

Republic of Iraq  
Ministry of Higher Education  
and Scientific Research  
University of Baghdad  
College of Science



*Using Swarm Intelligence in Cryptanalysis of Nonlinear Stream Cipher  
Cryptosystem*

A THESIS  
SUBMITTED TO THE COLLEGE OF SCIENCE AT THE UNIVERSITY OF  
BAGHDAD IN PARTIAL FULFILLMENT OF THE REQUIRMENTS FOR THE  
DEGREE OF MASTER OF SCIENCE IN DEPARTMENT OF APPLIED  
SCIENCES OF THE UNIVERSITY OF APPLIED MATHEMATICS

**BY**

**FATIN ABDUL RAHMAN HAMEED**

**SUPERVISED BY**

**Asst. Professor Dr. AYAD GHAZI NASER AI- SHAMMRY**

2017

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

وَأَوْحَىٰ رَبُّكَ إِلَى النَّحْلِ أَنِ اتَّخِذِي مِنَ الْجِبَالِ بُيُوتًا وَمِنَ



الشَّجَرِ وَمِمَّا يَعْرِشُونَ

صَدَقَ اللَّهُ الْعَظِيمُ

سُورَةُ النَّحْلِ آيَةُ (٦٨)

### **Supervisor Certification**

I certify that this thesis entitled “**Using Swarm Intelligence in Cryptanalysis of Nonlinear Stream Cipher Cryptosystem**” by **Fatin Abdul Rahman Hameed** was prepared under my supervision at the University of Baghdad / College of Science / Department of Mathematics in a partial fulfillment of the requirement for the degree of Master of Science in Applied Mathematics.

**Signature:**

**Name: Dr. Ayad Ghazi Naser Al- Shammry**

**Title: Assistant Professor**

**Date:     /     / 2017**

In view of the available recommendation, I forward this thesis for debate by the examining committee.

**Signature:**

**Name:**

**Title:**

**Date:     /     / 2017**

**Committee Certification**

We certify that we have read this thesis entitled “ **Using Swarm Intelligence in Cryptanalysis of Nonlinear Stream Cipher Cryptosystem** ” and as examining committee, examined the student “**Fatin Abdul Rahman Hameed**” on its content, and that in our opinion, it is adequate for the partial fulfillment of the requirements for the degree of Master of Science in Applied Mathematics.

**Signature:**

**Name:**

**Title:**

**Date:**

(Chairman)

**Signature:**

**Name:**

**Title:**

**Date:**

(Member)

**Signature:**

**Name:**

**Title:**

**Date:**

(Member)

Supervisor)

**Signature:**

**Name:**

**Title:**

**Date:**

(Member and

Approved by the Dean of the College of Science.

**Signature:**

**Name:**

**Title:**

**Date:**

TO

**My Beloved Father ... Mother,**

**Dear Husband,**

**and**

**My Children**

**With Love and Appreciation**

*Fatin*

## **Abstract**

The Swarm intelligence is a real an innovative computational way for solving some kinds of hard problems. This way is simulated by the behavior of social animals and insects such as bird flocks, fish schools, colonies of ants and hony bees etc.

The Particle swarm optimization (PSO) mimics the behavior of a swarm of bird flocks and school of fish. The Swarm behavior is modeled by number of particles in multidimensional space that have two parts: a position and a velocity.

Stream ciphers considered an important class of encryption algorithms. The Shift register sequences can be used in both cryptography and coding theory. There is a big wealth of theory about stream ciphers based on shift registers, which have been the workhorse of military cryptography since the beginnings of electronics.

This thesis aims to implement cryptanalysis system on stream cipher cryptosystems called PSO Cryptnslysis System (PSOCS) using probable word plaintext attack, choosing three study cases, single Linear Feedback Shift Register (LFSR), which considered as a basic unit of stream cipher systems, and Linear cryptosystem and Threshold generator (as nonlinear cryptosystem) in the performance of PSO by find the actual solution of the System of Linear Equations (SLE) for any number of variables of the output of LFSR.

The application of the proposed PSOSC divided into two stages, first, constructing SLE's for the combined LFSR's in the genrator, and the second, is attacking the variables of SLE's which they are also the initial key values the of LFSR's. This thesis shows the good performance of PSOCS in finding the actual key of the three study cases.

The results of this research are implemented in Delphi version 10.0 visual programming languages exploiting the object oriented tools of this language.

# Table of Contents

|  |   |    |
|--|---|----|
| <b>Chapter One</b>                         | <b>Basic Mathematical Concepts</b>                |    |
| 1.1 Introduction .....                     |   | 1  |
| 1.2 Number Theory .....                    |   | 1  |
| 1.2.1 Primarily .....                      |   | 1  |
| 1.2.2 Multiplicatively.....                |   | 2  |
| 1.2.3 Divisibility .....                   |   | 2  |
| 1.3 Arithmetic Functions .....             |   | 4  |
| 1.4 Prime Number.....                      |   | 5  |
| 1.5 Modular.....                           |   | 5  |
| 1.6 Group Theory.....                      |   | 7  |
| 1.7 Boolean Ring and Boolean Algebra ..... |   | 10 |
| 1.8 Logic Circuits .....                   |   | 12 |
| 1.9 Sequences and Series .....             |   | 14 |
| 1.9.1 Sequences.....                       |   | 14 |
| 1.9.2 Series .....                         |   | 14 |
| 1.10 Literature Survey .....               |   | 15 |
| 1.11 Objective of Thesis .....             |   | 17 |
| 1.12 Thesis Outlines .....                 |   | 18 |
| <br><b>Chapter Two</b>                     | <br><b>Cryptography and Stream Cipher Systems</b> |    |
| 2.1 Introduction.....                      |   | 20 |
| 2.2 Terminology.....                       |   | 20 |

|   |        |          |               |
|---|--------|----------|---------------|
| 2.3   |        |          |               |
| Cryptosystems.....                                  |        |          | 21            |
| 2.3.1 Public Key Cryptographic System.....          |        |          | 22            |
| 2.3.2   | Secret | Key      | Cryptographic |
| Systems.....  |        |          | 23            |
| 2.4   | Linear | Feedback | Shift         |
| Register.....                                       |        |          | 25            |
| 2.4 Basic Building-Blocks of Stream Ciphers.....    |        |          | 18            |
| 2.5 Stream Cipher Systems.....                      |        |          | 28            |
| 2.6 Combination Generator.....                      |        |          | 29            |
| 2.7 Examples of Known Generators.....               |        |          | 31            |
| 2.7.1 Linear Generator.....                         |        |          | 31            |
| 2.7.2 Product Generator .....                       |        |          | 31            |
| 2.7.3 Threshold Generator .....                     |        |          | 32            |
| 2.8 Cryptanalysis.....                              |        |          | 33            |
| 2.9 Attacking of Stream Cipher Methods.....         |        |          | 34            |
| 2.9.1 Classical Cryptanalysis Methods.....          |        |          | 36            |
| 2.9.2 Modern Cryptanalysis Methods.....             |        |          | 38            |
| 2.10 Adversarial Models.....                        |        |          |               |
| 2.10.1 <u>Ciphertext-only Attacks</u> .....         |        |          | 40            |
| 2.10.2 Known-plaintext Attacks.....                 |        |          | 40            |
| 2.10.3 Chosen-plaintext Attacks .....               |        |          | 40            |
| 2.10.4 Chosen-ciphertext Attacks.....               |        |          | 40            |
| 2.10.5 Adaptively Chosen-plaintext Attacks .....    |        |          | 41            |
| 2.10.6 Adaptively Chosen-ciphertext Attacks.. ..... |        |          | 41            |
| 2.10.7 Related-key Attacks .....                    |        |          | 41            |
| 2.11 Types of Attacks .....                         |        |          | 41            |
| 2.11.1 Exhaustive Search Attack .....               |        |          | 42            |
| 2.11.2 Algebraic Attack .....                       |        |          | 42            |



|  |    |
|--|----|
| 2.11.3 Correlation Attack .....          | 43 |
| 2.11.4 Fault Attack .....                | 43 |
| 2.11.5 Chosen-IV Attack .....            | 44 |
| 2.11.6 Slide Attack .....                | 44 |
| 2.11.7 Cube Attack.....                  | 45 |
| 2.11.8 Time-Memory Trade-off Attack..... | 45 |
| 2.11.9 Guess and Determine Attack .....  | 46 |

**Chapter Three          Particle swarm optimization**

**Chapter Four          Using swarm Intelligence in  
Cryptanalysis of nonlinear  
stream cipher cryptosystem**

**Chapter Five          Conclusions and Future Work**

**References**

## List of Symbols

| Symbol                 | Description   |
|------------------------|---|
| $B_i$ ,                | the observed number of blocks                             |
| $e_i$                  | simple events number $i$ .                                |
| $e_k$                  | encryption key  |
| $E_i$                  | Expected value of occurrence of outcome $i$               |
| $d_k$                  | decryption key  |
| $f$                    | Function $f$  |
| $F_n$                  | finite field with $n$ elements                            |
| $G_i$                  | The observed number of gaps                               |
| $\gcd(a,b)$            | greatest common divisor of $a$ and $b$                    |
| $\text{lcm}(a,b)$      | least common multiple of $a$ and $b$                      |
| $p$                    | Prime number  |
| $S_i$                  | the sequence $i$  |
| $n$ -KG                | Key Generator consists of $n$ LFSR's                      |
| $\oplus$               | Exclusive-OR (XOR)  |
| $s_i$                  | The element $i$ of the sequence $S$                       |
| $S_r$                  | Sequence generates from LFSR $r$                          |
| $S_L$                  | the sequences generates from the linear systems           |
| $S_B$                  | the sequences generates from the Brüer systems            |
| $S_P$                  | the sequences generates from the product systems          |
| $\langle G, * \rangle$ | mathematical system consists of set $G$ and operation $*$ |
| $\neq$                 | Not Congruent   |
|                        |   |
|                        |   |
|                        |   |

## List of Abbreviations

| <b>Abbreviatio</b> | <b>Meaning</b>                         |
|--------------------|--|
| AES                | Advanced Encryption Standard           |
| BDT                | Binary Derivative Test                 |
| C                  | Complex number                         |
| C                  | Ciphertext space                       |
| CE                 | Complementary Event                    |
| CF                 | Combining Function                     |
| CI                 | Correlation Immunity                   |
| D                  | Decryption process (algorithm)         |
| DES                | Data Encryption Standard               |
| E                  | Encryption process (algorithm)         |
| eSTREAM            | ECRYPT Stream Cipher Project.          |
| FSE                | Fast Software Encryption Conference    |
| GF                 | Galois field                           |
| KG                 | Key Generator                          |
| L                  | Length of Sequence                     |
| LC                 | Linear Complexity                      |
| LFSR               | Linear Feedback Shift Register         |
| M                  | Message space                          |
| MBSRT              | Main Binary Standard Randomness Tests  |
| MLFSR              | Maximal Linear Feedback Shift Register |
| N                  | Natural number                         |
| NLFSR              | Nonlinear Feedback Shift Register      |
| R                  | Real Number                            |
| RNG                | Random Number Generator                |
| S                  | Sequence                               |
| w.r.t.             | with respect to                        |
| Z                  | Integer number                         |
|                    |  |
|                    |  |
|                    |  |
|                    |  |
|                    |  |

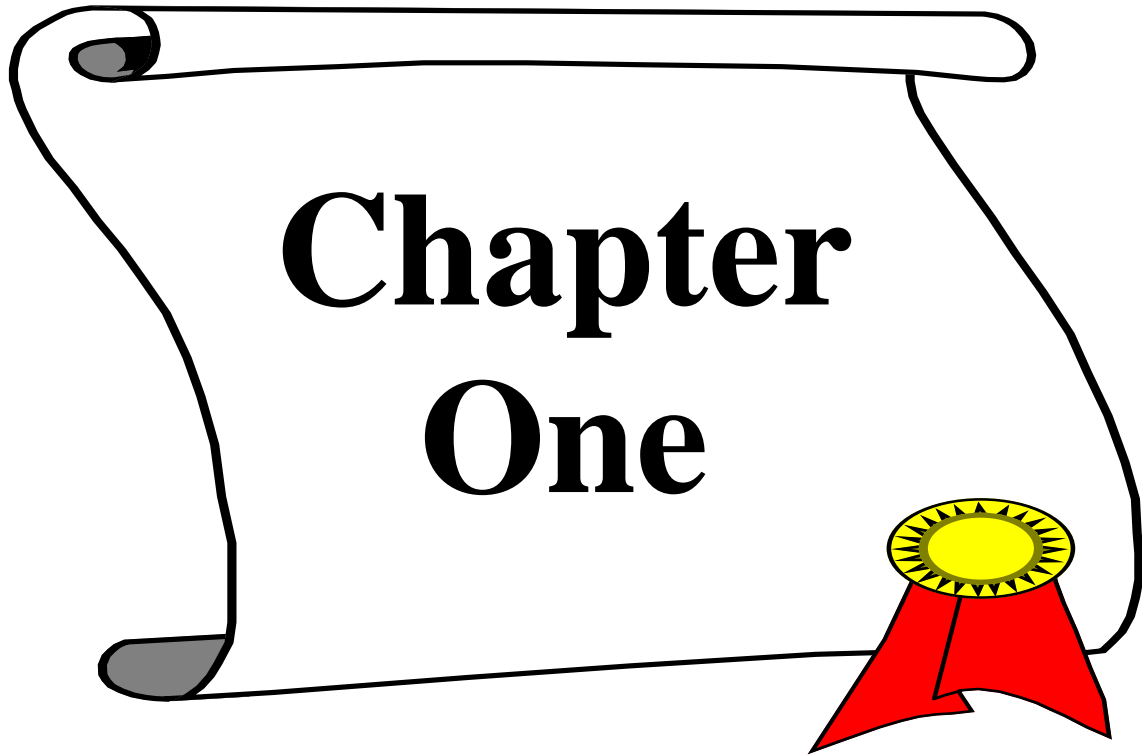
## List of Figures

---

### List of Tables

| <b>Table No.</b> | <b>Table Description</b>                                   | <b>Page No.</b> |
|------------------|--|-----------------|
| (1.1)            | The addition and multiplication for $F_5$                  | 8               |
| (1.2)            | The truth tables of the four gates                         | 11              |
| (3.1)            | the most common Parameters of PSO                          | 52              |
| (4.1)            | SLSCC with length $L=4$                                    | 54              |
| (4.2)            | The XOR relation for LSCC, $n=2$                           | 58              |
| (4.3)            | The 3-LFSR's of NTSCC Generator                            | 64              |
| (4.4)            | SLE for SRSCC with binary representation                   | 69              |
| (4.5)            | SLE for SLSCC with binary representation                   | 69              |
| (4.6)            | SLE for STSCC with binary representation                   | 70              |
| (4.7)            | Parameters selection of PSOCS                              | 75              |
| (4.8)            | PSOCS initialization of random initial particles for SLSCC | 76              |
| (4.9)            | The PSOCS results after 10 generations for SLCSS           | 77              |
| (4.10)           | The PSOCS results after 10 generations for SLCSS           | 77              |
| (4.11)           | The PSOCS finds actual key after 45 generations for SLCSS  | 78              |
| (4.12)           | Results of applying PSOSC on SLSCC                         | 78              |
| (4.13)           | PSOCS initialization of random initial particles for LSCC  | 79              |
| (4.14)           | Results of applying PSOSC on SLSCC                         | 80              |
| (4.15)           | PSOCS initialization of random initial particles for NTSCC | 81              |
| (4.16)           | Results of applying PSOSC on NTSCC                         | 81              |

| <b>Section No.</b> | <b>Figure No.</b> | <b>Figure Description</b>                      | <b>Page No.</b> |
|--------------------|-------------------|--|-----------------|
| 1.8                | (1.1)             | The boolean gates                              | <b>13</b>       |
| 2.2                | (2.1)             | Encryption Process                             | <b>21</b>       |
| 2.3                | (2.2)             | Cryptosystems classification                   | <b>22</b>       |
| 2.3.1              | (2.3)             | Modern public-key cryptosystem, $e_k \neq d_k$ | <b>23</b>       |
| 2.3.2              | (2.4)             | Conventional secret-key cryptosystems          | <b>23</b>       |
| 2.4                | (2.5)             | Feedback Shift Register                        | <b>25</b>       |
| 2.4                | (2.6)             | Linear Feedback Shift Register                 | <b>26</b>       |
| 2.5                | (2.7)             | A stream (Bit) cipher                          | 28              |
| 2.6                | (2.8)             | n-LFSR's Generator with Combining Function     | 30              |
| 2.7.1              | (2.9)             | n-Linear Generator                             | 31              |
| 2.7.2              | (2.10)            | n-Product Generator                            | 32              |
| 2.7.3              | (2.11)            | Threshold generator                            | 32              |
| 2.9                | (2.12)            | Attacking methods of Stream Cipher             | 35              |
|                    |                   |  |                 |
|                    |                   |  |                 |
|                    |                   |  |                 |
|                    |                   |  |                 |
|                    |                   |  |                 |



**BASIC  
MATHEMATICAL  
CONCEPTS**

## Chapter One

### Basic Mathematical concepts

#### 1.1 Introduction

The growth of the Internet has made government intelligence and police agencies nervous. They say that widely available encryption software could make wiretapping more difficult; their common reaction is to try to restrict the strength of encryption algorithms or require that spare copies of the keys are available somewhere for them to seize.

Cryptography is the study of mathematical techniques which are related to the aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication [1].

This chapter introduces the basic concepts in different fields in mathematics, which the cryptography and cryptanalysis are needed, specially, in stream cipher systems.

#### 1.2 Number Theory

**Number theory**, in mathematics, is primarily the theory of the properties of integers (whole numbers) such as parity, **divisibility**, **primarily**, **additively**, and **multiplicatively**, etc. In the next subsections we will investigate more detailed discussions about numbers.

##### *1.2.1 Primarily [2]*

**Definition (1.1):** A positive integer  $n > 1$  that has only two distinct factors, 1 and  $n$  itself (when these are different), is called **prime**; otherwise, it is called **composite**. The first few prime numbers are: 2,3,5,7,11,13,17,....

##### **Remark (1.1):**

1. It is interesting to note that primes thin out: there are eight up through 20, but only three between 80 and 100.
2. Note that 2 is the only even prime, all the rest are odd.

### 1.2.2 Multiplicatively

**Theorem (1.1):** [3] (the fundamental theorem of arithmetic)

Any positive integer  $n > 1$  can be written uniquely in the following prime factorization form:

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k} = \prod_{i=1}^k p_i^{\alpha_i} \quad \dots(1.1)$$

where  $p_1 < p_2 < \dots < p_k$  are primes, and  $\alpha_1, \alpha_2, \dots, \alpha_k$  are positive integers.

**Example (1.1)** [4]: The following are prime factorization of  $n$  for  $n=1999, 2000, \dots, 2010$ .

$$\begin{array}{lll} 1999 = 1999 & , & 2000 = 2^4 \cdot 5^3 & , & 2001 = 3 \cdot 23 \cdot 29 \\ 2002 = 2 \cdot 7 \cdot 11 \cdot 13 & , & 2003 = 2003 & , & 2004 = 2^3 \cdot 3 \cdot 167 \\ 2005 = 5 \cdot 401 & , & 2006 = 2 \cdot 17 \cdot 59 & , & 2007 = 3^2 \cdot 223 \\ 2008 = 2^3 \cdot 251 & , & 2009 = 7^2 \cdot 41 & , & 2010 = 2 \cdot 3 \cdot 5 \cdot 67 \end{array}$$

### 1.2.3 Divisibility

**Definition (1.2)**[5]: Let  $a$  and  $b$  be two integers, not both zero. The largest divisor  $d$  s.t.  $d|a$  and  $d|b$  is called the **greatest common divisor** (gcd) of  $a$  and  $b$ , which is denoted by  $\gcd(a,b)$ .

**Definition (1.3)**[5]: Let  $a$  and  $b$  be two integers, not both zero.  $d$  is a common multiple of  $a$  and  $b$ , the least common multiple (lcm) of  $a$  and  $b$ , is the **smallest common multiple**, which is denoted by  $\text{lcm}(a,b)$ .

**Definition (1.4)**[5]: Integers  $a$  and  $b$  are called **relatively prime** if  $\gcd(a,b)=1$ . we say that integers  $n_1, n_2, \dots, n_k$  are relatively prime if, whenever  $i \neq j$ , we have  $\gcd(n_i, n_j)=1, \forall i, j, 1 \leq i, j \leq k$ .

**Theorem (1.2)**[3]: Suppose  $a$  and  $b$  are two positive integers.

$$\text{If } a = \prod_{i=1}^k p_i^{\alpha_i} \text{ and } b = \prod_{i=1}^k p_i^{\beta_i}, \text{ then}$$



$$\gcd(a,b)=\prod_{i=1}^k p_i^{\varepsilon_i}, \text{ where } \varepsilon_i=\min(\alpha_i,\beta_i), \forall i, 1\leq i\leq k.$$

$$\text{lcm}(a,b)=\prod_{i=1}^k p_i^{\delta_i}, \text{ where } \delta_i=\max(\alpha_i,\beta_i), \forall i, 1\leq i\leq k.$$

**Theorem (1.3)**[4]: Suppose a and b are two positive integers.

If  $a=\prod_{i=1}^k p_i^{\alpha_i}$  and  $b=\prod_{i=1}^k p_i^{\beta_i}$ , then

$$\gcd(a,b)=\prod_{i=1}^k p_i^{\varepsilon_i}, \text{ where } \varepsilon_i=\min(\alpha_i,\beta_i), \forall i, 1\leq i\leq k.$$

$$\text{lcm}(a,b)=\prod_{i=1}^k p_i^{\delta_i}, \text{ where } \delta_i=\max(\alpha_i,\beta_i), \forall i, 1\leq i\leq k.$$

**Theorem (1.4)** [3]: Suppose a and b are two positive integers, then:

$$\text{lcm}(a,b)=\frac{a.b}{\gcd(a,b)}.$$

**Example (1.2)**: Since the prime factorization of 240 and 560 are:

$240=2^4.3.5$  and  $560=2^4.5.7$ , then the:

$$\gcd(240,560)=2^{\min(4,4)}.3^{\min(1,0)}.5^{\min(1,1)}.7^{\min(0,1)}=2^4.3^0.5^1.7^0=80.$$

$$\text{lcm}(240,560)=2^{\max(4,4)}.3^{\max(1,0)}.5^{\max(1,1)}.7^{\max(0,1)}=2^4.3^1.5^1.7^1=1680.$$

### **1.3 Arithmetic Functions** [6]

Arithmetic (or number theoretic) functions are the most fundamental functions in mathematics and computer science; for example, the computable functions studied in mathematical logic and computer science are actually arithmetic functions. In this section we shall study some basic arithmetic functions that are useful in number theory.

**Definition (1.5)** [7]: A **function**  $f$  is a rule that assigns to each element in a set  $D$  (called **Domain** of  $f$ ) one and only one element in a set  $B$ . the set of images called the **range** ( $R$ ) of  $f$ .

**Definition(1.6)**[8]: **Inverse of function** it's a relation from set  $Y$  (Codomain) to set  $D$  (Domain), such that.

$$f^{-1}:Y \rightarrow D$$

Where  $f$  is function on a set  $D$  to set  $Y$ ; let  $b \in Y$  the inverse of  $b$  denoted by  $f^{-1}(b)$  consist of these elements in  $D$  which mapped onto  $b$ , such that.

$$f^{-1}(b) = [x \in D, f(x) = b].$$

**Definition(1.7)**[9]: **Linear and Non-linear Function**

algebraically: any polynomial with highest degree equal to one ( $f(x) = mx + n$ ), where  $m$  and  $n$  are constant it's a **linear function**, Otherwise it's a **non-linear function**.

**Definition (1.8):** [7] A function  $f$  is called an **arithmetic function** or a **number theoretic** function if it assigns to each positive integer  $n$  a unique real or complex number  $f(n)$ . Typically, an arithmetic function is a real-valued function whose domain is the set of positive integer.

**Example (1.3):** the equation  $\sqrt{n}$ ,  $n \in \mathbb{N}$ , defines an arithmetic function  $f$  which assigns the real number  $\sqrt{n}$  to each positive integer.

## 1.4 Prime Number

**Definition (1.9)** [10]: An integer  $p$  is called a **Prime Number** if  $p$  greater than one and if the only positive dividing  $p$  are one and  $p$ .

**Example (1.4):** 2, 3, 5, 7 ..., 31, ..., 41, 43, ..., 61, ..., 67, ...

**Definition (1.10)** [11]: (**The Fundamental Theorem of Arithmetic**)

any integer positive number  $n$  greater than one can be written uniquely in the following prime factorization,  $n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \dots \cdot p_k^{\alpha_k}$ ,

where  $p_1 < p_2 < \dots < p_k$  are prime numbers,  $\alpha_1, \alpha_2 \dots \alpha_k$  are positive integer numbers.

## **1.5 Modular**

**Definition (1.11) [12]: divisor**

any non-zero integer number  $b$  satisfying the property

$$a = b \cdot m; (a, b, m \in \mathbb{Z}^+)$$

And  $b$  divided  $a$  with no remainder then  $b$  called divisor or factor of  $a$  and denoted by  $b|a$ .

**Example (1.5):** 1,2,3,4,6,8,12,24 are factors of 24.

**Definition (1.12) [13]: Modular Arithmetic**

The **modular arithmetic** is a system of arithmetic for integers uses a finite number values, where the values “warp around” until reaching to a certain value the modulus.

$$a + b \text{ mod } n \equiv [a \text{ mod } n + b \text{ mod } n] \text{ mod } n.$$

**Definition (1.13) [14]: Greatest Common Divisor (G.C.D)**

For any two positive integer numbers  $(a, b)$  the greatest common divisor (G.C.D) of them is the greatest number  $(d)$  that divisor of  $a$  and  $b$  ( $d|a$  and  $d|b$ )

**Definition (1.14) [14]: Smallest Common Multiple (L.C.M)**

The Smallest common Multiple (L.C.M) of  $a$  and  $b$ , is the smallest integer  $c$ , That  $a|c$  and  $b|c$ .

**Remark (1.2):** if  $G.C.D(a, b) = 1$  then  $a, b$  are called relatively prime.

**Theorem (1.5) [13]** suppose  $a$  and  $b$  are two positive integer numbers if

$$a = \prod_{i=1}^k p_i \alpha_i \quad \text{and} \quad b = \prod_{i=1}^k p_i \beta_i \text{ then}$$

$$G.C.D(a, b) = \prod_{i=1}^k p_i \gamma_i \text{ Where } \gamma_i = \min(\alpha_i, \beta_i) \text{ for every } i \text{ such that } 1 \leq i \leq k$$

L. C. M (a, b) =  $\prod_{i=1}^k p_i \mu_i$  Where  $\mu_i = \max(\alpha_i, \beta_i)$  for every  $i$  such that  $1 \leq i \leq k$

Where  $p_i$  are prime numbers and  $\alpha_i, \beta_i$  are non-negative numbers.

**Example(1.6):** G. C. D (240,560) = 80, and

L. C. M (240,560) = 1680

$240 = 2^4 * 3 * 5$  and  $560 = 2^4 * 5 * 7$

According to theorem (1.5)

$$\begin{aligned} \text{G. C. D (240,560)} &= 2^{\min(4,4)} \cdot 3^{\min(1,0)} \cdot 5^{\min(1,1)} \cdot 7^{\min(1,0)} \\ &= 2^4 \cdot 3^0 \cdot 5^1 \cdot 7^0 \\ &= 80 \end{aligned}$$

$$\begin{aligned} \text{L. C. M (240,560)} &= 2^{\max(4,4)} \cdot 3^{\max(1,0)} \cdot 5^{\max(1,1)} \cdot 7^{\max(1,0)} \\ &= 2^4 \cdot 3^1 \cdot 5^1 \cdot 7^1 \\ &= 1680. \end{aligned}$$

## **1.6 Group Theory** [15, 16]

### **Definition (1.15):**

1.  $Z_{>a}$  is the set of positive integers greater than a:

$$Z_{>a} = \{a+1, a+2, \dots\}.$$

2. The set of all residue classes modulo a positive integer denoted by  $Z_n$ :

$$Z_n = \{0, 1, 2, \dots, n-1\}.$$

### **Definition (1.16): Binary Operation**

A binary operation  $*$  on a set  $A$  is a rule that assigns to each ordered pair  $(a, b)$  of elements of  $A$  a unique element of  $A$ .

**Example (1.7):** Ordinary addition  $+$  and multiplication  $\cdot$  are binary operations on  $N, Z, R,$  or  $C$ .

### **Definition (1.17): Group**

A **group**, denoted by  $\langle G, * \rangle$  (or  $(G, *)$ ), or simply  $G$ , is a  $G \neq \emptyset$  of elements together with a binary operation  $*$ , s.t. the following axioms are satisfied:

1. **Closure:**  $a*b \in G, \forall a, b \in G$ .
2. **Associativity:**  $(a*b)*c = a*(b*c), \forall a, b, c \in G$ .
3. **Existence of identity:**  $\exists!$  element  $e \in G$ , called the identity, s.t.

$$e*a = a*e = a, \forall a \in G.$$

4. **Existence of inverse:**  $\forall a \in G, \exists!$  Element  $b \in G$ , s.t.

$$a*b = b*a = e. \text{ This } b \text{ is denoted by } a^{-1} \text{ and called the } \mathbf{inverse} \text{ of } a.$$

The group  $\langle G, * \rangle$  is called **commutative (abelian)** group if it satisfies further axiom:

5. **Commutatively:**  $a*b = b*a, \forall a, b \in G$ .

**Example (1.8):** the set  $Z^+$  with operation  $+$  is not group ( $\exists$  no identity element), and it's not group with operation  $\cdot$  ( $\exists$  no inverse element in  $Z^+$ ).

### **Definition (1.18): Additive and Multiplicative Group**

1. If the binary operation of a group is  $+$ , then the identity of group is 0 and the inverse of  $a \in G$  is  $-a$ ; this said to be an **additive group**.
2. If the binary operation of a group is  $\cdot$ , then the identity of a group is 1 or  $e$ , this group is said to be **multiplicative group**.

### **Definition (1.19): Finite Group**

A group is called a finite group if it has finite number of elements; otherwise it is called an infinite group.

**Definition (1.20):** The **order** of the group  $G$ , denoted by  $|G|$  (or by  $\#(G)$ ) is the number of elements of  $G$ .

**Example (1.9)**: the order of  $Z$  is  $|Z|=\infty$ .

**Definition (1.21): Subgroup**

Let  $a \in G$ , where  $G$  is multiplicative group. The elements  $a^r$ , where  $r$  is an integer, form a subgroup of  $G$ , called the subgroup generated by  $a$ .

**Definition (1.22): Cyclic Group**

A group  $G$  is **cyclic** if  $\exists a \in G$  s.t. the subgroup generated by  $a$  is the whole of  $G$ .

If  $G$  is a finite cyclic group with identity element  $e$ , the set of elements  $G$  may be written  $\{e, a, a^2, \dots, a^{n-1}\}$ , where  $a^n = e$  and  $n$  is the smallest such positive integer.

**Definition (1.23)[17]: Field**

A field by  $\langle F, \oplus, \otimes \rangle$  (or  $(F, \oplus, \otimes)$ ) or simply  $F$ , is abelian group w.r.t. addition, and  $F - \{0\}$  is abelian w.r.t. to multiplication.

**Definition (1.24)[17]: Finite Field**

A finite field is a field that has a finite number of elements in it; we call the number the order of the field.

**Theorem (1.6)[17]: Prime Power**

$\exists$  a field of order  $q$  iff  $q$  is prime power (i.e.  $q = p^r$ ) with  $p$  prime and  $r \in \mathbb{N}$ .

**Remark (1.3)[17]: Galois field**

A field of order  $q$  with  $q$  prime power is called Galois field and is denoted by  $GF(q)$  or just  $F_q$ .

**Example (1.7)[4]**: The finite field  $F_5$  has elements  $\{0, 1, 2, 3, 4\}$  and is described by the table(1.1) addition and multiplication table.

Table (1.1) The addition and multiplication for  $F_5$ .

| $\oplus$ | 0 | 1 | 2 | 3 | 4 |
|----------|---|---|---|---|---|
| 0        | 0 | 1 | 2 | 3 | 4 |
| 1        | 1 | 2 | 3 | 4 | 0 |
| 2        | 2 | 3 | 4 | 0 | 1 |
| 3        | 3 | 4 | 0 | 1 | 2 |
| 4        | 4 | 0 | 1 | 2 | 3 |

| $\otimes$ | 1 | 2 | 3 | 4 |
|-----------|---|---|---|---|
| 1         | 1 | 2 | 3 | 4 |
| 2         | 2 | 4 | 1 | 3 |
| 3         | 3 | 1 | 4 | 2 |
| 4         | 4 | 3 | 2 | 1 |

## 1.7 Boolean Ring and Boolean Algebra

**Definition (1.25)**[18]: Let  $A \neq \emptyset$  be a set,  $f$  be a binary operation on a set  $A$  ( $f: A \times A \rightarrow A$ ), we call the pair  $(A, f)$  as **mathematical system**.

**Definition (1.26)** [18]: Let  $X$  be the universal set, and let  $A$  and  $B$  be two subsets of  $X$ , then:

1. The operation  $+$  defined as  $A+B=A \cup B$ .
2. The operation  $\oplus$  defined on the power  $P(X)$  set of  $X$  by:

$$A \oplus B = (A - B) \cup (B - A) \text{ s.t. } A - B = A \cap B', B' \text{ is the complement set of } B.$$

The operation  $\oplus$  called Exclusive-OR (XOR) (or the symmetric difference).

3. The operation  $\cdot$  defined as  $A \cdot B = A \cap B$ .

**Definition (1.27)** [18]: **Boolean Ring**

Let  $(R, +, \cdot)$  be a ring with identity element, if the **Idempotency law** be satisfied  $a^2 = a, \forall a \in R$ , then the ring called **Boolean ring**.

**Example (1.8)**: Let  $P(X)$  represents the set of all the subsets of the universal set  $X$ , then the ring  $(P(X), \oplus, \cdot)$  is Boolean ring.

**Definition (1.28)** [18]: In Boolean ring  $(B, \oplus, \cdot)$ , we defined:

1. **Complement:**  $\bar{a} = a \oplus 1, \forall a \in B.$
2. **Sum (OR):**  $a + b = a \oplus b \oplus a \cdot b \forall a, b \in B.$

**Definition (1.29)** [18]: **Boolean Algebra**

The Boolean algebra is the mathematical system  $(B, \vee, \wedge)$  where  $B \neq \emptyset$ , and the binary operations  $\vee$  and  $\wedge$  defined on  $B$  as follows:

1. The operations  $\vee$  and  $\wedge$  are commutative.
2. The operations  $\vee$  and  $\wedge$  are satisfy the distribution law for each to other.
3.  $\exists$  two identity distinct elements 0 and 1 of the operations  $\vee$  and  $\wedge$  respectively s.t.  $a \vee 0 = a$  and  $a \wedge 1 = a, \forall a \in B.$

**Example (1.9)**: The system  $(P(X), \cup, \cap)$  is boolean algebra,  $X \neq \emptyset$ , we use  $\emptyset = 0$  and  $X = 1$ . If  $B$  be a set of subsets of  $X$  including  $\emptyset$  and  $X$  which is closed on  $\cup$  and complement then  $(B, \cup, \cap)$  is boolean algebra too.

**Theorem (1.7)**[19]: Every boolean algebra  $(B, \vee, \wedge)$  is boolean ring  $(B, \oplus, \cdot)$  when we defined the operations  $\oplus$  and  $\cdot$  as follows:

1.  $a \oplus b = (a \wedge b') \vee (a' \wedge b).$

2.  $a \cdot b = a \wedge b.$

$\forall a, b \in B.$

**Theorem (1.8)** [19]: Every ring  $(B, \oplus, \cdot)$  is Boolean algebra  $(B, \vee, \wedge)$  when we defined  $\vee$  and  $\wedge$  as follows:  $\forall a, b \in B.$

1.  $a \vee b = a \oplus b \oplus a \cdot b.$

2.  $a \wedge b = a \cdot b.$



**Theorem (1.9)**[19]: The ring  $(\mathbb{Z}_p, \oplus, \otimes)$  is field iff  $p$  is prime number s.t.

$$a \oplus b = a + b \pmod{p}.$$

$$a \otimes b = a \cdot b \pmod{p}.$$

This field is Galois field and is denoted by  $GF(p)$ ,  $\forall a, b \in \mathbb{Z}_p$ .

## 1.8 Logic Circuits[19]

In electronically logical circuits (which are subject to the Boolean algebra), there are small circuits called Gates which are, for example, part from transistors, diodes, capacitors, and etc, these gates are shown in figure (1.1):

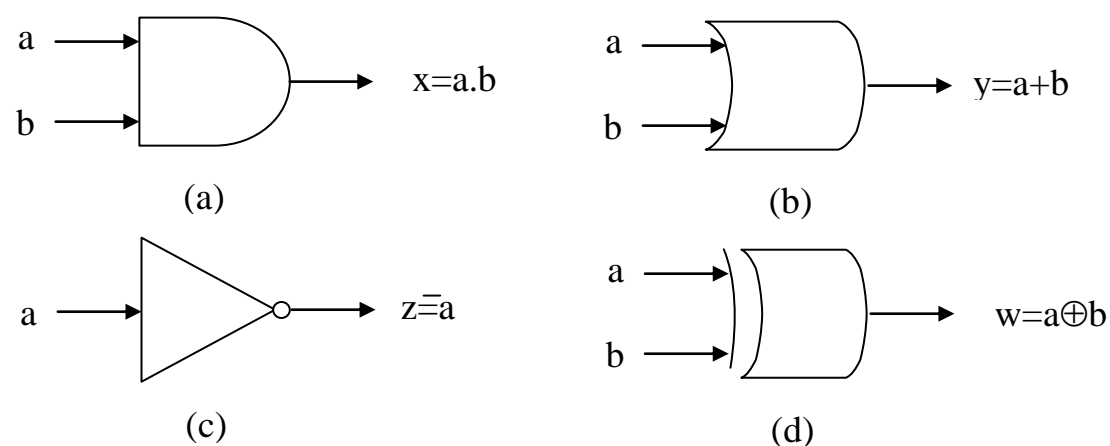


Figure (1.1) The boolean gates.

(a).The gate AND: is multiplying the input variables.

(b).The gate OR: summing the input variables.

(c).The gate NOT: complement of the input variable.

(d).The gate XOR: summing XOR the input variables.

These gates are shown in the table (1-2).

Table (1.2) The truth tables of the four gates.

|   |   |   |
|---|---|---|
| • | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 0 | 1 |

|   |   |   |
|---|---|---|
| + | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 1 |

|   |           |
|---|-----------|
| A | $\bar{a}$ |
| 0 | 1         |
| 1 | 0         |

|          |   |   |
|----------|---|---|
| $\oplus$ | 0 | 1 |
| 0        | 0 | 1 |
| 1        | 1 | 0 |

**Definition (1.30):** The logical function  $f$  is called the **output function** defined  $f: B^n \rightarrow B$ , where  $B^n$  is a set of  $n$  input binary data,  $f$  subject to the Boolean algebra laws and we can apply the gates concepts on it, s.t.  $x=f \cdot g$ ,  $y=f+g$ ,  $z=f$ , and  $w=f \oplus g$ , where  $f$  and  $g$  are Boolean functions.

## 1.9 Sequences and Series [20]

### 1.9.1 Sequences

**Definition (1.31): Sequence**

The sequence in the field  $F$  is a function  $f$ , whose domain is the set of non-negative (or could be positive) integer, s.t.  $f: Z \rightarrow F$ , and its denoted by  $S = \{S_n\}_{n=0}^{+\infty}$ .

**Definition (1.32):** The Sequence  $S$  is **periodic** when  $\exists p \in Z^+$  s.t.  $s_0=s_p, s_1=s_{p+1}, \dots$ , the minimum  $p$  is the **period** of  $S$ .

If  $Z_m = \{0, 1, \dots, m-1\}$ , where  $m \in Z^+$ , then  $S$  is digital sequence. In special case, if  $m=2$  then  $S$  is binary sequence, and if  $m=3$  then  $S$  is tri sequence.

### 1.9.2 Series

**Definition (1.33):** An infinite series is an expression of the form:

$$u_1 + u_2 + \dots + u_k + \dots = \sum_{k=1}^{\infty} u_k .$$

Let  $S_n$  denotes the sum of the first  $n$  terms of the series s.t.

$S_n = \sum_{k=1}^n u_k$ , and  $\{S_n\}_{n=1}^{+\infty}$  is called the **sequence of partial sums**.

$S = \sum_{k=1}^{\infty} u_k$  is called the **sum** of the series.

**Theorem(1.10)**: A geometric series  $a+ar+ar^2+\dots+ar^{k-1}+\dots(a \neq 0)$  is converges if  $|r| < 1$  and the sum is  $\frac{a}{1-r} = a+ar+ar^2+\dots+ar^{k-1}+\dots$ , and diverges if  $|r| \geq 1$ .

## **1.10 Literature Survey**

While there are have been many papers written on PSO and their application to various problems, there are relatively few papers that apply PSO to cryptanalysis.

In 1995, Kennedy J. and Eberhart R. introduced a concept for the optimization of nonlinear functions using particle swarm methodology. The evolution of several paradigms outlined, and an implementation of one of the paradigms had been discussed. Kennedy and Eberhart proposed Benchmark testing of the paradigm which describes the applications including nonlinear function optimization and neural network training. The described relationship is between particle swarm optimization and both artificial life and genetic algorithms [21].

In 1999 Eberhart R.C. and Hu X. arranged a new method for the analysis of human tremor using particle swarm optimization which is used to evolve a neural network that distinguishes between normal subject and those with tremor. Inputs to the neural network are normalized movement amplitudes obtained from an actigraph system.

The results from this preliminary investigation are quite promising, and the work is continuing [22].

In 2002, Venter G. and Sobieszczanski J. demonstrated the application of Particle Swarm Optimization to a realistic Multidisciplinary Optimization test problem. A new contribution to multidisciplinary optimization is the application of a new algorithm for dealing with the unique challenges associated with multidisciplinary optimization problems [23].

In 2002, Parsopoulos K. E. and Vrahatis M. N. adopted a new method PSO technique for the alleviation of local minima, and for detecting multiple minimizers are also described. Moreover, results on the ability of the PSO in tackling Multiobjective, Minimax and Integer Programming, as well as problems in noisy and continuously changing environments, are reported. Finally, a Composite PSO, in which the heuristic parameters of PSO are controlled by a Differential Evolution algorithm during the optimization that is described, and results for many well-known and widely used test functions are given [24].

In 2003, Rahmat-Samii Y., Gies D. , and Robinson J. are adopted a conceptual overview of the algorithm, keeping at all times the focus on implementing Particle Swarm Optimization to solve practical problems. In this vein, a selection of recent applications of the method to real-world design examples will also be presented [25].

On 2<sup>nd</sup> February 2003, Bliss L. adopted a new method which extends ANN training to include topological optimization of the network, especially the reduction of the number of inputs to the ANN [26].

On 6<sup>th</sup> may 2003, much work has been done by Bliss L. in the area of configuring ANN topology automatically using soft computing techniques such as GA. However, little time has been spent by researches on selecting the proper inputs to the ANN. NN used to predict the behavior of dynamical systems often have a choice of input information, much of which is redundant. Selecting a minimal set of inputs that produce acceptable behavior results in a lower cost solution. Researchers using trial and error methods currently do this input selection manually. The work shows several methods of automatically selecting a small set of inputs from a large candidate population. PSO is the primary network optimization technique used for processing ANN configuration [27].

In February 2004, Shi Y. surveyed the research and development of PSO in five categories: algorithms, topology, parameters, hybrid PSO algorithms, and applications. There are certainly other research works on PSO which are not included due to the space limitation. In general, the search process of a PSO algorithm should be a process consisted of both contraction and expansion so that it could have the ability to escape from local minima, and eventually find good enough solutions. A mathematical foundation of PSO needs to have a deep understanding of the dynamic process of PSO. There is also a need for a unique representation of the PSO topology and a need for a standard set of benchmark functions so that researchers can duplicate each other's work and compare their work with the others [28].

There are more literatures survey will be discussed in details chapter three.

## **1.11 Objective of Thesis**

The objective of this thesis is to investigate and evaluate one of the searching methods called Particle Swarm Optimization (PSO) and how

this algorithm can be implemented. The main goal of this thesis is to attack the stream cipher systems, specifically. The cryptanalysis issue includes attack the non-linear equations system and the output key sequence (which is known to the cryptanalyst) to retrieve the initial key values of LFSR(s) which generate this sequence. We propose a PSO cryptanalysis system to find the initial key of each LFSR which consists of particularized for construction the mathematical representation of nonlinear equation systems of each cryptosystem.

## **1.12 Thesis Outlines**

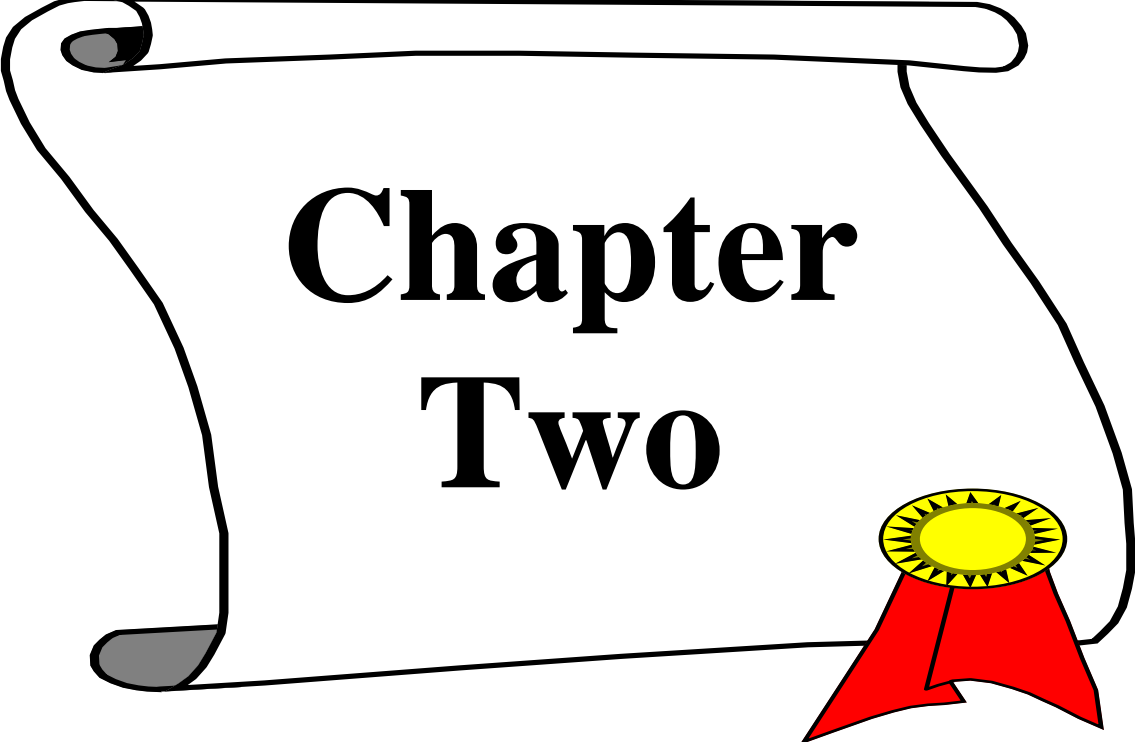
The solution of our problem has been covered through five chapters, the first three chapters are considered to provide the main concepts of this thesis, while the proposed system, practical work and the obtained results are explained in chapter four. Finally we introduced the conclusions and future work in the chapter five.

**Chapter Two:** deals with cryptology, classical cryptosystem types (simple substitution cipher and simple transposition cipher, cryptanalysis and cryptanalytic tools for classical cryptosystems. And shows the importance of stream cipher system and its classification, then showing some classical and modern cryptanalysis methods. Lastly, we will illustrate the types of attack.

**Chapter Three:** Presents the background of swarm intelligence and its applications, the background of PSO algorithm, biological collective behavior, and its applications. And the differences between PSO and classical methods. The operators and parameters of PSO are detailed in this chapter. In addition, the most application fields of PSO.

**Chapter Four:** covers the use of PSO algorithm as a cryptanalysis tool to attack such classical cryptosystems, and shows the experimental results for the implementation of the basic algorithm. We introduce how we can put the output results for each cryptosystem as a nonlinear equations system then put a new method to attack this systems depending on PSO.

**Chapter Five:** This chapter includes a conclusions and future work.

A black and white line drawing of a scroll, partially unrolled, with a red ribbon seal tied around it. The seal has a yellow sunburst design in the center.

# Chapter Two

## **CRYPTOGRAPHY AND STREAM CIPHER SYSTEMS**



# Chapter TWO

## Cryptography and Stream Cipher Systems

### 2.1 Introduction

**Cryptography** means hidden writing, the practice of using encryption to conceal text. A **Cryptanalyst** studies encryption and encrypted messages, with the goal of finding the hidden meanings of the messages. Both a cryptographer and a cryptanalyst attempt to translate coded material to its original form; normally a cryptographer works on behalf of a legitimate sender or receiver, while a cryptanalyst works on behalf of an unauthorized interceptor. Finally, **Cryptology** is the research into and study of encryption and decryption; it includes both cryptography and cryptanalysis [1].

This chapter introduces a description for the most important terminology in cryptography and some important known stream cipher systems which are depend on linear feedback shift register which are considered as a basic unit of stream cipher systems.

### 2.2 Terminology

**Cryptography** (from the Greek Kryptós, “**hidden**” and gráphein, “**to write**”) is the study of principles and techniques by which information can be concealed in ciphertexts and later revealed by legitimates users employing the secret key, but in which it is either impossible or computationally infeasible for an unauthorized person to do so. **Cryptanalysis** (from the Greek Kryptós, and analyéin “**to loosen**”) is the science (and art) of recovering information from ciphertexts without knowledge of the key. Both terms are subordinate to the more general

term **Cryptology** (from the Greek Kryptós, and logos, “word”). The cryptography concerned in **Encryption** and **Decryption** processes [1].

Now we have to present some important notations:

- **Message space M**: a set of strings (plaintext messages) over some alphabet, that needs to be encrypted.
- **Ciphertext space C**: a set of strings (ciphertexts) over some alphabet that has been encrypted.
- **Key space K**: a set of strings (keys) over some alphabet, which includes the encryption key  $e_k$  and the decryption key  $d_k$ .
- The **Encryption** process (algorithm) E:  $Ee_k(M)=C$ .
- The **Decryption** process (algorithm) D:  $Dd_k(C)=M$ .

The algorithms E and D must have the property that:

$$Dd_k(C)=Dd_k(Ee_k(M))=M.$$

The above situations shown in figure (2.1).



Figure (2.1) Encryption Process.

## **2.3 Cryptosystems**

The **Cryptosystem** are the systems which use the encryption and decryption processes, these systems can be classified as in figure (2.2).

Whenever **Cryptanalysis** is the science and study of methods of breaking ciphers. It is a system identification problem, and the goal of **Cryptography** is to build systems that are hard to identify [2]. To attack a cryptographic system successfully the cryptanalysis is forced to be based on subtle approaches, such as knowledge of at least part of the text encrypted, knowledge of characteristic features of the language used,..., with some

luck. However, in practice, some of this information may be inaccurate, imprecise, or missing, which, in turn, causes to decrease the possibility of attacking and increasing the time or the resources required by the analyst.

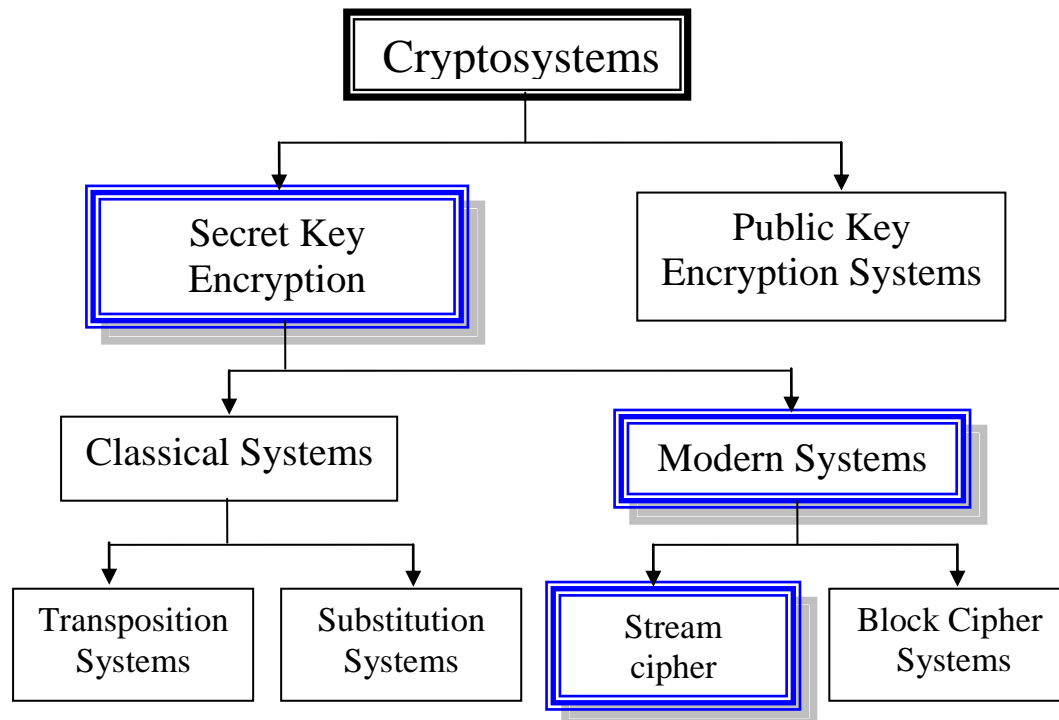


Figure (2.2) Cryptosystems classification

There are essentially two different types of cryptographic systems (cryptosystems), these cryptosystems are described in the next two subsections. [3]

### **2.3.1 Public Key Cryptographic System**

It is also called **asymmetric cryptosystems**. In a public key (**non-secret key**) cryptosystem (see figure (2.3)), the encryption key  $e_k$  and decryption key  $d_k$  are different, that is  $e_k \neq d_k$ .

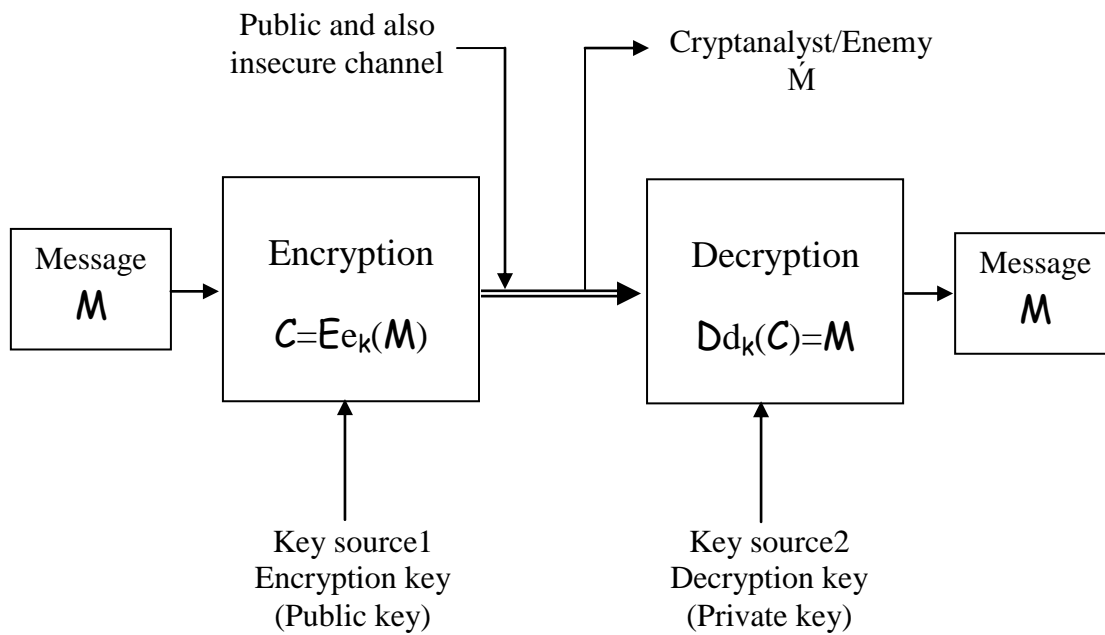


Figure (2.3) Modern public-key cryptosystem,  $e_k \neq d_k$ .

### 2.3.2 Secret Key Cryptographic System

It's also called **symmetric cryptosystems**. In a conventional secret-key cryptosystem (see figure (2.4)), the same key ( $e_k = d_k = k \in K$ ), called **secret key**, used in both encryption and decryption; we are interest in this type of cryptosystems.

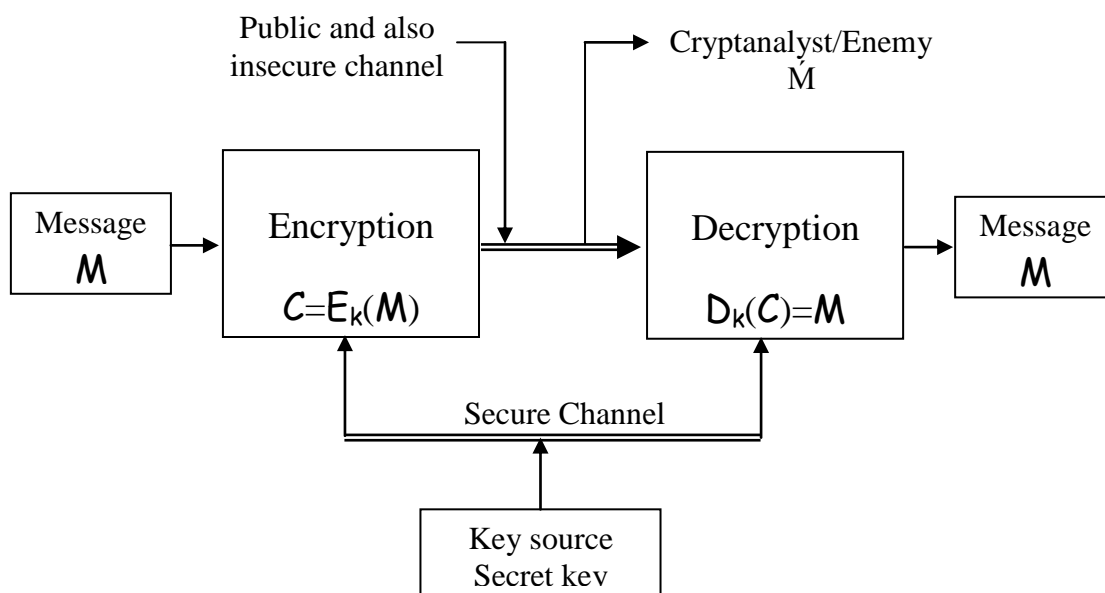


Figure (2.4) Conventional secret-key cryptosystems

The sender uses an invertible transformation  $f$  defined by:

$$f: M \xrightarrow{k} C$$

So produce the ciphertext:

$$c = (E_k(m)), m \in M \text{ and } c \in C.$$

and transmits it over the public insecure channel to the receiver. The key  $k$  should also be transmitted to the legitimate receiver for decryption but via a secure channel since the legitimate receiver knows the key  $k$ , he can decrypt  $c$  by transformation  $f^{-1}$  defined by:

$$f^{-1}: C \xrightarrow{k} M$$

and obtain:

$$D_k(c) = D_k(E_k(m)) = m, c \in C \text{ and } m \in M,$$

and it's the original plaintext message.

There are many different types of secret key cryptosystems. In what follows, we shall introduce some of these systems. [4]

### **I. Stream (Bit) Ciphers**

This kind of secret key cryptosystems is our interest, so we will detail it in the next section.

### **II. Monographic (Character) Ciphers**

Earlier ciphers (cryptosystems) were based on transforming each letter of the plaintext into a different letter to produce the ciphertext. Such ciphers are called **character substitution or monographic ciphers**, since each letter is shifted individually to another letter by a substitution.

### **III. Polygraphic (Block) Ciphers**

**Monographic ciphers** can be made more secure by splitting the plaintext into groups of letters (rather than a single letter) and then performing the encryption and decryption on these groups of letters.

This block technique is called **block ciphering**. Block cipher is also called a **polygraphic cipher**.

## ***2.4 Linear Feedback Shift Register***

A **feedback shift register** is made up of two parts: a shift register and a **feedback function** (see figure (2.5)). The shift register is a sequence of bits, (the length of a shift register is figured in bits). Each time a bit is needed, all of the bits in the shift register are shifted 1 bit to the right. The new left-most bit is computed as a function of the other bits in the register. The output of the shift register is 1 bit, often the least significant bit. The period of a shift register is the length of the output sequence before it starts repeating.

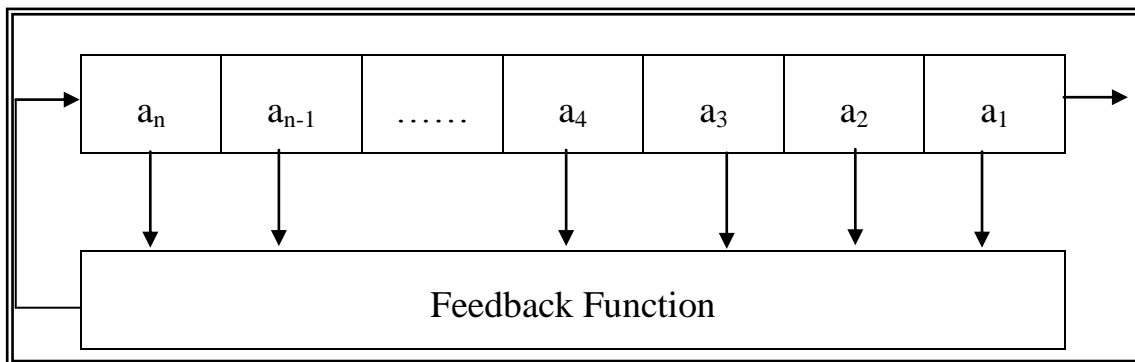


Figure (2.5) Feedback Shift Register.

Cryptographers have liked stream ciphers made up of shift registers: They are easily implemented in digital hardware. We will only touch on the mathematical theory.

Ernst Selmer, the Norwegian governments' chief cryptographer, worked out the theory of shift register sequences in 1965 [5]. Solomon Golomb, an NSA mathematician, wrote a book with Selmers results and some of his own [6].

The simplest kind of feedback shift register is a **Linear Feedback Shift Register** (LFSR), as described in figure (2.6). The feedback

function is simply the XOR of certain bits in the register; the list of these bits is called a **tap sequence**. Because of the simple feedback sequence, a large body of mathematical theory can be applied to analyzing LFSRs. Cryptographers like to analyze sequences to convince themselves that they are random enough to be secure. LFSR's are the most common type of shift registers used in cryptography.

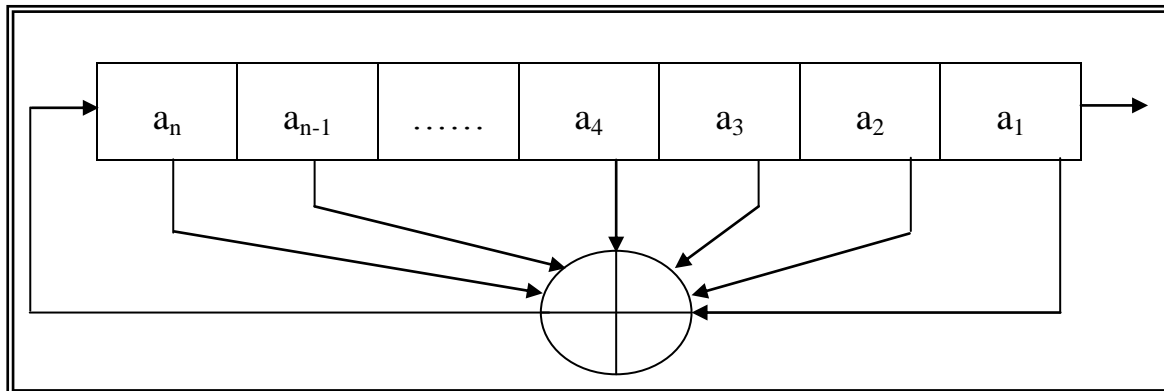


Figure (2.6) Linear Feedback Shift Register.

In order for a particular LFSR to be a maximal-period LFSR, the polynomial formed from a tap sequence plus the constant 1 must be a primitive polynomial mod 2. The degree of the polynomial is the length of the shift register. A primitive polynomial of degree  $n$  is an irreducible polynomial.

Linear feedback shift registers (LFSRs) are used in many of the keystream generators that have been proposed in the literature. There are several reasons for this [7]:

1. LFSRs are well-suited to hardware implementation.
2. They can produce sequences of large period.
3. They can produce sequences with good statistical properties.
4. Because of their structure, they can be readily analyzed using algebraic techniques.

A linear feedback shift register (LFSR) of length  $L$  consists of  $L$  stages (or delay elements) numbered  $0, 1, \dots, L-1$ , each capable of storing one bit and having one input and one output; and a clock which controls the movement of data. During each unit of time the following operations are performed:

1. The content of stage 0 is output and forms part of the output sequence.
2. The content of stage  $i$  is moved to stage  $i-1$  for each  $i$ ,  $1 \leq i \leq L-1$ .
3. The new content of stage  $L-1$  is the feedback bit  $s_j$  which is calculated by adding together modulo 2 the previous contents of a fixed subset of stages  $0, 1, \dots, L-1$ .

Every output sequence (i.e., for all possible initial states) of an LFSR  $\langle L, C(D) \rangle$  is periodic if and only if the connection polynomial  $C(D)$  has degree  $L$ .

If  $C(D) \in \mathbb{Z}_2[D]$  is a primitive polynomial of degree  $L$ , then  $\langle L, C(D) \rangle$  is called a **maximum-length LFSR**. The output of a maximum-length LFSR with non-zero initial state is called an **m-sequence**.

The basic approach to designing a keystream generator using LFSR is simple. First you take one or more LFSR, generally of different lengths and with different feedback polynomials. (If the lengths are all relatively prime and the feedback polynomials are all primitive, the whole generator is **maximal** length). The key is the initial state of the LFSR. Every time you want a bit, **shift** the LFSR once (this is sometimes called clocking). The output bit is a function, preferably a nonlinear function, of some of the bits of the LFSR. This function is called the **combining function**, and the whole generator is called a combination generator.

Three general methodologies for destroying the linearity properties of LFSRs are discussed in this section [7]:

1. Using a nonlinear combining function on the outputs of several LFSRs.



2. Using a nonlinear filtering function on the contents of a single LFSR.
3. Using the output of one (or more) LFSRs to control the clock of one (or more) other LFSRs

## 2.5 Stream Cipher Systems

In **stream ciphers**, the message units are bits, and the key is usual produced by a **random bit generator** (see figure (2.7)). The plaintext is encrypted on a bit-by-bit basis.

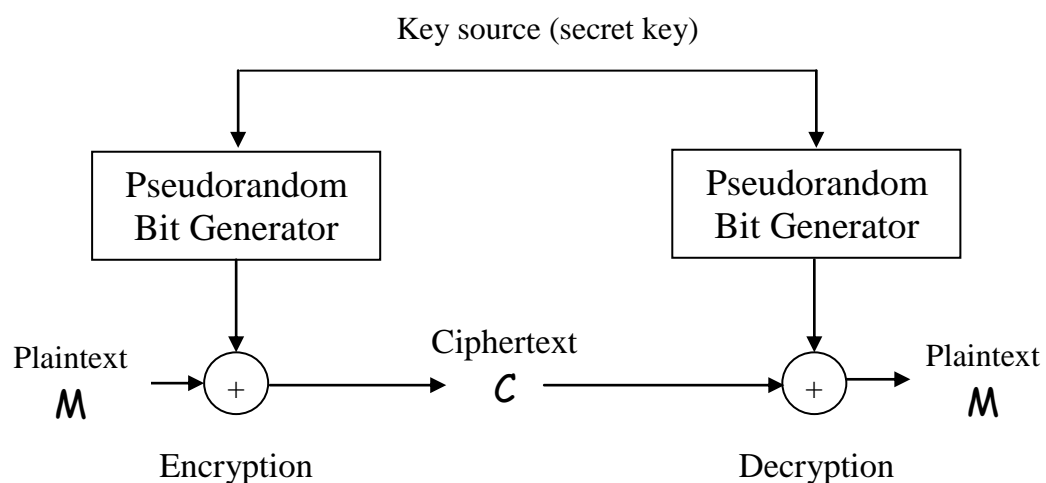


Figure (2.7) A stream (Bit) cipher.

The key is fed into random bit generator to create a long sequence of binary signals. This “key-stream”  $k$  is then mixed with plaintext  $m$ , usually by a bit wise XOR (Exclusive-OR modulo 2 addition) to produce the ciphertext stream, using the same random bit generator and seed.

Stream ciphers are an important class of encryption algorithms. They encrypt individual characters (usually binary digits) of a plaintext message one at a time, using an encryption transformation which varies with time. By contrast, block ciphers tend to simultaneously encrypt groups of characters of a plaintext message using a fixed encryption transformation. Stream ciphers are generally faster than block ciphers in hardware, and have less complex hardware circuitry. They are also more appropriate, and in some cases mandatory (e.g., in some

telecommunications applications), when buffering is limited or when characters must be individually processed as they are received. Because they have limited or no error propagation, stream ciphers may also be advantageous in situations where transmission errors are highly probable.

There is a vast body of theoretical knowledge on stream ciphers, and various design principles for stream ciphers have been proposed and extensively analyzed. However, there are relatively few fully-specified stream cipher algorithms in the open literature. This unfortunate state of affairs can partially be explained by the fact that most stream ciphers used in practice tend to be proprietary and confidential. By contrast, numerous concrete block cipher proposals have been published, some of which have been standardized or placed in the public domain. Nevertheless, because of their significant advantages, stream ciphers are widely used today, and one can expect increasingly more concrete proposals in the coming years.[8]

## **2.6 Combination Generator**

One approach is to use  $n$  LFSRs in parallel; their outputs combined using an  $n$ -input binary **Boolean function** or **combining function** (CF). Figure (2.8) shows the design of  $n$ -LFSR's generator with combining function.[9]

Because LFSRs are inherently linear, one technique for removing the linearity is to feed the outputs of several parallel LFSRs into a non-linear Boolean function to form a **combination generator**. Various properties of such a combining function are critical for ensuring the security of the resultant scheme, for example, in order to avoid correlation attacks.[10]

Since a well-designed system should be secure against known plaintext attacks, an LFSR should never be used by itself as a keystream generator. Nevertheless, LFSRs are desirable because of their very low implementation costs.[11]

For essentially all possible secret keys, the output sequence of an LFSR based keystream generator should have the following properties:

1. large period.
2. large linear complexity.
3. good statistical properties.

It is emphasized that these properties are only **necessary** conditions for a keystream generator to be considered cryptographically secure. Since mathematical proofs of security of such generators are not known, such generators can only be deemed **computationally secure** after having withstood sufficient public scrutiny. [7]

The LFSRs in an LFSR-based keystream generator may have known or secret connection polynomials. For known connections, the secret key generally consists of the initial contents of the component LFSRs. For secret connections, the secret key for the keystream generator generally consists of both the initial contents and the connections.

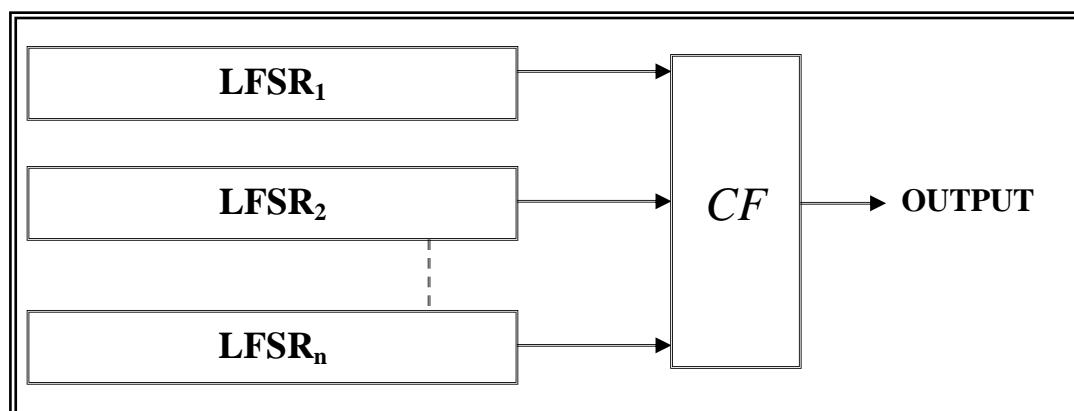


Figure (2.8) n-LFSR's Generator with Combining Function

## **2.7 Examples of Known Generators**

Some examples of known keystream generators are introduced.

### **2.7.1 Linear Generator** [12]

The **Linear generator**, illustrated in figure (2.9), is defined by n-maximum-length LFSRs whose lengths  $r_1, r_2, \dots, r_n$ , where  $n \in \mathbb{Z}^+$  are pair wise relatively prime, with XOR combining function:

$$F(x_1, x_2, \dots, x_n) = x_1 \oplus x_2 \oplus \dots \oplus x_n \quad \dots (2.1)$$

This generator considered weak, despite of his good randomness, because of his weak linear complexity.

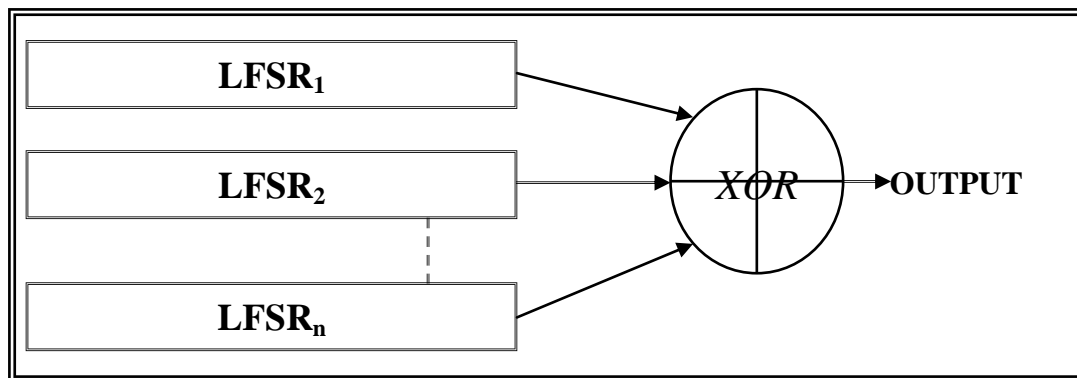


Figure (2.9) n-Linear Generator.

### 2.7.2 Product Generator [12]

The **Product generator**, illustrated in figure (2.10), is defined by n-maximum-length LFSRs whose lengths  $r_1, r_2, \dots, r_n$ , where  $n \in \mathbb{Z}^+$  are pair wise relatively prime, with AND combining function:

$$F(x_1, x_2, \dots, x_n) = x_1 \bullet x_2 \bullet \dots \bullet x_n = \prod_{i=1}^n x_i \quad \dots (2.2)$$

This generator considered weak, despite of his good linear complexity, because of his weak randomness.

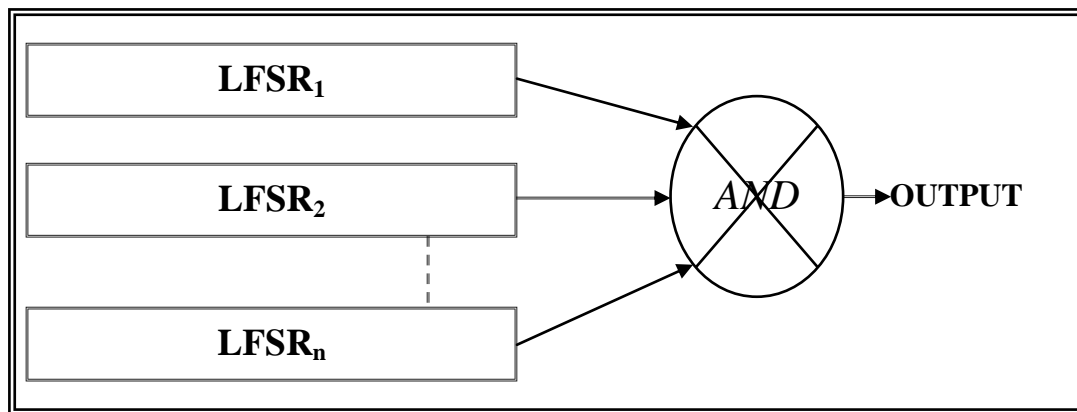


Figure (2.10) n-Product Generator.

### 2.7.3 Threshold Generator [13]

This generator as usual using combining function called Majority function which is balance and symmetric (which expect that this generator will produces pseudo random generator). This generator illustrated in figure (2.11) tries to get around the security problems by using a variable number of LFSR's. The theory is that if you use a lot of LFSRs, it's harder to break the cipher.

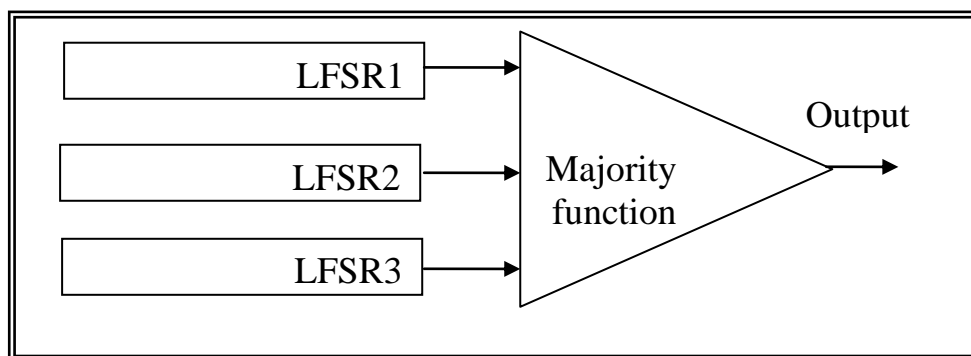


Figure (2.11) Threshold generator

Take the output of a large number of LFSRs (use an odd number of them). Make sure the lengths of all the LFSRs are relatively prime and all the feedback polynomials are primitive: maximize the period. If more

than half the output bits are 1, then the output of the generator is 1. If more than half the output bits are 0, then the output of the generator is 0.

With three LFSRs, the output generator can be written as:

The Threshold generator using the non-linear combining function s.t:

$$F_3(x_1, x_2, x_3) = x_1x_2 \oplus x_1x_3 \oplus x_2x_3.$$

From the combining function of this generator, we expect that it has a larger linear complexity (LC): [12]

$$LC(S) = r_1r_2 + r_1r_3 + r_2r_3$$

where  $r_1$ ,  $r_2$ , and  $r_3$  are the lengths of the first, second, and third LFSRs.

## **2.8 Cryptanalysis**

Cryptanalysis plays an essential role in the design of ciphers. A good cipher should be designed by taking into account all the known cryptanalysis techniques and the designer's insight into unknown attacks. For example, DES would not have been designed in the same way if the differential attack had not been invented at that time, and AES would not give adequate security margin if the square attack had not been developed at that time. In the following, we illustrate some general attacks on stream ciphers, followed by the dedicated attacks on LFSR based stream ciphers. The countermeasure against these attacks will be discussed.[14]

Unlike designing a cipher system, breaking a cipher system sometimes highly depends on guess and luck. Whatever method of cryptanalysis is used, the analyst must always expend some amount of time and resources (defined as work factor) to reach his goal. By increasing available resources, the time required to attack the cipher successfully can often be reduced.

Consequently, there is a relationship between cost and time for any given cryptanalytic attack against a cipher. Also, there is a relationship established between the value, of the information obtained by breaking a cipher, and time. These two relationships can be used to determine the practical secrecy of the cipher. Usually, cryptanalysis involves high-speed computers, and complex, sophisticated computer programs. This includes: first, computer processors, which may include special purpose hardware to execute the logical and arithmetic operations needed to obtain the solution, second, computer storage for the analysis programs and its data, and third human resources to devise and write analysis programs, gather data, and oversee the analysis.

The basic concepts of cryptanalysis were developed as a branch of applied mathematics; the cryptanalysis uses the following tools: [15]

1. Probability theory and statistics.
2. Linear algebra.
3. Abstract algebra (group theory).
4. Complexity theory.

## **2.9 Attacking of Stream Cipher Methods**

The attacking methods of stream cipher could be classified as displayed in Figure (2.12).

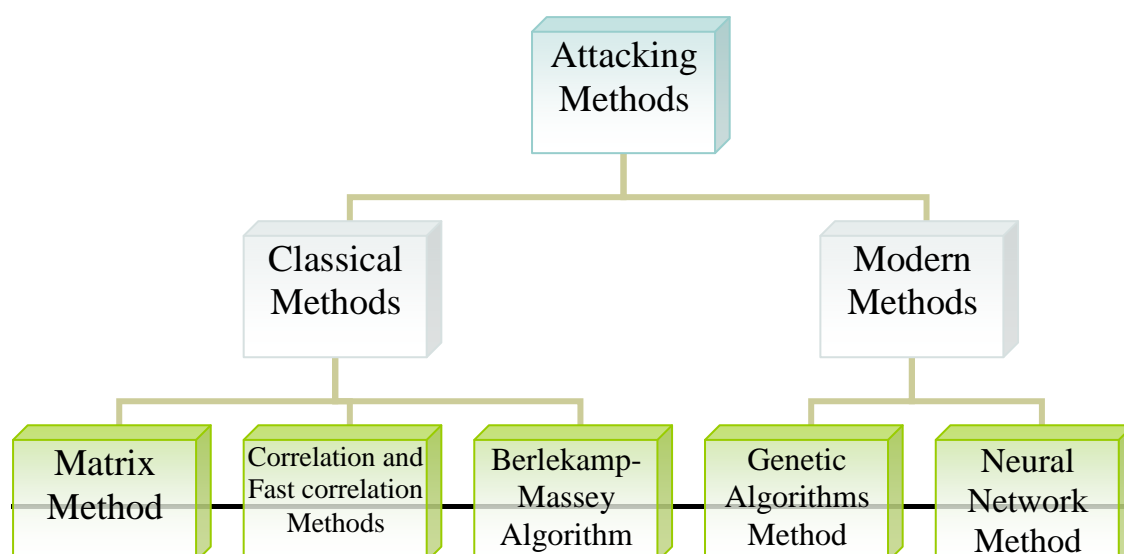


Figure (2.12) Attacking methods of Stream Cipher.

The classification is based on the information processing approaches and the tools that are utilized by the cryptanalysis. From figure (2.12), clearly the classification has two major parts: classical methods, and modern methods. The classical methods include the most efficient methods used for attacking the stream cipher, but they are not the only methods in this area; they are: matrix method, Massey algorithm, and correlation methods (including fast correlation). These methods usually may use some or all cryptanalysis requirements and tools given in next sections. However, the modern methods depend on different approaches for information processing, namely biological like processing. We classify them as modern methods since they depend on new tools, which are:

1. Genetic Algorithms. [16]
2. Neural Networks. [17]

These new tools can greatly facilitate cryptanalysis, as we shall explain in this chapter. In fact, the variety of new technologies increases the possibilities of attack. So that, each major advance in information technology must change our ideas about data security [18]. The



requirements summed up by the phrase "data security" do not stay the same; they change rapidly as the technology changes. One of the reasons for the speed of development in cryptography is, of course, the presence of cryptanalyst. [19]

### **2.9.1 Classical Cryptanalysis Methods**

Although there is a considerable literature on the design of cryptography system, relatively little public domain information exists on techniques for cryptanalysis. In this section we present an overview of the most efficient methods used to attack stream cipher (linear and non-linear type):

#### **I. Matrix Method**

Meyer et al [20,21], have demonstrated a method of breaking an "N-stage" LFSR given (2N) consecutive bits of known plaintext. The basic method is to set up the matrix equation:  $K = MC$

$$\begin{bmatrix} K_{n+1} \\ K_{n+2} \\ K_{n+3} \\ \vdots \\ K_{2n} \end{bmatrix} = \begin{bmatrix} M_n & M_{n-1} & M_{n-2} & \cdots & M_1 \\ 0 & M_n & M_{n-1} & \cdots & M_2 \\ 0 & 0 & M_n & \cdots & M_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & M_n \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ \vdots \\ C_n \end{bmatrix}$$

where

M : is the matrix of successive shifts of the first n bits of plaintext.

C : is the unknown matrix of switch states.

K. : is the key matrix.

Thus the switch values can be obtained by inverting M and solving:

$$C = M^{-1} K$$

Therefore, the entire key will be known completely. Although finding the inverse of matrix is not trivial (the time taken to find the inverse matrix over GF(2) is proportional to  $O(N^3)$ ), it is a straightforward process. But this method is weak, since if less than  $2N$  consecutive bits are known this may not be enough to determine the entire sequence.

Beker and Piper [19], showed two methods of breaking an “N-stage” LFSR given  $(2N)$  bits of the known plaintext sequence where the bits are not necessarily consecutive. These methods are based on trying every possibility for filling unknown entries in the sequence, and for each one they merely use the techniques of the matrix equation, so that, they involve a mixture of guessing and solving simultaneous equations. The choice between these two methods is usually made by seeing which will require fewer trials.

## **II. Berlekamp-Massey Shift Register Synthesis Algorithm**

The iterative algorithm introduced by Berlekamp [22] for decoding Bose-Chaudhuri-Hocquenghem (BCH) codes, provides efficient solution to the problem of synthesizing the shortest linear feedback shift register capable of generating a finite sequence of bits. The algorithm leads to the polynomial of smallest possible degree ( $N$ ) as providing  $(2N)$  bits from shift register of length ( $N$ ). [23]

## **III. Correlation and Fast Correlation Methods**

Siegenthaler [24], in 1985, demonstrated a method that the  $LFSR_i$  part of the key can be found independently of the other  $LFSR_s$  parts, by using the “divide and conquer” technique. This method showed that the number of trials to find the key can be reduced significantly in those cases where a correlation exists between the output of the running key generator employed in a stream cipher, and  $LFSR_i$  sequence with

correlation probability  $P$  (up to 0.75). These generators have been broken for LFSR lengths ( $N < 50$ ).

Meier and Staffelbach [25], in 1988, developed two algorithms (A and B), which are much faster than the above attack and are demonstrated to be successful against shift registers of considerable length  $N$  ( $N \gg 50$ ), provided that the number  $t$  of feedback taps is small ( $t < 10$  if  $P \leq 0.75$ ).

Both A and B algorithms are developed by using the statistical model of Siegenthaler. On the other hand, for correlation probabilities  $P < 0.75$  the attacks are proven to be infeasible against long LFSRs if they have a greater number of taps (roughly  $N \geq 100$  and  $t \geq 10$ ).

## **2.9.2 Modern Cryptanalysis Methods**

The modern methods of cryptanalysis are based on new and different approaches to minimize the time and the cost of attacking. These methods, as displayed in figure (3.10), are directed to utilize genetic algorithms and neural network concepts.

### **I. Genetic Algorithms Methods**

Genetic Algorithms are search algorithms based on the mechanics of natural selection and natural genetics. They use concepts drawn from the theory of evolution to "breed" progressively better solutions to problems with very large solution spaces [26]. With their ability to efficiently search huge solution spaces, genetic algorithms would seem a natural candidate for use in cryptanalysis. [27]

Spillman, et al showed the use of genetic algorithms in the cryptanalysis of simple substitution ciphers. [28]

Spillman and Rechar, used a new method to attack knapsack cipher using genetic algorithms to decrease the time for breaking. [29]

Matthews, demonstrated the use of genetic algorithms to break classical transposition ciphers by finding the transposition sequence used.

However, there are a number of cryptographic algorithms that cannot be attacked using genetic algorithms, a large number of cryptographic algorithms, including those underpinning many rotor-based systems (for example stream cipher systems), appear vulnerable to attack by genetic algorithms. [27]

## **II. Neural Networks Method**

The general interest in the neural networks arises from their fascinating properties which enable them to exceed the limitations of traditional information processing.

Although it seems that neural network is a valuable tool for use in cryptanalysis, neural network is not a panacea. Therefore an important question must be answered: Does cryptanalysis make a suitable problem to be solved by Neural Networks? To answer, we must know the characteristics that must be exhibited by the problem to be suitable for solution by neural network. They are: [30]

1. The rule used in solving the problem may be unknown, or very difficult to explain or formalize.
2. The problem makes use of noisy, or incomplete data.
3. The problem may evolve.
4. The problem needs very high speed processing.
5. There may be no current technical solutions.

### **2.10 Adversarial Models** [31]

The capabilities of an adversary in terms of operations he is able or allowed to execute, is another important factor during a cryptanalytic attack. These conditions are commonly summarized in adversarial models

and are categorized by the type of data and by the type of access an adversary requires to successfully mount a given attack. The type of data differentiates between inputs and outputs of a cryptosystem such as secret keys, plaintexts, and ciphertexts, and the type of access differentiates between reading, writing and adaptive writing access, which are denoted as known values, chosen values and adaptively chosen values, respectively. An overview on the main adversarial models in conventional cryptanalysis is given below:

### **2.10.1 Ciphertext-only Attacks [31]**

The adversary knows only the ciphertext and has no access to the plaintext. A cryptographic primitive vulnerable to such kind of attacks is considered exceptionally weak, since it is possible to distinguish it from a random permutation by analysing only ciphertexts.

### **2.10.2 Known-plaintext Attacks [32]**

The adversary has reading access to plain- and corresponding ciphertexts processed by the cipher. A representative of this category is, for example, linear cryptanalysis.

### **2.10.3 Chosen-plaintext Attacks [33]**

These are similar to known-plaintext attacks, with the difference that an adversary is allowed to choose the concrete plaintexts to be encrypted prior to the attack. A well-known attack type of this category is differential cryptanalysis.

### **2.10.4 Chosen-ciphertext Attacks [34]**

The adversary can choose ciphertexts to be decrypted by the cipher before the attack starts and has reading access to the resulting plaintexts.

### **2.10.5 Adaptively Chosen-plaintext Attacks** [34]

The adversary can select plaintexts to be encrypted during the attack and is not forced to choose them before the attack starts as in the case of the chosen plaintext scenario. The attacker also has access to the resulting ciphertexts.

### **2.10.6 Adaptively Chosen-ciphertext Attacks** [34]

The adversary can select ciphertexts to be decrypted during the attack and is not forced to choose them before the attack starts as in the case of the chosen ciphertext scenario. The attacker also has access to the resulting plaintexts.

### **2.10.7 Related-key Attacks** [35]

The adversary can encrypt plaintexts and decrypt ciphertexts with the attacked key and with keys related to the latter, which, for example, differ only at certain bit positions.

However, at the same time collecting data of a given type becomes more and more demanding the further we go down that list. The above categorization also presents an indication on the practicability of the attacks. The above models also form the basis for implementation attacks, however, an attacker is assumed to have additional capabilities.

## **2.11 Types of Attacks** [36]

In this section we will present the types of attacks, we also explain and discuss what is the use of the attack.

It is important to understand that all the attacks have one purpose. The purpose is to discover the key used in the process of ciphering and deciphering. Each attack has a method to try to discover the key that was used, we will give some examples of the application of the attack.

### **2.11.1 Exhaustive Search Attack** [37]

The exhaustive search attack is also called brute force. The method of this attack is to search through all possible states, checking for a match between the resulting and the observed keystream.

Fortunately, Babbage in 1995 improved the exhaustive search attack in stream ciphers. He defined two attacks in this area.

In the first attack, the attacker first produces a list of  $n$ -bit subsequences, sorted in lexicographic (or numeric) order. Then the attacker select a random candidate state in this list and check, if the selected state produces the output of cipher, then the attacker found the initial state else he continues try to find the initial state.

The second attack was defined by Babbage as:

“ Let  $V$  be a vector space of dimension  $n$  over  $GF(2)$ , with each possible KG(Keystream Generator) state an element of  $V$ . The initial state, which we wish to determine, is  $s_0$ , and the state transition function is linear, and so can be represented by an  $n \times n$  matrix  $A$ , so that  $s_i = s_0 A^i$ . The output function  $h : V \rightarrow GF(2)$ , so that the  $i$ th keystream bit  $k_i$  is equal to  $h(s_i)$ .”

### **2.11.2 Algebraic Attack** [38]

The algebraic attack is used in stream ciphers based in LFSRs. This attack try to find the initial state given some keystream bits.

The algebraic attacks has two steps. In the first step, the attack tries to find a system of equations in the bits of the secret key  $K$  and the output bits  $Z_i$ . If it has enough low degree equations and known key bits stream, then the secret key  $K$  can be recovered by solving this system of equations in a second step. This system could be solved using Groebner bases.

### **2.11.3 Correlation Attack** [39]

The correlation attack was proposed by Siegenthaler in 1985. An important work in this area was elaborated by Meier and Staffelbach.

After them, Mihaljevi and Goli was one of the promising work. Other important work is from Anderson, he started the search for the optimum correlation attack. They opened the world of cryptanalysis to correlation attack. The correlation attack is defined as:

“The correlation attack exploits the existence of a statistical dependence between the keystream and the output of a single constituent LFSR.”

#### **2.11.4 Fault Attack [40]**

The fault attack is a powerful cryptanalytic tool. It is widely applied in cryptosystems which are not vulnerable to direct attack. It is easy. The idea of Correlation attack involving several constituent LFSRs attacks in block ciphers, but the first application of this attack in stream cipher was developed by Hoch and Shamir.

In this attack, the attacker can apply some bit flipping faults to either the RAM or the internal register of the cryptographic device. However, he had only a partial control over their number, location and timing. This model tries to reflect a situation in which the attacker has the possession of the physical device, and the faults are transient rather than permanent.

A good work in this area was developed by Barengi et al, they talk about this technique and where it can be applied. In their work has examples using stream ciphers and block ciphers.

#### **2.11.5 Chosen-IV Attack [41]**

In the Chosen-IV attack one of the relevant work in this area is from Joux and Muller. To understand more about this attack we should bring the definition from Joux and Muller work:

“In general, a stream cipher produces a pseudo random sequence  $PRNG(K, IV)$  from a secret key  $K$  and an initialization vector  $IV$ . Then, the ciphertext  $C$  is computed from the plaintext  $P$  by:

$$C = PRNG(K, IV) \oplus P .$$



The main idea behind the use of initialization vectors is to generate different pseudorandom sequences without necessarily changing the secret key, since it is totally insecure to use twice the same sequence.” Then, this attack exploits the weaknesses in the key scheduling algorithm of the stream cipher. The attack tried to extract from the memory, the initial state of the LFSR. Like the algebraic attack.

### **2.11.6 Slide Attack** [42]

The first time that the slide attack appeared in the literature was with Biryukov and Wagner. They used the attack in TREYFER, WAKE-ROFB and others block ciphers. In 2000 they improved the slide attack and used in other block ciphers. More recently slide attacks have been applied to other stream ciphers, such as Trivium with Priemuth-Schmid and Biryukov.

The main idea of the attack is defined by Biryukov and Wagner like: “The idea is to slide one copy of the encryption process against another copy of the encryption process, so that the two processes are one round out of phase.”

### **2.11.7 Cube Attack** [43]

The cube attack is relative new. It has been introduced by Dinur and Shamir in 2009.

“The attack exploits the existence of low degree polynomial representation of a single output bit (as a function of the key and plaintext bits) in order to recover the secret key. In order to derive the secret key, the attacker sums this bit over all possible values of a subset of the plaintext bits. The summations are used in order to derive linear equations in the key bits which can be efficiently solved”. This attack can be applied in almost any cryptosystem.

### **2.11.8 Time-Memory Trade-off Attack** [44,45]

Biryukov and Shamir extended this attack for stream ciphers. The Time/Memory/Data Tradeoff Attack has two phases:

“During the preprocessing phase (which can take a very long time) the attacker explores the general structure of the cryptosystem, and summarizes his findings in large tables (which are not tied to particular keys). During the realtime phase, the attacker is given actual data produced from a particular unknown key, and his goal is to use the precomputed tables in order to find the key as quickly as possible.” In any time-memory tradeoff attack there are five key parameters:

N: represents the size of the search space.

P: represents the time required by the preprocessing phase of the attack.

M: represents the amount of RAM (hard disks or DVDs) available to the attacker.

T: represents the time required by the realtime phase of the attack.

D: represents the amount of realtime data available to the attacker.

Verdult et al recovered the key from Hitag2 stream cipher in 360 seconds. The importance of the Hitag2 is primarily used in RFID transponder systems manufactured by Philips/NXP, and used by many car manufacturers for unlocking car doors remotely.

### **2.11.9 Guess and Determine Attack** [46]

According with Ahmadi and Eghlidos the Guess and Determine Attack is defined as:

“ In GD attacks, the attacker first guesses (the values of) a set of state elements of the cryptosystem, called a basis; hence, the name. The basis can correspond to different elements of different states (multiple times). Next, she determines the remaining state elements and running key sequence, and compares the resulting key sequence with the observed key sequence. If these two sequences are equal, then the guessed values are

true and the cryptosystem has been broken, otherwise the attacker should repeat the above scenario with other guessed values” . [47]

Other application of this attack was proposed by Sha and Mahalanobis They used the GD attack on the A5/1 Stream cipher. Using the GD attack they recovered the key in a time complexity of  $2^{48} \cdot 5$ , wich is much less than the bruteforce attack with a complexity of  $2^{64}$ .

In the moment, Dunkelman and Keller made a cryptanalysis of the stream cipher LEX and in this cryptanalysis they used the GD attack. He started the GD attacks on LFSRs for stream ciphers.[48]



# Chapter Three

## **PARTIAL SWARM OPTIMIZATION**

# Chapter Three

## Particle Swarm Optimization

### **3.1 Introduction**

The social behavior of animals, and in some cases of humans, is governed by similar rules. However, human social behavior is more complex than a flock's movement. Besides physical motion, humans adjust their beliefs, moving, thus, in a belief space. Although two persons cannot occupy the same space of their physical environment, they can have the same beliefs, occupying the same position in the belief space, without collision [38].

PSO was originally developed by a social-psychologist (James Kennedy) and an electrical engineer (Russell Eberhart) in 1995 and emerged from earlier experiments with algorithms that modeled the "flocking behavior" seen in many species of birds. Where birds are attracted to a roosting area in simulations they would begin by flying around with no particular destination and in spontaneously formed flocks until one of the birds flew over the roosting area [41].

PSO has been an increasingly hot topic in the area of computational intelligence. PSO is yet another optimization algorithm that falls under the soft computing umbrella that covers genetic and evolutionary computing algorithms as well. As such, it lends itself as being applicable to a wide variety of optimization problems.

### **3.2 Simulation Social Behavior** [24]

A number of scientists have created computer simulations of various interpretations of the movement of organisms in a bird flock or fish school. Notably and et al presented simulations of bird flocking.

Reynolds was intrigued by the aesthetics of bird flocking choreography, and Heppner, a zoologist, was interested in discovering the underlying rules that enabled large numbers of birds to flock synchronously, often changing direction suddenly, scattering and regrouping, etc. Both of these scientists had the insight that local processes, such as those modeled by cellular automata, might underlie the unpredictable group dynamics of bird social behavior. Both models relied heavily on manipulation of inter-individual distances; that is, the synchrony of flocking behavior was thought to be a function of birds' efforts to maintain an optimum distance between themselves and their neighbors. It does not seem a too-large leap of logic to suppose that some same rules underlie animal social behavior, including herds, schools, and flocks, and that of humans. As sociobiologist “E. O. Wilson” has written, in reference to fish schooling, “In theory at least, individual members of the school can profit from the discoveries and previous experience of all other members of the school during the search for food. This advantage can become decisive, outweighing the disadvantages of competition for food items, whenever the resource is unpredictably distributed in patches”.

### **3.3 Genetic Algorithms** [34]

Genetic Algorithms are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. While randomized, Genetic Algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search point with expected improved performance.

GAs attempt to identify optimal solution by applying the techniques of natural selection to a population of solutions, the solutions are evaluated, the bad solutions are killed, and the remaining solutions are recombined (Mate) to form a new generation of solution.

The general principle is that of Darwinism the good traits will survive overtime as they are less likely to be on solutions that are killed a generation. Over a number generation in an overall increase in the quality of solution in the population.

Thus, a GA is an iterative procedure, which maintains a constant size population of candidate solution. During each iteration step (Generation) the structures in the current population are evaluated, and, on the basic of those evaluations, a new population of candidate solutions formed.

### **3.3.1 The Basic Cycle of GA**

The basic GA cycle based on the three processes (selection, mating and mutation) as shown in figure (3.1) [51].

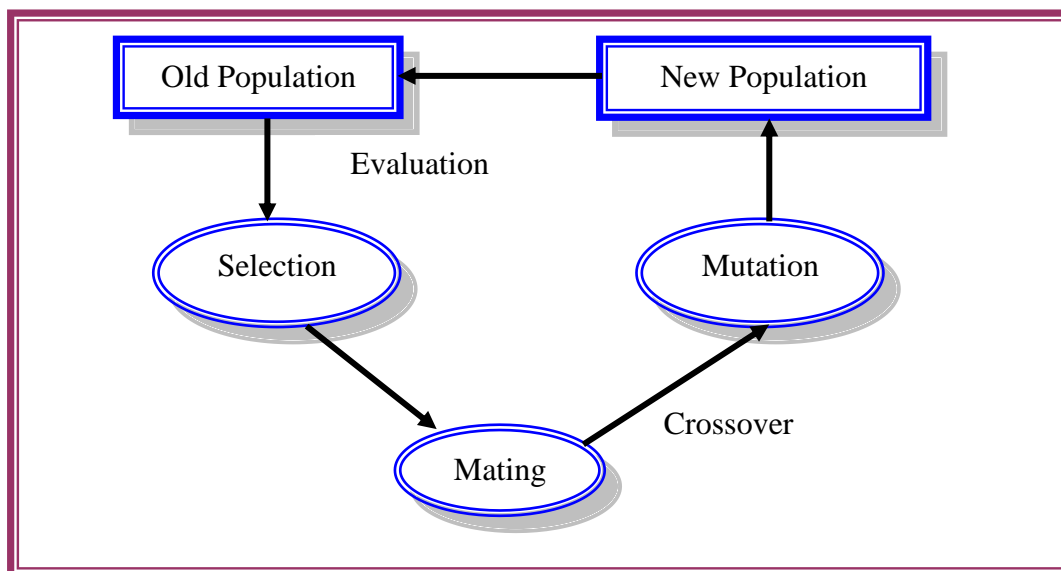


Figure (3.1) The Basic Cycle of GA

An abstract view of the GA is:

Generation=0;

Initialize G(P); {G=Generation ; P=Population }

Evaluate G(P);

While (GA has not converged or terminated)

    Generation = Generation + 1;

    Select G(P) from G(P-1);

    Crossover G(P);

    Mutate G(P);

    Evaluate G(P);

End (While)

Terminate the GA. [54]

The GA main steps are as follows:

1. **The first step** is the selection process, which determines which string in the current generation will be used to create the next generation. By using move in the most promising direction in the overall search.
2. **The second step** is the mating process that determines the actual form of the string in the next generation [54]. At this point, two of the selected parents are paired. If the length of the each string is  $r$ , then a random number between 1 and  $r$  is selected, says, the mating process is one swapping bits  $s+1$  to  $r$  of the first parent with bits  $s+1$  to  $r$  of the second parent. In this way two new strings are created.
3. **The final step** is one of mutation. A fixed small mutation probability is set at the start of the algorithm. Bits in all the new string are then subject to change based on this mutation probability.

These three steps are repeated to create each new generation It continues in this fashion until some stopping condition is reached such as a maximum number of generation or a specified fitness value (threshold).

[16]



The GA is usually stopped when a given termination condition is met. Some common termination conditions are:

1. A pre-determined number of generations have passed.
2. A satisfactory solution has been found.
3. No improvement in solution quality has taken place for a certain number of generations.

The different termination conditions are possible since a GA is not guaranteed to converge to a solution. [54]

### **3.4 Swarms and Particles** [24]

It became obvious during the simplification of the paradigm that the behavior of the population of agents is now more like a swarm than a flock. The swarm has a basis in the literature. Millonas, developed his models for applications in “ALife”, and articulated five basic principles of swarm intelligence, these principles are:

1. **Proximity principle:** the population should be able to carry out a simple space and time computations.
2. **Quality principle:** the population should be able to respond to quality factors in the environment.
3. **Principle of diverse response:** the population should not commit its activities along excessively narrow channels.
4. **Principle of stability:** the population should not change its mode of behavior every time the environment changes.
5. **Principle of adaptability:** the population must be able to change the behavior mode when it is worth the computational price.

Note that principles four and five above are the opposite sides of the same coin. The particle swarm optimization concept and paradigm

presented in adhere to all five principles. Basic to the paradigm are n-dimensional space calculations carried out over series of time steps.

The population members are mass-less and volume-less, and thus could be called "points," it is felt that velocities and accelerations are more appropriately applied to particles, even if each is defined to have arbitrarily small mass and volume. Further, Reeves discusses particle systems consisting of clouds of primitive particles as models of diffuse objects such as clouds, fire and smoke.

### **3.5 Swarm Intelligence (SI) [7]**

In the last two decades, the computational researchers have been increasingly interested to the natural sciences, and especially biology, as sources of modeling paradigms.

Many research areas are massively influenced by the behavior of various biological entities and phenomena. It gave birth to most of population-based metaheuristics such as Evolutionary Algorithms (EAs), particle swarm optimization, and BA etc.

SI is a modern AI discipline that is concerned with the design of multi agent systems with applications, e.g. in optimization and robotics. The design paradigm for these systems is fundamentally different from many traditional approaches.

Instead of the sophisticated controller that governs the global behavior of the system, the SI principle is based on many unsophisticated entities that cooperate in order to exhibit a desired behavior. Inspiration for the design is taken from the collective behavior of social insects such as ants, termites, bees and wasps, as well as from the behavior of other animal societies such as flocks of birds or schools of fish. Colonies of social insects have mesmerized researchers for many years.

However, the principles that govern their behavior remained unknown for a long time. Even though the single members of these societies are unsophisticated individuals, they are able to achieve complex tasks in cooperation.

Optimization techniques inspired by SI have become increasingly popular during the last decade. They are characterized by a decentralized way of working that mimics the behavior of swarms of social insects. The advantage of these approaches over traditional techniques is their robustness and flexibility.

These properties make SI a successful design paradigm for algorithm that deals with increasingly complex problems concerned with collective behavior in self-organized.

Bonabeau has defined the swarm intelligence as “any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies”

### **3.5.1 Concepts of Swarm Intelligence** [23]

Two fundamental concepts, self-organization and division of labor, are necessary and sufficient properties to obtain swarm intelligent behavior such as distributed problem solving systems that self-organize and adapt to the given environment these are:-

1. Self-organization can be defined as a set of dynamical mechanisms, which result in structures at the global level of a system by means of interactions among its low-level components. These mechanisms establish basic rules for the interactions between the components of the system. The rules ensure that the interactions are executed on the basis of purely local information without any relation to the global pattern. Bonabeau et al. [6] have characterized four basic properties on which

self-organization rely: Positive feedback, negative feedback, fluctuations and multiple interactions.

**I)** Positive feedback is a simple behavioral “rules of thumb” that promotes the creation of convenient structures. Recruitment and reinforcement such as trail laying and following in some ant species or dances in bees can be shown as the examples of positive feedback.

**II)** Negative feedback counterbalances positive feedback and helps to stabilize the collective pattern. In order to avoid the saturation which might occur in terms of available foragers, food source exhaustion, crowding or competition at the food sources, a negative feedback mechanism is needed.

**III)** Fluctuations such as random walks, errors, random task switching among swarm individuals are vital for creativity and innovation. Randomness is often crucial for emergent structures since it enables the discovery of new solutions.

**IV)** Self-Organization requires a minimal degree of mutuality to learned

individuals, enabling them to make use of the results from their own activities as well as others.

**2.** Inside a swarm, there are different tasks, which are performed simultaneously by specialized individuals. This kind of phenomenon is called division of labour. Simultaneous task performance by cooperating specialized individuals is believed to be more efficient than the sequential task performance by unspecialized individuals. Division of labour also enables the swarm to respond to changed conditions in the search space.

Two fundamental concepts for the collective performance of a swarm presented above, self-organization and division of labour are

necessary and sufficient properties to obtain swarm intelligent behaviour such as distributed problem-solving systems that self-organize and adapt to the given environment.

### **3.5.2 Ant Colony Optimization** [37]

ACO is a class of optimization algorithms modeled on the actions of an ant colony. ACO methods are useful in problems that need to find paths to goals.

Artificial 'ants'—simulation agents—locate optimal solutions by moving through a parameter space representing all possible solutions. Natural ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate better solutions.

### **3.5.3 Bee Algorithm Optimization** [58]

The Bees Algorithm (BA) is a population-based search algorithm, first developed in 2005 by Pham DT etc. and Karaboga independently. The algorithm mimics the food foraging behavior of swarms of honey bees. In its basic version, the algorithm performs a kind of neighborhood search combined with random search and can be used for optimization problems.

### **3.5.4 Particle Swarm Optimization** [37]

PSO is an optimization algorithm for dealing with problems in which a best solution can be represented as a point or surface in an n-dimensional space.

Hypotheses are plotted in this space and seeded with an initial velocity, as well as a communication channel between the particles.

Particles then move through the solution space, and are evaluated according to some fitness criterion after each time step.

Over the time, particles are accelerated towards those particles within their communication grouping which have better fitness values. The main advantage of such an approach over other global minimization strategies such as simulated annealing is that the large numbers of members that make up the particle swarm make the technique impressively resilient to the problem of local minima.

### **3.6 PSO Topology** [24]

The common uses of PSOs are either global version or local version of PSO. In the global version of PSO, each particle flies through the search space with a velocity that is dynamically adjusted according to the particles of personal best performance achieved so far and the best performance achieved so far by all the particles. While in the local version of PSO, each particle's velocity is adjusted according to its personal best and the best performance achieved as far within its neighborhood. The neighborhood of each particle is generally defined as topologically nearest particle to the particle at each side.

Since then, a lot of researchers have worked on improving its performance by designing or implementing different types of neighborhood structures in PSOs. Each neighborhood structure has its strength and weakness. It works better in one kind of problems, but worse on the other kind of problems. When using PSO to solve a problem, not only the problem needs to be specified, but the neighborhood structure of the PSO utilized, should also be clearly specified.

### 3.7 PSO Algorithm [24]

The PSO algorithm depends in its implementation in the following two relations:

$$v_{id} = w * v_{id} + c_1 * r_1 * (p_{id} - x_{id}) + c_2 * r_2 * (p_{gd} - x_{id})$$

...(3.1a)

$$x_{id} = x_{id} + v_{id}$$

...(3.1b)

where  $c_1$  and  $c_2$  are positive constants, and  $r_1$  and  $r_2$  are random function in the range  $[0,1]$ ,  $x_i = (x_{i1}, x_{i2}, \dots, x_{id})$  represents the  $i^{\text{th}}$  particle;  $p_i = (p_{i1}, p_{i2}, \dots, p_{id})$  represents the best previous position (the position giving the best fitness value) of the  $i^{\text{th}}$  particle; the symbol  $g$  represents the index of the best particle among all the particles in the population,  $v = (v_{i1}, v_{i2}, \dots, v_{id})$  represents the rate of the position change (velocity) for particle  $i$ .

Equation (3.1) is the equation describing the flying trajectory of a population of particles. Equation (3.1a) describes how the velocity is dynamically updated and Equation (3.1b) the position update of the changed abruptly. It is changed from the “flying” particles. Equation (3.1a) consists of three parts. The first part is the momentum part. The velocity can't be current velocity. The second part is the “cognitive” part which represents private thinking of itself - learning from its own flying experience. The third part is the “social” part that represents the collaboration among particles learning from group flying experience.

In equation (3.1a), if the sum of the three parts on the right side exceeds a constant value specified by user, then the velocity on that dimension is assigned to be  $\pm v_{max}$ , that is, particle's velocities on each dimension is clamped, to a maximum velocity  $v_{max}$ , which is an important parameter, and originally is the only parameter required to be adjusted by

users. Big  $v_{max}$  has particles that have the potential to fly far past good solution areas while a small  $v_{max}$  has particles that have the potential to be trapped into local minima, therefore they are unable to fly into better solution areas. Usually a fixed constant value is used as the  $v_{max}$ , but a well-designed dynamically changing  $v_{max}$  might improve the PSO's performance.

The PSO algorithm is simple in concept, easy to implement and computational efficient. The original procedure for implementing PSO is as follows:

1. Initialize a population of particles with random positions and velocities on  $d$ -dimensions in the problem space.
2. PSO operation includes:
  - a. For each particle, evaluate the desired optimization fitness function in  $d$  variables.
  - b. Compare particle's fitness evaluation with its pbest. If current value is better than pbest, then set pbest equal to the current value, and  $p_i$  equals to the current location  $x_i$  in  $d$ -dimensional space.
  - c. Identify the particle in the neighborhood with the best success so far, and assign its index to the variable  $g$ .
  - d. Change the velocity and position of the particle according to equation (3.1a) and (3.1b).
3. Loop to step (2) until a criterion is met, usually a sufficiently good fitness or a maximum number of iterations.

Like the other evolutionary algorithms, a PSO algorithm is a population based on search algorithm with random initialization, and there is an interaction among population members. Unlike the other evolutionary algorithms, in PSO, each particle flies through the solution space, and has the ability to remember its previous best position, survives from generation to generation. Furthermore, compared with the other evolutionary algorithms,



e.g. evolutionary programming, the original version of PSO is faster in initial convergence through simplified social model while it is lower in fine tuning. The flow chart of PSO algorithm is shown in figure (3.2). [59]

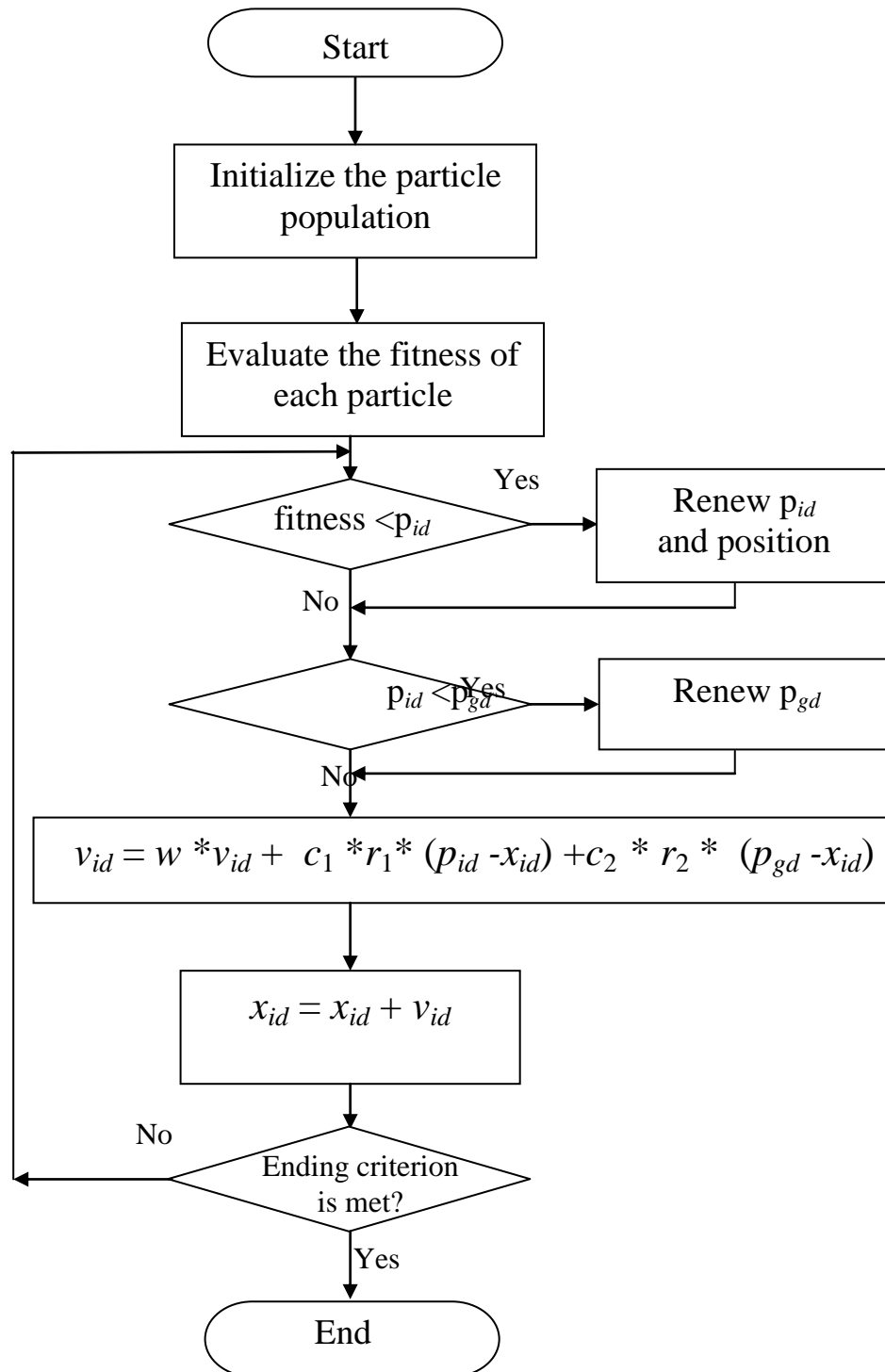


Figure (3.2) Flowchart of PSO Algorithm [58].

### **3.8 The Parameters of PSO**

A number of factors will affect the performance of the PSO. These factors are called PSO parameters, these parameters are:

1. Number of particles in the swarm affects the run-time significantly, thus a balance between variety (more particles) and speed (less particles) must be sought.
2. Maximum velocity ( $v_{\max}$ ) parameter. This parameter limits the maximum jump that a particle can make in one step, thus a too large value for this parameter will result in oscillations, while a too small value could cause the particle to become trapped in a local minima. The maximum velocity parameter is set to the default value of 2 (this value was suggested in earlier works in PSO). [48]
3. The role of the inertia weight  $w$ , in equation (3.2a), is considered critical for the PSO's convergence behavior. The inertia weight is employed to control the impact of the previous history of velocities on the current one. A large inertia weight facilitates global exploration (searching new areas); while a small one tends to facilitate local exploration, i.e., fine-tuning the current search area. Experimental results indicate that it is better to initially set the inertia to a large value, in order to promote global exploration of the search space, and gradually decrease it to get more refined solutions. Thus, an initial value around 1.2 and a gradual decline towards 0 can be considered as a good choice for  $w$ . [41]
4. The parameters  $c_1$  and  $c_2$ , in equation (3.1a), are not critical for PSO's Convergence. However, proper fine-tuning may result in faster convergence and alleviation of local minima. An extended study of the acceleration parameter in the first version of PSO is given in

recent work reports that it might be even better to choose a larger cognitive parameter,  $c_1$  than a social parameter  $c_2$  but with  $c_1 + c_2 = 4$ .

5. The parameters  $r_1$  and  $r_2$  are used to maintain the diversity of the population, and they are uniformly distributed in the range  $[0,1]$ . [41]

From the above case, we can learn that there are two key steps when applying PSO to optimization problems: the representation of the solution and the fitness function. One of the advantages of PSO is that PSO takes real numbers as particles.

Secondly global version vs. local version: We have introduced two versions of PSO; global and local versions. Global version is faster but might converge to local optimum for some problems. Local version is a little bit slower but not easy to be trapped into local optimum. One may use global version to get quick results and use local version to refine the search [20]. The most common parameters of PSO are shown in table (3.1).

Table (3.1) the most common Parameters of PSO.

| No. | Parameter                     | Symbol     | Parameter value                |
|-----|-------------------------------|------------|--------------------------------|
| 1   | No. of particles              | Psize      | $Psize \in [10-40]$            |
| 2   | Maximum velocity              | $v_{max}$  | $v_{max} = 2$                  |
|     | Minimum velocity              | $v_{min}$  | $v_{min} = -v_{max}$           |
| 3   | Inertia weight                | $w$        | $w \in [0.4, 0.9]$             |
| 4   | First acceleration parameter  | $c_1$      | $c_1 \in [0.5, 2]$             |
|     | Second acceleration parameter | $c_2$      | $c_1 = c_2$ or $c_1 + c_2 = 4$ |
| 5   | Diversity of the population   | $r_1, r_2$ | $r_1, r_2 \in [0,1]$           |
| 6   | Iterations                    | Iter.      | $\leq 30000$                   |



# Chapter Four

**PSO Cryptanalysis**  
**Stream Cipher**  
**Cryptosystems**

## Chapter four

### PSO Cryptanalysis Stream Cipher Cryptosystems

#### 4.1 Introduction

In stream cipher systems field, the Linear Feedback Shift Register (LFSR) cryptosystems used widely. In this chapter, first, a method introduced to construct a linear equations system of a single LFSR. This method developed to construct a system of linear equations (SLE) of the attacked a LFSR cryptosystem, where the effect of combining function (CF) of LFSR is considerable. Three study cases are suggested to be cryptanalyzed: Single LFSR, Linear, and Threshold.

This thesis aims to find the initial values of every LFSR in the attacked cryptosystem, using proposed PSO cryptanalysis system, depending on the following information:

1. The length of every LFSR and its feedback polynomial are known.
2. The algebraic description of CF is known.
3. The keystream (output sequence S) generated from the LFSRS is known, or part of it, practically, that means, a known plain attack be applied [47].

This research consists of three stages, constructing the SLE, applying the proposed PSO cryptanalysis system to solve SLE and find the actual initial key for the attacked cryptosystem, and lastly, decrypt the ciphertext to obtain the real plaintext.

#### 4.2 Constructing SLE for Stream Cipher Cryptosystems

LFSRs are used widely in stream cipher systems field. A LFSR System consists of two main basic units. First, a LFSR function and initial state values. The second one is, the *Combining Function* (CF), which is represented by a Boolean function. Most of all stream cipher

systems depend on these two basic units. This section, describes the attack of three case studies of stream ciphers using PSO algorithm.

#### 4.2.1 Single LFSR Stream Cipher Cryptosystem (SLSCC)

The first case study which we want to attack, is a single LFSR, which this system has no combining function, so we expect that it's can be expressed as one linear equations system.

Most practical stream cipher cryptosystems designs center around LFSR. In the early days of electronics machines, they were easy to build. A SR is an array of bit memories and the feedback sequence is only a series of XOR gates. A LFSR-based on stream cipher gives a lot of security with only a few logic gates.

To see how the SLSCC acts, let's assume, a 4-bit LFSR has  $1+x+x^4$  recursive polynomial. If it is initialized with the value 1010, it produces the following sequence of internal states before repeating; See Table (4.1):

Table (4.1): SLSCC with length  $L=4$ .

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |

The output sequence is the string of least significant bits: 0101100.

Let  $SR_L$  be a single LFSR with length  $L$ , let  $A_0=(a_1,a_2,\dots,a_L)$  be the initial value key vector of  $SR_L$ , s.t.  $a_j$ ,  $1 \leq j \leq L$ , be the component  $j$  of the vector  $A_0$ , this mean,  $a_j$  is the initial bit of stage  $j$  of  $SR_L$ . Let  $C_0^T=(c_1,\dots,c_L)$  be the feedback vector, where  $c_j \in \{0,1\}$ , if  $c_j=1$  this means that the stage  $j$  is connected else it's not connected. Let  $S=\{s_i\}_{i=0}^{m-1}$  or

$S=(s_0,s_1,\dots,s_{m-1})$  be the sequence with length  $m$  generated from  $SR_L$ . The generation of  $S$  depending on the following relation:

$$s_i = a_i = \sum_{j=1}^L a_{i-j} c_j \quad i=0,1,\dots \quad \dots(4.1)$$

Relation (4.1) represents the linear recurrence relation [4].

The objective is finding the vector  $A_0$ , when  $L$ ,  $C_0$  and  $S$  are all known.

Let  $M$  be a  $L \times L$  matrix, which is describes the initial phase of  $SR_L$ :

$$M=(C_0|I_{L \times L-1}), \text{ where } M^0=I.$$

Let  $A_1$  represents the new initial of  $SR_L$  after one shift, s.t.

$$A_1=A_0 \times M=(a_{-1},a_{-2},\dots,a_{-L}) \begin{pmatrix} c_1 & 1 & \dots & 0 \\ c_2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ c_L & 0 & \dots & 0 \end{pmatrix} = (\sum_{j=1}^L a_{-j} c_j, a_{-1}, \dots, a_{-L}).$$

In general,

$$A_i=A_{i-1} \times M, \quad i=0,1,2,\dots \quad \dots(4.2)$$

Equation (4.2) can be considered as a recurrence relation, so we have:

$$A_i=A_{i-1} \times M=A_{i-2} \times M^2=\dots=A_0 \times M^i \quad \dots(4.3)$$

notice that:

$M^2=[C_1 C_0 | I_{L \times L-2}]$  and so on until get  $M^i=[C_{i-1} \dots C_0 | I_{L \times L-i}]$ , where  $1 \leq i < L$ .

When  $C_P=C_0$  then  $M^{P+1}=M$ , where  $P$  is the period of the vector  $C$ .

Now we can calculate  $C_i$  [4] as follows:

$$C_i=M \times C_{i-1}, \quad i=1,2,\dots \quad \dots(4.4)$$

Relation (4.3) can be rewritten in matrix form:

$$A_0 \times C_i = s_i, \quad i=0,1,\dots,L-1 \quad \dots(4.5)$$

if  $i=0$  then  $A_0 \times C_0 = s_0$  is the 1<sup>st</sup> equation of the SLE,

if  $i=1$  then  $A_0 \times C_1 = s_1$  is the 2<sup>nd</sup> equation of the SLE, and

if  $i=L-1$  then  $A_0 \times C_{L-1} = s_{L-1}$  is the  $L^{\text{th}}$  equation of the SLE.

In general:

$$A_0 \times \Omega = S \quad \dots(4.6)$$

$\Omega$  represents the matrix of all  $C_i$  vectors s.t.

$$\Omega = (C_0 C_1 \dots C_{L-1}) \quad \dots(4.7)$$

The SLE can be formulated as follows:

$$\Lambda = [\Omega^T | S^T] \quad \dots(4.8)$$

$\Lambda$  represents the extended (augmented) matrix of the SLE.

**Example (4.1)**

Let the  $SR_3$  has  $C_0^T = (0,1,1)$  and  $S = (0,0,1)$ , by using equation (4.6), we get:

$$C_1 = M \times C_0 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \text{ in the same way, } C_2 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix},$$

From equation (4.8) we have:

$$A_0 \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} = (0,0,1), \text{ this system can be written as equations:}$$

$$a_2 + a_3 = 0$$

$$a_1 + a_2 = 0$$

$$a_1 + a_2 + a_3 = 1$$

Then the augmented matrix  $\Lambda$  of SLE after using formula (4.8) is:

$$\Lambda = \left[ \begin{array}{ccc|c} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{array} \right] \quad \dots(4.9)$$

The pseudo code of constructing the SLE (find the extended matrix  $\Lambda$ ) for single LFSR algorithm is:



**NAME : Constructing SLE for SLSCC Algorithm  
(CSLESRA).**

:  $L, C_0, S$  {length of LFSR, feedback vector, output seq.}     **INPUT**  
: Construct the Matrix  $M$  { $M=[C_0/I_{L \times L-1}]$ }     **PROCESS**  
Construct the Matrix  $\Omega=(C_0C_1 \dots C_{L-1})$   
: The SLE  $\Lambda=[\Omega^T|S^T]$  {the extended matrix of LES }     **OUTPUT**

**4.2.2 Linear Stream Cipher Cryptosystem (LSCC)**

The second case study that is applied in this thesis in the cryptanalysis of stream cipher is the linear function or XOR gate. Figure (4.1) describes the sequence  $S$  generated from the linear system.

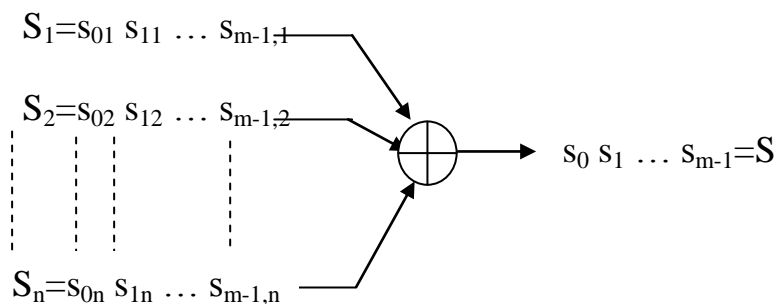


Figure (4.1): The keystream sequences generated from LSCC.

The output of each LFSR in the system is known as the simplest balance system. It is expected that each LFSR can be expressed by one SLE, but, each has unknown absolute values, since the output of each LFSR is unknown. Then, we concatenate the constructed LSCC with each other to construct one SLE with known absolute values.

For example, the logical truth table of the combining function of 2-LFSR's linear cryptosystem is as shown in Table (4.2).

Table (4.2): The XOR relation for LSCC,  $n=2$ .

|                      |          |                      |          |
|----------------------|----------|----------------------|----------|
|                      |          | <b>x<sub>1</sub></b> |          |
|                      |          | <b>0</b>             | <b>1</b> |
| <b>x<sub>2</sub></b> | <b>0</b> | <b>0</b>             | <b>1</b> |
|                      | <b>1</b> | <b>1</b>             | <b>0</b> |

Let's have n of  $SR_{L_j}$  with length  $L_j$ ,  $j=1,2,\dots,n$ , with following feedback vector:

$$C_{0j} = \begin{pmatrix} c_{01j} \\ c_{02j} \\ \vdots \\ c_{0r_jj} \end{pmatrix}, \text{ and has unknown initial value vector } A_{0j} = (a_{-1j}, \dots, a_{-L_jj}), \text{ so}$$

$$SR_{L_j} \text{ has } M_j = (C_{0j} | I_{L_j \times L_j - 1})$$

By using recurrence equation (4.4),

$$C_{ij} = M_j \times C_{i-1,j}, \quad i=1,2,\dots \quad \dots(4.10)$$

by using equation (4.5):

$$A_{0j} \times C_{ij} = s_{ij}, \quad i=0,1,\dots,L-1 \text{ and } S_j = (s_{0j}, s_{1j}, \dots, s_{m-1,j}).$$

$S_j$  represents the output vector of  $SR_{L_j}$ , which of course, is unknown too.  $m$  represents the number of variables produced from the LFSR's with consider to CF, in the same time its represents the number of equations which are be needed to solve the SLE. Of course, there is n of SLE (one SLE for each  $SR_{L_j}$  with unknown absolute values).

Now, let  $A_0$  be the extended vector for  $m$  variables, which consists of initial values from all LFSR's and  $\Omega$  is the matrix of  $C_i$  vectors considering the CF,  $C_i$  represents the extended vector of all feedback vectors  $C_{ij}$ , then  $A_0 \times \Omega = S$ .

As known, the outputs of every LFSR of the LSCC are XORed with each other to obtain the sequence  $S$  which is generating from this cryptosystem.

Since the  $SR_{L_j}$  has  $L_j$  number of unknown initial values, then  $m = \sum_{j=1}^n L_j$ .

Now, all the vectors  $A_{0j}$  are extended from  $r_j$  to  $m$  as follows:

$$A_{01} = (a_{11}, \dots, a_{L_1}, 0 \dots 0, \dots, 0 \dots 0)$$

$$A_{02} = (0 \dots 0, a_{12}, \dots, a_{L_2}, \dots, 0 \dots 0)$$

And so on..

$$A_{0n} = (0 \dots 0, 0 \dots 0, \dots, a_{1n}, \dots, a_{L_n})$$

And let:

$$A_0 = \sum_{j=1}^n A_{0j} = (a_{11}, \dots, a_{L_1}, a_{12}, \dots, a_{L_2}, \dots, a_{1n}, \dots, a_{L_n}) = (l_0, l_1, \dots, l_{m-1})$$

Where  $l_0 = a_{11}$ ,  $l_1 = a_{21}$ ,  $\dots$ ,  $l_{m-1} = a_{L_n}$ , or it can be deduced from the following formula:

$$l_k = a_{ij}, \text{ where } k = (i-1) + \sum_{h=1}^{j-1} L_h, \quad j = 1, 2, \dots, n, \quad i = 1, 2, \dots, L_j. \quad \dots(4.11)$$

In fact,  $A_0$  represents a concatenation of all  $A_{0j}$  vectors respectively. The same process will be done on the feedback vectors  $C_{ij}$  which must be found first from equation (4.10). Therefore,  $C_i$  will be the extended concatenation vector of all feedback  $C_{ij}$  vectors too, s.t.

$$C_i = \begin{pmatrix} C_{i1} \\ C_{i2} \\ \vdots \\ C_{in} \end{pmatrix}, \quad i = 0, 1, \dots, m-1$$

Since the CF of LSCC is XOR, then  $S$  can be obtained from XORed all unknowns  $S_j$ . Since we need  $m$  equations, that means every LFSR shifts  $m$  movements, then:

$$S_j = (s_{0j}, s_{1j}, \dots, s_{m-1,j}), j=1, 2, \dots, n, \text{ and } s_i = \sum_{j=1}^n s_{ij}, i=0, 1, \dots, m-1, \text{ (the sum}$$

here is XOR), then:

$$S = \sum_{j=1}^n S_j = (s_0, s_1, \dots, s_{m-1})$$

So  $\Omega$  can be gotten from equation (4.7) and by applying equation (4.6), the SLE can be constructed.

The pseudo code of constructing the SLE for linear system algorithm is:

**NAME : Constructing SLE for LSCC Algorithm (CSLELSA).**

{n: number of LFSR's} : n, S **INPUT**

$L_j, C_{0j}, j=1 : n$  {length of LFSR(i), feedback vector(i)}

: calculate  $m = L_1 + L_2 + \dots + L_n$  **PROCESS**

**for**  $j = 1 : n$

Construct the Matrix  $M_j$

$$C_{0j} = C_{01j}C_{02j} \dots C_{0L_jj}$$

**end;** {for j }

**for**  $i = 1 : m$

**for**  $j = 1 : n$

$$C_{ij} = M_j \times C_{i-1j}$$

**end;** {for i,j }

$C_i =$  concatenation  $(C_{i1}, C_{i2}, \dots, C_{in}); \{i=0, 1, \dots, m-1\}$

Construct the Matrix  $\Omega = (C_0 C_1 \dots C_{m-1})$

: The LES  $\Lambda = [\Omega^T | S^T]$  {the extended matrix of SLE } **OUTPUT**

**END.**

### Example (4.2)

Let's have the following feedback vectors for 3 LFSR with length 2, 3 and 4:

$$C_{01} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, C_{02} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \text{ and } C_{03} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \text{ then } m=9.$$

And let  $S = (1, 1, 1, 0, 1, 1, 0, 1, 1)$ .

By using equation (4.4),

$$C_{01} = C_{31} = C_{61} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, C_{11} = C_{41} = C_{71} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \text{ and } C_{21} = C_{51} = C_{81} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

$$C_{02}=C_{72}=\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, C_{12}=C_{82}=\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, C_{22}=\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, C_{32}=\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, C_{42}=\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, C_{52}=\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix},$$

$$C_{62}=\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

$$C_{13}=\begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}, C_{23}=\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, C_{33}=\begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}, C_{43}=\begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, C_{53}=\begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, C_{63}=\begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, C_{73}=\begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix},$$

$$C_{83}=\begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

Then  $C_0^T=(1,1,1,0,1,1,0,0,1)$ .

The SLE can be written as follows:

$$A_0 \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} = (1,1,1,0,1,1,0,1,1)$$

s.t.

$A_0=(a_{11},a_{21},a_{12},a_{22},a_{32},a_{13},a_{23},a_{33},a_{43})=(l_0,l_1,l_2,l_3,l_4,l_5,l_6,l_7,l_8)$ , and the extended matrix  $\Lambda$  which is can be calculated from equation (4.8) is:

$$\Lambda = \left[ \begin{array}{cccccccc|c} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \end{array} \right] \quad \dots(4.12)$$

#### 4.2.3 Nonlinear Threshold Stream Cipher Cryptosystem (NTSCC)

The third proposed study case, is the Threshold Generator known. Its CF called the majority function. This cryptosystem consists of odd numbers of LFSRs, so naturally, there are a majority in the output bits of one or another, this means, which one is the major, will be the output. So it can be represented by following equation:

$$BF_3(x_1,x_2,x_3)=x_1 \cdot x_2 \oplus x_1 \cdot x_3 \oplus x_2 \cdot x_3$$

Table (4.3) shows the truth table of 3-LFSR's of Threshold generator cryptosystem.

Table (4.3): The 3-LFSR's of NTSCC Generator.

| $x_1$ | $x_2$ | $x_3$ | $BF_3$ |
|-------|-------|-------|--------|
| 0     | 0     | 0     | 0      |
| 0     | 0     | 1     | 0      |
| 0     | 1     | 0     | 0      |
| 0     | 1     | 1     | 1      |

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

for this reason  $m=L_1L_2+L_1L_3+L_2L_3$ .

The initial value of this cryptosystem is:

$$A_0=A_{01}A_{02}+A_{01}A_{03}+A_{02}A_{03}=(d_0,d_1,\dots,d_{m-1}) \quad \dots(4.13)$$

(where + is concatenation to the vectors) s.t.

$d_0=a_{-11}a_{-12}$ ,  $d_1=a_{-11}a_{-22}$ , ...,  $d_{r-1}=a_{L_2,2}a_{L_3,3}$ , or it can be taken from the

following System of NonLinear Equations (SNLE):

$$d_k = \begin{cases} a_{-i1}a_{-j2}, & \text{when } k = i * L_2 + j, \text{ s.t. } i = 0, \dots, L_1 - 1, j = 0, \dots, L_2 - 1 \\ a_{-i1}a_{-j3}, & \text{when } k = i * L_3 + j + L_1L_2, \text{ s.t. } i = 0, \dots, L_1 - 1, j = 0, \dots, L_3 - 1 \\ a_{-i2}a_{-j3}, & \text{when } k = i * L_3 + j + L_1L_2 + L_1L_3, \text{ s.t. } i = 0, \dots, L_2 - 1, j = 0, \dots, L_3 - 1 \end{cases} \quad \dots(4.14)$$

(this arrangement of unknowns can be changed according to the researcher requirements so it is not standard).

In the same way, equation (4.14) can be applied on the feedback vector

$C_{ij}$ :

$$C_i=C_{i1}C_{i2}+C_{i1}C_{i3}+C_{i2}C_{i3}.$$

And the sequence S will be:

$$S=S_1S_2+S_1S_3+S_2S_3 \text{ s.t. } s_i=s_{i1}s_{i2} \oplus s_{i1}s_{i3} \oplus s_{i2}s_{i3},$$

$s_i$  is the element i of S.

So the SNLE which be changed to SLE can be gotten by equation (4.13).

Figure (4.2) shows the sequence S which is generated from Brür Generator [4].



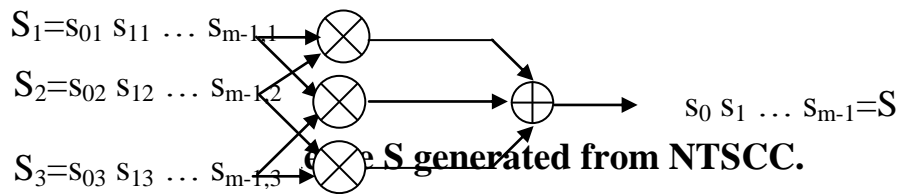


Figure 4.3

The pseudo code of constructing the SLE for NTSCC algorithm is:

```

NAME : Constructing SLE for NTSCC Algorithm (CSLENTSA).
: S      INPUT
Lj, C0j, j = 1 : 3
: Calculate m = L1*L2+ L1*L3+ L2*L3 PROCESS
for j = 1 : 3
Construct the Matrix Mj
end; {for j }
C0j = C01j*C02j+C01j*C03j+C02j*C03j
for i = 0 : m-1
Cij = Ci1j*Ci2j+Ci1j*Ci3j+Ci2j*Ci3j
end; {for i }
for i = 0 : m-1
Ci = Ci1*Ci2+Ci1*Ci3+Ci2*Ci3
end; {for i }
Construct the Matrix Ω=(C0C1...Cm-1)
: The SLE Λ=[ ΩT|ST] {the extended matrix of SLE } OUTPUT
END.

```

**Example (4.3)**

Let's have the following feedback vectors for 3 LFSR with length 2,3 and 4:

$$C_{01}=\begin{pmatrix} 1 \\ 1 \end{pmatrix}, C_{02}=\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \text{ and } C_{03}=\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \text{ then } m=2*3+2*4+3*4=26.$$

And let the required sequence is:

$$S=(1,0,1,1,0,1,1,1,1,1,0,1,1,0,1,0,0,1,1,0,0,1,0,1,0,1,1,0).$$

By using equation (4.4),

$$C_{01}=C_{31}=C_{61}=C_{91}=C_{12,1}=C_{15,1}=C_{18,1}=C_{21,1}=C_{24,1}=\begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

$$C_{11}=C_{41}=C_{71}=C_{10,1}=C_{13,1}=C_{16,1}=C_{19,1}=C_{22,1}=C_{25,1}=\begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

$$C_{21}=C_{51}=C_{81}=C_{11,1}=C_{14,1}=C_{17,1}=C_{20,1}=C_{23,1}=\begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

$$C_{02}=C_{72}=C_{14,2}=C_{21,2}=\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, C_{12}=C_{82}=C_{15,2}=C_{22,2}=\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix},$$

$$C_{22}=C_{92}=C_{16,2}=C_{23,2}=\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, C_{32}=C_{10,2}=C_{17,2}=C_{24,2}=\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix},$$

$$C_{42}=C_{11,2}=C_{18,2}=C_{25,2}=\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, C_{52}=C_{12,2}=C_{19,2}=\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, C_{62}=C_{13,2}=C_{20,2}=\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

$$C_{03}=C_{15,3}=\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}, C_{13}=C_{16,3}=\begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}, C_{23}=C_{17,3}=\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, C_{33}=C_{18,3}=\begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}, C_{43}=C_{19,3}=\begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

,

$$C_{53}=C_{20,3}=\begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, C_{63}=C_{21,3}=\begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, C_{73}=C_{22,3}=\begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}, C_{83}=C_{23,3}=\begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix},$$

$$C_{93}=C_{24,3}=\begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}, C_{10,3}=C_{25,3}=\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, C_{11,3}=\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, C_{12,3}=\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, C_{13,3}=\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, C_{14,3}=\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

by applying equation (4.4),  $C_0^T$  will be:

$$C_0^T = (1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1).$$

Therefore the augmented matrix will be:

$$\Lambda = \left[ \begin{array}{cccccccccccccccccccccccc|c} 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{array} \right] \dots(4.15)$$

### 4.3 The Proposed PSO Cryptanalysis System (PSOCS)

PSO is an extremely simple concept, and can be implemented without complex data structure. No complex or costly mathematical functions are used, and it doesn't require a great amount of memory [49]. The facts of PSO has fast convergence, only a small number of control parameters, very simple computations, good performance, and the lack of derivative computations made it an attractive option for solving the problems.

In this thesis we will use one of important methods of swarm intelligence methods; it's the PSO method, in the field of cryptanalysis of stream cipher systems, which it has good achievement in the cryptanalysis field. Before we discuss the operators and parameters of PSOCS we will show how the system constructs the SLE.

#### 4.3.1 PSOCS Constructing a SLE

Before we start the attack of a certain class of stream ciphers using PSOSC, the first step is to select the known algorithm that will be attacked. The second step is to prepare the constructed key equations for m steps, where m is the length of string. These equations will be represented in binary form.

#### I. PSOCS Constructing a SLE for SLSCC

For example, let us use 10-stage m-LFSR, which has  $1+x^3+x^{10}$  as characteristic primitive polynomial. Table (4.4) shows the 10-stage

equations of LES for single LFSR with binary representation.

Table (4.4) SLE for SRSCC with binary representation.

| <b>Eq. No.</b> | <b>Equation</b>            | <b>Binary Code</b> |
|----------------|----------------------------|--------------------|
| 1              | $a_3+a_{10}=1$             | 0010000001 1       |
| 2              | $a_2+a_9=0$                | 0100000010 0       |
| 3              | $a_1+a_8=1$                | 1000000100 1       |
| 4              | $a_3+a_7+a_{10}=1$         | 0010001001 1       |
| 5              | $a_2+a_6+a_9=0$            | 0100010010 0       |
| 6              | $a_1+a_5+a_8=1$            | 1000100100 1       |
| 7              | $a_3+a_4+a_7+a_{10}=1$     | 0011001001 1       |
| 8              | $a_2+a_3+a_6+a_9=0$        | 0110010010 0       |
| 9              | $a_1+a_2+a_5+a_8=1$        | 1100100100 1       |
| 10             | $a_1+a_3+a_4+a_7+a_{10}=0$ | 1011001001 0       |

## II. PSOCS Constructing a SLE for SLSCC

For example, let us use two LFSR's. Each has the following information:

- First LFSR has characteristic polynomial  $1+x+x^4$  with initial key values 1001.
- Second LFSR has characteristic polynomial  $1+x+x^7$  with initial key values 1000001.

The equations of SLE for the linear cryptosystem are shown in Table (4.5).

Table (4.5) SLE for SLSCC with binary representation.

| Eq. No. | Separating Equations                           | Single Equations                        | Binary Code   |
|---------|--|---|---------------|
| 1       | $a_1+a_4=0\dots b_1+b_7=0$                     | $a_1+a_4+b_1+b_7=0$                     | 10011000001 0 |
| 2       | $a_1+a_3+a_4=0\dots b_1+b_6+b_7=0$             | $a_1+a_3+a_4+b_1+b_6+b_7=0$             | 10111000011 0 |
| 3       | $a_1+a_2+a_3+a_4=0\dots b_1+b_5+b_6+b_7=0$     | $a_1+a_2+a_3+a_4+b_1+b_5+b_6+b_7=0$     | 11111000111 0 |
| 4       | $a_2+a_3+a_4=1\dots b_1+b_4+b_5+b_6+b_7=0$     | $a_2+a_3+a_4+b_1+b_4+b_5+b_6+b_7=1$     | 01111001111 1 |
| 5       | $a_1+a_2+a_3=1\dots b_1+b_3+b_4+b_5+b_6+b_7=0$ | $a_1+a_2+a_3+b_1+b_3+b_4+b_5+b_6+b_7=1$ | 11101011111 1 |
| 6       | $a_2+a_4=1\dots b_1+b_2+b_3+b_4+b_5+b_6+b_7=0$ | $a_2+a_4+b_1+b_2+b_3+b_4+b_5+b_6+b_7=1$ | 01011111111 1 |
| 7       | $a_1+a_3=1\dots b_2+b_3+b_4+b_5+b_6+b_7=1$     | $a_1+a_3+b_2+b_3+b_4+b_5+b_6+b_7=0$     | 10100111111 0 |
| 8       | $a_1+a_2+a_4=0\dots b_1+b_2+b_3+b_4+b_5+b_6=1$ | $a_1+a_2+a_4+b_1+b_2+b_3+b_4+b_5+b_6=1$ | 1101111110 1  |
| 9       | $a_3+a_4=1\dots b_2+b_3+b_4+b_5+b_7=1$         | $a_3+a_4+b_2+b_3+b_4+b_5+b_7=0$         | 00110111101 0 |
| 10      | $a_2+a_3=0\dots b_1+b_2+b_3+b_4+b_6=1$         | $a_2+a_3+b_1+b_2+b_3+b_4+b_6=1$         | 01101111010 1 |
| 11      | $a_1+a_2=1\dots b_2+b_3+b_5+b_7=1$             | $a_1+a_2+b_2+b_3+b_5+b_7=0$             | 11000110101 0 |

### III. PSOCS Constructing a SLE for STSCC

Three LFSR's are used, each has the following information:

- 1<sup>st</sup> LFSR has characteristic polynomial  $1+x+x^3$  with initial key values 101.
- 2<sup>nd</sup> LFSR has characteristic polynomial  $1+x+x^4$  with initial key values 1001.
- 3<sup>rd</sup> LFSR has characteristic polynomial  $1+x^2+x^5$  with initial key values 10001.

When the SNLE changes, 47 equations of SLE are obtained for the Threshold generator cryptosystem shown in Table (4.6).

Table (4.6) SLE for STSCC with binary representation.

| Eq. No. | Equations   | Binary Code       | Extension of Equations in Binary Code                |
|---------|---|-------------------|--|
| 1       | $a_1+a_3=0\dots b_1+b_4=0\dots c_2+c_5=1$             | 101100101001<br>0 | 1001000010010100100000010010100100000000001001<br>0  |
| 2       | $a_1+a_2+a_3=0\dots b_1+b_3+b_4=0\dots c_1+c_4=1$     | 111101110010<br>0 | 10111011101110010100101001010010000001001010010<br>0 |
| 3       | $a_2+a_3=1\dots b_1+b_2+b_3+b_4=0\dots c_2+c_3+c_5=1$ | 011111101101<br>1 | 00001111111100000011010110101101011010110101101<br>1 |
| 4       | $a_1+a_2=1\dots b_2+b_3+b_4=1\dots c_1+c_2+c_4=1$     | 110011111010<br>1 | 01110111000011010110100000000000110101101011010<br>1 |
| 5       | $a_3=1\dots b_1+b_2+b_3=1\dots c_1+c_2+c_3+c_5=0$     | 001111011101<br>1 | 00000000111000000000001110111101111011110100000<br>1 |

| □  | □   | □                 | □  |
|----|---|-------------------|--|
| 41 | $a_2=0..b_1+b_2=1..c_4+c_5=1$                 | 010110000011<br>1 | 00001100000000000000110000000011000110000000000<br>1         |
| 42 | $a_1=1..b_4=1..c_3+c_4=0$                     | 100000100110<br>1 | 0001000000000001100000000000000000000000000000000000110<br>1 |
| 43 | $a_1+a_3=0..b_3=0..c_2+c_3=0$                 | 101001001100<br>0 | 001000000010011000000011000000000000001100000000<br>0        |
| 44 | $a_1+a_2+a_3=0..b_2=0..c_1+c_2=1$             | 111010011000<br>0 | 0100010001001100011000110000000011000000000000000<br>0       |
| 45 | $a_2+a_3=1..b_1=1..c_1+c_2+c_5=0$             | 011100011001<br>1 | 1000101001001010010000100000000001000000000000000<br>0       |
| 46 | $a_1+a_2=1..b_1+b_4=0..c_1+c_2+c_4+c_5=0$     | 110100111011<br>0 | 0010001000100100001100011000100001100000000000000<br>0       |
| 47 | $a_3=1..b_1+b_3+b_4=0..c_1+c_2+c_3+c_4+c_5=0$ | 001101111111<br>0 | 01000100010011000110001100010000011000010000000<br>0         |

Each time, m equations are needed to solve the system for each cryptosystem. First, one is the SLE of SRSCC, where m here equals the length of LFSR. The second, is the SLE of SLSCC, m represents the summation of the two combined LFSR's. While the third, is the SNLE of STSCC, which needs  $m = r_1 * r_2 + r_1 * r_3 + r_2 * r_3$ , where  $r_i$  is length of register i,  $i=1,2,3$ , to solve the system.

### **4.3.2 Representation of a Random Solution**

For the purpose of this study, a SLE or SNLE is decoded by a binary code. For example the equation  $a_2 \oplus a_2 = 1$  of a single LFSR with length 5 is decoded by the equation string (01001-1), where the absolute value (right hand) of the equation is the real output key of the attacked cryptosystem. The equations must be constructed and stored in data base file, since these equations are constant for fixed LFSR's length according to the connection function and combining function (if it exist). This representation indicates that the size of the solutions space is  $2^m - 1$  (ignoring the zero string). When m is as large as possible, then a purely random search is not acceptable.

### **4.3.3 Swarm Solution Initialization**

For the initialization process, it can initialize the population of the swarm by a random sample of combinations of 0 and 1 with  $m$ -string length which represents the probable initial values LFSR's. The creation of the swarm's population must submit to what we called non-zero initial condition. In this condition, the zero's initial of LFSR's must be avoided. For example, you wish to initiate initial values of LFSR with length 5; you ignore the initial value 00000 for single LFSR or other cryptosystems. Another example the string 00001100100 is ignored for linear cryptosystem, which consists of two LFSR's with lengths 4 and 7 respectively, since the initial of the first register, is 0000. A pseudo code for generating the swarm initialization is shown below:

```
NAME : Swarm Initialization Algorithm (SIA).  
{ Swarm Size, Length of Particle } : SS, L INPUT  
: for i = 1 : SS PROCESS  
    for j = 1 : L  
        pj = Random(2);  
        vj = Random(2);  
    end;  
    Particle_P(i)=(s1,s2,...,sL); { position }  
    Particle_V(i)=(v1,v2,...,vL); { velocity }  
{ position and velocity } : Swarm of Particles OUTPUT
```

### **4.3.4 PSOSC Fitness Function**

The fitness function is used to determine the “best” solution. The process of the fitness function selection is as follows:

1. From swarm, a particle initial solution of length n-bits is extended to  $m$  bits;  $m=L_1 \oplus L_2 \oplus L_3$  for linear system,  $m=L_1 * L_2 \oplus L_1 * L_3 \oplus L_2 * L_3$  for Threshold cryptosystem and it remains as it is for the Single LFSR ( $m=n=L$ ), so we get the vector  $A=(a_1, a_2, \dots, a_m)$  after extension.
2. The extended string bit  $a_j$  product with corresponding equation vector bit  $x_j$ , where  $1 \leq j \leq m$  s.t. the equation string is  $X=(x_1, x_2, \dots, x_m)$  and calculate the observed value:

$$o_i = a_1 * x_1 \oplus a_2 * x_2 \oplus \dots \oplus a_m * x_m = \sum_{j=1}^m a_j * x_j = AX^T \quad \dots(4.16)$$

3. Compare the observed value  $o_i$  with key value  $b_i$  which represents the known output value of the cryptosystem, by using mean absolute error (MAE), in general  $O=(o_1, o_2, \dots, o_m)$  and  $B=(b_1, b_2, \dots, b_m)$  s.t.

$$MAE = \frac{1}{m} \sum_{i=1}^m |o_i - b_i| = \frac{1}{m} |O - B| \quad \dots(4.17)$$

4. The Fitness value is

$$Fitness = 1 - MAE = 1 - \frac{1}{m} \sum_{i=1}^m |O_i - K_i| \quad \dots(4.18)$$

where

$m$  : The length of the individual string or equation string.

$a_j$ : is the initial value  $j$  in vector  $A$ .

$x_j$ : is the equation variable  $j$  in the string  $X$ .

$o_i$ : is the measured or observed value  $i$  in the vector  $O$  calculated from equation (4.16).

$b_i$ : is the key bit (actual value)  $I$  in the vector  $B$ .

When the observed vector  $O$  matches the key vector  $B$ , for all  $1 \leq i \leq m$ , then the summation terms MAE in equation (4.17) evaluate to 0 so the fitness value is 1. The fitness equation is bounded below by 0 though it does not actually evaluate to 0. The fact that a fitness value of 0 is never achieved does not affect the algorithm since high fitness values are more



important than low fitness values. As a result, the search process is always moving towards fitness values closer to or equal 1. The steps of the *Stream Cipher Fitness Algorithm* are shown below:

|  |
|--|
| <p><b>NAME : Stream Cipher Fitness Algorithm (SCFA).</b></p> <p><b>INPUT</b> : <b>READ</b> A vector; {Initial string with length L}</p> <p style="padding-left: 40px;"><b>READ</b> X vector; {Equation string from data base file}</p> <p style="padding-left: 40px;"><b>READ</b> B vector ;{Actual key=absolute value of SLE}</p> <p style="padding-left: 40px;"><b>OUTPUT</b> : Fitness value;</p> <p style="padding-left: 40px;"><b>PROCESS</b> : <b>FOR</b> i = 1 : m</p> <p style="padding-left: 80px;"><math>o_i = \sum_{j=1}^m a_j * x_j</math>; {XOR sum, O is observed key vector}</p> <p style="padding-left: 120px;"><math>AE_i =  o_i - b_i </math>;</p> <p style="padding-left: 80px;"><b>END</b>;</p> <p style="padding-left: 40px;"><math>MAE = \frac{1}{m} \sum_{i=1}^m AE_i</math>; { MAE is the Mean Absolute Error }</p> <p style="padding-left: 80px;">Fitness = 1-MAE;</p> <p style="padding-left: 40px;"><b>END.</b></p> |
|--|

#### 4.3.5 PSOSC Parameters Selection

As the main objective of this research is to verify the impact of the selection of social topologies in the behavior of the PSO, the tuning parameters are fixed. They are tuned to the values that are widely used by the community and that are seemed to be the most appropriate ones.

Table (4.7) shows the different parameters used.

Table 4.7: Parameters selection of PSOCS

| Parameter Symbol | Description     | Value     |
|------------------|-----------------|-----------|
| $c_1$            | Self confidence | 1.5 – 2.0 |

|                   |                                  |             |
|-------------------|----------------------------------|-------------|
| $c_2$             | Swarm Confidence                 | $c_1$       |
| $w$               | Inertia weight                   | 0.9,...,0.4 |
| $v_{\max}$        | maximum of velocity              | 0.5         |
| $v_{\min}$        | minimum of velocity              | $-v_{\max}$ |
| <i>Swarm_Size</i> | Number of particles in the swarm | 30-50       |
| <i>Max_Iter</i>   | Maximum number of iteration      | 100-2000    |

#### 4.3.6 The PSOCS Algorithm

The following are the main steps of an algorithmic description of the attack on stream cipher using PSOCS:

|   |
|---|
| <p><b>NAME : PSOCS algorithm (PSOCSA).</b></p> <p><b>INPUT</b> : The construction SLE/SNLE, parameters (<math>c_1</math>, <math>w</math>, <math>v_{\max}</math>, <math>SS</math>, <math>m</math>, <math>Max\_Iter</math>).</p> <p><b>OUTPUT</b> : The key having the highest fitness as found by PSOCS;</p> <p style="text-align: right;"><b>PROCESS :</b></p> <p><b>Step 1:</b> Randomly generate the initial particles (keys) and velocities to a swarm.</p> <p><b>Step 2:</b> Calculate the fitness value of each of the particles (keys) using equation (4.18).</p> <p><b>Step 3:</b> If the current position of the particle is better than the previous history, update the particles to indicate this fact.</p> <p><b>Step 4:</b> Find the best particle (key) of the swarm. Update the positions of the particles by using equations (3.1-a, 3.1-b):</p> <p><b>Step 5:</b> If the <math>Max\_iter</math> has been reached or if the key with fitness=1.0 value is found, then goto <b>step 6</b> or else goto <b>step 2</b>.</p> <p><b>Step 6:</b> Keep the best key obtained so far in the output key variables.</p> |
|---|

#### 4.4 Experimental Results of Applying PSOCS

Two stopping criteria are used to stop the system of the PSOCS, the first criterion, is to reach the  $max\_iter$  that is enough to reach this level of fitness. The second, when the fitness reaches the value equal to

1.0, then there is no need to reach the maximum number of iterations. Now we will describes the experimental results of applying PSOCS for each study case.

The best way to illustrate the proceeding of PSOCS algorithm is by looking at the development of the swarm fitness results over time.

#### **4.4.1 Experimental Results of Applying PSOCS on SLSCC**

For this study case, only 10 initial particles (keys) are appear in the swarm. The PSOCS began by generating 10 random initial particles (keys) as shown table (4.8).

Table (4.8): PSOCS initialization of random initial particles for SLSCC.

| <b>Key</b>                | <b>Strings of Swarm of SLSCC</b> | <b>Fitness</b> |
|---------------------------|----------------------------------|----------------|
| <b>1</b>                  | 00010111110                      | 0.3636         |
| <b>2</b>                  | 00110010110                      | 0.4545         |
| <b>3</b>                  | 01111100111                      | 0.5455         |
| <b>4</b>                  | 00110010110                      | 0.4545         |
| <b>5</b>                  | 10111111100                      | 0.3636         |
| <b>6</b>                  | 01110011010                      | 0.4545         |
| <b>7</b>                  | 11010101111                      | 0.6364         |
| <b>8</b>                  | 11011001100                      | 0.5455         |
| <b>9</b>                  | 00010111110                      | 0.3636         |
| <b>10</b>                 | 11011000101                      | 0.5455         |
| <b>Average of Fitness</b> |                                  | <b>0.4727</b>  |

As expected, none of the random keys are close to the actual key which is reflected in the fact that the average fitness for these keys is 0.4727. The best of these 10 random keys (key7) has a fitness value of only 0.6364.

After only 10 iterations, this swarm of 10 keys begins to take on a set of common features that approach the exact key: The PSOCS results after 10 generations, the initial particles (keys) as shown table (4.9).

Table (4.9): The PSOCS results after 10 generations for SLCSS.

| <b>Key</b>                | <b>Strings of Swarm of SLCSS</b> | <b>Fitness</b> |
|---------------------------|----------------------------------|----------------|
| <b>1</b>                  | 11010001010                      | 0.8182         |
| <b>2</b>                  | 00000001000                      | 0.4545         |
| <b>3</b>                  | 00000000000                      | 0.0909         |
| <b>4</b>                  | 11010001010                      | 0.8182         |
| <b>5</b>                  | 00000001000                      | 0.4545         |
| <b>6</b>                  | 00000001000                      | 0.4545         |
| <b>7</b>                  | 00000001000                      | 0.4545         |
| <b>8</b>                  | 00000000000                      | 0.0909         |
| <b>9</b>                  | 00000001000                      | 0.4545         |
| <b>10</b>                 | 11010001010                      | 0.8182         |
| <b>Average of Fitness</b> |                                  | <b>0.4909</b>  |

Now the average fitness is 0.4909. The best key (key1) has fitness value of 0.8182. After 30 iterations, the swarm begins to converge at a high rate of speed.

After only 20 iterations, this swarm starts to take on a set of common features that approach the exact key: The PSOCS results after 20 generations, the initial particles (keys) are shown table (4.10).

Table (4.10): The PSOCS results after 20 generations for SLCSS.

| <b>Key</b>                | <b>Strings of Swarm of SLCSS</b> | <b>Fitness</b> |
|---------------------------|----------------------------------|----------------|
| <b>1</b>                  | 10000000100                      | 0.9091         |
| <b>2</b>                  | 00000000100                      | 0.3636         |
| <b>3</b>                  | 10000000100                      | 0.9091         |
| <b>4</b>                  | 00000000100                      | 0.3636         |
| <b>5</b>                  | 00000000100                      | 0.3636         |
| <b>6</b>                  | 10000000100                      | 0.9091         |
| <b>7</b>                  | 10000000100                      | 0.9091         |
| <b>8</b>                  | 10000000100                      | 0.9091         |
| <b>9</b>                  | 10000000100                      | 0.9091         |
| <b>10</b>                 | 00000000100                      | 0.3636         |
| <b>Average of Fitness</b> |                                  | <b>0.6909</b>  |

Now the average fitness is 0.6909. The best key (key1) has fitness value of 0.9091. After 20 iterations, the swarm begins to converge at a high rate of speed.

After 45 iterations the PSOCS is stopped since it finds the actual key as shown in table (4.11).

Table (4.11): The PSOCS finds actual key after 45 generations for SLCSS.

| <b>Key</b>                | <b>Strings of Swarm of SLCSS</b> | <b>Fitness</b> |
|---------------------------|----------------------------------|----------------|
| <b>1</b>                  | 10000000001                      | 1.0000         |
| <b>2</b>                  | 11010101000                      | 0.8182         |
| <b>3</b>                  | 11010101000                      | 0.8182         |
| <b>4</b>                  | 11010101000                      | 0.8182         |
| <b>5</b>                  | 00000000000                      | 0.0909         |
| <b>6</b>                  | 11010101000                      | 0.8182         |
| <b>7</b>                  | 01000001000                      | 0.7273         |
| <b>8</b>                  | 11010101000                      | 0.8182         |
| <b>9</b>                  | 11010101000                      | 0.8182         |
| <b>10</b>                 | 10000000001                      | 1.0000         |
| <b>Average of Fitness</b> |                                  | <b>0.7364</b>  |

The average fitness is now at 0.7364, key1 and key8 turn out to have the highest fitness (1.0) and on test is the exact key.

Table (4.12), provides the iteration number for which improvement is noted in the evaluation function, together with the value of the function.

Table (4.12): Results of applying PSOSC on SLSCC.

| <b>Iter.</b> | <b>Key</b> | <b>Fitness</b> | <b>Average</b> | <b>Best Initial Key</b> |
|--------------|------------|----------------|----------------|-------------------------|
| 0            | 7          | 0.6364         | 0.4727         | 11010101111             |
| 10           | 4          | 0.8182         | 0.4909         | 11010001010             |
| 20           | 1          | 0.9091         | 0.6909         | 10000000100             |
| 45           | 1          | 1.0000         | 0.7364         | 10000000001             |

The best initial key is: 1 0 0 0 0 0 0 0 0 1, which is the real initial key for SLSCC.

Figure (4.3) shows the achievement of PSOCS to find the actual key for SLSCC.

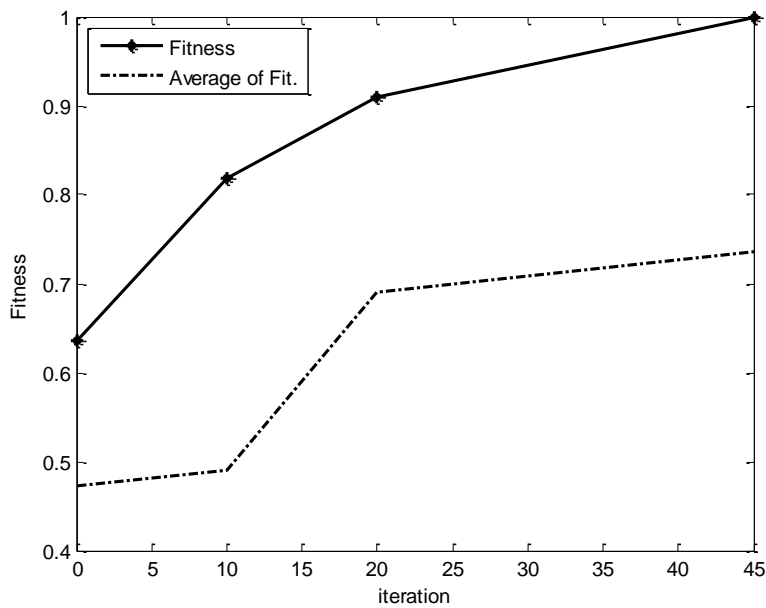


Figure (4.3) PSOCS's results to find the actual key for SLSCC.

#### 4.4.2 Experimental Results of Applying PSOCS on LSCC

For this study case (LSCC), 10 initial particles (keys) are appear in the swarm. The PSOCS began by generating 10 random initial particles (keys) as shown table (4.13):

Table (4.13): PSOCS initialization of random initial particles for LSCC.

| <b>Key</b>                | <b>Strings of Swarm of LSCC</b> | <b>Fitness</b> |
|---------------------------|---------------------------------|----------------|
| <b>1</b>                  | 101100011010                    | 0.4167         |
| <b>2</b>                  | 111001011011                    | 0.7500         |
| <b>3</b>                  | 000010011001                    | 0.5000         |
| <b>4</b>                  | 000101010011                    | 0.5000         |
| <b>5</b>                  | 111001011011                    | 0.7500         |
| <b>6</b>                  | 110100100100                    | 0.5000         |
| <b>7</b>                  | 111011110111                    | 0.5833         |
| <b>8</b>                  | 000010011001                    | 0.5000         |
| <b>9</b>                  | 111011110111                    | 0.5833         |
| <b>10</b>                 | 111001011011                    | 0.7500         |
| <b>Average of Fitness</b> |                                 | <b>0.5333</b>  |

As expected, none of the random keys are close to the actual key which is described in the fact that the average fitness for these keys is 0.5333. The best of these 10 random keys (key2) has a fitness value of only 0.7500.

Table (4.14) shows the improvement in the results of finding the actual initial key for LSCC.

Table (4.14): Results of applying PSOSC on SLSCC.

| <b>Iter.</b> | <b>Key</b> | <b>Fitness</b> | <b>Average</b> | <b>Best Initial Key</b> |
|--------------|------------|----------------|----------------|-------------------------|
| 0            | 2          | 0.7500         | 0.5333         | 111001011011            |
| 10           | 5          | 0.7500         | 0.5417         | 000110000000            |
| 30           | 3          | 0.8333         | 0.6083         | 011100110100            |
| 50           | 1          | 0.8333         | 0.6167         | 011010010101            |
| 120          | 2          | 0.9166         | 0.6333         | 010111010110            |
| 219          | 9          | 1.0000         | 0.6583         | 100011000001            |

The best initial key is: 100011000001, which is the actual initial key.

Figure (4.4) shows the achievement of PSOCS to find the actual key for SLSCC.

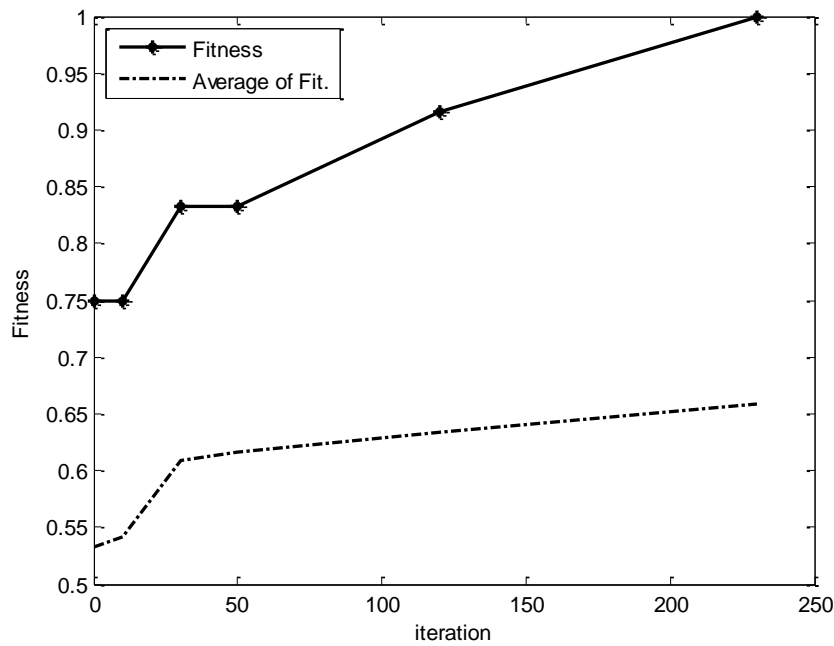


Figure (4.4) PSOCS's results to find the actual key for LSCC.

#### **4.4.3 Experimental Results of Applying PSOCS on NTSCC**

For another example of attack non-linear function, we have the NTSCC, only 10 initial keys were in the swarm. The system began by generating 10 random initial key as shown table (4.15):

Table (4.15): PSOCS initialization of random initial particles for NTSCC.

| Key | Best Key         | Extended Strings of Swarm of NTSCC                       | Fitness |
|-----|------------------|--|---------|
| 1   | 00100010000<br>1 | 00000000000010000000000000001000000000000000000000000001 | 0.5106  |
| 2   | 00010010000<br>1 | 0000100010000000000001000010000100000000000000000000     | 0.5319  |
| 3   | 00001001000<br>1 | 010101010101110111101111011100000110110000011011         | 0.4894  |
| 4   | 01000010000<br>1 | 000010110000000000010110000001011000000101101011         | 0.4468  |
| 5   | 00100010000<br>1 | 110111011101100111001110011100111001110000010011         | 0.4043  |
| 6   | 00010110000<br>1 | 00000111011100000111001110000000111001110011100          | 0.5745  |
| 7   | 01100010000<br>1 | 00000000101100000000000111101111000000111101111          | 0.4468  |
| 8   | 00100011000<br>1 | 000000000111000000000001011000000101101011010110         | 0.4468  |
| 9   | 00010010100<br>1 | 00000111011100000111001110000000111001110011100          | 0.5745  |



|    |                           |   |        |
|----|---------------------------|---|--------|
| 10 | 01000100000<br>1          | 10111011101101101011010110101101000000110101101 | 0.4894 |
|    | <b>Average of Fitness</b> |   | 0.4787 |

We notice that, none of the random keys are close to the actual key which is described in the fact that the average fitness for these keys is 0.4787. The best of these 10 random keys (key6) has a fitness value of only 0.5745.

Table 4.16 shows the improvement in the results of finding the real initial key for NTSCC.

Table (4.16): Results of applying PSOSC on NTSCC.

| Iter. | Fitness | Ave.   | Best Key     | Extended Best Initial Key                       |
|-------|---------|--------|--------------|---|
| 0     | 0.5745  | 0.4787 | 011011111100 | 0000011101110000011100111000000111001110011100  |
| 20    | 0.6596  | 0.5372 | 101001001000 | 0010000000100100000000010000000000000100000000  |
| 50    | 0.7021  | 0.5670 | 101011110010 | 01110000011110010000001001000000100101001010010 |
| 100   | 0.7021  | 0.5957 | 101010110100 | 01010000010110100000001010000000101000000010100 |
| 200   | 0.7447  | 0.6915 | 101100100100 | 10010000100100100000000010000100000000000000100 |
| 384   | 1.0000  | 0.7281 | 101100110001 | 10010000100110001000001000110001000000000010001 |

The best initial key is: 101100110001, which is the actual initial key.

Figure (4.5) shows the achievement of PSOCS to find the actual key for NTSCC.

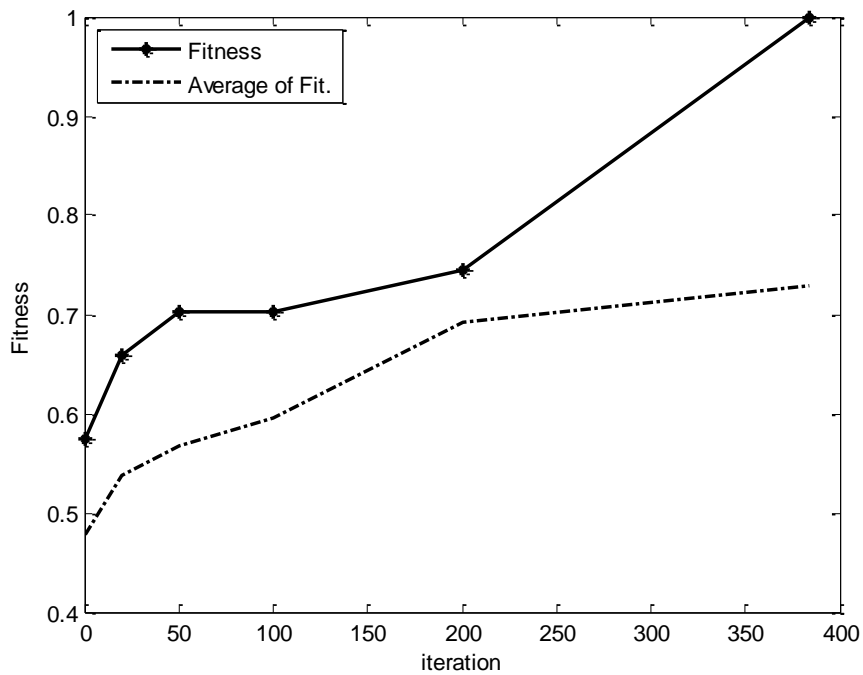


Figure (4.5) PSOCS's results to find the actual key for NTSCC.

The attack on the class of stream ciphers must be run a number of times with a variety of parameter values. Figures (4.3), (4.4), and (4.5) present a plot comparing the values of fitness function and their averages with the number of iterations in the attack for the SLSCC, LSCC and NTSCC.



**Chapter  
Five**

**Conclusions  
and  
Future Works**

## **Chapter five**

### **Conclusions and Future Work**

#### **5.1 Conclusion**

From this work, several conclusions can be drawn as follows:

1. In this thesis, we attack three study cases in stream cipher cryptosystem, these study cases are: Single LFSR, Linear System and Threshold Nonlinear system.
2. The results illustrated in this thesis show that the PSO has a powerful algorithm that succeeds in cryptanalysis of stream cipher cryptosystems using a small number of fitness evaluations to reach the best fitness. It also requires only a small number of particles. The actual computation times required are also small compared to other algorithms.
3. As the stream cipher be complicated as the needed number of generations and the consuming time are increased to find the actual initial key.
4. This thesis can be considered as a warning for a stream cipher designer to avoid the weak points, which may be found in the stream cipher, which are exploit by the cryptanalysts.

#### **5.2 Suggestions for Future Work**

As future work, several suggestions can be put forward to improve PSO.

1. PSOCS can be used to attack more complicated cryptosystems in the field of stream cipher (Geffe, or Bruer with more than 3 registers).

2. Further research in the cryptanalysis area, PSOCS can be treated to be suitable to attack other complicated cryptosystems such as knapsack cipher and block cipher.
3. To improve the performance of PSO, we suggest making a hybrid between this algorithm and other soft computing algorithms (e.g. simulated annealing or descent algorithm).
4. Some improvements can be done in the main evolving equations of velocity or position of particles of PSO to improve the attack achievement.

# References

1. Andrews, G. G. 1994. **“Number Theory”**, Dover Publications, October.
2. Apostol, T. M. **“Introduction to Analytic Number Theory”**, Corrected 5<sup>th</sup> Printing, Undergraduate Texts in Mathematics, Springer-Verlag.
3. August, D. **1985. “Information Theoretic Approach to Secure LFSR Ciphers”**, Cryptologia, pp. 351-359, October.
4. Ayad G. Naser and Fatin A. Hameed, **2017. “Constructing of Analysis Mathematical Model for Stream Cipher Cryptosystems”**, Iraqi Journal of Science.
5. Beker, H., Piper, F. **1982. “Cipher Systems, the Protection of Communications”**, Northwood Publications, U.K, 2.
6. Bishop C. **1995. ” Neural Networks for Pattern Recognition”** Clarendon Press. Oxford, first edition.
7. Bonabeau E., Dorigo M., Theraulaz G. **1999. “Swarm Intelligence: From Natural to Artificial Systems”**, New York, NY: Oxford University Press, 1999.
8. Bronshtein, I. N. and Semendyayev, K. A. **2004. “Handbook of Mathematics”**, 4th Ed. New York: Springer-Verlag, ISBN 3-540-43491-7.
9. Brüer, J. O. **1983. “On Nonlinear Combination of Linear Shift Register Sequences”**, Internal Report, Cryptologia Magazine, Vol. XVII, No. 2, pp. 187-201.
10. Davalo, E., Nairn, P. **1991. “Neural Networks”**, Machllan, 1991.
11. Davies, D. W., Price W. L. **1989. “Security for Computer Networks, An Introduction To Data Security In Teleprocessing And Electronic Funds Transfer”**, John Wiley and Sons Ltd., 1989.
12. Eberhart R. C. and Hu X. **1999. “Human Tremor Analysis Using Particle Swarm Optimization”** Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999), Washington D.C. pp. 1927-1930, 1999.
13. Epp, S. S. **1995. “Discrete Mathematics with Applications”** 2<sup>nd</sup> Edition, PWS Publishing Company, Boston.
14. Fraleigh, J. B. **1994. “A First Course in Abstract Algebra”** 5<sup>th</sup> Edition, Addison-Wesley.
15. Gilbert, W. J. **2002. “Modern Algebra with Applications”**, Wiley-Interscience, March.
16. Goldberg, D. E. **1989. “Genetic Algorithm in Search, Optimization, and Machine Learning”**, Boston: Addison-Wesley, 1989.

17. Golomb, S.W. 1982. **“Shift Register Sequences”** San Francisco: Holden Day 1967,(Reprinted by Aegean Park Press in 1982).
18. Gries, D. and Schneider, F. B., **“A Logical Approach to Discrete Mathematics”**, Texts and Monographs in Computer Science, Springer-Verlag, 1993.
19. Henkel, W., **“Another Description of the Berlekamp-Massey Algorithm”**, IEEE Proceedings, vol. 136, pt. 1, no. 3, pp. 197-200, June, 1989.
20. Hu X., **“Particle Swarm Optimization”**, Tutorials 5-7-2002, <http://www.swarmintelligence.org/>.
21. Ir. Bart Preneel, **“Cryptanalysis and Design of Stream Ciphers”**, July 2008
22. Johnson, D. W. and Johnson, F. P., **“Joining Together: Group Theory and Group Skills”**, Allyn & Bacon, July 2002.
23. karaboga D.” **An idea based on honey bee swarm for numerical optimization ”** technical report-TR06,October,2005.
24. Kennedy J. and Eberhart R. C., **“Particle Swarm Optimization”**, Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ. pp. 1942-1948, 1995.
25. Konheim, A. G., **“Cryptography: A Primer”**, John Wiley and Sons, Inc., 1981.
26. Massey, J.L., **“Shift Register Synthesis and BCH Decoding”**, IEEE Transaction on Information Theory, Vol. IT-15, No.1, 1969.
27. Matt, J. B., **“Stream Ciphers Technical Report TR-701”**, RSA Laboratories, 1995.
28. Matthews, R. A., **“The Use of Genetic Algorithms in Cryptanalysis”**, Cryptologia, vol. XVII, no 2, pp. 187-201, April, 1993.
29. McEliece, R. J., **“Finite Fields for Computer Scientists and Engineers”**, Kluwer Academic Publishers, 1987.
30. Menezes, A. P. van Oorschot, P. and Vanstone, S., **“Handbook of Applied Cryptography”**, CRC Press, 1996.
31. Meyer, C, Tuchman W., **“Pseudo-Random Codes Can be Cracked”**, Electronic Design, Vol. 23, pp. 74-76, 1972.
32. Meyer, C. H., Matyas S. M, **“Cryptography: A New Dimension in Computer Data Security”**, John Wiley and Sons, New York, 1982.
33. Mister S., **“Cryptanalysis of RC4-like Stream Ciphers”**, M.Sc. thesis, Queen’s University, May 1998.



34. Mitchell M., “**An Introduction of Genetic Algorithms**”, Abroad Book 1998.
35. Motwani, R. and Raghavan, P., “**Randomized Algorithms**”, Cambridge University Press, 1995.
36. Nicholas, J., Hunter, J., Hargreaves, J., 1997, “**Functions and Their Graphs**”, Mathematics Learning Centre, University of Sydney.
37. Panigrahi B., Shi Y., and Lim M., ” **Handbook of Swarm Intelligence Concepts, Principles and Applications**”, ISBN 978-3-642-17389-9, Adaptation, Learning, and Optimization, Volume 8, Springer, 2011.
38. Parsopoulos K. E. and Vrahatis M. N., “**Recent Approaches to Global Optimization Problems Through Particle Swarm Optimization**”, Kluwer Academic Publishers. Printed in the Netherlands, Natural Computing 1, pp 235–306, 2002.
39. Pinch, R. G. E., “ **Mathematics for Cryptography** ”, Queen’s College, University of Cambridge, 1997.
40. Philipp Jovanovic, **Analysis and Design of Symmetric Cryptographic Algorithms**, PHD thesis, May 2015.
- 41.[41]. Pomeroy P., “**An Introduction to Particle Swarm Optimization**”, Article, March, [www.adaptiveview.com](http://www.adaptiveview.com) , page 1-7, 2003.
- 42.[42]. Rahmat-Samii Y. R., Gies D. and Robinson J., “**Particle Swarm Optimization (PSO): A Novel Paradigm for Antenna Designs**”, 2003. <http://www.ee.ucla.edu/~dgies/files/gies-ursi-2003.pdf>.
- 43.[43]. Rechberger, C., “**Side Channel Analysis of Stream Ciphers**”, Master’s Thesis, Institute for Applied Information Processing and Communications (IAIK) Graz University of Technology, Graz, Austria, 2004.
- 44.[44]. Ribenboim, P., “**The New Book of Prime Number Records**”, Springer-Verlag , 1996.
- 45.[45]. Rivest, R. L., “**Hand Book of Applied Cryptography**”, John Wiley & Sons, 1997.
- 46.[46]. Staffelbach, O. J. and Meier, W., “**Fast Correlation Attack on Stream Cipher**”, Springer-Verlag, 1988.
- 47.[47]. Schneier, B., “**Applied Cryptography**”, John Wiley & Sons, 1996.
- 48.[48]. Settles M. and Rylander B., “**Neural Network Learning using Particle Swarm Optimizers**”, Advances in Information Science and soft computing, pp. 224-226, 2002.

49. Shi Y., “**Particle Swarm Optimization**”, Electronic Data Systems, Inc. Kokomo, IN 46902, USA Feature Article, IEEE Neural Networks Society, February 2004.
50. Siegenthaler, T., “**Decrypting A Class Of Stream Cipher Using Ciphertext Only**”, IEEE Transaction on Computers, vol. c-34, no.1, pp. 81-85, January, 1985.
51. Spillman R., Janssen M., Nelson B., Kepner M, “**Use Of A Genetic Algorithm In The Cryptanalysis Of Simple Substitution Ciphers**”, Cryptologia, vol.16, no. 1, pp. 31-44, 1993.
52. Stinson, D. R., “**Cryptography: Theory and Practice**” CRC Press, 1995.
53. Thomas Johansson, “**Lecture Notes in Cryptography**”, 2006.
54. Tomassini, M., “**Parallel and Distributed Evolutionary Algorithms**”, Evolutionary Algorithms in Engineering and computer Science (pp.113-133) Chichester: J. Wiley and sons, 1998.
55. Venter G., Sobieszczanski J., “**Multidisciplinary Optimization of a Transport Aircraft Wing using Particle Swarm Optimization**”, 9<sup>th</sup> AIAA / ISSMO Symposium on Multidisciplinary Analysis and optimization, Atlanta, GA, 2002.
56. Whitesitt, J. E, “**Boolean Algebra and its Application**”, Dover Publications, April 1995.
57. Yan, S. Y., “**Number Theory for Computing**”, Springer-Verlag Berlin Heidelberg, New York, 2000.
58. Yang X.S.,” **Nature-Inspired Metaheuristic Algorithms**”, second edition, Luniver Press, ISBN-13: 978-1-905986-28-6, 2010.
59. Zhou Y., Zeng G., and Yu F, “**Particle Swarm Optimization-Based Approach for Optical Finite Impulse Response Filter Design**”, Optical Society of America, 2003.

## مستخلص

ذكاء السرب طريق مبتكر لحل المشكلات الصعبة الحقيقية وهي مستوحاة من السلوك والتنظيم الاجتماعي لمجتمع الحيوانات والحشرات مثل الطيور والاسماك والنمل والنحل .. الخ. ان امثلية السرب الجزئي (PSO) هي خوارزمية مثالية تحاكي سلوك سرب من الطيور او الاسماك. ان سلوك السرب يتمثل بعدد من الاجسام (طيور او اسماك) في فضاء متعدد الابعاد وله خاصيتين: الموقع والسرعة.

اعتبر التشفير الانسيابي من اهم فئات خوارزميات التشفير. لقد استخدمت متتابعات المسجل الزاحف بشكل واسع في كل من علم التشفير ونظرية الترميز. وهناك اساس رياضي غزير عن التشفير الانسيابي المعتمد على المسجل الزاحف، وكان هو اساس العمل في التشفير للاغراض العسكرية كونه الاساس في الاتصالات الالكترونية.

هدف البحث هو تنفيذ نظام تحليل شفري باستخدام PSO سمي نظام PSO لتحليل نظم التشفير (PSOCS) بالاعتماد على الكلمة المحتملة في اسلوب المهاجمة بالنص الواضح وباستخدام ثلاث حالات دراسية هي: مسجل زاحف منفرد باعتباره الوحدة الاساسية لنظم التشفير الانسيابي، المولد الخطي بالاضافة الى مولد يعتمد دالة العتبة (Threshold) باعتباره مولد غير خطي، حيث يقوم نظام التحليل بايجاد الحل الحقيقي لنظام المعادلات الخطية (SLE) للمسجلات الزاحفة المشاركة في المنظومة ولاي عدد من المتغيرات.

ان تنفيذ PSOCS يتمثل بمرحلتين، الاولى تتضمن بناء نظام المعادلات الخطية للمسجلات الزاحفة الخاصة بالحالة الدراسية، والثانية تمثل مهاجمة المتغيرات الخاصة بالنظام SLE والتي تمثل القيم الابتدائية للمسجلات الزاحفة. هذا البحث يظهر مدى كفاءة وانجازية PSOCS في ايجاد القيم الابتدائية للمسجلات الزاحفة.

نتائج هذا البحث تم الحصول عليها من خلال برامج معدة باستخدام لغة دلفي من الجيل العاشر وبلاستفادة من وسائل البرمجة المرئية لهذه اللغة.



جمهورية العراق

وزارة التعليم العالي والبحث العلمي  
جامعة بغداد  
كلية العلوم

# استخدام السرب الذكي في تحليل الانظمة الغير خطية في نظم التشفير الانسيابي

رسالة

مقدمة إلى كلية العلوم – جامعة بغداد  
وهي جزء من متطلبات نيل درجة الماجستير في علوم الرياضيات التطبيقية

من قبل

فاتن عبد الرحمن حميد

اشراف

أ.م.د اياد غازي ناصر الشمري