Cryptanalysis of the RC4 Stream Cipher using Evolutionary Computation Methods

by

Benjamin Nicholas Ferriman

A Thesis
presented to
The University of Guelph

In partial fulfilment of requirements for the degree of Master of Science in Computer Science

Guelph, Ontario, Canada ©Benjamin Nicholas Ferriman, December, 2013
ABSTRACT

CRYPTANALYSIS OF THE RC4 STREAM CIPHER USING EVOLUTIONARY COMPUTATION METHODS

Benjamin Nicholas Ferriman
University of Guelph, 2013

Advisors:
Dr. Charlie Obimbo

In order to protect the actions and information on the Internet, secure protocols are implemented to facilitate secure communication between clients and services. The most common protocol used is the Secure Socket Layer protocol that wraps around HTTP to obfuscate network data from prying eyes. RC4 is the most widely used encryption algorithm over this protocol and has held strong to many attempts of cryptanalysis over its twenty-six year existence. The following Thesis will outline the RC4 algorithm and present a new approach for cryptanalysis of the cipher by attacking RC4’s state register which is required for all the algorithms communication. We intend to improve the exhaustive search process for a permutation using genetic and particle swarm optimization algorithms from the evolutionary algorithm family. It will be shown that thus far, the genetic algorithm has been the most successful of the two methods and on several occasions, has been able to recover 25% and on average, 10% of the keystream with a candidate solution. The attack can theoretically on average recover the state register in $2^{121.5}$ generations.
Acknowledgements

First and foremost I would like to thank my advisor Dr. Charlie Obimbo who took me on and allowed me to develop and experiment with my own ideas in the field of security. I would also like to acknowledge my advisory committee consisting of Dr. Gary Grewal and Dr. Joe Sawada. I also would like to thank my parents Lies and Bill for teaching me to always have a backup plan and who have been supportive of me for all these years. I would like to thank my brother Sasha for showing me just what perseverance can achieve. Thank you to my Aunt Julia for all the support along the way and always being there for me. To my Oma and Opa, I thank you for always instilling the power of education in me and being my number one supporters. I would like to thank my friends David Jozefacki and David Doerksen for always being there for me and supportive when I needed some time away from my studies. Thank you to my friend Lauren for helping motivate me as my work approached the finish line. Finally I would like to thank the entire faculty, administrative, and support staff at the School of Computer Science for all the assistance they have provided and always assisting in the behind the scenes aspects of this thesis. Thank you everyone!
# Contents

1 Introduction  
   1.1 RC4 State Register .................................................. 4  
   1.2 Thesis Statement ....................................................... 4  
   1.3 Contributions .......................................................... 5  
   1.4 Overview of Thesis ................................................... 5  

2 Background  
   2.1 The Study of Codes ..................................................... 7  
      2.1.1 Cryptography ....................................................... 8  
      2.1.2 Cryptanalysis ..................................................... 9  
   2.2 RC4 Stream Cipher ..................................................... 10  
      2.2.1 Key Scheduling Algorithm ....................................... 11  
      2.2.2 Pseudo-Random Generation Algorithm ......................... 11  
      2.2.3 Data Encryption and Decryption ............................... 12
2.3 Genetic Algorithms .................................................. 13
  2.3.1 Chromosome Representation ................................. 16
  2.3.2 Selection .......................................................... 16
  2.3.3 Crossover .......................................................... 17
  2.3.4 Mutation .......................................................... 18

2.4 Particle Swarm Optimization .................................. 19
  2.4.1 Parameters ......................................................... 19
  2.4.2 Particle Representation ....................................... 21

3 Literature Review ....................................................... 22
  3.1 Review of RC4 Literature ....................................... 22
    3.1.1 Successful Cryptanalysis of RC4 ......................... 23
    3.1.2 Other Methods for Cryptanalysis of RC4 ............... 25
  3.2 Review of Evolutionary Algorithms Literature ............ 28
    3.2.1 Cryptanalysis Using Genetic Algorithms ............... 28
    3.2.2 Cryptanalysis Using Particle Swarm Optimization .... 30

4 Methodology ............................................................... 32
  4.1 Problem .............................................................. 33
  4.2 Assumptions .......................................................... 34
4.3 Proposed Solution .............................................. 35
4.4 State Register Representation ............................... 36
4.5 Genetic Algorithm Representation .......................... 37
  4.5.1 Selection Methods ........................................ 37
  4.5.2 Fitness Evaluation ....................................... 39
  4.5.3 Crossover Methods ....................................... 40
  4.5.4 Mutation Methods ....................................... 44
  4.5.5 Parameter Configurations ............................... 45
4.6 Particle Swarm Optimization Representation ............. 46
  4.6.1 Solution Representation ................................. 47
  4.6.2 Fitness Evaluation ....................................... 47
  4.6.3 Parameter Configurations ............................... 47
4.7 Non-evolutionary Methods .................................... 48

5 Evaluation ......................................................... 49
  5.1 Genetic Algorithm Results .................................. 50
    5.1.1 Comparison of Crossover Operators .................... 50
    5.1.2 Comparison of Mutation Operators .................... 51
    5.1.3 Results ................................................ 53
List of Tables

5.1 Percentage increase using adaptive mutation at 1,000,000 generations . . . . 53

5.2 Best average fitness for word size $n$ using a genetic algorithm . . . . . . . . . 53

5.3 Best average fitness for word size $n$ using particle swarm optimization . . . 55

7.1 Percentage increase using adaptive mutation over non-adaptive mutation for
the genetic algorithm ($n = 6$) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63

7.2 Percentage increase using adaptive mutation over non-adaptive mutation for
the genetic algorithm ($n = 7$) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63

7.3 Percentage increase using adaptive mutation over non-adaptive mutation for
the genetic algorithm ($n = 8$) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64

7.4 Best average fitness for word size $n$ using simulated annealing . . . . . . . . 64
List of Figures

2.1 Single-point crossover operation .............................................. 17
2.2 Example of duplication introduced by non-ordered crossover method single-point crossover (duplications seen in red) .............................................. 18
3.1 Plaintext encrypted using a broadcast protocol ............................... 26
4.1 State Register bit/byte Representation ........................................ 36
4.2 Fitness proportional representation of different fitness values (f) ........ 38
4.3 Example of partially mapped crossover (PMX) ................................ 40
4.4 Example of the steps involved in edge recombination crossover (ER) ..... 43
4.5 Example of swap mutation (SwM) ................................................. 44
5.1 Partially mapped crossover vs. edge recombination for n = 6 using adaptive mutation ................................................................. 51
5.2 Swap mutation vs. inversion mutation for n = 6 using adaptive mutation 52
5.3 Genetic algorithm best average fitness for various word sizes \( n \) using adaptive mutation .................................................. 54

5.4 Particle swarm optimization best average fitness for various word sizes \( n \) ........................................ 56

5.5 Simulated annealing best average fitness for various word sizes \( n \) (cooling rate: \( 1.0 \times 10^{-8} \)) .................................................. 57

5.6 Comparison of algorithms for \( n = 8 \) .................................................. 59
Chapter 1

Introduction

Over the past twenty years the Internet has evolved astronomically as a tool for education, pleasure, and economics, to name a few applications. In today’s society there are very little tasks in one’s daily life which are not facilitated by the Internet in some way, shape, or form. As the services available over the Internet continue to expand, new and old problems of security arise and must be accounted for in order to properly facilitate these applications.

One of the largest sectors which continues to grow is on-line banking and other electronic financial transactions (conveniently distinguished as E-Commerce). These two applications face many of the same problems as traditional physical banking, but also a new set of challenges that have amounted due to the use of the Internet. The most obvious contemporary issue is that of communication. Traditionally a customer simply communicated with a bank teller where the environment could be controlled as well as the manor of communication (i.e. whether something could simply be conveyed through speech or read privately by the customer). With the advent of on-line banking there is an unknown communication between
the customer (client computer) and the teller (bank servers). The very fact that on-line banking improves the ease of use for a customer by virtually letting them do their banking anywhere with an Internet connection also hinders their ability to know specifically how their private communication with the bank system is being conducted.

Imagine a person at a sporting event buying food from a vendor walking up and down the aisles with an electronic payment system. If the patron is at the aisle seat they can be directly handed the machine and pay with their payment card of choice knowing that only themselves and the vendor are involved in the transaction. This is quite like the communication between a customer and a teller at a bank. Now picture a patron in the middle of a long row of seats who wants to make a purchase and the other spectators between that patron and the vendor as the various links (or nodes) that make up the Internet. The patron must first convey a message of what they desire to the vendor, which is passed down the row of spectators to the vendor. The patron then must pay with an electronic payment method and provide their payment information including their card and personal identification number (PIN) across the row of spectators. This scenario is quite like the Internet since the patron has no idea which, if any, of the other spectators are trustworthy with their information. Several outcomes could arise including (but not limited to) the wrong food item being stated to the vendor, the incorrect banking information being given voiding the transaction, or everything appearing to work in the patron’s favour with one of the spectators storing the banking information of the patron for another time.

To try and solve the problem of secure communication over the Internet, a secure lan-
guage to communicate as well as a set of rules on how to use this language are needed. The secret language needed is called encryption, which is a method of turning data from a decipherable state into an indecipherable state and the ability to reverse those actions with one or more pieces of information commonly called a key. The set of rules needed to moderate the secret language is called a protocol. The importance of the latter is to have a pre-existing understanding of how certain sensitive information should be shared and how to agree on different secret languages or encryption schemes.

A modern encryption cipher used to protect our data while it travels across the Internet is the RC4 stream cipher. The main protocols RC4 is implemented over are the Secure Socket Layer (SSL) and the Transport Layer Security (TLS). Many websites incorporate RC4 over one of the two protocols to protect services such as electronic banking and social media. Since a vast amount of services on the Internet rely on RC4, a successful practical attack on the cipher would cause a vast abundance of information to be exposed.

As of the summer of 2013, there has been an extensive surveillance campaign headed by the National Security Agency. The agency was able to circumvent many of the encryption algorithms used over SSL and attack the protocol itself [3]. Though this is a highly contemporary issue, the following Thesis will focus on one of the underlying encryption schemes of SSL and not on the deficiencies of the protocol itself.
1.1 RC4 State Register

When looking at RC4 one can make the comparison to a pseudo random number generator (PRNG). The values that are output from a PRNG can be compared to what is called a keystream in RC4. These streams of numbers are, by design, created to allow as little correlation as possible, but can still be replicated through the initialization of the sequence through a value called a seed. A seed is normally set through a value such as a time stamp to ensure that repetitive initializations will yield different results. In the case of RC4, both communicating parties want to initialize the keystream with the same seed so they can interpret the information they exchange correctly. This seed value is known as a shared key.

The state register is the second important part of a PRNG since it requires a state to be stored in some form or another to continue the stream of values each time the generator is called. In RC4 the state register is an array of permutations which are operated on to get the current pseudo random value and mutate the register further. Simply put, if we can reconstruct the initial state register of the cipher, it is as good as having the key that initialized it. This Thesis will examine methods of discovering the state register for the RC4 stream cipher.

1.2 Thesis Statement

The following Thesis intends to solve for as many unique values in the state register of the RC4 stream cipher as possible in order to re-create the keystream and therefore have
the ability to encrypt and decrypt further communications between multiple parties. We propose to solve this permutation problem by improving an exhaustive search with the implementation of a genetic algorithm and particle swarm optimization, both tailored to solve for the state register of RC4.

1.3 Contributions

This Thesis is presented to contribute to the ongoing cryptanalysis of RC4 and investigate a new avenue of attack against the cipher that thus far has not been examined. While arguments can still be made for the speed and lightweight characteristics of RC4 over certain power consumption sensitive devices such as mobile devices, the processing improvements of personal computers and servers should be exploited for better encryption algorithms. Modern web security requirements should not be constrained by limitations of processing power and load balancing that was far more of a concern when the Internet boom first occurred.

1.4 Overview of Thesis

This Thesis has been organized into the following chapters to provide a clear understanding of the proposed methodology and results of experiments created to verify the proposed solution. In Chapter 2, the reader will gain the necessary background knowledge to understand the remainder of the thesis. A literature review is presented in Chapter 3. In Chapter 4, the outline of our proposed attack is provided including the methodology and implementation.
Then in Chapter 5, we analyze the resulting algorithms and demonstrate which algorithm proved to be the most successful. The Thesis concludes in Chapter 6, with a summary of the work completed and direction for future research.
Chapter 2

Background

Several topics will be elaborated upon in order to familiarize the reader with specific terminology and concepts of this Thesis. Terminology and notation introduced in the following sections will remain constant throughout the rest of the composition.

In Section 2.1 the terms associated with cryptology will be introduced. Next, Section 2.2 will acquaint the reader with the RC4 stream cipher. In Sections 2.3 and 2.4, genetic algorithms and Particle Swarm Optimization will be discussed respectively with an overview of the specific algorithms used in this thesis.

2.1 The Study of Codes

The study of codes is referred to as cryptology which is the art and science of transforming legible data into indecipherable data and returning it into a comprehensible state whether it is completed through an intended function or other method. These two methodologies
are associated to the terms cryptography and cryptanalysis respectively.

2.1.1 Cryptography

Cryptography is the science of creating codes or encryption algorithms. In this process a form of mathematics is usually the foundation of the algorithm whether it uses simple operations such as the Caesar Cipher that uses simple modulo addition and subtraction on an alphabet or more advanced number theory in algorithms such as the RSA asymmetric key encryption where prime numbers and modulo exponentiation are used [32].

Cryptography has general terminology and notations assigned to it. The text or data being coded is known as the plaintext $P$ and the information that has been scrambled is known as the ciphertext $C$. The process of coding the plaintext to the ciphertext is done with a key $K$ and is called encryption which can be written as $\mathcal{E}(P, K) = C$. The process of returning the ciphertext back to plaintext is known as decryption and is similarly expressed as $\mathcal{D}(C, K) = P$. These operations can be performed with the same shared key which is called symmetrical cryptography or a set of keys referred to as asymmetrical cryptography. A cryptographer’s goal is to try and utilize Mathematics that is notably difficult or time-consuming for a computer to process to improve the cracking resilience of modern computing. Cracking resilience relates to the amount of time it would take to try every key for a cryptographic function which is called brute-force cracking.
2.1.2 Cryptanalysis

The science of Cryptanalysis is in reference to the analysis of cryptographic systems, specifically how to improve (or shorten) the time it would take to brute-force a key. Improvements can be short falls of an algorithm itself such as a short period of a pseudo-random generator or even statistical data leaked by the encrypted data which can be used to reconstruct the key. Cryptanalysis attacks are generally clustered into two distinct labels: practical and theoretical. Despite these classifications with continuous improvements in computing power it is not unheard of for a theoretical attack to become practical over time. An example of this is is found with the DES encryption algorithm where in 1977 a brute-force attack was theorized to cost $20 million (USD) to get the key in one day. The attack moved into the practical space in 1998 when a single board was built to crack a key at a cost of $250 000 (USD).

Attacks can use a variety of tactics to accomplish their goals. One method used is known as differential cryptanalysis and is performed by inserting known data into an algorithm and measuring the difference in the result of the function. Another tactic is to use statistical analysis to improve solution discovery using probabilistic methods.

Much of the notation associated with cryptanalysis is the same as that of cryptography. Any new notations introduced will be done at first use.
2.2 RC4 Stream Cipher

RC4 was invented by Ron Rivest in 1987, while working at RSA Security and was designed as a non-linear feedback shift register (non-LFSR). It is a stream cipher meaning: given identical initialization keys, the algorithm will produce the same keystream for all parties involved in communication; encryption/decryption takes place when each word of data is XOR’ed with the keystream word. RC4 allows the initialization key $K$ to be of length 40 to 2048 bits. The cipher produces a keystream $z$ of word size $n$. Generally RC4 is implemented with $n = 8$ bit words. What sets RC4 apart from other ciphers is its speed (execution time) as well as its simplicity to implement. It can be implemented efficiently over both software and hardware architectures making it highly portable for a vast array of different applications [19].

Since the RC4 cipher is a stream cipher that encrypts and decrypts one word at a time, it needs several variables to hold states as the algorithm iterates through each word. RC4 uses three variables to store its state. The main variable is the state register $S$ which is an array of size $2^n$, where $n$ is the size of a word. The state register is a permutation register that holds inclusively the values $\{0, 1, \ldots, 2^n - 1\}$. The notation for a state register is $S_i$, where $i$ is also a value from $\{0, 1, \ldots, 2^n - 1\}$. There are also two $n$ word-sized index pointers $i$ and $j$ that store temporary index values to the state register.
2.2.1 Key Scheduling Algorithm

The *Key Scheduling Algorithm* or KSA is used to initialize the state register’s values in a pseudo-random order that is based on the key $K$. When communicating with other parties, each party initializes their state register to the exact same order using the key they have shared (hopefully through secure communication mechanisms). The scheduling algorithm of the state register $S$ can be seen below in Algorithm 1. Note that in this case, it is assuming the key is of size $2^n$. For smaller keys the algorithm would simply be modified to $j \leftarrow j + S_i + (K_i \mod \text{key length}) \mod 2^n$.

**Algorithm 1** Key Scheduling Algorithm (KSA) for RC4  
Input: Shared Key $K$  
Output: State Register $S$  
for $i = 0 \rightarrow 2^n - 1$ do  
    $S_i \leftarrow i$  
    $j \leftarrow 0$  
for $i = 0 \rightarrow 2^n - 1$ do  
    $j \leftarrow j + S_i + K_i \mod 2^n$  
    Swap($S_i, S_j$)  
return($S$)

2.2.2 Pseudo-Random Generation Algorithm

The *Pseudo-Random Generation Algorithm* or PRGA is the algorithm that actually facilitates the process of keystream generation. Algorithm 2 illustrates the theoretical communication functionality of a stream cipher through an infinite loop as seen in the `WHILE`
1 statement. In practicality, communication would eventually cease and the loop would be broken. As communication progresses several actions occur to generate each word-sized keystream value which is denoted as \( z \). As stated in at the beginning Section 2.2, since the RC4 cipher relies on the previous state of its variables, the variables \( i, j \), and \( S \) are saved for the next iteration and not re-initialized.

### Algorithm 2 Pseudo-random Generation Algorithm (PRGA) for RC4

<table>
<thead>
<tr>
<th>Input: State Register ( S )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: Keystream bytes ( z )</td>
</tr>
<tr>
<td>( i \leftarrow 0 )</td>
</tr>
<tr>
<td>( j \leftarrow 0 )</td>
</tr>
<tr>
<td>while 1 do</td>
</tr>
<tr>
<td>( i \leftarrow i + 1 \mod 2^n )</td>
</tr>
<tr>
<td>( j \leftarrow j + S_i \mod 2^n )</td>
</tr>
<tr>
<td>Swap(( S_i, S_j ))</td>
</tr>
<tr>
<td>( z \leftarrow S_{S_i+S_j} \mod 2^n )</td>
</tr>
<tr>
<td>output(( z ))</td>
</tr>
</tbody>
</table>

#### 2.2.3 Data Encryption and Decryption

Encryption and decryption in conjunction with RC4 are done using the PRGA and the keystream word \( z \) and each word of data to be encrypted or decrypted. The actual process of encryption and decryption is through XOR’ing \( z \) with either \( P \) or \( C \) to get the other respectively. Encryption can be written as:

\[
C = \mathcal{E}(P, z) = P \oplus z.
\]
Decryption can therefore be written as:

\[ \mathcal{P} = \mathcal{D}(\mathcal{C}, z) = \mathcal{C} \oplus z. \]

For example, if \( n = 4 \) and Alice would like to send Bob the word of data \( 0110 \). Assuming Alice and Bob have both already agreed on a shared key and performed the KSA algorithm, Alice runs through an iteration of the PRGA and receives the keystream word \( 1100 \). Alice then creates the ciphertext word \( 1010 = 0110 \oplus 1100 \), and sends it to Bob. The ciphertext word is received by Bob and since he initialized his state register with the same key as Alice, he generates \( 1100 \) as his first word as well. He performs \( 1010 \oplus 1100 = 0110 \) to generate the plaintext again. For larger messages each word is simply encrypted, buffered, and sent as its original size. The same process occurs for decryption. It is interesting to note that since encryption and decryption rely on synchronous indexing, two messages sent simultaneously (one from each party) are unreadable to each user unless they have stored their previous keystream values. In addition any attempt to further communicate with the same state register can lead to the inability to correspond between parties.

### 2.3 Genetic Algorithms

Genetic Algorithms (GAs) are a type of optimization algorithm that utilize methods found in biology. They are members of the evolutionary family of algorithms due to their implementation of various evolutionary operators such as reproduction and mutation. Modern genetic algorithms were popularized by John Holland in 1975 [15]. This particular algorithm is used in optimization problems in an attempt to find the best solution in a group
of candidate solutions or chromosomes over time ([13],[28],[24],[5]). A solutions strength or fitness is determined by a fitness function \( f(x) \) which quantifies the quality of the solution with a desired requirement [30]. Since a group of solutions or population are just a small sample of all possible solutions, there is no guarantee that the best fit solution will be the global optimum for the problem. When reproduction or crossover is conducted the stronger candidate solutions have a better probability of entering the next generation of the algorithms life cycle. This is known as selection. Each generation is also referred to as an iteration. The basic algorithm of a GA can be seen in Algorithm 3. The algorithm depicts two processes: reproduction and evaluation. The steps of the reproduction process are as follows:

1. An initial population is randomly generated of size \( N \)

2. For \( N/2 \) iterations the reproduction process is conducted; The method uses \( N/2 \) since two children are introduced every iteration

3. Two parent chromosomes (\( p_1 \) and \( p_2 \)) are selected by a selection method

4. Crossover is performed on the two parents creating two children chromosomes (\( c_1 \) and \( c_2 \))

5. Given a rate of mutation, each child may or may not be mutated by the mutation operator

6. The children are then inserted into the new population
7. Finally, the population is replaced by the new population

The second process in the GA is the evaluation process. As the name suggests, the candidate solutions are evaluated to determine which members improved in fitness from the previous generation. Evaluation is conducted by:

1. Every member of the population \((p_i)\) is evaluated by a fitness function unique to the problem the GA is solving

2. Depending on the selection process the population might be sorted by fitness or percentage fitness

---

**Algorithm 3 General Genetic Algorithm**

Input: Generate initial population \(p\) of size \(N\) randomly

while Max iteration not met or fitness not satisfactory do

    for \(i = 1 \rightarrow N/2\) do

        Select \(p_1\) and \(p_2\) from population

        \((c_1,c_2) \leftarrow \text{Crossover}(p_1, p_2)\)

        Mutate\((c_1)\)

        Mutate\((c_2)\)

        Insert \(c_1\) and \(c_2\) into newpopulation

    Replace population with newpopulation

    for \(i = 1 \rightarrow N\) do

        fitness\((p_i)\)
2.3.1 Chromosome Representation

Potential solutions can be represented in several ways but the most common representation is through binary strings where each bit represents a characteristic flag. The other common representation is through a numerical value as the bit string, whether it is an integer or real number. For the $S$ register of RC4 each candidate solution will be represented as $2^n$ sets of $n$-bit binary strings. For example, a candidate solution is represented as $C_i = (w_1, w_2, \ldots, w_{2^n})$ where $w_i = \{0, 1, \ldots, 2^n - 1\}$ and each value of $w_i$ is unique due to its permutation property. All operations documented are performed on a whole $n$-bit word $w_i$ or multiple words and never on a subset of $w_i$ to maintain the chromosomes permutation property.

2.3.2 Selection

Selection is the process of probabilistically choosing the best candidate solutions for the next generation. This is done by evaluating the fitness of each chromosome in order to assign a better probability to the more fit solutions to enter the next generation. This Thesis will use two popular methods as a way to contrast their influence on the best particular approach for a permutation problem. The methods are fitness proportional representation and tournament selection which will be elaborated in Chapter 4.
2.3.3 Crossover

Crossover is the reproduction operator that takes two chromosomes and introduces parts of each of their solutions to the other. When using this operator the two original chromosomes are called the *parents* whereas the resulting chromosome(s) are referred to as the *child* or *children*. An example of crossover can be seen in Figure 2.1. The rate at which crossover occurs is regulated by a probability for crossover to occur. Crossover methods are dependent on the nature of the problem.

![Figure 2.1: Single-point crossover operation](image)

For a permutation problem like the RC4 $S$ register, methods of crossover (or ordered crossover) which do not destroy the permutation property must be implemented or duplicate values could be introduced to the children during crossover invalidating the solution altogether. An example of this problem is exhibited in Figure 2.2. The red values in the two children chromosomes represent the repetition in the solution that is introduced when
non-ordered crossover methods are used for permutation problems.

![Example of duplication introduced by non-ordered crossover method single-point crossover (duplications seen in red)](image)

**2.3.4 Mutation**

The process of mutation is to introduce random changes to a solution. This process is important to ensure the diversity of a population remains intact and reproduction does not cause all the chromosomes to diverge on a similar solution to quickly. Mutation also plays another important role in the reproduction process which is to help the solution not fall into a local optimum as much as possible (though this cannot be guaranteed). Similar to the crossover operation, a mutation rate dictates whether mutation shall be introduced or not. Typically the act of mutation is conducted by flipping random bits of a chromosome. Like the considerations for the crossover method, solutions that are encodes as permutations must also have mutation functions that take this property into consideration.
2.4 Particle Swarm Optimization

Particle swarm optimization (PSO) is another type of optimization algorithm with some similarities to GAs. Unlike GAs though, PSOs call candidate solutions particles and assign them velocities. PSOs were first published about in 1995 by James Kennedy and Russell Eberhart [17]. Instead of using biological methods to produce new candidate solutions, PSOs manipulate the velocity of a particle as it travels through a solution space. Though this method has traditionally been used for modelling behaviours of living creatures such as bird flocks, bee swarms, and schools of fish [17], it has also more recently been used for cryptanalysis (see Section 3.2.2).

2.4.1 Parameters

There are various differences between the variables that control PSOs and other evolutionary algorithms. The variables are as follows:

- $S$ - the size of a swarm of particles
- $p_i$ - a particle (or candidate solution) in the swarm
- $g$ - the best solution found globally
- $b_l$ - the lower bounds of a search space
- $b_u$ - the upper bounds of a search space
- $x_i$ - the current position of a particle
• $v_i$ - the current velocity of a particle

• $d$ - the dimension of a current calculation

• $r_p$ - a uniform random value for a particle between $[0,1]$

• $r_g$ - a uniform random value for the global solution between $[0,1]$

There are also some new constants introduced that differ from other algorithms. These constants are:

• $w$ - an inertia weight value in the velocity function

• $\phi_p$ - a local particle weight

• $\phi_g$ - a global particle weight

Like the GA, quality of a candidate solution or particle is evaluated using a fitness function that quantifies a proposed solution with respect to a desired result. All of the previous variables can be seen implemented in Algorithm 4.
Algorithm 4 Particle Swarm Optimization Algorithm

Input: Create a swarm $p$ of size $S$

Output: $g$ will hold the best position found globally

for $i = 1 \rightarrow S$ do
    $x_i \leftarrow \text{UniformRandom}(b_l, b_u)$
    $p_i \leftarrow x_i$
    if fitness($p_i$) < fitness($g$) then
        $g \leftarrow p_i$
    $v_i \leftarrow \text{UniformRandom}(-|b_u - b_l|, |b_u - b_l|)$
while max iterations not reached or acceptable solution not found do
    for $i = 1 \rightarrow S$ do
        for $d = 1 \rightarrow n$ do
            $(r_p, r_g) \leftarrow \text{UniformRandom}(0, 1)$
            $v_{i,d} \leftarrow w v_{i,d} + \phi_p r_p (p_{i,d} - x_{i,d}) + \phi_g r_g (g_d - x_{i,d})$
            $x_i \leftarrow x_i + v_i$
        if fitness($x_i$) < fitness($p_i$) then
            $p_i \leftarrow x_i$
        if fitness($p_i$) < fitness($g$) then
            $g \leftarrow p_i$

2.4.2 Particle Representation

Each particle in the swarm is represented through a position and velocity. These values are real numbers and as such must be represented as an integer permutation in order to let the fitness function $f(x)$ evaluate the current position. To perform such a transformation, each dimension of the position is translated into a corresponding unique value of the permutation.
Chapter 3

Literature Review

The literature review is organized into two main sections. Section 3.1 will explore the previous literature pertaining to the RC4 encryption algorithm and cryptanalysis that has been conducted on it. Next, Section 3.2 will delve into a review of certain evolutionary algorithms that compliment permutation optimization problems well. Additional methods of cryptanalysis using evolutionary algorithms will also be examined.

3.1 Review of RC4 Literature

This review will focus on several cryptanalysis methods for RC4. First, RC4 will be investigated for how it was found unfit to provide adequate protection to a popular wireless protocol. Secondly, various other cryptanalysis methods such as differential and linear cryptanalysis will be probed.
3.1.1 Successful Cryptanalysis of RC4

Despite RC4 having been around for 26 years, there have been very few successful attacks on it. In the mid 2000’s the Internet became more accessible largely due to the availability of wireless access points. One of the first major protocols adopted was the *Wired Equivalent Privacy (WEP)* wireless protocol. WEP utilized RC4 in order to obfuscate the Internet traffic of its users as well as maintaining access to into the wireless network. Fluhrer, Mantin, and Shamir found it was found that when a secret key was concatenated with a globally viewable *initialization vector (IV)*, as long as the first byte of the transmission was accessible, one could decipher the secret key rather effortlessly [10]. Since routers using the WEP protocol rarely change their keys’ and there is an abundance of synchronizing and query traffic utilized in maintaining a wireless access point, the attack proved theoretically highly successful on the protocol. In 2005, Erik Tews et al. utilized the proof-of-concept attack proposed in [10] and were able to implement it in a practical software tool called *aircrack-ptw* [37]. The tool was able to crack WEP protected networks in under one minute by collecting packets holding the IV and deciphering the secret key thus gaining access to the network. Mantin and Klein independently improved the WEP attack on RC4 to consider throwing away the first 256 bytes which was becoming standard practise in the algorithms use ([25], [20]). As a result the attack only needed $2^{17}$ keystreams generated from different IVs [25]. Work by [40] improved the attack to require only $2^{15}$ keystreams. The attack was further improved to only requiring $2^{14}$ faulted keystreams if the attacker can inject values into the cipher [25].
There is another method that would be successful at theoretically cracking RC4. Carter, Dawson, and Wong performed an algebraic cryptanalysis attack on RC4 and were able to algebraically represent the RC4 algorithm as a set of equations. In [9] equation generation methods as well as a small proof-of-concept exercise was proposed. Since the stream cipher does hold a state from one keystream word generation to the next, let \( t \) be the clock value where \( t \) is initialized to \( t_0 \). Also addition modulo \( 2^n \) is re-represented as \( +_{2^n} \) by the authors. Finally, another matrix \( M \) is introduced and is a square permutation matrix of the dimension \( 2^n \). The equations algebraically expressed are generalized as follows:

- \( i^t = i^{t-1} +_{2^n} 1 \) (pointer increment),
- \( j^t = j^{t-1} +_{2^n} S^{t-1}_i \) (pointer addition),
- \( S^t = MS^{t-1} \) (state permutation),
- \( z^t = S^t_{S^t_i +_{2^n} S^t_j} \) (keystream generation).

The exercise demonstrated that although the attack is correct (shown for 2-bit words), the attack’s success is hindered by time complexities due to methods of solving multi-variate polynomial equations. Since this attack is a case where it is hindered by the processing power of modern systems, it is still accepted as not a practical tool to attack the RC4 stream cipher.

The resiliency of the RC4 stream cipher really becomes apparent at this point. The most successful attack on RC4 requires a specific protocol for the algorithm to follow and the
best proposed theoretical attack cannot become practical until modern computer systems can facilitate it. At this point one must start to break down the cipher into its components and attack them individually. This will require an array of different cryptanalysis methods in order to attack the individual modules.

In 2013 researchers at Kobe University and Hiroshima University presented a new full plaintext attack on RC4 when used in conjunction with a broadcast protocol [16]. The researchers used previously discovered biases in the keystream. The attack used 1 billion ciphertexts with different keys to reproduce the same plaintext (See Figure 3.1). The attack worked since the keystream being XOR’ed with the ciphertext could be predicted with a very high probability. One downfall of this attack is the fact it relies on the same information to be encrypted by streams generated by many keys. Even protocols such as SSL that use different keys for each session and connection would not be able to generate the amount of keys needed for this attack to be practically feasible. Though this was successful attack, like WEP it was specific to the protocol RC4 was implemented in and did not actually attack any variables of the PRGA but rather the statistical anomalies of the beginning of the keystream which are typically thrown out at the encryption phase ([29],[26]).

3.1.2 Other Methods for Cryptanalysis of RC4

RC4 uses a non-linear feedback shift register (non-LFSR) to avoid various linear cryptanalysis methods that work against LFSRs. One approach that has shown results is the use of differential cryptanalysis. This method first was tried on RC4 in 1997 by Golić. In [14],[12],
and [11], researchers found that when a unique non repeated key was used (256 bytes), each key had related keys that resulted in very similar keystreams. The reason this approach was not widely implemented was due to the fact that most commercial implementations of RC4 utilize a 128-bit key. Weak keys used for the KSA of RC4 were first discovered as far back as 1995 by Andrew Roos [33].

From the former work, Eli Biham gathered that long 256 byte keys were preferable key sizes to attempt to perform differential cryptanalysis ([6],[7]). In fact using differential techniques, [6] was able to conduct a differential analysis of RC4 and resolve key states with the exception of a couple of bytes that had a constant difference (usually a difference of 2 or 3).

Since there are three variables that continually hold the state of the cipher many attacks
try and diagnose the pseudo-random properties associated with each variable. The index pointer $i$ does not need to be accounted for since we always know $i$ for every byte of the keystream as long as we are counting bytes from the beginning of transmission. This is due to the fact that $i$ simply increments its value by one every round of the PRGA with modulo arithmetic implying that every $2^n$ bytes the counter returns to 0.

The variable $j$ on the other hand is far more obfuscated by the algorithm. Basu et al. were able to predict the value of $j$ using the premise that $j$’s sequence was unique to the secret key used for encryption [4]. The researchers were able to take their predictions and create a theoretical attack that could victimize keys of size 2048-bits with a success probability of 0.1409. Their research hit a snag when their algorithm was found to require a very high time complexity. Like [9], the time complexity of this attack is what makes it a purely theoretical attack at this point in time.

The main variable to attack in the PRGA is the $S$ state register. One particular attack on $S$ is by Maximov et al. and is able to find the register in $2^{241}$ [27]. Their proposed method utilizes pattern guessing of the keystream in conjunction with an algebraic analysis of the PRGA in order to try and fill in some of the blanks in their mathematical model. For $n = 8$, this method is only shown through extrapolation as the algorithm was too computationally complex. One method of solving the $S$ register is the hill-climbing strategy. One reason for this strategy is that it has a far better complexity than an exhaustive search. There has also been a proposed a hill-climbing algorithm that yielded better results by an order of magnitude of 2 than the previous best state-solving algorithm [38].
As has become apparent, the RC4 stream cipher is very well designed to combat cryptanalysis. Conversely, its security appears to rely on the lack of computer power of modern computers to remain secure.

3.2 Review of Evolutionary Algorithms Literature

Since evolutionary algorithms help to reduce a search space through the use of fitness functions and methods relating to areas such as physics and biology, they have long been explored as a viable alternative to brute-force methods for finding a cryptographic system key. Many of these keys have vast search spaces that these cryptosystems rely on to make key derivation impractical with modern computing power and the computing power of future generations. Any methods that can help direct a search for a key over an extremely vast search space is one worth investigating. In Section 3.2.1 various cryptanalysis methods pertaining to GAs will be examined. This will be followed by a review of cryptanalysis methods that utilize PSOs in Section 3.2.2.

3.2.1 Cryptanalysis Using Genetic Algorithms

One of the first uses of soft computing to aid in the cryptanalysis of a cipher was in 1993 by Richard Spillman. He chose to attack the Knapsack cipher using a genetic algorithm. His research utilized a GA to decrypt cipher text letter by letter with an average decryption time of 84 seconds making the attack a practical use of soft computing methods [36]. Yaseen and Sahasrabuddhe used a genetic algorithm to attack the Chor-Rivest Knapsack public key
crypto system with very promising results in all test cases [42]. This was also the first public
attack on the crypto system. The algorithm is able to attack the cipher with a minimal
search space as well as a very small number of generations. The encryption algorithm
uses modular multiplication as well as logarithmic functions leading one to believe that
a genetic algorithm should be able to tackle the far simpler arithmetic involved in the
RC4 cipher. Surprisingly no work has been done on the cryptanalysis of RC4 using soft
computing methods. These methods have had a recent introduction as a form of not only
cryptanalysis, but cryptography as well.

Sharma et al. incorporated the use of a genetic algorithm to break a simplified implementa-
tion of the Data Encryption Standard (DES) called S-DES. The research was successful
in retrieving the key used for encryption as well as performing the task in less than 1/5 the
time it would take to brute force the key [35] (remember this is S-DES and is far easier to
brute force than regular DES would be). There are a few areas of the previous paper that
this thesis will deviate from. The first area is the method used as a crossover method; the
previous paper used a ring crossover method that does not respect the properties of ordered
chromosomes such as permutations. The second deviation is the use of letter frequency
analysis as the fitness function which is not in the proposed research since the author is
solving for the keystream and not a decrypted plaintext.

Alvarez et al. used a genetic algorithm to encode image information in a watermark
that could then be hidden or kept in plain sight within the image. The method utilized
by the authors was also highly resilient to many common attacks of detection [2]. What
makes this paper relevant to the research proposed is that the problem is encoded as a permutation problem. As a result, the authors examine several ordered crossover (OX) methods including partially mapped crossover (PMX) and edge recombination crossover (ER) as well as a method called cycle crossover (CX) that was not chosen for this task due to its larger computational needs over the other methods proposed. The authors also experimented with both the swap mutation method (SwM) and the inversion mutation method (InvM). The paper ultimately used a method created by the authors which claimed to be superior to the standard OX methods previously stated but since no algorithm for this method was stated, it cannot be replicated or utilized in this Thesis.

In Brown et al. the use of a genetic algorithm was used to attack a substitution permutation network. While the study was intended to find weak keys, it was later noted that a GA would be a good tool to attack the scheme [8]. The paper is important to this research as it used SwM to attack the network and it was also a method proposed to attacking other relevant crypto systems such as AES and 3DES.

3.2.2 Cryptanalysis Using Particle Swarm Optimization

The use of PSOs in cryptanalysis is relatively new to the field but it seems to be showing promise as a successful avenue of thought. In 2012 Valarmathi et al. attacked a simplified version of the Advanced Encryption Standard (S-AES). The research yielded a great amount of success in the recovery of the key used to encrypt data. One main observation of this paper was that utilizing this method, as the amount of ciphertext used increased, the
amount of generations needed to solve for the key greatly reduced by almost a factor of 3/4 the ciphertext size [39]. It is also noted that the cost function used in this experiment was letter frequency analysis which is irrelevant for this proposed thesis.

PSOs have also been used to attack DES ([22],[34],[41],[1]) and are definitely a soft computing method that should be exploited to their fullest in this analysis of soft computing methods which can aide in the cryptanalysis of RC4. Pandey et al. found the PSO attack useful for exploiting typical weak keys but had a far harder time breaking keys that have been deemed strong [31]. One point made by the researchers was that the use of PSO and methods similar to it require an exuberant amount of massaging certain variables to attain the desired results. This should be considered when choosing the values of certain variables for the various soft computing methods.

Finally, research performed by Laskari et al. found that PSOs can be used for various discrete optimization tasks in cryptography [23]. Out of the problem set selected by the authors, one problem in particular was interesting as it utilized PSO to help factor two prime numbers as a way of attacking RSA. Though the numbers were trivial in comparison to practical values used by the algorithm, it did demonstrate the ability to perform this task on a smaller scale. This work is relevant to the following research as it demonstrates the abilities of PSO over a vast search space such as integer factorization.
Chapter 4

Methodology

The methodology is divided into the following sections. A more in-depth description of the problem will be outlined in Section 4.1 followed by assumptions for the proposed problem in Section 4.2. Next the proposed solution will be stated in Section 4.3. Since problem representation is a very important issue for both algorithm optimization and storage space, Section 4.4 will discuss the specific representation of the state register for the two evolutionary methods proposed. In Section 4.5 we will explore the specific methods used to make the genetic algorithm as accurate as possible. Following the previous method, Section 4.6 will investigate the specific methods to make particle swarm optimization as devastating an attack as possible. We will conclude with a brief look at a non-evolutionary method in Section 4.7 to provide a comparison between evolutionary and non-evolutionary methods.
4.1 Problem

When attacking the RC4 stream cipher there are generally two main avenues of exploitation. The first method is to attack the KSA in an effort to recover the shared key of the crypto system. This method is a favourable choice when the shared key does not change often. Such a circumstance was the case with the implementation of RC4 in the WEP wireless protocol. While this method has been utilized in the past, it is not always beneficial especially if the specific attack is very computationally expensive when a protocol continually changes keys. Such key changing schemes are present in the SSL protocol where encryption is initialized on a session basis where each session is for a different connection and each connection has expiratory features such as the *time to live* (TTL) ability.

The other prominent vector of attack on the RC4 cipher is by attacking the PRGA. This approach can be conducted in several ways. The first way is to attack the pseudo-random properties of the algorithm. The second method is to recover the state variables that make RC4 a stream cipher. From the point of view of a brute force attack the key used in the KSA can be a maximum of 2048 bits. Using the max length of this key the brute force approach would yield $2^{2048} = 3.232\times10^{616}$ possible keys. On the other hand solving for the state register reduces this space by being a permutation. Though its storage space is also 2048 bits, they are arranged into a register of 256 unique values from 0 to 255. Therefore a brute force attack is trying to find all possible permutations of 256 values. This results in $256!$ possible permutations (see Equation 4.1) which works out to $256! = 8.578\times10^{506}$. While this value is still not a desirable number for a brute-force attack, it is still over one
hundred orders of magnitude better than brute-forcing the key.

\[ nPr = \frac{n!}{(n-r)!} = \frac{256!}{(256-256)!} = \frac{256!}{0!} = \frac{256!}{1} \]  

(4.1)

In this Thesis the desired vector of attack is through the PRGA utilizing evolutionary algorithms to improve the brute-force method of traversing through the vast search space of all the state register permutations. Solving for the state register is still as desirable as solving for the shared key because one can still read previously encrypted bytes with it as well as forge future encrypted messages to the other communicating party.

4.2 Assumptions

Since the proposed attack is not connected to any specific implemented protocols there are a few theoretic assumptions. The assumptions are as follows:

- Time for solution derivation may be infinite. This implies that part-way through solution finding a session key will never expire
- 256 keystream bytes are required to solve for a unique state register solution (as proposed in [9])
- The keystream is alway available to the attacker due to Kerckhoff’s principle [18]
- The cipher is always initialized for communication at the beginning (i.e. no bytes are thrown out by default)
4.3 Proposed Solution

This thesis proposes to solve the state register $S$ of RC4 using two evolutionary algorithms to provide a strong metric and evaluation process to evolve a population of solutions as close to the correct solution as possible. This method does introduce its own risks to the problem since a local optimal solution can solve for a portion of the global solution and be completely different from the desired global solution and thus be rather useless. Due to this pitfall, only solutions that re-render 85% of the state register or greater are looked at as solutions with potential fragments of the global solution (and even then they might be irrelevant to the global solution entirely).

The proposed methodology is to utilize a fitness function to measure the quality of solutions with the assistance of both evolutionary reproduction and properties of motion in physics to improve candidate state registers to be able to replicate as many bytes of the keystream sequence as possible. The overall goal is for the methods to be able to reproduce all 256 keystream bytes of the state register and be able to accurately reproduce all further bytes that the PRGA produces.

The first proposed method is a genetic algorithm which is modified to handle permutation problems. The GA was chosen for its ability to evolve a set of candidate solutions in an attempt to find a solution that fits the basic optimization requirements of the problem. The second method that has been proposed is particle swarm optimization. This method has been chosen due to its success in the cryptanalysis of other ciphers found in literature.
4.4 State Register Representation

In the following algorithms the state register $S$ is represented in the following way. The register as a whole is represented as an array $S$ while each word in the array is represented as $S[i]$, where $i \in \{0, \ldots, 2^n - 1\}$. Within each permutation of $S[i]$, there exists a binary value denoted as $b_{(i)}$ where $i \in \{0, \ldots, n\}$. An illustration of this representation can be seen in Figure 4.1. During the course of the Thesis, $n = 8$ bits (unless otherwise stated) which implies that the total storage of a particular candidate solution in the algorithm is 2048 bits or 256 bytes; practical implementations of RC4 would require 2064 bits to account for the two index pointers ($i$ and $j$) but since one of the assumptions is that the algorithm always start from initialization, these values do not need to be stored.

![State Register bit/byte Representation](image)

Figure 4.1: State Register bit/byte Representation

All keystream bytes $z$ are similarly stored as $n$ bit words in an array of $2^n$ words but remain constant throughout the duration of the algorithm and are simply loaded in at the beginning.
4.5 Genetic Algorithm Representation

Since the state register of the RC4 stream cipher is a permutation, modifications must be made to the evolutionary operators of the GA in order to preserve the permutations integrity while performing such operations. The subsequent modifications take place in the crossover methods, as well as the mutation methods. To further fine-tune the GA for the specific problem, runtime modifications also occur to the parameters used.

4.5.1 Selection Methods

There are two selection algorithms that have been chosen for this particular GA algorithm. The first algorithm which will be examined is called fitness proportional representation selection. This method utilizes the percentage fitness of each population member to give a higher probability for survival to the next generation based on the solutions fitness proportional to other solution fitnesses in the population. As seen in Figure 4.2 $f_3$ has the best probability for selection to the next generation.

Until a new population has been re-populated, the fitness proportional algorithm will search a sorted list of parent candidate solutions using a key which is a random value from $[0.0,1.0]$. To improve on search time while using fitness proportional representation, the algorithm will be implemented using a binary search to reduce search time from $O(n)$ to $O(\log n)$.

The second selection method to be utilized is tournament selection. In tournament selection, candidate solutions compete for inclusion into the next generation. Generally, the number of competing candidates is two or three; in the algorithm used for the experiment
Figure 4.2: Fitness proportional representation of different fitness values \( f \)

A tournament of two will be used. The fittest member of the tournament will be the one that survives to the next generation. Tournament selection can be seen in Algorithm 5.

**Algorithm 5** Tournament Selection Algorithm

Input: The number of participants in a tournament \( k \) and the population size \( N \)

Output: Winner of tournament

for \( i = 1 \rightarrow k \) do

    \( \text{index} \leftarrow \text{random}(0, N - 1) \)

    if \( \text{fitness}(\text{population}[\text{index}]) > \text{fitness}(\text{population}[\text{bestFitness}]) \) then

        bestFitness \( \leftarrow \text{index} \)

    Return(\text{population}[\text{bestFitness}])

Aside from the previously stated methods which will be used in the selection process for the GA, the practice of *elitism* will also be introduced. The function and benefit of elitism
is to automatically carry over the most fit member of the population to the next generation in order to ensure the best solutions are not being discarded or destroyed by the selection process.

4.5.2 Fitness Evaluation

When using evolutionary algorithms for cryptanalysis, the fitness function is usually a functional representation of the success of letter frequency analysis as to whether the ciphertext decrypts to the correct language that the plaintext is in. Since RC4 is a stream cipher that creates a keystream of bytes there is no ciphertext to decrypt or plaintext to verify with. On the other hand because the PRGA is essentially a byte random number generator which is seeded by the shared key, we can expect that a properly arranged state register will produce the exact same keystream every time. The GA uses 256 keystream bytes to evaluate the generated candidate solutions.

The fitness value is the number of consecutive successful keystream bytes that a candidate solution can produce. This value can simply be represented as $f(x) \in \mathbb{Z}^+$ for a simple count or represented as a percentage of correctness (seen in Equation 4.2) where $f(x) \in \mathbb{R}$.

$$f(x) = \frac{\text{No. of bytes correct}}{256} \quad \text{(4.2)}$$
4.5.3 Crossover Methods

Since the problem being solved is represented as a permutation the traditional single-point and two-point crossover methods will destroy the permutation property of the chromosome. For this reason, special methods of crossover must be employed.

The first algorithm used for ordered crossover is *partially mapped crossover* (PMX). This form of crossover resembles two-point crossover in that two points are selected and the bits (bytes in our case) between them are exchanged to create two new children. Where it differs is when the original parents are added to the child chromosomes, each value is added to the child in order of appearance until that value is already present in the child from the original crossover. At this point the duplicated value is replaced in the child chromosome by one of the values that was removed also due to the initial crossover. This method can be seen in Figure 4.3.

![Figure 4.3: Example of partially mapped crossover (PMX)](image)
The steps in which this algorithm operates are as follows:

1. Two crossover points are randomly selected

2. A map of the corresponding values in between the crossover points of the two parents is created; this is seen in the middle of Figure 4.3

3. The values between the two crossover points are swapped between the two parents commencing the creation of two children

4. The remaining values in each parent are directly copied to its respective position in each child

   (a) If that value already exists in the crossed-over section of the child, the map is referred to and the value is replaced with its corresponding map value

   (b) If this value is also in the child already, it too is looked up in the map for its corresponding value until a value unique to the child chromosome is found

The second algorithm that will be used for ordered crossover is edge recombination crossover (ER). This method uses an adjacency matrix which is a list of each node and their respected neighbours. Figure 4.4 outlines how edge recombination works. A master adjacency matrix is constructed by taking the union ($\cup$) of the two parent matrices. From the new matrix a starting node is randomly selected and removed from all neighbouring sets and the node is appended to the empty child list. The next node appended is the smallest nodes set of the previous set. For example, if A is the first node appended to the child list,
the member with the smallest set in its set is the next to be removed from all neighbouring sets. Since $A = \{B, C, D\}$, the smallest set of either B, C, or D is next to be appended to the list. Which in this case happens to be $B = \{C, D\}$ since A was removed when it was appended to the child list. In the event which there are multiple sets that are the smallest, the set to use is randomly chosen. The process is repeated until the child list is the same length as the parent chromosome.
Figure 4.4: Example of the steps involved in edge recombination crossover (ER)

1. Parent 1: A B D E F C
2. Parent 2: A B C E F D

Steps:

A: Remove A from all neighbour sets, find smallest set of B, C, and D
B: Remove B, find smallest set of C and D; randomly select one since they are the same (D is selected)
ABD: Remove D, find smallest set of E and F. Randomly select one (F is selected)
ABDF: Remove F, find smallest set of C and E. Randomly select one (C is selected)
ABDFC: E = {} is the smallest set
ABDFCE: Child chromosome is same length as the parent so stop

A: B C D = (C,B) U (B,D)
B: A C D = (A,D) U (A,C)
C: A B E F = (F,A) U (B,E)
D: A B E F = (B,E) U (F,A)
E: C D F = (D,F) U (C,F)
F: C D E = (E,C) U (E,D)
4.5.4 Mutation Methods

Similar to the crossover methods, the mutation methods employed must uphold the permutation property that exists for the candidate solution. Two main algorithms are implemented using the Poisson mutation methodology.

The first method for mutation that is called the swap mutation method (SwM). The method is very straightforward and is conducted by selecting two random indices in the state register and simply swapping their contents. The simplicity in this method is witnessed in Figure 4.5.

![Figure 4.5: Example of swap mutation (SwM)](image)

The second mutation method that is implemented is the inversion mutation method (InvM). This algorithm also preserves the permutation requirement of the chromosome. The process requires two indices in the chromosome to be selected with the sub-sequence between the two indices being simply reversed. Algorithm 6 shows this process.
**Algorithm 6** Inversion Mutation (InvM)

Input: Chromosome $c$ and population size $N$

\begin{align*}
i &\leftarrow \text{random}(0, N - 1) \\
j &\leftarrow \text{random}(0, N - 1) \\
\text{if } i > j \text{ then} & \\
&\text{Swap}(i, j) \\
\text{for } i \rightarrow j & \text{ do} \\
&\text{Swap}(c_i, c_j) \\
&j = j - 1 \\
\text{return}(c)
\end{align*}

4.5.5 Parameter Configurations

There are several parameters of the GA which by default are constant and several that dynamically change to adapt to how the GA is progressing through the solution space. The specific values will be examined in Chapter 5 as part of the evaluation.

Our implementation will use a dynamic mutation rate. The default rate of mutation is set to 4% but there is a ceiling mutation rate also implemented that is 15%. During the generations, the best fitness is sampled at a predetermined rate proportional to the total amount of iterations. If the fitness appears to stagnate over these samples, the mutation rate is increased by 1% until it hits the ceiling rate in order to encourage more diversity in the population. Conversely if the fitness seems to improve over this period, the mutation rate will decrement by 1% until it is back at the original rate of mutation.

The GA has a hard set number of candidates for tournament selection which remains
at two chromosomes competing each tournament. The other main parameter that remains constant is the number of elite candidate solutions preserved to the next generation stays at one.

4.6 Particle Swarm Optimization Representation

Particle swarm optimization is used in this Thesis to contrast the results of the GA as it is another form of evolutionary algorithms. PSOs are generally utilized to model swarm formations which are found in nature such as the patterns of bees or schools of fish. In comparison to GAs, they use a particle system that is a part of a swarm instead of the chromosome and population terminology of the GA respectively.

Particles are given positions and velocities that are inside the search space of the problem. They apply these velocities to each particle to move them through the search space at different rates. One major difference between PSOs and GAs is the fact that an individual particle also stores its best solution as well as a global best solution of the swarm (this is mildly comparable to elitism in GAs).

This method is not predicted to be the more successful method of cryptanalysis of the RC4 stream cipher but rather a comparison of its ability to solve this unique permutation problem with respect to the other evolutionary computation method.
4.6.1 Solution Representation

To maintain the integrity of a permutation solution, a separate method from the operators in the GA is used. All positions, $x_i$, and velocities, $v_{i,d}$, are real numbers such that $x, v \in \mathbb{R}$. Since these values are real numbers they must be converted to integer permutations. Such a task is achieved by assigning the permutation values $0, 1, \ldots, 2^n - 1$ in ascending order to each float position from the lowest value to the highest value.

For example, the float position $x_i = [0.12, 0.03, 0.31, 0.01]$ (when $n = 2$) would translate to the integer permutation $x_i = [2, 1, 3, 0]$.

4.6.2 Fitness Evaluation

Since PSOs generally work in the domain of real numbers, the fitness of the PSO is represented by the Equation 4.2. Therefore, the fitness is represented as $f(x) \in \mathbb{R}$. This evaluation measures how many consecutive $n$-sized words of the keystream are replicated by a particular particle’s register representation.

4.6.3 Parameter Configurations

There are three basic parameters that are used in conjunction with a PSO. They include: $w$, $\phi_p$, and $\phi_g$. In the following experiments $w$ is set to 1.0. This value was common among the cryptanalysis literature using PSOs seen in Section 3.2.2. Both $\phi_p$ and $\phi_g$ are set to 2.0 which was also the common value presented in the literature. A swarm of size 10 is to be used in the experiments.
4.7 Non-evolutionary Methods

For the sake of comparison, a non-evolutionary method has been chosen to compare and contrast the results that will be presented. The method chosen is simulated annealing (SA) which is a probabilistic metaheuristic search approach. This single point search method has been chosen to compare the results of the two evolutionary algorithms and determine whether they are acceptable candidates to solve the state register problem.
Chapter 5

Evaluation

In the following evaluation of the results, each evolutionary algorithm will be examined on its own. First we will evaluate the results from the genetic algorithm in Section 5.1. This will be followed by an examination of the results pertaining to particle swarm optimization and simulated annealing respectively (Sections 5.2 and 5.3). We will conclude with a discussion of these results and a comparison of the different methods evaluated in Section 5.4.

All experiments were conducted on an Intel i7 dual-core CPU running at 2.8 GHz. The machine had 4 Gb of DDR3 RAM available to it. All three methods of exploration were programmed using C and were compiled with gcc version 4.2.1. Aside from certain programming optimizations including reducing any use of system functions that could be time intensive (i.e. malloc() and free()), optimizations at the compiler level were done using the -O3 flag built into the gcc compiler. All experiments were run 25 times with all graphs and tables reflecting the average performance of their results.
5.1 Genetic Algorithm Results

The genetic algorithm was by far the most promising approach used in the experiment. There were many components to the genetic algorithm that were used including:

- Adaptive mutation rates
- Tournament selection
- Partially-mapped crossover (PMX) and edge recombination crossover (ER)
- Swap mutation (SwM) and inversion mutation (InvM)

From a number of experiments which were conducted using the GA, the use of these additional operators was narrowed down to those that had the most profound impact on the results.

5.1.1 Comparison of Crossover Operators

In an attempt to solve the RC4 state register, there were two crossover methods implemented. These operators were PMX and ER. While it was found that ER was more successful when solving the traveling salesman problem [21], when solving the RC4 permutation problem, PMX was found to be the superior technique. It can only be speculated but it is believed that due to the requirement that the register produces the keystream consecutively, PMX disturbs the candidate solution far less than ER. Thus PMX maintains the integrity of higher fitness solutions in the crossover phase.
PMX was able to evolve a solution far better than ER by almost 10%. This is exhibited in Figure 5.1 where \( n = 6 \) and adaptive mutation is utilized. Both methods grow logarithmically, but the ER method just does not produce the results that the PMX achieves. Finally ER was far more time consuming when running the GA (especially for generations of 1 million and greater).

![Figure 5.1: Partially mapped crossover vs. edge recombination for \( n = 6 \) using adaptive mutation](image)

**5.1.2 Comparison of Mutation Operators**

There were also two mutation methods that were compared as well as adaptive mutation which was implemented. The two methods chosen were SwM and InvM. Each of these approaches preserve the permutation property of the state register \( S \). Initial experiments showed that SwM was a far better candidate for evolving fitness than InvM. The improvements that SwM give to the percentage fitness over InvM can be seen in Figure 5.2. In
fact InvM seemed to reduce the best fitness found after about 1 million generations. This is believed to be the case because SwM make a number of small changes to the candidate solution while InvM makes a larger impact on the solution and can potentially mutate the entire solution if the indexes randomly selected were 0 and $2^n - 1$.

Figure 5.2: Swap mutation vs. inversion mutation for $n = 6$ using adaptive mutation

Adaptive mutation was implemented into the GA. This method took a sample of the fitness at certain intervals during the generation process and determined if the fitness value was stagnating. If this was the case, the mutation rate would be increased by a step size until a ceiling value was reached. If the fitness rate increased, the mutation rate would step down until it was at its original rate again. The improvements of the adaptive mutation method can be seen in Table 5.1 for 1,000,000 generations of the GA.
Table 5.1: Percentage increase using adaptive mutation at 1,000,000 generations

<table>
<thead>
<tr>
<th>n</th>
<th>non-adaptive mutation</th>
<th>adaptive mutation</th>
<th>percentage increase (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>11.8</td>
<td>11.8</td>
<td>0%</td>
</tr>
<tr>
<td>7</td>
<td>17.4</td>
<td>16.9</td>
<td>-2.87%</td>
</tr>
<tr>
<td>8</td>
<td>18.1</td>
<td>20.7</td>
<td>14.36%</td>
</tr>
</tbody>
</table>

5.1.3 Results

The main results collected for the genetic algorithm utilized PMX in conjunction with SwM. These results were collected at various generations and both using the adaptive and non-adaptive mutation method. For the experiments, the population was restrained to 5 candidates with one candidate always making it through to the next generation due to elitism. The crossover rate was established at 99%, while the starting mutation rate was set at 4%. Each test was run for a number of trials in order to create a statistical average.

Table 5.2: Best average fitness for word size $n$ using a genetic algorithm

<table>
<thead>
<tr>
<th>Generations</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>non-adaptive</td>
<td>adaptive</td>
<td>non-adaptive</td>
</tr>
<tr>
<td>10000</td>
<td>4.5</td>
<td>4.7</td>
<td>4.7</td>
</tr>
<tr>
<td>100000</td>
<td>8.4</td>
<td>10.6</td>
<td>11.6</td>
</tr>
<tr>
<td>1000000</td>
<td>11.8</td>
<td>11.8</td>
<td>17.4</td>
</tr>
<tr>
<td>10000000</td>
<td>12.8</td>
<td>16.2</td>
<td>21.4</td>
</tr>
</tbody>
</table>

The results shown in Table 5.2 display several interesting properties observed when the problem size ($n$) increases. It can be seen that the improvements of the GA at each increase in generations is a logarithmic function (See Figure 5.3). The function for $n = 8$ using adaptive mutation is represented in Equation 5.1.
Using this extrapolation, it can be predicted that on average the keystream could be completely replicated after approximately $4.0 \times 10^{36}$ or approximately $2^{121.5}$ generations. While this value is extremely large (there are less stars in our solar system), it is still a great improvement over $8.578 \times 10^{506}$, which is what is needed to brute-force the state register permutation. Further if the GA was implemented as a hardware chip, it could compute one generation each cycle.

Figure 5.3: Genetic algorithm best average fitness for various word sizes $n$ using adaptive mutation
5.2 Particle Swarm Optimization Results

The second evolutionary algorithm implemented was the PSO. Unlike the GA, this algorithm presented highly lackluster results and was not very effective for this sort of permutation problem. For every word size, the PSO regressed in improvement between 100000 generation and 1 million generations before slowly improving again (see Table 5.3).

<table>
<thead>
<tr>
<th>Generations</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>3.125%</td>
<td>1.5625%</td>
<td>0.5859%</td>
</tr>
<tr>
<td>100000</td>
<td>3.28125%</td>
<td>1.71876%</td>
<td>0.82027%</td>
</tr>
<tr>
<td>1000000</td>
<td>3.125%</td>
<td>1.64063%</td>
<td>0.74217%</td>
</tr>
<tr>
<td>10000000</td>
<td>4.0625%</td>
<td>2.03128%</td>
<td>0.7812%</td>
</tr>
</tbody>
</table>

Table 5.3: Best average fitness for word size \( n \) using particle swarm optimization

5.2.1 Evaluation of PSO Effectiveness

Traversal through the search space of this particular permutation problem appeared a little problematic for the PSO. The search space was originally treated as integer bits like in previous successful cryptanalysis attempts of other ciphers. This method was quickly abandoned for the use real numbers like in traditional swarm modelling and immediately improved the results gathered. Though these results were an improvement on the original methodology, they still were not able to traverse the search space as well as the genetic algorithm or the simulated annealing algorithm.
Figure 5.4: Particle swarm optimization best average fitness for various word sizes $n$

The graph in Figure 5.4 shows the results of the PSO trials. It demonstrated how little improvement could be made as the generations increased. The algorithm improved its solutions at a very slow rate. As the size of the problem increased, improvements from each generation did not increase proportionally as witnessed in the GA.

Using a PSO has shown to be a poor choice for trying to solve the RC4 state register as predicted and is an avenue of pursuit that should not be further explored (in relation to this specific problem).

### 5.3 Simulated Annealing Results

The simulated annealing method was used to evaluate if any improvements could be made to the results over the evolutionary algorithms by using a single step method. The use
of the SA showed that the best solutions plateaued after temperature of 1 million and a cooling rate of 1.0x10^{-8}. While given enough time a correct solution could be found, the goal of these attacks is to improve on the time it would take to brute force the permutation.

The graph in Figure 5.5 shows this stagnation of improvement.

![Figure 5.5: Simulated annealing best average fitness for various word sizes n (cooling rate: 1.0x10^{-8})](image)

Using the SA method was extremely slow in comparison to the previous methods examined and the results had very little, if any, improvements when the temperature was increased. This method is believed to be not as satisfactory as the GA due to the swap operator used to single step through the search space on a single $S$ register. Unlike problems like the traveling salesman problem, where one swap can improve a current result, the state register solution of the RC4 stream cipher usually is destroyed by a simple swap since it requires the permutations that have already been used to generate a keystream to remain
intact. For this reason a population larger than one and the functionality of elitism appears to improve the candidate solution(s) over time (as utilized in the GA). A table of complete results related to the SA can be found in Chapter 7.

5.4 Comparison of Results

The experiments showed that while the results were not as initially intended, the GA as predicted, still vastly out performed the PSO and SA. The GA was far superior to every other method and was shown to be the best area to focus further efforts in attempting the solve the state register permutation in these experiments. The SA showed a 55% improvement over the PSO for 10 million generations while the GA showed a 1210% improvement over the PSO for the same number of generations. This confirms the genetic operators chosen in our experiments were the best tools (of the methods evaluated) for attacking the permutation problem presented over the single step methodology of the SA and the velocity traversal of the PSO.

It should also be noted that the performance time of the GA was far superior to the other two methods. This further emphasizes the that the GA would likely be the best method for any practical attack in the future as computing power continues to improve.

The GAs vastly better results are contrasted against the other two methods in Figure 5.6. These results are for a word size \( n = 8 \) and reflect when adaptive mutation was used with the GA. While these results were not as initially desired, they show a promising new form of cryptanalysis for the RC4 stream cipher over the other main method of attacking the
Figure 5.6: Comparison of algorithms for $n = 8$

PRGA, the algebraic attack.
Chapter 6

Conclusions and Future Work

In summary, we explored three methods to attack the state register $S$ of the RC4 stream cipher. The three algorithms include: the genetic algorithm, particle swarm optimization, and simulated annealing. The first two algorithms were the evolutionary algorithms being tested and the third algorithm was a non-evolutionary method to compare results of the evolutionary operators with a single step algorithm.

Through the experiment data presented in Chapter 5, it was found that the GA performed far better than the other algorithms. On average the GA was able to yield 10% of the keystream with a generated state register when $n = 8$, 18% of the keystream when $n = 7$, and 25% of the keystream when $n = 6$. Further it was found that the GA was the better choice over the PSO and the SA and on average could find 100% of the keystream in about $4.0 \times 10^{36}$ or approximately $2^{121.5}$ generations which could help improve the state recovery attack of [27]. If the GA was implemented on a hardware chip, it would have the potential to find the state register quicker than brute-forcing the permutation.
We also found that for the state register permutation problem several operators used with the GA were better than others. The partially mapped crossover method proved to be much better at improving the fitness of the population than the edge recombination method. Also the swap mutation operator was far more successful at reducing convergence while improving fitness over the inverse mutation operator. The adaptive mutation method helped ensure a diverse population as it converged on the most fit solution it was generating. This method was especially helpful due to the high number of generations that were needed for the results achieved.

In closing, the GA has shown that it can be used to aid in the cryptanalysis of the RC4 stream cipher. This is a promising method for future cryptanalysis and should be tried on other encryption algorithms.

6.1 Future Work

There are several areas where further experimentation could be evaluated for the state register permutation problem. Since the GA performed far better than the other algorithms, the following proposed work will be in relation to the GA.

Finally, other operators for ordered problems such as cycle crossover should be investigated to confirm whether the ones used in this Thesis were the best candidates for this particular problem. The GA can also be parallelized to further improve the running time of the algorithm and allow higher amounts of generations to be tested in reasonable time. Other encryption algorithms would also make an interesting study of the versatility of the
GA when used for cryptanalysis.
Chapter 7

Appendix A

<table>
<thead>
<tr>
<th>Generations</th>
<th>non-adaptive</th>
<th>adaptive</th>
<th>percentage increase (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>4.5</td>
<td>4.7</td>
<td>4.4%</td>
</tr>
<tr>
<td>100000</td>
<td>8.4</td>
<td>10.6</td>
<td>26.2%</td>
</tr>
<tr>
<td>1000000</td>
<td>11.8</td>
<td>11.8</td>
<td>0%</td>
</tr>
<tr>
<td>10000000</td>
<td>12.8</td>
<td>16.2</td>
<td>26.6%</td>
</tr>
</tbody>
</table>

Table 7.1: Percentage increase using adaptive mutation over non-adaptive mutation for the genetic algorithm ($n = 6$)

<table>
<thead>
<tr>
<th>Generations</th>
<th>non-adaptive</th>
<th>adaptive</th>
<th>percentage increase (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>4.7</td>
<td>4.2</td>
<td>-10.6%</td>
</tr>
<tr>
<td>100000</td>
<td>11.6</td>
<td>11.3</td>
<td>-2.6%</td>
</tr>
<tr>
<td>1000000</td>
<td>17.4</td>
<td>16.9</td>
<td>-2.87%</td>
</tr>
<tr>
<td>10000000</td>
<td>21.4</td>
<td>23.9</td>
<td>11.7%</td>
</tr>
</tbody>
</table>

Table 7.2: Percentage increase using adaptive mutation over non-adaptive mutation for the genetic algorithm ($n = 7$)
Table 7.3: Percentage increase using adaptive mutation over non-adaptive mutation for the genetic algorithm ($n = 8$)

<table>
<thead>
<tr>
<th>Generations</th>
<th>non-adaptive</th>
<th>adaptive</th>
<th>percentage increase (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>3</td>
<td>3.8</td>
<td>26.7%</td>
</tr>
<tr>
<td>100000</td>
<td>10.5</td>
<td>10.5</td>
<td>0%</td>
</tr>
<tr>
<td>1000000</td>
<td>18.1</td>
<td>20.7</td>
<td>14.36%</td>
</tr>
<tr>
<td>10000000</td>
<td>29.5</td>
<td>26.2</td>
<td>-11.2%</td>
</tr>
</tbody>
</table>

Table 7.4: Best average fitness for word size $n$ using simulated annealing

<table>
<thead>
<tr>
<th>Temperature</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>4.3</td>
<td>3.4</td>
<td>3</td>
</tr>
<tr>
<td>100000</td>
<td>4.2</td>
<td>3.4</td>
<td>3</td>
</tr>
<tr>
<td>1000000</td>
<td>4.5</td>
<td>3.7</td>
<td>3</td>
</tr>
<tr>
<td>10000000</td>
<td>4.5</td>
<td>3.7</td>
<td>3.1</td>
</tr>
</tbody>
</table>

Table 7.4: Best average fitness for word size $n$ using simulated annealing
Bibliography


