) Ants are following the shorter path as more pheromone is laid down on this

obstacle is placed on the path between nest to food (Fig.3.1.(b)). At position C (if going from nest to food source) or position D (if returning from food source to nest) have to decide which path will it follow (upper path or downer path) (Fig.3.1.(c)).The choice is influenced by the intensity of the pheromone trails left by preceding ants. Initially for the first ant which reaches point C can choose either path CAD or CBD with equal probability as there is no previous pheromone on any path. But as the path CAD is smaller than path CBD an ant will take less time to get to the D point from B point by path CAD than by path CBD. As a result number of ants following the path CAD will be higher than that of path CBD per unit time. As more ants follow path CAD more pheromone will be on that path and other ants will start following that path. So after some times all ants will follow the path CAD (Fig.3.1.(d)) which is the shortest path.

3.2.1 ACO Algorithm

Ant colony optimization algorithm was first used to produce near-optimal solutions to the traveling salesman problem (TSP) [36]. Given a collection of cities, TSP can be stated as the problem of finding a minimum length closed tour that visits each town once.

At first, all ants (m) are placed on the towns in a random fashion. An ant k will choose to go to the next town j from the current town i by the intensity of the pheromone trails $\tau(i,j)$ left by preceding ants during previous iterations and also by the *a priori* desirability of the choice $\eta(i,j)$. The choice of next node j from current node i for an ant k is given by probability $p_k(i \rightarrow j)$

$$p_k(i \to j) = \begin{cases} 0, & \textit{if } j \textit{ already visited} \\ \dfrac{[\tau(i,j)]^{\alpha}\,[\eta(i,j)]^{\beta}}{\sum_{k \text{ not visited}} [\tau(i,k)]^{\alpha}\,[\eta(i,k)]^{\beta}}, & \textit{otherwise.} \end{cases}$$

Here, the *a priori* desirability of the choice $\eta(i,j)$ is called visibility, and is given by

$$\eta(i,j) = \frac{1}{d(i,j)}$$

where, $d(i,j)$ is the distance between the two cities.

α and β are two controlling parameters that controls the relative importance between pheromone trail and visibility. On the first run (iteration) when there is no pheromone trail on the edge of the towns $\tau(i,j) = 0$; the probability of choosing next town totally depends on visibility.

A *tabu list* is maintained to force ant to make legal tours. The *tabu list* disallows the ant to visit the same town more than once before a tour is completed.

Once a tour is completed, every ant updates the pheromone $\tau(i,j)$ over the edge $(i \to j)$ along its visited path as follows:

$$\tau(i,j) = \tau(i,j) + \Delta\tau(i,j)$$

where,

$$\Delta\tau(i,j) = \sum_{k=1}^{m} \Delta\tau_k(i,j)$$

and,

$$\Delta\tau_k(i,j) = \begin{cases} \dfrac{Q}{L_k}, & \textit{if ant } k \textit{ uses the edge} (i \to j) \\ 0, & \textit{otherwise.} \end{cases}$$

Here, Q is a constant and L_k is the tour length of k-th ant.

After each iteration, a portion of the pheromone of the edge is evaporated according to a local updating rule,

$$\tau(i,j) = \rho \times \tau(i,j)$$

where, ρ is the evaporation rate and the value of ρ is smaller than 1. It prevents the unlimited accumulation of pheromone trail on the edges. It increases diversity of the system.

3.3 Particle Swarm Optimization

Particle swarm optimization (PSO) is a population based, self-adaptive stochastic optimization technique developed by Eberhart and Kennedy [37] [38] in 1995, inspired by social behavior of bird flocking or fish schooling.

Like other population-based optimization methods such as genetic algorithm (GA), the particle swarm algorithm is initialized with a population of random solutions in the search space [40]. However, unlike GA, typical PSO has no evolution operators such as crossover and mutation. The PSO algorithm works on the social behavior of particles in the swarm. Therefore it searches for optimum solution by simply adjusting the trajectory of each potential solution towards its own best location and towards the best particle of the entire swarm over generations [38] [41] [42]. However recently evolutionary PSO (EPSO) [43][44] has also been proposed. The main advantage of PSO over other population-based optimizer is that, it is very easy to implement and there are very few parameters to adjust and it has also the ability to quickly converge to a reasonably good solution.

3.3.1 PSO Algorithm

PSO simulates the behaviors of bird flocking. If we consider a scenario that a group of bird is searching for food in an area and there is only one piece of food available in that area and the birds don't know where the food is but they know how far the food is in each iteration. So the best strategy to find the food is to follow the bird which is nearest to the food.

PSO learned from the scenario and used it to solve the optimization problems. In PSO, each single solution is a 'bird' in the search space and is called 'particle'. Each particle keeps track of its coordinates in the search space which are associated with the best solution it has achieved so far. This value is called particle best or *pbest.* Another value is the best value achieved so far by any particle in the population. This value is called global best or *gbest.* After finding the two best values each particle updates its velocity ($v_{i,j}$) and position ($P_{i,j}$) towards its *pbest* and *gbest* locations as follows:

Velocity update:

$$v_{i,j} = v_{i,j} + c_1 r_1 (Ppbest_{i,j} - p_{i,j}) + c_2 r_2 (Pgbest_{i,j} - P_{i,j})$$

Particle position update :

$$P_{i,j} = P_{i,j} + v_{i,j}$$

Where, $Ppbest_{i,j}$ and $Pgbest_{i,j}$ are the particle best and global best position of the particles respectively. r_1 , r_2 are uniformly distributed random variables where, $1 \le r_1, r_2 \le 0$ and c_1, c_2 are learning factors. These learning factors made PSO an attractive optimization technique. Varying these factors it is possible to use PSO in a wide variety of applications.

3.4 *Search Space*

While solving a problem, the objective is to find a solution which will be the best among others. The space of all feasible solutions (the set of solutions among which the desired solution resides) is called search space. There are two types of search spaces for optimization problem, vector spaces and permutation spaces.

3.4.1 Vector Spaces and Permutation Spaces

Unlike vector spaces where all the elements are independent from each other, in permutation space the elements are totally dependent to each other as the order of the elements which constitute the n-tupla of values is the fact which differentiates one solution from another solution [45].

Suppose a three variable function, $f(x,y,z)$, is being optimized in vector space and x,y,z all takes values from 0 to 3. So, a combination of values of (x,y,z) = (3, 3, 3) can be possible. But if x,y,z are assumed as positions of a vector then the optimization problem changes into a permutation problem and in this case no two elements from x,y,z can have the same position. In this case only followed 6 solutions are possible.

(x,y,z) = (1, 2, 3), (2, 3, 1), (1, 3, 2), (3, 2, 1), (2, 1, 3) and (3, 2, 1)

3.4.2 Dealing with Permutation Spaces in Optimization Techniques

Regardless of optimization technique, dealing with permutations is always bit harder than dealing with vectors of parameters. There are few techniques that have been applied to solve permutation problems:

1. **Penalty function:** In this technique if input sequence is not a legal permutation a penalty factor is added to its cost function. As far the input sequence will be from

a legal permutation the penalty factor will be larger. This is a very popular technique used with genetic algorithm

2. **Only legal input sequence generation:** In this technique during the iterative process only legal input sequence are generated. A *tabu list* can be maintained, where all the illegal elements of a permutation will be saved and generation of those elements will be prohibited through out the iteration. Another way is to use swapping between the elements of a permutation. Simulated annealing and hill climbing algorithms use this technique for permutation space.
3. **Simple mapping from vector space to permutation space:** Simple mapping from vector search space to permutation space can be constructed by sorting the elements of the position vector in an ascending or descending order to get an ordered set and use the index of the position as legal sequence.

3.5 Optimization Heuristics for Cryptanalysis of Classical Ciphers

The Arabs were among the first to make significant advances in cryptanalysis. As discussed before Early in the 15th century, Qalqashandi, wrote down a technique for solving ciphers using the average frequency of each letter of the language. But it is only last twenty-five years; several optimization heuristics have shown promise for automated cryptanalysis of different ciphers especially classical ciphers [46].

One of the early proposals was given by Peleg and Rosenfeld [47]. They modeld the problem of breaking substitution ciphers as a probabilistic labeling problem. Every coded alphabet was assigned probabilities of representing plaintext alphabets and they updated the probabilities using the joint letters. Using this scheme in an iterative way they were able to break the cipher.

Carrol and Martin [48] developed an expert system approach to solve simple substitution ciphers using hand-coded heuristics.

Forsyth and Safavi-Naini [49] recast the problem as a combinatorial optimization problem and presented an attack on simple substitution cipher using simulated annealing algorithm.

Spillman *et. al* [50] presented an attack on simple substitution cipher using genetic algorithm. Clerk [46] re-implemented the genetic algorithm and simulated annealing attack in order to compare them and also evaluate a third technique using *tabu* search. He successfully deciphered both substitution and transposition ciphers.

Bahler and King [51] used trigram statistics and relaxation scheme to iterate towards the most probable key as previously done by Peleg and Rosenfeld.

Lucks [52] used a word pattern dictionary and search over it with the constraint that all ciphertext characters must decrypt to the same plaintext character.

Hart [53] improved upon this method by directing this combinatorial search towards more frequent English words.

Jakobsen [54] introduced a fast algorithm for the cryptanalysis of simple substitution ciphers based on a process where an initial key guess is refined through a number of iterations. In each step the plaintext corresponding to the current key was evaluated and the result was used as a measure of how close we are in having discovered the correct key.

Carroll and Robbins [55] presented an attack on polyalphabetic substitution cipher using relaxation algorithm. King [56] again used the same attacking technique on polyalphabetic ciphers and obtained better results than them.

Matthews [57] utilized the genetic algorithm in cryptanalysis of the transposition cipher.

Recently, Russell, Clark and Stepney [58] successfully used Ant Colony Algorithm (ACO) in breaking transposition ciphers.

Optimization heuristics have also been used to cryptanalyze other modern ciphers. For example, Hernandez *et al* [59][60][61] used genetic algorithm successfully to cryptanalize TEA (Tiny Encryption Algorithm) and XTEA (Extended Tiny Encryption Algorithm).

3.6 Evolutionary Search for Boolean functions

The design of Boolean functions and S-boxes with desirable cryptographic properties such as high nonlinearity, high algebraic degree, and reasonable order of correlation immunity are very important for cryptographic research. The application of evolutionary techniques in this specific field of research started only in the last decade. The first attempt at local search algorithm was made by Forre [62] in 1990. But the results were poor and generated a little interest. Latter Millan, Clark and Dawson used the different evolutionary techniques to successfully design many Boolean functions with different properties. They used Hill Climbing algorithm to generate Boolean functions with high nonlinearity [63] [64] [65]. They also used genetic algorithm for the same purpose [66].

Dimovski and Gligoroski [12] also used both genetic algorithm and hill climbing algorithm to generate highly nonlinear Boolean functions.

Clark and Jacob [67] introduced the use of simulated annealing algorithm in Boolean function design. Clark, Jacob and Stepney designed S-boxes [68] and they also found Boolean function with particular desirable cryptographic properties of a small number of variables [69] using simulated annealing algorithm. Clark, Jacob, Maitra and Stanica used spectral inversion technique with simulated annealing to solve some open problem of Boolean function with specific properties [29].

Chapter 4

Cryptanalysis of Simple Substitution Ciphers

4.1 *Classical Ciphers*

Classical ciphers [1] [4] are the earliest schemes of cryptography. These ciphers were developed and used before the computer age, and are usually carried out by pen and paper or by simple mechanical devices rather than by electronic media. Although, from the security point of view, classical ciphers are no match to the recently developed ciphers, they have not lost their importance because most of the commonly used modern ciphers use classical ciphers as their building blocks. In fact, most complex algorithms are formed by mixing substitution and transposition in a product cipher. Modern block ciphers such as DES and AES iterate through several stages of substitution and transposition. Given their simplicity, and the fact that they are used to construct other ciphers, the classical ciphers are usually the first ones considered when researching new attack techniques such as the ones discussed in this chapter.

There are usually two types of classical ciphers, namely transposition ciphers and substitution ciphers. The substitution ciphers can yet be devided into to types, simple substitution ciphers and polyalphabetic substitution ciphers.

In this literature we only concentrate on the cryptanalysis of simple substitution ciphers.

4.2 Simple Substitution Ciphers

Simple substitution cipher is a well-known cryptosystem. It is also known as mono-alphabetic substitution ciphers. It is the simplest form of substitution ciphers. Each symbol in the plaintext maps to a different symbol in the ciphertext [46]. It is a one-to-one substitution. The simple substitution cipher used in this work operates on the English alphabet of 26 letters ("A-Z"). We assume that all the punctuations and structure (sentences/paragraphs, space characters, and newline characters) are removed from the plaintext in order to hide these obvious statistics from the ciphertext.

Key:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
X	N	Y	A	H	P	O	G	Z	Q	W	B	T	S	F	L	R	C	V	M	U	E	K	J	D	I

Encryption:
Plaintext:
SWARMOPTIMIZATIONISAPOWERFULTOOL
Ciphertext:
VKXCTFLMZTZIXMZFSZVXLFKHCPUBMFFB

Table 4.1: Example of a Simple Substitution Cipher

Let x be an n-character alphabet $\{x_0, x_1, x_2, x_3, \ldots, x_{n-1}\}$, and y is also an n-character alphabet $\{K(x_0), K(x_1), K(x_2), K(x_3), \ldots\ldots, K(x_{n-1})\}$, where $K : x \rightarrow y$ is a one to one mapping of every alphabet of x to the corresponding alphabets of y. Here 'K' is the cipher key function which can be looked at as a permutation of the 26 character. The transmitter enciphers the plaintext into ciphertext with a predetermined key function (K) and sends it to the receiver. The receiver deciphers the ciphertext to plaintext with the inverse key function (K^{-1}).

An example for a simple substitution cipher key and encryption operation is shown in Table 4.1.

4. 3 Attacks on Simple Substitution Ciphers

For an alphabet of 26 characters there exist ($26! \approx 4.03291461 \times 10^{26} \approx 2^{88}$) possible keys for a simple substitution cipher. This number is far too large to allow any kind of exhaustive search attack - even on the fastest of today's computers. However, one special property of simple substitution cipher that makes it relatively easy to cryptanalyze, is that the language statistics remain unchanged by the encryption process and hence frequency analysis presents a basic tool for breaking classical ciphers. In simple substitution ciphers the language statistics remain unchanged by the encryption process but in encrypted message the language statistics is the permutation of the original message language statistics. So the search for the corresponding language frequencies can be found using swarm inspired artificial life algorithms (described in Chapter 3) to decrypt the original message from the encrypted message.

The length of the intercepted ciphertext message which is being cryptanalyzed plays a major role in the cryptanalysis process. As the length of intercepted ciphertext increases the chance of recovering the original message (or entire key) also increases. Study show [46] if intercepted ciphertext length is 1000 or more then it is possible to decrypt almost hundred percent of the message correctly. Note that in practice it is impossible to find a simple substitution cipher key which differs in exactly one place from the correct key.

4.3.1 N-Gram Statistics

In all languages, certain letters of the alphabet appear more frequently than others. The *n-gram* statistics is the character frequency statistics. It indicates the frequency distribution of all possible instances of '*n*' adjacent characters.

English Alphabet	Percentage of Frequency
A	8.13254
B	1.47264
C	2.79902
D	4.23232
E	12.6696
F	2.35908
G	1.76541
H	5.9186
I	7.11386
J	0.0820647
K	0.672795
L	3.96895
M	2.43187
N	7.0501
O	7.36819
P	1.89857
Q	0.104446
R	5.61341
S	6.61197
T	9.62056
U	3.03097
V	0.920797
W	2.3532
X	0.188771
Y	1.55222
Z	0.0680481

Table 4.2: Uni-gram (1-gram) Statistics of English Language

For example 'E' is the most common uni-gram (1-gram) in English language, and one of the common bi-gram (2-gram) in English language is 'TH'. These n-gram statistics can be used to measure the fitness of a suggested decryption key. In Table 4.2 the uni-gram (1-gram) statistics of English language alphabets are given. This statistics are calculated from an English book named "20,000 Leagues under the Sea" written by famous science-fiction writer Jules Verne. In our cryptanalysis process we only consider uni-gram and bi-gram statistics. The bi-gram (2-gram) statistics used for the cryptanalysis process is also calculated from the same book.

It can be noted that to obtain a readable message it is not necessary to recover each elements of the key. Experiments [46] have shown that the correctly determination of the key element for the five most frequent alphabets of English language can recover almost fifty percent of the message. And the eleven most infrequent characters account for only ten percent of the message.

4.3.2 Cost Function

The general idea to compare candidate keys for simple substitution cipher is to calculate the cost for certain candidate key by comparing the *n-gram* statistics of the decrypted message using the candidate key with those of the reference language statistics (discussed in 4.3.1). Using optimization techniques with this cost functions the objective is to minimizing the cost function, that the decrypted message *n-gram* statistics can match the reference language statistics as close as possible. The candidate key generating the lowest cost for decrypted message is regarded as the solution of the cryptanalysis problem.

The following equation is used as the cost function equation for our cryptanalysis problem:

$$Cost(K) = \lambda_1 \sum_{c \in \{A,B,\ldots,Z\}} \left| R^U_{(c)} - DK^U_{(c)} \right| + \lambda_2 \sum_{c_1,c_2 \in \{A,B,\ldots,Z\}} \left| R^B_{(c_1,c_2)} - DK^B_{(c_1,c_2)} \right| \qquad (4.1)$$

Here. R^U, R^B and DK^U, DK^B be the reference language uni-gram and bi-gram statistics and decrypted message uni-gram and bi-gram statistics (using a candidate key *K*) respectively. The superscript U and B denote the uni-gram and bi-gram. The values λ_1 and λ_2 allow assigning of different weights to each of the two *n-gram* types. In this literature we individually consider purely uni-gram ($\lambda_1 = 1$ and $\lambda_2 = 0$) and purely bi-gram ($\lambda_1 = 0$ and $\lambda_2 = 1$) statistics and compare the results.

Forsyth and Safavi-Naini, in their simulated annealing attack on the substitution cipher [49] and Jakobsen in his attack [54] used a very similar cost function based on purely bi-gram statistics.

$$Cost(K) = \sum_{c_1,c_2 \in \{A,B,\ldots,Z\}} \left| R^B_{(c_1,c_2)} - DK^B_{(c_1,c_2)} \right| \qquad (4.2)$$

Spillman et al [50] used both uni-gram and bi-gram statistics for his genetic algorithm attack on the simple substitution ciphers.

$$Cost(K) = \sum_{c \in \{A,B,\ldots,Z\}} \left| R^U_{(c)} - DK^U_{(c)} \right| + \sum_{c_1,c_2 \in \{A,B,\ldots,Z\}} \left| R^B_{(c_1,c_2)} - DK^B_{(c_1,c_2)} \right| \qquad (4.3)$$

Andrew Clerk [46] also used uni-gram and bi-gram statistics for his attack on simple substitution ciphers using optimization techniques like genetic algorithm, simulated annealing and tabu search. He also used tri-gram statistics successfully for the same.

$$Cost(K) = \lambda_1 \sum_{c \in \{A,B,\ldots,Z\}} \left| R^U_{(c)} - DK^U_{(c)} \right| + \lambda_2 \sum_{c_1,c_2 \in \{A,B,\ldots,Z\}} \left| R^B_{(c_1,c_2)} - DK^B_{(c_1,c_2)} \right|$$

$$+ \lambda_3 \sum_{c_1,c_2,c_3 \in \{A,B,\ldots,Z\}} \left| R^T_{(c_1,c_2,c_3)} - DK^T_{(c_1,c_2,c_3)} \right| \tag{4.4}$$

Here, superscript T is used to denote tri-gram statistics.

The calculation of trigram statistics is usually an expensive task compare to the accuracy gain can be achieved using tri-gram statistics. So we, like most of the other researchers, omitted tri-gram statistics based calculation in our cryptanalysis problem.

4.4 Ant Colony Optimization (ACO) Attack

The use of Ant Colony optimization for cryptanalysis problem is rather straight forward and modeled similarly to the TSP problem discussed in chapter three.

4.4.1 ACO for Permutation Space

As stated in chapter three ACO uses *tabu list* to overcome the permutation problem. The typical ACO algorithm is designed to solve permutation problems.

4.4.2 Algorithm

For our cryptanalysis problem each complete path constructed by ants is a permutation of the nodes (alphabetic characters) corresponding to a key. So in our algorithm, we use the distance between the unigram frequency of the reference language statistics and the target test key to represent the *a priori* desirability of choice of a particular key element,

i.e. we set, $$d(i,j) = \left| R^U_i - DK^U_j \right| \tag{4.5}$$

The system is initialized with a group of ants moving across a fully connected bidirectional graph of 26 nodes $(n_1, n_2, ..., n_{26})$. A *tabu* list is maintained to prevent any ant from visiting the same node twice. Every possible decryption key $K = (k_1, k_2, ..., k_{26})$ to a unique path along this graph $(n_1 \rightarrow n_{k_1}, n_2 \rightarrow n_{k_2}, ..., n_{26} \rightarrow n_{k_{26}})$.

The algorithm proceeds by iterating through the following three basic steps:

1. **Construct a solution for all ants:** At each node, each ant has to make a (statistical) decision regarding the next node to visit. At the first iteration, all the ants will move randomly. However, on subsequent iterations, the ants' choices will be influenced by the intensity of the pheromone trails left by preceding ants during previous iterations. A higher level of pheromone on a given path gives an ant a stronger stimulus and thus a higher probability to follow this path. In particular, at node i, the ant expand its tour to node j with probability p that is given by:

$$p(i \rightarrow j) = \begin{cases} 0, & \textit{node j already visited} \\ \dfrac{[\tau(i,j)]^{\alpha}[\eta(i,j)]^{\beta}}{\sum\limits_{k \text{ not visited}} [\tau(i,k)]^{\alpha}[\eta(i,k)]^{\beta}}, & \textit{otherwise.} \end{cases} \qquad (4.6)$$

Here, $\tau(i, j)$ is the pheromone trail and $\eta(i, j)$ is the *a priori* desirability of choice.

where, $$\eta(i,j) = \frac{1}{d(i,j)} \qquad (4.7)$$

Setting $\alpha = 0$ in equation 4.6, corresponds to the system that relies only on the uni-gram statistics for the cryptanalysis. For the bi-gram based system, the optimum value of α and β is found by a heuristic trial and error.

2. **Do a global pheromone update:** Once the tour is completed, every ant updates the pheromone $\tau(i,j)$ over the arc $(i \to j)$ along its visited path as follows:

$$\tau(i,j) = \tau(i,j) + \Delta\tau(i,j) \tag{4.8}$$

where,

$$\Delta\tau(i,j) = \frac{1}{Cost(K)} \tag{4.9}$$

the equation for $Cost(K)$ is given in equation (4.1).

3. **Evaporate pheromone:** After each iteration, a portion of the pheromone of the edge is evaporated according to a local updating rule,

$$\tau(i,j) = \rho \times \tau(i,j) \qquad \rho < 1 \tag{4.10}$$

such that the probability of the selection of that edge by other ants decreases. This prevents construction of similar paths by the set of ants and increases the diversity of the system. The rate of evaporation provides a compromise between the rate of convergence and reliability of convergence. Fast evaporation causes the search algorithm to be stuck at local optima, while slow evaporation lowers the rate of convergence.

After enough iteration of the algorithm, the pheromone of the good edges which are used in constructing of low-cost paths will increase and the pheromone of the other edges will evaporate. Thus, in the higher iterations the probability of constructing low-cost paths increases.

4.4.3 Experimental Results

Throughout all of our experiments, the number of ants and number of passes (iterations) were set to 1000 and 100 respectively. The rest of the parameters were varied in an ad-hoc way to optimize the results. The results have been taken by using both purely uni-gram and bi-gram statistics. Figure 4.1 shows how the average (over 100 randomly selected keys) number of corrected key elements varies with the amount of known ciphertext. Usually in other cases we run our algorithm and take the best result but here in this case, because of the random nature of the algorithm it is better to represent the results by averaging the results of a large number of attacks. In this way we can get a good idea on the ability of the algorithm.

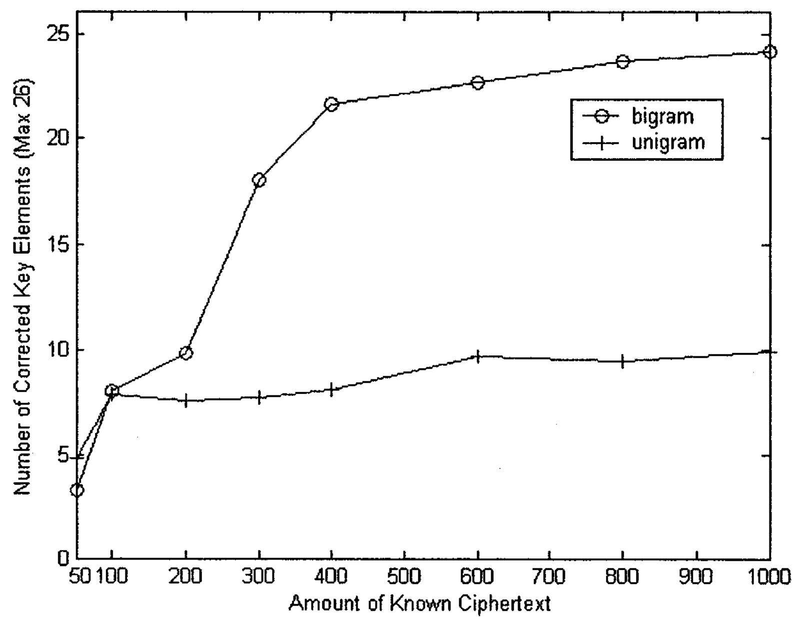

Figure 4.1: Number of corrected key elements versus the amount of known ciphertext

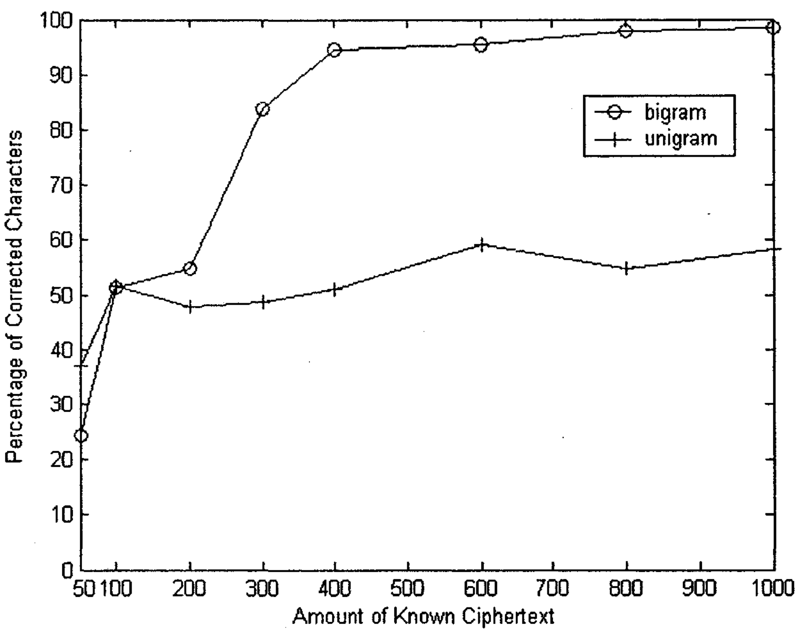

Figure 4.2: Percentage of corrected characters versus the amount of known ciphertext

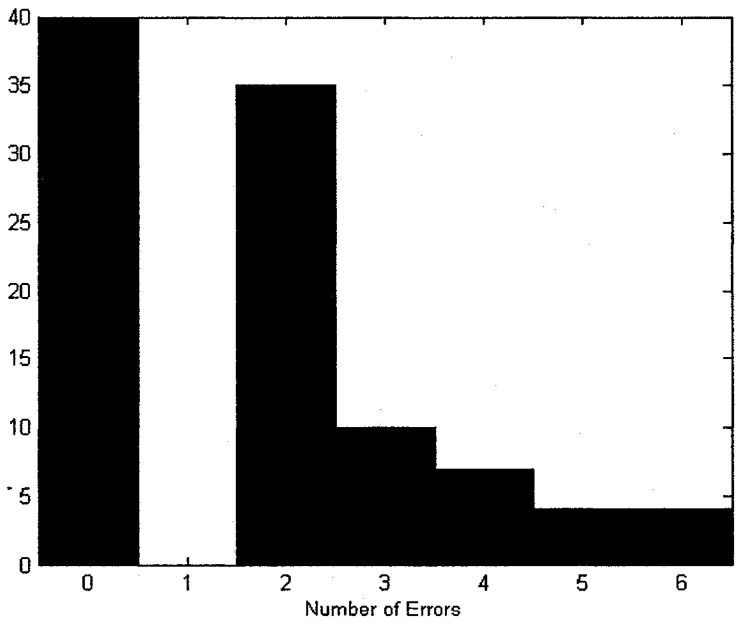

Figure 4.3: Error distribution for bi-gram based system (100 random keys and 900 known ciphertext characters)

Similarly, Figure 4.2 shows the percentage of corrected characters versus the amount of known ciphertext. Figure 4.3 shows the error distribution for 100 randomly selected keys when the amount of known ciphertext is 900 characters.

For this case, the average and variance of the number of errors in the recovered key characters are 1.72 and 2.9303 respectively.

4.5 Particle Swarm Optimization (PSO) Attack

For PSO dealing with permutation space is not as straight forward as in the case of ACO.

4.5.1 PSO for Permutation Space

In traditional PSO, the particle is encoded as a string of positions, which represent a multidimensional space. All the dimensions typically are independent of each other, thus the updates of the velocity and the particle are performed independently in each dimension. But for a problem which deals with permutation space and since the elements are not independent to each other in this case; it is possible to get two or more positions with same value after the update, which breaks the permutation rule. So the tradition PSO can not be used to solve permutation problem. One idea is, as stated on chapter three and suggested in [70] [71], simple mapping from search space to permutation space by sorting the elements in the particle position vector to get an ordered set $(P_{i\pi_1}, P_{i\pi_2}, P_{i\pi_3} ...., P_{i\pi_n})$ where, $(P_{i\pi_1} \geq P_{i\pi_2} \geq P_{i\pi_3} \geq .... \geq P_{i\pi_n})$ to get the permutation $(\pi_1, \pi_2, \pi_3, ..., \pi_n)$. But our experiments indicate that the non linearity inherent in the sorting process limits the success of this approach when the size of the permutation grows above 20 elements.

Another strategy was proposed by Hu *et al* [72] to overcome the permutation problem.

In typical PSO system, for updating the position of the particle the velocity is added to the particle on each dimension. So, when the velocity is larger the particle may explore distant areas. In the new updating method [72], it also does the same thing in a bit different way. In this strategy when the velocity is larger, the particle is more likely to change to a new permutation sequence rather than adding the velocity value to the particle. If the velocity of a position in a particle is high the probability of a swap taking place in that position increases. Each position randomly determines if there is a swap with a probability determined by the velocity corresponding to this position. If a swap is required, the position is set to the value of same position in $gbest$ by swapping a pair of values in the particle. But this type of modification in the particle position has a problem. Since the particle tries to follow the $gbest$, it would stay in its current position forever when it was identical to $gbest$. So a mutation factor is added. The particle will randomly swap one pair of positions in the permutation if the permutation is identical to $gbest$.

4.5.2 Algorithm

In our cryptanalysis problem each solution in the parameter space represented by each particle is a permutation of the alphabetic characters corresponding to a key. At first we initialize a set of particles with random permutation sequence for the alphabetic characters.

Each particle keeps track the best solution it has achieved so far ($pbest$) and the best value achieved so far by any particle in the population ($gbest$) is also calculated. We also keep track of the positions responsible for *pbest* and *gbest*, which is denoted as *Ppbest* and *Pgbest* respectively.

After finding the particle positions for two best values, each particle updates its velocity ($v_{i,j}$) and position ($P_{i,j}$) towards its *pbest* and *gbest* locations.

Velocity update:

This velocity update is done individually for each position of a particle by the following equation:

$$v_{i,j} = v_{i,j} + c_1 r_1 (Ppbest_{i,j} - p_{i,j}) + c_2 r_2 (Pgbest_{i,j} - P_{i,j}) \qquad (4.11)$$

here, r_1, r_2 are random numbers vary from 0 to 1, and c_1, c_2 are learning factors. Suitable values for c_1 and c_2 is found by a heuristic trial and error.

Particle position update:

For updating particle positions we slightly modified the permutation strategy suggested by Hu *et al* [72] and used that strategy for the particle updating in our problem.

We can describe the updating process by the following steps:

Step 1: The velocity is limited to an absolute value and normalized to the range of 0 to 1 by dividing all the velocities by the maximum velocity.

Step 2: For each position we determine if there is a swap with a probability determined by the velocity corresponding to this position.

Step 3: If a swap is required, the position is set to the value of same position in *gbest* by swapping value.

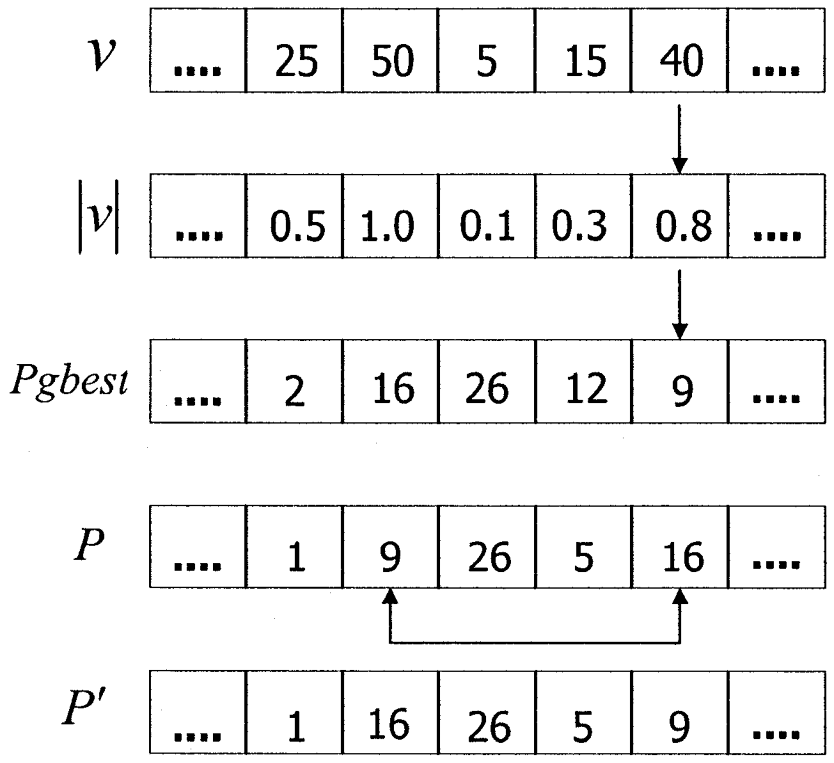

Figure 4.4: Particle position update strategy [72]

Step 4: This step is identical to step 3 except that the update is done with respect to the position in *pbest*.

Figure 4.4 shows the velocity update strategy proposed by Hu *et al* [72]. It should be noted that *Step 4* above is not described by them. However, our experiments show that instead of their mutation proposal, using this extra step helps in reducing the number of iterations required before a good solution is found.

4.5.3 Experimental Results

Throughout all of our experiments, the number of particles and number of passes (iterations) were set to 500 and 200 respectively. The rest of the parameters were varied in an ad-hoc way to optimize the results.

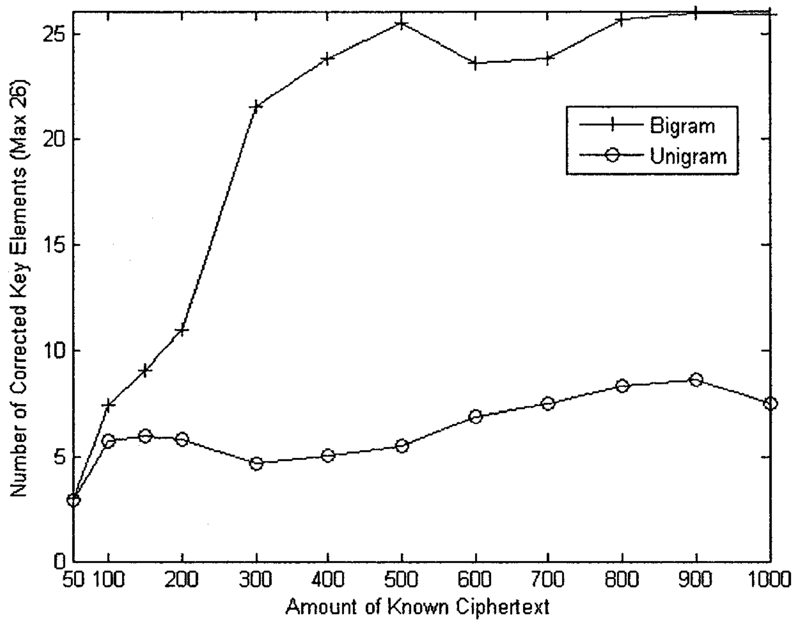

Figure 4.5: Number of corrected key elements versus the amount of known ciphertext

Figure 4.5 shows how the average number of corrected key elements varies with the amount of known ciphertext by using both purely uni-gram and purely bi-gram statistics. The average has taken over 100 randomly selected keys like we have done for the ACO attack. Similarly, Figure 4.6 shows the percentage of corrected characters versus the amount of known ciphertext. In Figure 4.7 the error distribution for 100 randomly selected keys are shown when the amount of known ciphertext is 900. The average and variance of the error distribution is 0.04 and 0.0792 respectively.

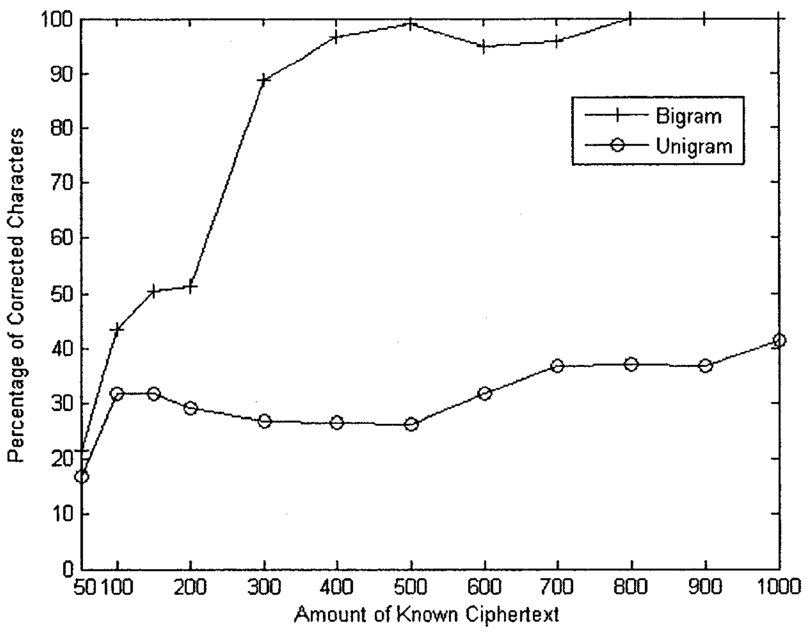

Figure 4.6: Percentage of corrected characters versus the amount of known ciphertext

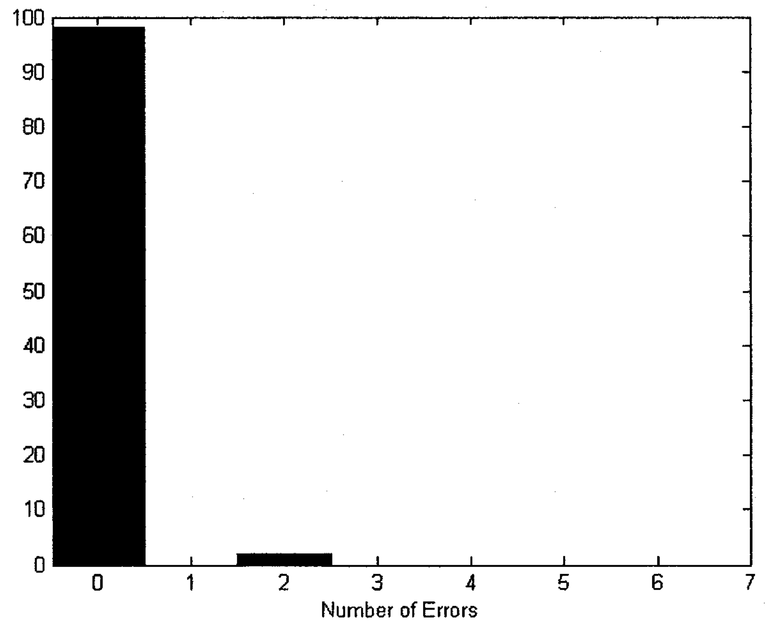

Figure 4.7: Error distribution for bi-gram based system (100 random keys and 900 known ciphertext characters)

Chapter 5

Spectral Inversion Construction of Bent and Resilient Functions

5.1 Introduction

There are lots of techniques to construct different classes of Boolean functions. Many people have suggested many techniques to construct bent and resilience functions. Evolutionary techniques are one of the recently proposed construction techniques for generating this type of functions. In our work, we used swarm inspired artificial life technique to generate bent and resilient functions with different properties. In our study we used spectral inversion technique to find legitimate Boolean functions. We also used concatenation of two (sometimes four) Boolean functions with non-intersecting Walsh spectrum to construct a Boolean function of higher inputs. This concatenation technique proved to be very effective for construction of Boolean functions of input $n > 7$.

5.2 Spectral Inversion Technique for Boolean Functions

Various important cryptographic criteria such as balancedness, nonlinearity, correlation immunity, and resiliency are defined in terms of the Walsh Hadamard values of that function. From a given spectrum of Walsh-Hadamard values it is possible to identify whether the function met those properties. Clark *et al* [29] introduced the idea of Boolean functions construction by spectral inversion and applied it for the construction of several cryptographic functions of interest. The basic idea is to start with a set of Walsh coefficients $\{W(0), W(1), \ldots\ldots, W(2^n - 1)\}$ that satisfy the required constraints (nonlinearity, resiliency, etc). However, since it is not guaranteed that such a spectrum W, will be the Walsh spectrum for some Boolean function, our problem is reduced to finding a permutation $\Pi = \{\lambda_1, \lambda_2, \ldots\ldots\ldots, \lambda_{2^n}\}$ such that when is Π applied to the set W the resulting function obtained by applying the Inverse Walsh Transform to the permuted spectrum is Boolean.

While using both ACO and PSO for the cryptanalysis of simple substitution ciphers, we observed that PSO shows a better performance over ACO for solving optimization problems with a large permutation space. So, for the Boolean functions construction, we decided to focus only on using PSO.

5.3 Suitable Cost Function

While a few permutations, after Inverse Walsh Transform, will correspond to Boolean functions, most will not. Note that many permuted spectra may collapse to the same desired function. With each permutation Π, we can associate a cost that indicates how far $\Pi(W)$ is from the spectrum of a valid Boolean function.

For our work, we tried a couple of cost function suggested in [29].

As only a few parameters after inverse Walsh Transform of $\Pi(W)$, will correspond to Boolean functions, the inverse value of most of the permutations will not actually be +1 or -1. Let us consider the inverse Walsh Transform of those functions as $\hat{f} = \{f(0), f(1), \ldots\ldots, f(2^n - 1)\}$. The association f with a Boolean function $\hat{b}$, by choosing,

$$\hat{b}(i) = +1 \text{ if } \hat{f}(i) > 0$$

$$\hat{b}(i) = -1 \text{ if } \hat{f}(i) < 0$$

$$\hat{b}(i) = +1 \text{ or -1 (randomly chosen) if } \hat{f}(i) = 0$$

The following equation may used as the cost function.

$$Cost = \sum_{i=0}^{2^n-1} [\hat{f}(i) - \hat{b}(i)]^2$$

By this equation, each function $\hat{f}$ generated using optimization technique can be associated a cost that indicates how far is $\hat{f}$ from a legitimate Boolean function. This cost represents how far the Walsh spectrum $\Pi(W)$ is from a legitimate spectrum of a Boolean function. However, the problem of this cost function is that it does not directly represent the distance of the spectrum from a legitimate spectrum of a Boolean function and thus does not work properly for most of the cases when the number of inputs of Boolean functions grows more than 6. So, we changed our cost function to a more efficient one and use it throughout our experiment.

This cost function was developed using Titsworth's theorem [73]. This State that, $F(\omega)$ is a Walsh spectrum of a binary Boolean function if and only if,

$$\sum_{\omega} F(\omega)F(s \oplus \omega) = 2^{2n} \delta(s)$$

where, $s \in Z_2^n$ and $\delta(s)$ =1 if s =0 and $\delta(s)$ =0 otherwise. Clark *et al* in [29] suggested a cost function that punishes a deviation from the previous equation.

$$Cost = \sum_{s} (|\sum_{\omega} F(\omega)F(\omega \oplus s)|)^R$$

If $F(\omega)$ is the Walsh transform of a Boolean function f then when $s = 0$ the inner summation should be non-zero (and is constant for all permutations of the spectrum $F(\omega)$). For $s \neq 0$ the inner term should be zero. The value of R is determined by trail and error method. However, for most of our constructions of Boolean functions we used $R = 1$.

5.4 Reducing the Search Space

For a Boolean function with n input variables, the number of positions in the permutation space is $2^n!$. For input $n \geq 8$ the permutation search space increases to 2^8 !=256! and more. For any optimization technique, this permutation space is a huge one to handle. To reduce this huge permutation space and generate Boolean functions of input $n \geq 8$, we have used a technique of concatenating two or four functions to get the desired function.

For example, it is possible to construct an (n, m, n-m-1, 2^{n-1}-2^{m+1}) function where $m > \left[\frac{n}{2} - 2\right]$ from the concatenation of two (n-1, m, n-m-2, 2^{n-2}-2^{m+1}) or four (n-2, m, n-m-3, 2^{n-3}-2^{m+1}) functions with non overlapping Walsh coefficients, if such two or four functions exist.

Any two functions f and g are said to have non overlapping Walsh transform coefficients if $G(\omega) \neq 0 \Rightarrow F(\omega) = 0$ and $F(\omega) \neq 0 \Rightarrow G(\omega) = 0$.

5.5 Resilient Functions Construction

The spectrum of any (n, m, -, 2^{n-1}-2^{m+1}) function is necessarily three valued function $(0, \pm\lambda)$ where, $\lambda = 2^{m+2}$ noting that $m > \left[\frac{n}{2} - 2\right]$ [14]. In our work we consider all functions with resiliency degree $m > \left[\frac{n}{2} - 2\right]$. These functions with three valued spectrum are known as plateaued functions [74].

The algebraic degree of the function (n, m, -, 2^{n-1}-2^{m+1}) is always maximum and equal to *n-m-1* [32][33].

We have two types of positions corresponding to zero Walsh coefficients: the fixed position zeros and the non fixed position zeros. We also have numbers of positive and negative λ. The fixed position zeros are placed in the position in the starting spectrum corresponding to $0 \le \omega \le m$ (these element remain fixed throughout the search). These zeros are for the resiliency. Thus the number of fixed zeros is given by

$$N_{\text{fixed '0'}} = \sum_{i=0}^{m} \binom{n}{i}.$$

Then we put the non-fixed position zeros and the positive and negative λ in arbitrary positions in our search space.

From Parseval's inequality, we have

$$N_{+\lambda} + N_{-\lambda} = \left(\frac{2^{2n}}{\lambda^2}\right) = 2^{2n-2m-4}$$

By noting that $\sum_{w} F(w) = 2^{m+2}(N_{+\lambda} - N_{-\lambda}) = \pm 2^n$, then we get

$$N_{+\lambda} - N_{-\lambda} = \pm 2^{n-m-2}$$

The numbers of positive and negative coefficients are interchangeable. The number of non fixed position zeroes is given by

$$N_{\text{Non-fixed '0'}} = 2^n - (N_{\text{fixed '0'}} + N_{+\lambda} + N_{-\lambda}).$$

After putting all the values of the Walsh coefficients in the search space of particle swarm optimizer (PSO), we use the previously described cost function in PSO to permute the coefficients in such a way that the Inverse Walsh Transform of that spectrum corresponds to a Boolean function, and that Boolean function will be our desired Boolean function (n, m, *n-m-1*, 2^{n-1}-2^{m+1}). Note that in all these cases, we used modified PSO algorithm as described in chapter 4.

5.5.1 Construction of (6,2,3,24)

For input $n = 6$, when resiliency $m = 2$, the desired resilient function will be (6,2,3,24). The initial spectrum will contain 6 '16', 10 '-16' and 48 '0's. For retaining the resiliency degree $m = 2$ of the function, zeros in the positions corresponding to $0 \leq \omega \leq 2$ are made fixed throughout the search and the remaining Walsh values (i.e. 6 '16', 10 '-16' and 26 '0's) are arbitrarily allocated to the remaining positions. Then PSO is used to permute the above spectrum such the resulting spectrum corresponds to a Boolean function.

Example:

1110001110001001010010010111100100100110110101010111110000101010

5.5.2 Construction of (6,3,2,16)

For input $n = 6$, when resiliency $m = 2$, the desired resilient function will be (6,3,2,16). The initial spectrum will contain 1 '32', 3 '-32' and 60 '0's. For retaining the resiliency degree $m = 3$ of the function, zeros in the positions corresponding to $0 \leq \omega \leq 3$ are made fixed throughout the search. So the number of fixed zeros in this case is 42. The remaining Walsh values (i.e. 1 '32', 3 '-32' and 18 '0's) are arbitrarily allocated to the remaining positions. PSO is then used to permute the spectrum in such a way that the resulting spectrum corresponds to a Boolean function.

Example:

1001011010010110011010011001011001101001011010010110100110010110

5.5.3 Construction of (7,2,4,56)

The resilient function of input $n = 7$ and resiliency degree $m = 2$ is (7,2,4,56). This function can be constructed by concatenating two non overlapping (6,2,3,24). First, we use PSO to obtain a (6,2,3,24) function f_1 as described before. To construct the second (6,2,3,24) function f_2 which doesn't overlap f_1 we need to increase the number of fixed zeros by fixing zeros in the initial position of F_2, where F_1 has non- zero values (i.e. $F_1(\omega) \neq 0 \Rightarrow F_2(\omega) = 0.$). The remaining Walsh values (i.e. 6 '16', 10 '-16' and 10 '0's) are arbitrarily allocated to the remaining positions and use PSO to construct the non overlapping resilient function f_2. The concatenation of f_1 and f_2 generates a (7,2,4,56) function. Since it is not guaranteed that every f_1 may have a non overlapping f_2, if the search for f_2 failed after a pre-specified number of iterations, we have to start all again by generating a new f_1 function.

Example:

1110001110001001010010010111100100100110110101010111110000101010
1001011001001011111001000011100110100101011110000001101111000110

5.5.4 Construction of (7,3,3,48)

Using PSO with the initial spectrum that contains 10 '32', 6 '-32', 64 fixed '0's and 48 non-fixed '0's, we were able to construct a (7,3,3,48) function. We were also able to obtain (7,3,3,48) function by concatenating two (6,3,2,16) functions.

Example:

1001001101000111111010001001011001101001111010000001110100110110
0110110010111000000101111001011001101001000101111110001011001001

5.5.5 Construction of (7,4,2,32)

For (7,4,2,32) the initial spectrum consists with 1 '64', 3 '-64', 99 fixed '0's and 25 non-fixed '0's. Using PSO for permuting 1 '64', 3 '-64'and 25 non-fixed '0's we can directly construct a (7,4,2,32) function.

Example:

1100001100111100001111001100001101101001100101101001011001101001
1001011001101001011010011001011000111100110000111100001100111100

5.5.6 Construction of (8,3,4,112)

Two non overlapping (7,3,3,48) functions were constructed to construct a (8,3,4,112) function. For first (7,3,3,48) function the number of fixed '0's were 64 and for the second function the number of fixed '0's increases to 80. And the number of '32' and '-32' were 10 and 6 respectively all trough the construction.

Example:

1001001101000111111010001001011001101001111010000001110100110110
0110110010111000000101111001011001101001000101111110001011001001
1000111001110001011101001000101100111001110001100110100110010110
1001001101101100110000110011110011010100001010110010111011010001

5.5.7 Construction of (8,4,3,96)

The construction of (8,4,3,96) functions was done directly using the PSO. In this case, the initial spectrum consists with 6 '64', 10 '-64', 163 fixed '0's and 77 non-fixed '0's.

Other constructions were achieved by concatenating two (7,4,2,32) functions.

Example:

1001011010010110010110101010010110100101010110100110100101101001
0110100101101001101001010101101001011010101001011001011010010110
1001101001010110011001011010100101100101101010011001101001010110
1001010110100110011010100101100101101010010110011001010110100110

5.5.8 Construction of (9,3,5,240)

The construction of (9,3,5,240) function is done by concatenating four non-overlapping (7,3,3,48) functions. The construction procedure is summarized as follows:

1) Use PSO to obtain a (7, 3, 3, 48) function, f_1. The initial spectrum should contain 10 '+32', 6 '-32' and 112 '0's. Zeros are placed in the position in the starting spectrum corresponding to $0 \le \omega \le 3$. These element remain fixed throughout the search) and the remaining Walsh values (i.e. 10 '+32, 6 '-32' and 48 '0's) are arbitrarily allocated to the remaining positions.

2) To construct f_2 use the conditions in step 1 but with 32 nonfixed'0's with the extra condition that $F_1(\omega) \neq 0 \Rightarrow F_2(\omega) = 0$.

3) To construct f_3 use the conditions in step 2 but with 16 non-fixed '0's and the extra two conditions that $F_1(\omega) \neq 0 \Rightarrow F_3(\omega) = 0$, and $F_2(\omega) \neq 0 \Rightarrow F_3(\omega) = 0$

4) To construct f_4 use the conditions in step 3 but with all '0's fixed, in this case there are no non-fixed zeros and extra three conditions that $F_1(\omega) \neq 0 \Rightarrow F_4(\omega) = 0$, $F_2(\omega) \neq 0 \Rightarrow F_4(\omega) = 0$, and $F_3(\omega) \neq 0 \Rightarrow F_4(\omega) = 0$.

Since it is not guaranteed that for every (7, 3, 3, 48) we can find another three functions satisfying the above constraints, if the search failed after a predetermined number of steps, then go to step 1 and begin with another (7, 3, 3, 48) function.

Example:

1010100110011001010101100101101001101001011001011001011010100110
0110010110010110101001100110100101011010011010101001100110010101
1100001101101001100101100110100110010110100101101001011000111100
0011110011000011011010010011110000111100011010011100001111000011
1000010101111010110110100010010100111011110001000101100010100111
1110011000011001000100111110110001100100100110111010110101010010
1111000000100111000111011100101000001111110110001110001000110101
1000101101011100011001101011000101110100101000111001100101001110

5.5.9 Construction of (9,4,4,224)

The construction steps for (9,4,4,224) follows the same way of (9,3,5,240). In this case, the function is constructed using four non-overlapping (7,4,2,32) functions.

Example:

1100001100111100001111001100001101101001100101101001011001101001
1001011001101001011010011001011000111100110000111100001100111100
1001011001101001011001101001100110011001011001100110100110010110
1001011001101001100110010110011001100110100110010110100110010110
1001011010010110010110101010010110100101010110100110100101101001
0110100101101001101001010101101001011010101001011001011010010110
1001101001010110011001011010100101100101101010011001101001010110
1001010110100110011010100101100101101010010110011001010110100110

5.5.10 Construction of (10,4,5,480)

Similarly, (10,4,5,480) function is constructed using four non-overlapping (8,4,3,96) functions.

Example:

1100001100111100001111001100001101101001100101101001011001101001
1001011001101001011010011001011000111100110000111100001100111100
1001011001101001011001101001100110011001011001100110100110010110
1001011001101001100110010110011001100110100110010110100110010110
1001011010010110010110101010010110100101010110100110100101101001
0110100101101001101001010101101001011010101001011001011010010110
1001101001010110011001011010100101100101101010011001101001010110
1001010110100110011010100101100101101010010110011001010110100110

1001100000101011101011011101010001010110101100010011000010110110 1
1011011010000110010001101011010100010101110110101110101000011001
0110011110101000101001000101011101010010100111011001111010010010
0100100101111001101110010100101011101010001001010001010111100110
1001001111001001011011000011011001101100001101101001001111001001
0110110000110110100100111100100110010011110010010110110000110110
0001111011100001101101000100101111100001000111100100101110110100
1110000100011110010010111011010000011110111000011011010001001011

5.6 Bent Function Construction

The spectrum of a bent function is always a two valued ($\pm\lambda$) function. Where, $\lambda = 2^{\frac{n}{2}}$. For bent functions, the Number of $+\lambda$ ($N_{+\lambda}$) and $-\lambda$ ($N_{-\lambda}$) follows the following equations:

$$N_{+\lambda} = 2^{n-1} - 2^{\frac{n}{2}-1}$$

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

where, n is number of variables or input for the bent function. The numbers of positive and negative coefficients are interchangeable.

5.6.1 Construction of 6-variable Bent Function

For 6-varible bent function, the initial spectrum contains 36 ‘8’s and 28 ‘-8’s. Using PSO, we permute this spectrum with respect to the cost function described before to construct bent function of 6-variables.

Example:

0011010111110110100010111000010011100111011100011111001101010110

5.6.2 Construction of 8-variable Bent Function

For $n \geq 8$, it is hard to construct bent functions directly. So, to make the job easier, we concatenate two non overlapping functions with 7-variables. Both of these functions should be three valued plateaued function [74] with spectrum values $(0, \pm 16)$. For making the construction easier (by making the search space smaller) we start by constructing (7,2,4,56) function as described before. Note that for bent functions there are no resiliency constrains. As the resilient function (7,2,4,56) is a plateaued function and easy to construct we start with this function. Then for the second function we construct a (7,0,-,56) function . In the initial spectrum of (7,0,-,56) function we fixed the zeros in the non-zero positions of the previous spectrum but make the fixed zero position of the previous function spectrum non-fixed in the current spectrum as we do not need the resiliency constrain for this function. For both the cases the numbers of +16 and -16 were same, 28 and 36 respectively. Using PSO, we were able to find this type of functions and concatenating these two functions we got 8-variable bent function.

Example:

11011111111111101010011110000010000000001101110100000011100101010
10001000101110000001100000001100110000101000111010000110000000011011
1110001110001001010010010111100100100110110101010111110000101010
1001011001001011111001000011100110100101011110000001101111000110

5.6.3 Construction of 10-variable Bent Function

The construction of 10-variable Bent function starts with (7,3,3,48) functions. Using four non overlapping (7,3,3,48) functions, we constructed two non overlapping (8,3,4,112) functions. Then make the fixed zeros position for the resiliency degree 3 non-fixed, we construct a (8,0,-,112) using two non overlapping (7,0,-,48). Note that this two

(7,0,-,48) can not overlap the four (7,3,3,48) functions. Thus, these three functions of 8-variables do not overlap each other. Then we directly construct a (8,0,-,112) function which does not overlap any of the three previously constructed 8- variable functions. By concatenating the two (8,3,4,112) and two (8,0,-,112) functions which do not overlap each other, we constructed a 10-variable bent function. For the 7-variable functions the initial spectrum contains 10 '32' and 6 '-32'. For 8-variable functions the initial spectrum contains 28 '32' and 36 '-32'. In this case also starting with a resilient function for bent function construction is not needed. We use resilient functions to make the search space smaller and also these resilient functions are plateaued functions which are needed for this type of constructions.

Example:

1010100110011001010101100101101001101001011001011001011010100110
0110010110010110101001100110100101011010011010101001100110010101
1100011000111001101001100101100100110110110010011001010101101010
1101000100101110000110111110010000101110110100011101100000100111
1100001101111000001111000010110101001011101001011110000101011010
1001011010000111100101101101001001111000001111000010110111000011
1100011100110010001010011101110010011000011011010111011010000011
0011100011001101110101100010001101100111100100101000100101111100
1010101011111111000010100000101000000000001010101010111110101111
1010000011110101111100001111000010101111000001011111111100000000
1010101010101010000100010001000100100010111011100011001100000000
1110111000100010110011001111111101100110011001101110111011101110
1100110011000000111100111100001111111100111100001111111111001111
1100110011000000111100111100001111111100111100001111111111001111
0010010100110001001001010011000111101100010111011110110001011101
1110010110101011-1110010110101011111100000000010111111000000000001011

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, both ACO and PSO were used for automated cryptanalysis of classical simple substitution ciphers. Based on our experimental results, both PSO and ACO-based attacks proved to be very effective on various sets of encoding keys.

Previously, PSO was mostly used to solve vector space problems or permutation problem with small search space. In this thesis, we used a modified version of PSO for solving optimization problems in a large permutation space.

Our modified PSO optimization was used construct Boolean functions of cryptographic interest such as bent functions and resilient functions. While previous works have shown how such heuristic search could equal the best achievements of all theoretical constructions for eight or fewer inputs, mixing these techniques with some algebraic ideas, as we have done in this thesis, helps in extending the successful range of these search techniques.

Using these techniques, we affirmatively answered the open problem of the existence of the resilient function (9,3,5,240) by constructing several examples of this function.

6.2 Future Work

One main disadvantage of heuristic optimization techniques (including ACO and PSO) is its large sensitivity to parameter variations (e.g. ρ, α, β for ACO and c_1, c_2 for PSO). Although fine tuning of these parameters can be done by trial and error, it will be interesting to find analytical formula for the optimal (regions) of these parameters. Very recently, an adaptive version of PSO is proposed which is parameter free. The use of this type of optimization technique can add a more reliability flavor to these search technique and is worth more investigation.

In our cryptanalysis problem, we only considered pure uni-gram and bi-gram statistics. It is interesting to try tri-gram statistics in the evaluation function. One may also try to use a cost function that is based on a weighted combination of the different n-gram statistics. Finding the optimum weight is a challenging optimization problem given the fact that there are several other parameters (related to the optimization algorithm itself) that also need to be tuned.

It is also interesting to extend the application of these optimization techniques to other cryptographic problems such as the cryptanalysis of some of the newly proposed block or stream ciphers.

Solving other open problems related to cryptographic Boolean functions (e.g. the construction of $(10,2,-,488)$) is a natural extension for the work presented in this thesis.

References

[1] D. Kahn, "*The Codebreakers: The Story of Secret Writing,*" New York: Macmillan, 1967.

[2] http://www.ridex.co.uk/cryptology

[3] http://www.resonancepub.com/homecrypto.htm

[4] Ibraham A. "*Al-Kindi: The origins of cryptology: The Arab Contributions,*" Cryptologia, vol.16(2), pp. 97–126, 1992.

[5] C. E. Shannon, "*Communication Theory of Secrecy Systems,*" Bell Systems Technical Journal, vol.28, pp. 656–715, 1949.

[6] A. Menezes, P. Oorschot, S. Vanstone, "*Handbook of Applied Cryptography,*" Boca Raton, FL : CRC Press, 1996.

[7] Bruce Schneier, "*Applied Cryptography 2nd Edition,*" John Wiley & Sons, 1996.

[8] Mohammad Faisal Uddin and Amr M. Youssef, *"An Artificial Life Technique for Cryptanalysis of Simple Substitution Ciphers,"* Proc. of IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2006), Ottawa, May 2006.

[9] Mohammad Faisal Uddin and Amr M. Youssef, *"Cryptanalysis of Simple Substitution Ciphers Using Particle Swarm Optimization,"* Proc. of the Congress on Evolutionary Computation (CEC '06), pp.2692-2695, July 2006.

[10] Ziad Saber, Mohammad Faisal Uddin and Amr M. Youssef, *"On The Existence of (9,3,5,240) Resilient Functions,"* IEEE Transactions on Information Theory, vol. 52(5), pp. 2269-2270, May, 2006.

[11] Ziad Saber, Mohammad Faisal Uddin and Amr M. Youssef, "On Some Resilient Functions Constructions using PSO-based Spectral Inversion," Proc. of the 2006 IEEE on Swarm Intelligence Symposium, May 2006.

[12] A. Dimovski, D. Gligoroski . "*Generating Highly Nonlinear Boolean Functions Using a Genetic Algorithm,*" 6th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Service (TELSIKS 2003), vol. 2, pp.604 – 607, 2003.

[13] M. Matsui, *"Linear Cryptanalysis Method for DES Cipher,"* Advances in Cryptology: Proc. of EUROCRYPT '93, Berlin, pp. 386–397, Springer-Verlag, 1994.

[14] P. Sarkar and S. Maitra, *"Nonlinearity Bounds and Constructions of Resilient Boolean Functions,"* Proc. of Cypto 2000, pp. 516-533, LNCS 1880, Speinger-Verlag, 2000.

[15] W. Meier and O. Staffelbach, *"Nonlinearity Criteria for Cryptographic Functions,"* In Advances in Cryptology: Proc. of EUROCRYPTt '89, LNCS, vol. 434, pp. 549–562. Springer-Verlag, 1990.

[16] F. J. MacWilliams and N. J. A. Sloane, *"The Theory of Error Correcting Codes,"* North-Holland Publishing Company, Amsterdam, 1978.

[17] T. Siegenthaler, *"Corrolation Immunity of Nonlinear Combining Functions for Cryptographic Applications,"* IEEE Transactions on Information Theory, vol. IT-30, pp.776-780, oct. 1984.

[18]O. S . Rothaus, *"On Bent Functions,"* J. Combinatorial theory, vol. 20(A), pp. 300-305, 1976.

[19] C. Adams and S. Tavares, *"Generating and Counting Binary Bent Sequences,"* IEEE Transactions on Information Theory, vol. 36(5) pp.1170-1173, September, 1990.

[20] J. D. Olsen, R. A. Scholtz, and L. R. Welch, *"Bent-Function Sequences,"* IEEE Transactions on Information Theory, vol. IT-28(6), 1982 .

[21] R. Yarlagadda and J.E. Hershey, *"Analysis and Synthesis of Bent Sequences,"* Computers and Digital Techniques, IEE Proceedings E, vol. 136(2) , pp. 112 – 123, March 1989.

[22] C. Carlet, *"A Construction of Bent Functions,"* Finite Fields and Applications, London Mathmatical Society ,Lecture Series 233,Cambridge University Press, pp. 47-58, 1996.

[23] P. V. Kumar, R. A. Scholtz and L. R. Welch, *"Generalized Bent Functions and Their Properties,"* J. Combinatorial Theory A 40, pp. 90-107, 1985.

[24] C. Adams, *"Constructing Symmetric Ciphers Using the CAST Design Procedure,"* Designs, Codes and Cryptography, vol. 12(3), pp. 283-316, November 1997.

[25] X.Zhen, J.Massey, *"A Spectral Characterization of Corrolation-Immune Combining Functions,"* IEEE Transactions on Information Theory, vol. 34(3), May 1988.

[26] E. Pasalic, T. Johansson, S. Maitra, and P. Sarkar, *"New Constructions of Resilient and Correlation-Immune Boolean Functions achieving Upper Bounds on Nonlinearity,"* WCC2001 International Workshop on Coding and Cryptography, Electronic Notes in Discrete Mathematics, vol. 6, Elsevier Science, 2001.

[27] E. Pasalic, S. Maitra, *"Construction of Nonlinear Resilient Boolean Functions Using "Small" Affine Functions,"* IEEE Transactions on Information Theory, vol. 50(9), September 2004.

[28] Selcuk Kavut, Subhamoy Maitra and Melek D. Yucel, "*There exist Boolean functions on n (odd) variables having nonlinearity*> $2^{n-1}-2^{\frac{n-1}{2}}$ *if and only if n > 7,*" Cryptography ePrint Archive Report, http://eprint.iacr.org/2006/181.pdf

[29] J.A. Clark, J.L. Jacob, S. Maitra, P. Stanica. "*Almost Boolean Functions: The Design of Boolean Functions by Spectral Inversion,*" The Congress on Evolutionary Computation (CEC '03), vol. 3, pp. 2173 - 2180, 2003.

[30] S. Maitra, P. Sarkar, *"Construction of Nonlieanear Boolean Functions with Important Cryptographic Properties,"* Advances in Cryptology-EUROCRYPT 2000(Lecture Notes in Computer Science), vol. 2551 in Lecture Notes in Computer Science). Berlin, Germany, vol.1807, pp. 491-512, Springer Verlag, 2000.

[31] S. Maitra, and E. Pasalic, *"Further Construction of Resilient Boolean Fnction with Very High Nonlinearity,"* IEEE Transactions on Information Theory, vol.48, pp. 1825-1834, July 2002.

[32] P. Sarkar, *"Spectral Domain of Correlation-Immune and Resilient Boolean Functions ,"* Cryptography ePrint Archive Report, 2000/49, http://eprint.iacr.org./

[33] C.Carlet, *"On the Coset of Weight Divisibility and Nonlinearity of Resilient and Correlation Immune Functions,"* In Advances in Cryptography CRYPTO 1991, pp. 86-100, Springer Verlag, 1992.

[34] L. Aleksander, and H.Morton, "*An introduction to neural computing*", Chapman and Hall (1990)

[35] David E.Goldberg "*Genetic Algorithms in Search Optimization & Machine Learning*" Addison-Wesley Publishing Company, INC. 1989

[36] M. Dorigo, V. Maniezzo, and A. Colorni, "*The Ant System: Optimization by a Colony of Cooperating Agents,*" IEEE Transactions on Systems, Man, and Cybernetics. B, vol.26, no.2, pp. 29–41, 1996.

[37] R. C.Eberhart and J. A Kennedy "*New Optimizer using Particle Swarm Theory,*" Proc. of the Sixth Intenational Symposium on Micromachine and Human Science, Nagoya. Japan. pp. 39-43. 1995.

[38] J. Kennedy and R. Eberhart, "*Particle swarm optimization,*" Proc. of IEEE Int. Conf. Neural Networks, 1995, pp. 1942–1948.

[39] R. Beckers, J.L. Deneubourg and S. Goss, "*Trails and U-turns in the Selection of the Shortest Path by the Ant Lasius Niger,*" Journal of Theoretical Biology, vol.159, pp.397–415, 1992.

[40] D. E. Goldberg, "*Genetic Algorithms in Search, Optimization, and Machine Learning,*" Reading, MA: Addison-Wesley, 1989

[41] M. Clerc, "*The swarm and the queen: Toward a deterministic and adaptive particle swarm optimization,*" Proc. of The Congress on Evolutionary Computation (CEC '99), vol. 3, 1999, p. 1957.

[42] M. Clerc and J.Kennedy, "*The particle swarm—Explosion, stability, and convergence in a multi-dimensional complex space,*" Proc. of The Congress on Evolutionary Computation (CEC' 02), vol. 6, pp. 58–73, Feb. 2002.

[43] Vladimiro Miranda and Nuno Fonseca, "*EPSO-Evolutionary Particle Swarm Optimization, a new algorithm with applications in power systems,*" on Proc. of IEEE T&D Asia-Pacific, vol. 2, pp.745-750,2002.

[44] Vladimiro Miranda and Nuno Fonseca, "*EPSO-Best of Two Worlds Meta-Heuristic applied to power system problems,*" The Congress on Evolutionary Computation (CEC '02), Vol. 2, pp.1080- 1085 ,2002.

[45] Silvio Turrini, "*Optimization in Permutation Spaces,*" Western Research Laboratory, Research Report 96/1, November 1996

[46] Andrew Clerk, "*Optimisation Heuristics for Cryptology,*" PhD thesis, Queensland University of Technology, 1998.

[47] S. Peleg and A. Rosenfeld, "*Breaking substitution ciphers using a relaxation algorithm,*" Communications of the ACM, vol. 22(11), pp.598–605, 1979.

[48] J. Carrol and S. Martin, "*The automated cryptanalysis of substitution ciphers,*" Cryptologia, vol.10(4), pp.193–209, 1986.

[49] W. S. Forsyth and R. Safavi-Naini, "*Automated cryptanalysis of substitution ciphers,*" Cryptologia, vol.17(4), pp.407–418, 1993.

[50] R. Spillman, M. Janssen, B. Nelson and M. Kepner, "*Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers,*" Cryptologia, vol.17(1), pp.31–44, 1993.

[51] D. Bahler and J. King, "*An implementation of probabilistic relaxation in the cryptanalysis of simple substitution systems,*" Cryptologia, vol.16(3), pp.219–225, 1992.

[52] M. Lucks, "*A constraint satisfaction algorithm for the automated decryption of simple substitution Ciphers,*" In Proceedings of CRYPTO'88, pp. 132–144, 1988.

[53] G. W. Hart, "*To decode short cryptograms,*" Communications of the ACM, vol.37(9), pp.102–108, 1994.

[54] Thomas Jakobsen, "*A fast method for cryptanalysis of substitution ciphers,*" Cryptologia, pp. 265-274, July 1995.

[55] John M. Carroll and Lynda Robbins. *The automated cryptanalysis of polyalphabetic ciphers,*" Cryptologia, 11(3):193–205, July 1987.

[56] John C. King, "*An algorithm for the complete automated cryptanalysis of periodic polyalphabetic substitution ciphers,*" Cryptologia, 18(4):332–355, October 1994.

[57] Robert A. J. Matthews, "*The use of genetic algorithms in cryptanalysis,*" Cryptologia, 17(2):187–201, April 1993.

[58] M.D. Russell, J.A. Clark, and S. Stepney, "*Making the most of two heuristics: breaking transposition ciphers with ants,*" The Congress on Evolutionary Computation (CEC '03), Vol. 4, pp.2653- 2658, 2003.

[59] J.C. Hernandez, P. Isasi, A. Ribagorda. "*Easing collision finding in cryptographic primitives with genetic algorithms,*" The Congress on Evolutionary Computation (CEC '02), Vol. 1, pp. 535 – 539, 2002.

[60] J.C. Hernandez, P. Isasi ."*Finding efficient distinguishers for cryptographic mappings, with an application to the block cipher TEA,*" The Congress on Evolutionary Computation (CEC '03), Vol. 3, pp.2189 – 2193, 2003.

[61] J.C. Hernandez, P. Isasi . "*New results on the genetic cryptanalysis of TEA and reduced-round versions of XTEA,*" The Congress on Evolutionary Computation (CEC '04), Vol. 2, pp. 2124 - 2129, 2004.

[62] R. Forre, "*Methods and Instruments for Designing S-boxes,*" Journal of Cryptology, 2(3):115-130, 1990.

[63] W. Millan. A. Clark and E. Dawson. "*Smart Hill Climhing Finds Better Boolean Functions,*" Workshop on Selected Areas in Cryptography 1997 (SAC'97), Workshop Record, pp 50-63, 1997.

[64] W. Millan, A. Clark and E. Dawson, "*Heuristic Design of Cryptographically Strong Balanced Boolean Functions*," Advances in Cryptology, Proceedings of EUROCRYPT'98, LNCS vol 1403, page 489, Springer-Verlag.

[65] W. Millan and A. Clark and E. Dawson, "*Boolean Function Design Using Hill Climbing Methods,*" 4th Australasian Conference on Information Security and Privacy, ACISP'99, LNCS vol 1587, page 1, Springer-Verlag.

[66] W. Millan, A. Clark and E. Dawson, "*An effective genetic algorithm for finding highly nonlinear Boolean functions*," First International Conference on Information and Communications Security, ICICS'97, LNCS vol 1334, page149, Springer-Verlag.

[67] J.A. Clark and J.L. Jacob, "*Two-Stage Optimisation in the Design of Boolean Functions*," Proc. of ACISP 2000, LNCS vol 1841, pages 242-254, Springer-Verlag. 2000.

[68] J.A. Clark, J.L. Jacob and S. Stepney, "*The Design of S-Boxes by Simulated Annealing,*" The Congress on Evolutionary Computation (CEC '04), Vol. 2, pp. 1533 - 1537, 2004.

[69] J.A. Clark, J.L. Jacob and S. Stepney. "*Searching for cost functions,*" The Congress on Evolutionary Computation (CEC '04), Vol. 2, pp. 1517 - 1524, 2004.

[70] W. Pang, K.Wang, C. Zhou, L. Dong, M. Liu, H. Zhang, and J. Wang, "*Modified particle swarm optimization based on space transformation for solving traveling salesman problem,*" Proc. of 2004 International Conference on Machine Learning and Cybernetics, 2004. vol. 4, 26-29, pp.2342 – 2346, 2004.

[71] L. Cagnina, S. Esquivel, S, and R. Gallard, "*Particle swarm optimization for sequencing problems: a case study,*" Congress onEvolutionary Computation, CEC2004. vol 1, pp. 536 – 541, 2004.

[72] Xiaohui Hu, R.C. Eberhart, and Yuhui Shi, "*Swarm intelligence for permutation optimization: a case study of n-queens problem,*" Proc. of the 2003 IEEE on Swarm Intelligence Symposium, pp.243-246, 2003.

[73] C. Ding , G. Xiao and W. Shan. "*The Stability Theory of Stream Ciphers,*" Number 561 in Lecture Notes in Computer Science. Springer-Verlag. 1991.

[74] Xian-Mo Zhang and Yuliang Zheng, "*Plateaued functions,*" Proc. of International Conference on Information and Communications Security, ICICS'99, Sydney, November 1999, LNCS 1726, Springer - Verlag, pp. 284-300, 1999.