Advances in String Algorithms for Information Security Applications

Aljamea, Mudhi Mohammed

Awarding institution:
King's College London

Advances in String Algorithms for Information Security Applications

Moudhi Mohammed Aljamea

Department of Informatics

King's College London

This dissertation is submitted for the degree of

Doctor of Philosophy

King's College September 2016

Supervisors:

First Supervisor: *Prof. Costas S. Iliopoulos*

King's College London.

Second Supervisor: *Prof. Maxime Crochemore*

King's College London.

Examiners:

Dr. Aris T. Pagourtzis

National Technical University of Athens (NTUA)

Doc. Ing. Jan Janousek

Czech Technical University (CTU)

I would like to dedicate this thesis to my loving parents …

Mohammed Jasim Aljamea

Monerah Moahmmed Alsoqair

Proud Daughter

Moudhi Mohammed Aljamea

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains some work done in collaboration with others,as specified in the text and Acknowledgements.

Moudhi Mohammed Aljamea

September 2016

Abstract

This thesis focuses on introducing novel algorithms in information security through studying successful algorithms in bioinformatics and adapting them to solve some open problems in information security. Although, the problems in both bioinformatics and information security are different, yet, they might be considered very similar when it comes to identifying and solving them using Stringology techniques. Different successful bioinformatics algorithms have been studied and introduced to solve different security problems such as malware detection, biometrics and big data. Firstly, we present a dynamic computer malware detection model; a novel approach for detecting malware code embedded in different types of computer files, with consistency, accuracy and in high speed without excessive memory usages. This model was inspired by REAL; an efficient read aligner used by next generation sequencing for processing biological data. In addition, we introduce a novel algorithmic approach to detect malicious URLs in image secret communications. Secondly, we also focus on biometrics, specifically fingerprint which is considered to be one of the most reliable and used technique to identify individuals. In particular, we introduce a new fingerprint matching technique, which matches the fingerprint information using circular approximate string matching to solve the rotation problem overcoming the previous methods' obstacles. Finally, we conclude with an algorithmic approach to analyse big data readings from smart meters to confirm some privacy issues concerns.

List of Publications

Portions of the work detailed in this thesis have been presented in national and international scholarly publications, as follows:

- Chapter 3 (first section):
  - Aljamea, M., Ghanaei, V., Iliopoulos, C.S. and Overill, R.E., 2013. Static Analysis and Clustering of Malware Applying Text Based Search. *In The International Conference on Digital Information Processing, E-Business and Cloud Computing (DIPECC2013)* (pp. 188-193). The Society of Digital Information and Wireless Communication.

  - Alatabbi, A., Al-Jamea, M. and Iliopoulos, C.S., 2013. Malware Detection using Computational Biology Tools. *International Journal of Engineering and Technology,* 5(2), p.315.

- Chapter 3 (second section):
  - Aljamea, M., Iliopoulos, C.S., and Samiruzzaman, M. Detection of URL In Image Steganography *Proceedings of the 2016 ACM International Conference on Internet of things and Cloud Computing (ICC 2016).* ACM, 2016.(Accepted to be published by ACM DL (dl.acm.org)) and (invitation to submit the extension of the accepted paper at a journal).

- Chapter 4:
  - Aljamea, M, Athar, T, Iliopoulos, C.S., Pissis, S and Rahman, MS 2015. A Novel Pattern Matching Approach For Fingerprint-Based Authentication. *in Patterns 2015. IARIA,* pp. 45-49.
  - Ajala, O., Aljamea, M., Alzamel. M, and Iliopoulos, C.S. Fast Fingerprint Recognition Using Circular String Pattern Matching Techniques. *in Patterns 2016. IARIA,* (accepted).
- Chapter 5:
  - Aljamea, M.,Brankovic, L., Gao, J. Iliopoulos, C.S., and Samiruzzaman, M. Smart Meter Data analysis *Proceedings of the 2016 ACM International Conference on Internet of things and Cloud Computing (ICC 2016).* ACM, 2016. (Accepted to be published by ACM DL (dl.acm.org)) and (invitation to submit the extension of the accepted paper at a journal).

The above papers are the only conjoint work included in this thesis. The candidate contributed at least 50% work to each paper mentioned above. The rest of the material in this thesis is entirely the candidate's contribution.

Acknowledgements

My experience in the Department of Informatics at King's College London was intellectually exciting and fun, an unforgettable journey for which I thank God for making it part of my life. The first person I would like to thank is my supervisor, who I consider myself very lucky to have, Professor Costas S. Iliopoulos. He has made my PhD journey not only about knowledge and research but also about how to enjoy it, and he helped me to be confident enough to publish my work and present it at conferences all over the world. Something that I know for sure not every supervisor is capable of doing.

Special thanks also go to my second supervisor, Professor Maxime Crochemore, for his invaluable insights and advice and to Dr Solon Pissis for being a great source of ideas and motivating discussions. I would like to express my special appreciation to Dr Manal Mohammed for her valuable comments on my work. Also, I'm grateful to my AB group for making me feel like part of a family, rather than a research group, through all of the sleepless nights we worked together before deadlines, and for all of the fun we have had in the last four years. To my biggest support, my parents, who believed in me since I was young and encouraged me to succeed and achieve more. They made my dream part of their dream. I can't express how lucky I am to have a sister who supported me while she is also going through a tough PhD journey in London with her family. My two brothers, Abdullah and Hamid, and Hamid's wife, May, supported me spiritually and kept reminding me every day how proud they are of me. I am sincerely thankful to

my aunt Moudhi and every person in her family for their love, understanding and unlimited support. To my best friend Dr Nouf Alnumair who felt my pain and happiness and was there for me whenever I needed her with constant support and encouragement. And thanks to all my friends who filled my life with joy and happiness every day. Finally, I owe my gratitude to my beloved country who fully sponsored my studies and made it easy for me to concentrate on my research.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Information Security

Information security is one of today's most interesting and challenging topics, especially if one considers the ever-developing Internet and the increasing number of online social platforms and services. These services are generating an enormous amount of data every day and causing a great threat of cybercrime along with it. For that reason, people, companies and governments are worried about privacy and how to secure these online data. Information security is no longer just about technology devices and anti-virus, it is now more about three factors – social, people and technology – all coming together.

However, we are beginning to see some changes that can improve how we manage these problems, yet, information security is still a very big aspect when it comes to handling information.

In this era, everything revolves around information, but not just any type of information, specifically cyber information. Everything has evolved around technology; from an application that lets you order a cup of coffee, to more complex applications like building factories and performing surgeries.

Technology has been incorporated into our lives in so many ways; it is now hard to completely cut it out of our lives. Instead, information is moving at a fast pace all over the web, so having your information available is a must, but aiming to ensure the integrity and confidentiality of these information is a true challenge. Regardless of the type of information, securing it is crucial, depending on it location, size and accessibility.

The Internet is considered a rich platform of information where many people get benefits from accessing it, yet, they are being attacked by computer malware and various other threats which distract them from their normal and efficient work flow. These types of malware; software that is harmful to both computers and networks, are so pervasive that anti-virus software companies receive extensive amounts of malware variants daily; therefore, there is an essential need to improve their detection techniques, accuracy and speed in order to protect their customers. Chapter 3 in this thesis is concerned with this type of security problem, we provide a novel malware detection algorithm and present the implementation and experiment results. Our work on this area has been published in [9] and [4].

Meanwhile, hackers techniques are changing and getting harder and more challenging for security experts to produce solutions as fast as new types of attacks emerge by the clock. Hackers have the capability to experiment persistently and to hit the web with new manifestations of malware, creating cyber-terrorism.

Therefore, “white hat hackers” have been on the chase to create and ramp up the existing solutions to help mitigate or even sometimes eliminate the risks that are being exposed. Some hacking techniques are quite old but they are being modified to engage with the new digital era, such as the art of hiding information, or “steganography”.

Steganographic techniques started back in ancient Greece, for example, writing text on wax-covered tablets or shaving the head of a messenger to tattoo a message

on his head, and after the hair grew back, the message would be undetected until the head was shaved again.

However, these steganographic techniques now has been developed to hide malware in images by embedding the malware code along with the image code so when downloading an innocent image, it will attack the victims' computer. This technique only started two years ago, at the end of 2013. In the second part of Chapter 3 we study the steganography in details and introduce a new detection technique that can prevent some of the image steganography malicious attacks, the work on this problem has been published in [5].

Furthermore, focusing on other aspects of information security would be through looking into problems that may arise with current solutions and try to modify and update the ways these solutions work. Constantly, creating better, more efficient and effective solutions to existing problems is a key, the more solutions that emerge, the stronger and more immune the security is.

Without information security, problems like identity theft and cybercrimes would be done more often and more easily, for instance, identification and verifications via biometrics (fingerprint for example) is part of essential regular individuals activities, therefore, in chapter 4 we revisit the fingerprint matching problem and try to solve it in a novel technique using approximate circular string matching to overcome the weaknesses in the previous methods, the corresponding achievements has been published in [8] and [7].

Moreover, industries now have changed their mind-set when it comes to information security, since they have realised that it may only take one action to affect the whole dynamic of that industry and cost them their reputation and business continuity. It is not a very different case when it comes to individuals.

Having increasing amounts of identity theft and cybercrime has proved that securing your information is essential. According to [133] "Cyber-crime has been

estimated to cost the global economy in excess of $445 billion each year."
One way of providing security control, is to ensure that certain organisations are abiding by the privacy acts and making sure that the information of individuals they collect stays confidential during use, transmission and destruction. Therefore, the United States Privacy Act of 1974 states the following:

> *"The Privacy Act guarantees three primary rights:*
>
> - *The right to see records about oneself, subject to Privacy Act exemptions;*
> - *The right to request the amendment of records that are not accurate, relevant, timely or complete; and*
> - *The right of individuals to be protected against unwarranted invasion of their privacy resulting from the collection, maintenance, use, and disclosure of personal information."*

(The U.S. Department of State, Freedom of Information Act, 22-05-2015) [129].

Information security consists of many different aspects, such as the technology aspect, as well as the human aspect. whereas the latter, has become an increasing point of attack and always a factor to target, as it is the weakest link in a security model. Providing sufficient human awareness may help to mitigate the problems that may arise from social engineered attacks which consist of the most efficient techniques in gaining classified information. Securing the human is always the goal, whether it is physically or intellectually.
Providing a global understanding of privacy is also crucial since everything is connected. Companies today are providing their customers with more integrated services that will give them more access to their data and daily activities. One such case, is electricity companies marketing the new smart meters technology as a beneficial service to reduce electricity usage by monitoring the electricity readings in real time. Although the users might benefit from this extra service, it will compromise their privacy by giving the utility constant access to their electricity

readings. Any kind of information can be used by numerous types of people, however, unauthorised use of this information is an invasion of privacy and may lead to severe consequences. In chapter 5 we define the smart meters privacy problem in details and provide algorithmic data analysis technique to detect private in-house activities, The corresponding work has be been published in [6].

Finally, securing information is not solely for corporate data, as it should also consider all types of information since technology is encompassed in our day to day activities. Creating a canopy of security for all aspects is crucial since it not only provides confidentiality of the data, but also assures integrity of the data itself. Information security often talks about the security triad which is the confidentiality, integrity and availability of data.

1.2 Bioinformatics

Algorithms in computers and information technology are considered as a tool to solve problems in many fields and research science areas. This is particularly true in biological research. Bioinformatics is a relatively new research area that addresses the need to manage and interpret the data that were generated by genomic research in the past decade.
According to the *Oxford English Dictionary* bioinformatics is conceptualising biology in terms of molecules and applying informatics techniques. In other words, bioinformatics is a management information system for molecular biology and has many practical applications [75].

The National Center for Biotechnology Information (NCBI)[84] defines bioinformatics as follows:

"Bioinformatics is the field of science in which biology, computer science, and information technology merge into a single discipline. The ultimate goal of the field is to enable the discovery of new biological insights as well as to create a global perspective from which unifying principles in biology can be discerned different types of information."

Also, the Swiss Institute of Bioinformatics describes it as:

"Bioinformatics provides novel methods to store, analyse and visualise this information — creating new knowledge to enhance our standard of life."

Moreover, there are three sub-disciplines in bioinformatics. The first, focuses on the informatics side, and concentrates on the development of new algorithms and statistics with which to assess relationships among members of large data sets. The second, focuses on the biology side, and concentrates on the analysis and interpretation of various types of data including nucleotide and amino acid

sequences, protein domains and protein structures. Finally, the third, is the result of the previous two. It concentrates on the development and the implementation of tools that enable efficient access and management of different types of information. In general, bionformatices can be defined as well as any use of computers or specifically algorithms for processing any biologically-derived information [15].

1.3 Bioinformatics and Information Security

In this thesis, different successful bioinformatics algorithms have been studied and introduced to help in solving different security problems such as malware detection, biometrics and big data, mixing more than one filed together to solve similar problems will give the solutions strength and empower them through looking at similar problems from different aspects. And that is a key factor in this thesis from detecting malware using an efficient read aligner used by next generation sequencing for processing biological data to fingerprint matching technique using circular string matching algorithm that was used to solve human virus problem in bioinformatics.

1.4 Algorithms and Complexity

The task of representing information is considered fundamental in computer science; however, the main purpose of computer programs is not only to perform calculations, but also to store, analyse, secure and retrieve information as quickly as possible. For this reason, the study of data structures and the algorithms that manipulate them is at the heart of computer science [112].

According to the Oxford Dictionary of Word Origins (2 ed.) the word "algorithm" originally meant the Arabic or decimal notation of numbers. It is a variant, influenced by the Greek arithmos for "number", and the Middle English algorism, which came via Old French from Mediaeval Latin algorismus, derived from the Muhammad ibn Musa al-Khwarizmi, a 9th century Persian mathematician [3].

Furthermore, the OED defined algorithm as:

"A procedure or set of rules used in calculation and problem-solving; (in later use spec.) a precisely defined set of mathematical or logical operations for the performance of a particular task."

An algorithm is a procedure to accomplish a specific task. It is a method or a collection of finite number of steps followed to solve a problem. If the problem is viewed as a function, then the algorithm is an implementation for the function that transforms an input to the corresponding output. A problem can be solved by many different algorithms. A given algorithm solves only one defined problem. Algorithms is the core of every program, and each program will be evaluated by its speed, accuracy and memory as these are the main factors in designing an efficient algorithm.

In the most general sense, algorithms are involved in every person's life, not just computer programs, as it is the well-defined steps to use the right resources to solve a well-defined problem. A simple example, is that preparing a meal will

require specific steps (the recipe) using specific resources (the ingredients), and manipulating these resources results in an output (the meal), regardless of whether it is a desirable result or not. However, introducing some modern cooking techniques might speed up the process and provide a better result.

This is the case in different life problems or situations where a person needs to identify specific steps to solve or reach the best result. Therefore, algorithm design won't only solve computational problems but will as well help to solve daily life problems. Since technology is involved in our life more and more day after day, creating programs to manage these technologies will be absolutely essential, from designing the right algorithm to providing the matching required output. Algorithms are not limited to computer science, but can be involved in solving any kind of problem related to any field or science (e.g. finance, marketing, security, biology).

Furthermore, many successful companies that are worth billions of pounds started by designing an algorithm to solve a specific problem, like an online search engine. Designing an algorithm is done through the description of the algorithm at an abstract level in pseudo code, and proving its correctness by solving the given problem in all inputs.

A key factor in designing an efficient algorithm is that the problem should be formally defined, where the given instance and question are stated clearly. There are three required properties for a good algorithm design: it should be correct, efficient and easy to implement [113].

Algorithm analysis is a field in computer science which concentrates primarily on the algorithm's performance evaluation and understanding the time complexity of that algorithm. The "Big O" is the time complexity notation of an algorithm and it is usually estimated by counting the number of elementary functions performed by the algorithm in basic operations; multiplication, single addition or subtraction are

considered to count as a unit time.

The complexity of an algorithm is a function $f(n)$ where is n is the input size. Generally, there are three cases to find the complexity function $f(n)$:

- Best case: The minimum value of for any possible number of inputs n.
- Worst case: The maximum value of for any possible number of inputs n.
- Average case: The value which is in between maximum and minimum for any possible number of inputs n.

Algorithms are compared by evaluating their performance and how they respond in their processing time or working space requirements to different input size. In fact, the order of growth of the running time of an algorithm gives a simple evaluation of the algorithm's efficiency and also allows us to compare the relative performance of similar algorithms.

The analysis of an algorithm helps us to understand it better, and can suggest informed improvements. However, measuring the time complexity of an algorithm usually comes after proving the correctness of the algorithm for the given problem.

1.5 Thesis Main Contributions

The key contributions of this thesis are as follows:

1. The first contribution in the "Malware Detection Techniques" chapter is solving the malware detection problem using a computational biology tool through combining the concepts of DNA mapping algorithms and malware signature-base detection problems. Therefore, this section introduced a signature-based detection algorithm which uses approximate string matching techniques, word-level parallelism, sorting algorithms, pigeonhole principle and other techniques to build and present an efficient linear time and space algorithm which can be considered a promising solution for detecting malware signatures, different to what has been done so far in the field. The main contribution is the novelty of the new detection approach which was designed in respect to the great threat of daily introduced malware signatures. The experiment results proved that the proposed algorithm performed with high accuracy, speed, and less space as required.

   In fact, the presented experimental results are very promising, both in terms of efficiency and sensitivity on the detection process, especially with respect to the space, as it was able to complete the assignment on high speed, despite not using a stored pre-processed index of the scanned files. This becomes especially important considering the problem of the massive amount of malware-signatures generated daily. The second contribution in the "Malware Detection Techniques" chapter involves developing an algorithm to extract the hidden URL in images, and it can be considered as one of the first tools to detect the hidden URL in images and extract it as it is. The proposed approach used fast string matching and sorting algorithms to develop the proposed solution. The experimental results showed very promising accu-

rate results for this new security problem which encouraged us to take the solution to the next level and consider detecting and extracting the encrypted hidden URL in the images.

2. The main contribution in the "Fingerprint Recognition Techniques" chapter is developing a new approximate pattern matching based approach for fast and accurate recognition of fingerprints, and solving the rotation problem through making use of the approximate circular string matching algorithms. The proposed solution is a first attempt to address the issue of fingerprint recognition systems through using circular string matching algorithms and proposing a solution based on the comparison of the circular binary string representation of fingerprints. The experiment results showed that the false match rate (FMR) and the false non-match rate (FNMR) mean values for the proposed solution match and sometimes outperform the current solutions and give more accurate results in high speed of average time 4.5 seconds, regardless of the rotation degree of the scanned fingerprint. Furthermore, detecting the fingerprint information is usually done through storing the information about ridge endings and bifurcations as sets of coordinates. However, the algorithm in the proposed solution will extract and store the ridge and furrow information in the form of circular strings, and that is the novelty of the solution. This will help in solving the rotation problem while keeping the high speed and accuracy of the detection process.

3. The smart meter analysis problem is a relatively new problem in security and privacy concerns, the contribution of this chapter is through proposing an algorithmic approach in terms of some probabilistic conditions to detect private in-house activities. However, in order to achieve the desired level of efficiency, the approach combined the use of fast sorting, searching and

matching algorithms in a way that through the analysis of time and space complexity for the overall approach proved that the proposed algorithmic approach will achieve a very promising result in development.

1.6 Structure of the Thesis

This dissertation is divided into six chapters as follows:

Chapter 1 - Introduction:

This chapter, attempted to introduce the two major fields of interest for this thesis, namely information security, and bioinformatics, from an algorithmic perspective.

Chapter 2 - Notions and Definitions:

Provides the basic notions required to properly follow the work and results presented in the subsequent chapters; in particular, it introduces the basic definitions and notations on alphabet and strings, string similarity, notations and also describes some elementary data structures and related techniques.

Chapter 3 - Malware Detection Techniques:

The first section of the third chapter describes the malware detection problem by presenting the existing malware evasion techniques along with the latest related work in malware detection. We propose a dynamic computer malware detection model, using a biology tool, that can detect malwares to prevent attacks which might cause damaging or stealing sensitive information. Then, we present the results of implementing the proposed model.

In the second section of this chapter, we define the concept of steganography and its applications and briefly review some of the current image steganography techniques which is the focus of this section. Next, we introduce an algorithm for detecting malware URLs or (any kind) of URLs in image steganography with full analysis of the time and space complexity for each step and then we take the solution to the next level to consider detecting and extracting encrypted URLs. Finally, briefly concluding the section with some discussion and future proposals.

Chapter 4 - Fingerprint Recognition Using Circular String Pattern Matching Techniques:

This chapter, studies the performance of the existing Automated Fingerprint Identification System (AFIS) and then proposes a novel and new pattern matching based approach for quick and accurate recognition of fingerprints regardless of its location and rotation on the scan surface. With the help of approximate circular string matching algorithms. Next, we present the results after implementing the proposed solution.

Chapter 5 - Smart Meter Data Analysis:

Provides a detailed understanding of the new "Smart Meters" technology and discusses the strength and weakness behind it. Then, propose an algorithmic approach for the comparison and analysis of Smart Meter data readings, considering the time and temperature factors at each second to identify the users patterns at each house by identifying the appliances activities at each second with a with full time and space analysis of the proposed algorithm.

Chapter 6 - Concluding Remarks:

Summarises the results of the work presented in the previous chapters, conclusions are drawn. Moreover, the chapter discusses future work based on the presented algorithms.

Chapter 2

Notions and Definitions

2.1 Alphabets and Strings

Strings are the main data type used in this thesis which also can be called words or sequences. A string is an ordered sequence of characters or symbols taken from a finite set called alphabet Σ. This definition implies that any file is composed of finite distinct characters can be treated as a string. For example, all the biological sequences used throughout the thesis are strings over 4 characters which is the DNA alphabet {A,C,G,T}, another example, is the binary strings which are over the alphabet {0,1}.

However, at the beginning of the 21st century the interest of studying strings similarities and properties has increased dramatically and developed into a respectful computer science field known as Stringology or algorithms on Strings. The term stringology first used by Zvi Galil in 1984, denotes a science of algorithms on strings and sequence. Algorithms mainly solves problems like exact and approximate pattern matching, searching for repetitions in various texts,..etc [124].

Stringology can be implemented in many areas that utilize its results, for example, informations security, information retrieval, computer vision, bioinformatics, DNA processing,..etc. And it is expected to grow even further due to the increasing demand for efficient high speed stringology algorithms.

Formally, Let Σ be a finite alphabet which consists of a set of characters (or symbols). The cardinality of an alphabet is denoted by $|\Sigma|$. The set of all non-empty strings over the alphabet Σ is denoted by Σ^+. The empty string is the empty sequence (of zero length) is denoted by ε; that is $\Sigma^* = \Sigma^+ \cup \varepsilon$.

For an alphabet Σ, a string is a sequence of zero or more characters (or symbols). A **string** x of length n is represented by $x[0 \cdots n-1]$, where $x[i] \in \Sigma$ for $0 \leq i \leq n$. The i-th symbol of a string x is denoted by $x[i]$.

Example : $x = aabbaba$ is a non-empty string over $\Sigma = \{a,b\}$. of length $|x| = 7$.

The **string** xy is a concatenation of two strings x and y. The concatenation of k copies of x is denoted by x^k and its called the $k-th$ power of x.

Example: The string $x = bbbb$, is the 4th power of b, denoted for short as $x = b^4$.

A **substring** of x, can be denoted by $x[i \cdots j]$ that starts at position i and ends at position j. Then a string w is a substring of x if $x = uwv$, where $u, v \in \Sigma^*$. Conversely, x is called a **super-string** of w.

Example: If $x = abbaaba$ and $w = baab$ then w can be called a 4 substring of x starts at position 4.

Also, the substring $x[1..i]$ is called a **prefix** of x,and **a proper prefix** of x if $i < n$. Similarly, the substring $x[j..n]$ will be called a **suffix** of x and a **proper suffix** of x if $j > 1$.

Example: If $x = abbaaba$, then $x[1,5] = abbaa$ is the longest proper prefix of x. And $x[4,7] = aaba$ is the proper suffix of x.

A **repeat** in string x is a substring that occurs in x at least two times. Moreover, the repeat can appear in the string x as discrete, consecutive or as overlapping.

Example: If $x = bababab$, then both strings *ba* and *bab* are repeats in x.

The substring w of x can be called **period** of x if x can be written as $x = w^k w'$, where $k \geq 1$ and w' is a prefix of w. The shortest period of x is called **the period** of x.

Example: If $x = abcabcab$, then the substrings *abc*, *abcabc* and the string x itself are all called periods of x but only *abc* is the period of x.

A **cover** is the substring w of x such that x can be constructed by concatenations and superpositions copies of w.

Example: If the string $x = ababaaba$ then the substring *aba* and x cover x. If x has a cover w does not equal to x then x is said to be **quasi-period**; otherwise x is said to be **super-primitive**.

A **seed** is an extended cover in the sense of a cover of a super-string of x.

Example: The string $x = bbabab$ has a proper seed *bab*, since *bab* covers a super-string *babbabab* of x.

2.2 Strings Similarity Measurements

The string similarity is originally a bioinformatics' term which is used to measure how similar two aligned strings are. However, its also can be used on one string to measure which parts of the string are alike.

Usually there are two methods to measure the similarities between two strings or the distances between them as follows:

2.2.1 Distances and Alignment

The Edit Distance defined as the distance $\delta_E(x,y)$ between two strings x and y as the minimal number of a sequence of operations that transform x into y [47]. The number of a sequence of operations is the sum of the individual operations. The operations are a finite set of rules of the form $\delta_E(x,y) = n$, where x and y are different strings and n is a non-negative real number. Once the sequence of operations has converted the string x into y, no further operations can be done on y. The edit distance is symmetrical and, it holds $0 \leq \delta_E(x,y) \leq max(|x|,|y|)$.

- Insertion: $\delta_E(\varepsilon,a)$, i.e. inserting the letter a.
- Deletion: $\delta_E(a,\varepsilon)$, i.e. deleting the letter a.
- Substitution or Replacement: $\delta_E(a,b)$ for $a \neq b$, i.e. substituting a by b.

The Hamming Distance defined as given two strings of equal length, the hamming distance between them is the number of positions for which the corresponding symbols are different. In other words, the hamming distance between two strings of equal length is the minimum number of symbol substitutions required to change one string into the other.

Hamming distance allows only substitutions, which cost 1. also, it is symmetric, and finite. In this case it holds $0 \leq \delta(x,y) \leq |y|$ where $|x| = |y|$.

$$H_{Dist}(x,y) = |I|, I = \{i | x[i] \neq y[i], 1 \leq i \leq n\},$$
$$\text{where } |x| = |y| = n.$$

Alignment of Two Strings: In general, the alignment between two strings $x, y \in \Sigma^*$ whose respective lengths are n and m, is a way to visualize their similarities. Formally, an alignment A between x and y is a string z, such that $(\Sigma \cup \varepsilon) \times (\Sigma \cup \varepsilon) \times (\varepsilon, \varepsilon)$. Furthermore, there are two kinds of string alignment, global alignment where the comparison is done between two complete strings. And, the local alignment where the comparison is done between substrings of two strings.

2.3 Searching and Sorting Algorithms

The **sorting algorithms** is one of the most important operations in computer science. In fact, sorting is usually used as a pre-step by most programmers before they start solving any computational problem to improve the running time of the algorithm. Radix sort for example, considered to be one of the fastest sorting algorithms that runs in linear time.

To illustrate the Radix sort, if there were n numbers to sort, such that each number has J digits, and each digit is in the set $\{1,2,..k\}$, then they can be sorted in $O(J(n+k))$ time [28] by distributing the numbers into temporary positions according to the least significant digit, by repeating the grouping process with the next least significant digit each time the numbers will get closer to their final positions. So, as a result, the numbers will be sorted after reaching the last digit k.

As mentioned before, most of the algorithms especially **searching algorithms** sort the items before preforming any further instructions. For example, if we consider the binary search which is as well considered to be one of the fastest algorithms for searching in a sorted array [113].

For instance, searching for the key T, the algorithm will test whether the key T is in the array A of length n in $O(\log n)$ time by comparing T to the middle key $A[n/2]$ in the sorted array A. If T appears to be less than the middle key then it will result that the key T's position is located in the first sorted half of the array. If not, then it will be located in the second half of the sorted array. By repeating this step recursively on the correct half, it will define the exact position of the key T in a total of $O(\log n)$ comparisons.

2.4 Exact and Approximate string Matching Problem

String matching problem is one of the main problems in computer science. In fact, it arises in wide range of applications [47]. The formal definition of exact pattern matching problem is; given two strings a text t and a pattern p, to determine whether the text t contains an exact occurrence of the pattern p [126] such that $t = wpy$, while considering the speed and efficiency of the searching process.

For example: If $p = bab$ and $t = aababxbababx$, then p occurs in t starting at position 3, 7 and 9. Note that two occurrences of p may overlap, as illustrated by the occurrences of p at locations 7 and 9 [47].

Similarly, an approximate pattern matching problem is an extension of the exact pattern matching by allowing mismatch or edit distance [29]. It is defined as; given two strings a text t and a pattern p, determine whether the text t contains an approximate occurrence of the pattern p as a substring with at most k-differences [26].

2.4.1 Circular String Matching

A circular string of length n can be viewed as a traditional linear string, which has the left- and the right-most symbols wrapped around and stuck together in some way [114]. Under this notion, the same circular string can be seen as n different linear strings, which would all be considered equivalent. Given a string x of length n, we denote by $x^i = x[i..n-1]x[0..i-1]$, $0 < i < n$, the i-th rotation of x and $x^0 = x$. Consider, for instance, the string $x = x^0 = abababbc$; this string has the following rotations: $x^1 = bababbca$, $x^2 = ababbcab$, $x^3 = babbcaba$, $x^4 = abbcabab$, $x^5 = bbcababa$, $x^6 = bcababab$, $x^7 = cababab$.

Here we consider the problem of finding occurrences of a pattern x of length m with circular structure in a text t of length n with linear structure. In fact, This is the problem of circular string matching.

Exact Circular String Matching

The problem of exact circular string matching has been considered in [74], where an $O(n)$-time algorithm was presented. The approach presented in [74] consists of preprocessing x by constructing a suffix automaton of the string xx, by noting that every rotation of x is a factor of xx.

Then, by feeding t into the automaton, the lengths of the longest factors of xx occurring in t can be found by the links followed in the automaton in time $O(n)$.

In [41], an average-case optimal algorithm for exact circular string matching was presented and it was also shown that the average-case lower bound for single string matching of $\Omega(n \log_\sigma m/m)$ also holds for circular string matching.

Recently, in [21], the authors presented two fast average-case algorithms based on word-level parallelism. The first algorithm requires average-case time $O(n \log_\sigma m/w)$, where w is the number of bits in the computer word. The second one is based on a mixture of word-level parallelism and q-grams.

The authors showed that with the addition of q-grams, and by setting $q = \Theta(\log_\sigma m)$, an average-case optimal time of $O(n \log_\sigma m/m)$ is achieved.

Approximate Circular String Matching

The approximate circular string matching pattern problem is an extension of the exact circular string matching by allowing mismatch or edit distance. It will be defined in details through the approximate circular string matching via Filtering (ACSMF) algorithm [14] where it will be used in Chapter 4.

2.5 Fundamental Data Structures

Data structure is the most fundamental and building block concept in computer science, It's a key factor behind designing and developing efficient software systems. Moreover, Creating efficient programs has little to do with " programming tricks " but rather is based on good organization of information and good algorithms [111]. Therefore, data structure is the science of structuring information to support efficient processing and to organize data so it can be accessed quickly and usefully.

Choosing the right data structure will help in processing, retrieving data and extracting information easily. Furthermore, it is important to know that different kinds of data structures are needed to organize different kind of data (text, images, videos, relational data..etc) for different purposes.

Program efficiency will always be the most important aspect in computer science. No matter how fast and powerful computers are getting or how the processor's speed and memory's size are improving, it will never decrease the importance of the programme efficiency. In fact, the fast growing of computer applications will result in producing more data with more complex problems demanding more computational solutions to store, process and secure this massive amount of data. Therefore, there will always be a great need to develop efficient algorithms, and with that comes the need for efficient improved data structures.

2.5.1 Arrays and Linked Lists

An **array** is the most common data structure that contains a collection mainly of similar data types. Elements of data are logically stored sequentially (contiguous) in blocks within the array. Also, the array is entirely allocated as one block in the memory. Each element in the array gets it own space. And any element can be

accessed directly using the index (location) of that particular element (as shown in Figure 2.1).

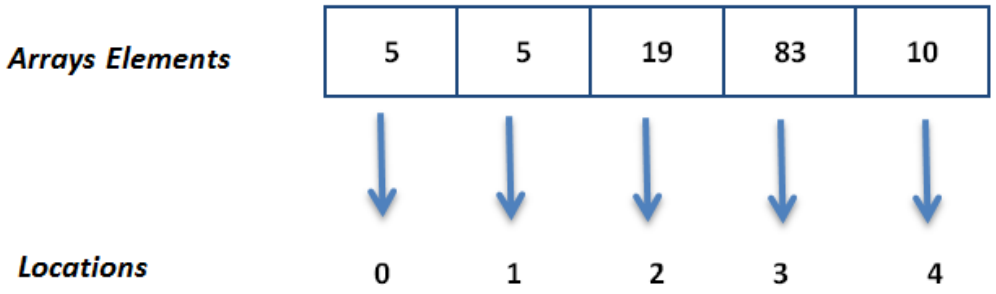


Fig. 2.1 Array-Basic Data Structure

Thereafter, array is the most efficient data structure for storing and accessing a sequence of objects in a constant time $O(1)$. Naturally, if the starting address (base address) is known, then, it will be easy to calculate the addresses of the other elements. However, the main disadvantage of the array data structure is that the size of the array is fixed since it has been created, and in the case of an array overflow a new array should be created as solution to this problem with double the size and all the elements should then be shifted to the new array with an extra cost of time and memory.

Similarly, inserting a new element at the beginning of the array is potentially expensive since it will require shifting all the existing elements over to make room for the new one. In contrast, adding an element at the end of the array will require constant time if the space is available. Moreover, another disadvantage of the array data structure, is the wasted space if the allocated memory for the array was more than the required space then the remaining memory space will be wasted (as shown in Figure 2.2).

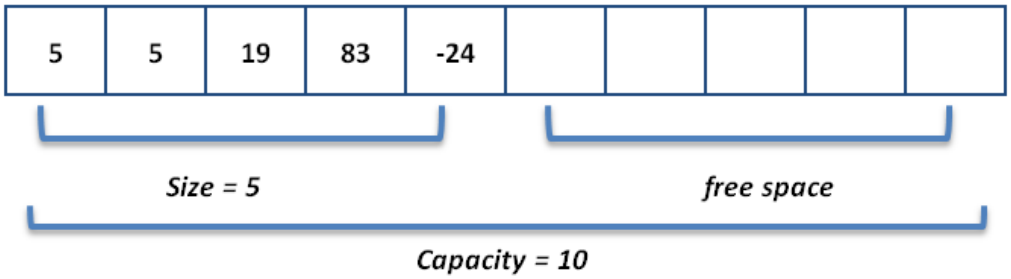


Fig. 2.2 Wasted Space in Arrays

On the other hand, **linked list** is also a common data structure for dynamic list, and uses the memory more efficiently than array. For instance, in the array overflow case (when the array grows more than it size) to add more elements a new array will be created (double size of the previous one) even if the extra required space is only for one element, and that is as result of the fixed arrays size, but in the linked list, the size is not limited, whereas the blocks are created along each time a new element is inserted, also there is no unused memory.

However, there is an extra used memory which is used to store the pointers between the nodes. In fact, the linked list data, is stored in multiple non-contiguous blocks of memory, each block of memory is called *"node"*, the identity of the linked list is the pointer to the head node (pointer to the first node), each node have two blocks one to store the data (the value) the second block to store the pointer (the address to the next node).

For Example, an integer element will require 4 bytes. So, the node will be a block of 8 bytes (4 bytes to store the integer and 4 bytes for the pointer to store the address to the next node) as shown in Figure 2.3.

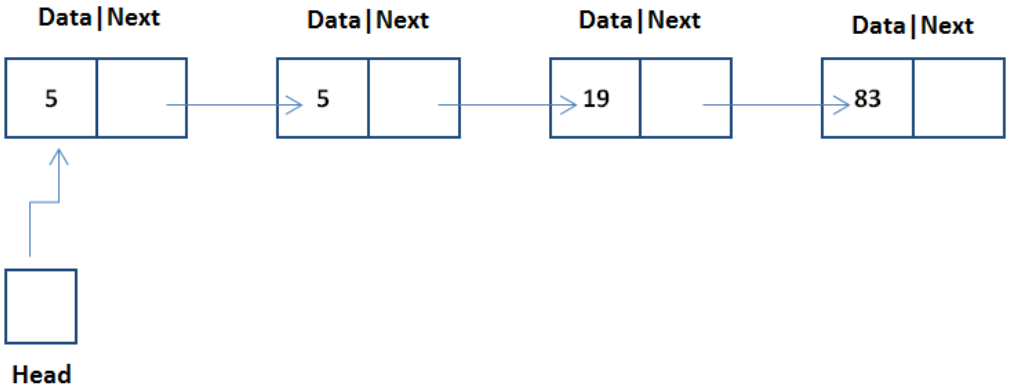


Fig. 2.3 Linked List - Data Structure

Thereafter, in an array it is a constant time $O(1)$ to access an element by knowing the starting address and calculating the addresses of the other elements in the array. Whereas, in linked list we have to travel from one node to anther to find and reach the required element in $O(n)$ time as linked list does not allow random access to the data.

2.6 DNA Sequencing

Deoxyribose Nucleic Acid (DNA) is a molecule that encodes the genetic instructions used in the development and functioning of all living organisms and viruses; it contains the biological instructions that make each species unique. In fact, it is the building block of the life [86]. The DNA was well defined in 1953 by James Watson, Francis Crick, Maurice Wilkins and Rosalind Franklin.

Furthermore, most of DNA molecules consist of double stranded helix which contain two long biopolymers made of nucleotides. In other words, DNA molecule is composed of four types of smaller chemical molecules (nucleotide bases): adenine (A), cytosine (C), guanine (G), and thymine (T); these four letters $\{A, C, G, T\}$ are called DNA alphabet. The order of these four nucleotides within a DNA molecule bases is called the DNA sequence, and segments of the DNA that carry genetic information are called genes. However, each base of these four nucleotides bases bonds with its complementary base to form a base pair. The rules are as follows:

A always bonds to T,

C always bonds to G.

Therefore, a single strand of DNA can be presented as a string composed of the four letters DNA alphabet. Also, if the sequence of one strand of a DNA is known, then the sequence of the strand that will pair with, or "complement" it can be predicted (as shown in the Figure 2.4).

Fig. 2.4 Sample genetic code with complementary strands [15].

DNA sequencing is the process of reading the nucleotide bases and determine the exact order of these nucleotide bases within a DNA molecule. This sequencing process was first developed by Frederick Sanger in 1975. In fact, it was named after him (Sanger Sequencing). The accuracy of this sequencing was high up to 99.9% producing long sequences between 400-900 base pairs.
However, it was mainly a laboratory method [87] which was time consuming in practice and very expensive [97]. Yet, Sanger Sequencing was the workhorse technology for DNA sequencing for so many years until it was replaced slowly with new DNA sequencing technologies that were developed in the mid to late 1990s [10].
Also, in 2006 new technologies were developed to allow rapid sequencing of large amounts of DNA called next-generation sequencing that are much faster and less expensive but with lower accuracy 98 - 99.9% and shorter sequences.
One example of these new next-generation sequencing technologies is the technology that was developed by Illumina called (Genome Analyser), that generates in a single experiment millions of very short reads ranging from 25 to 50 bp. However, the properties that should be considered in any sequencing process are (cost, speed, amount of data, sequence length and error rate) [94].

Consequently, the new high throughput sequencing technology methods have redefined the previous sequencing properties and how the genome is sequenced. By producing tens of millions of short sequences (reads) in a single experiment with much lower cost than the previous old methods. And due to this massive amount of the generated data, an efficient algorithms for mapping these short sequences to a reference genome are in great demand [10]. And as mentioned in Section 1.2, bioinformatics can be defined as the use of algorithms or more specifically computers for processing any biologically-derived information such as DNA sequencing.

Reads: Are DNA fragments which are the outputs of the DNA sequencer instrument, after breaking the DNA into fragments. Each nucleotide sequences is called a “read”. The reads are used later as an inputs to reconstruct the original sequence using a reference DNA.

Chapter 3

Malware Detection Techniques

3.1 Introduction

The Internet is considered to be a rich platform of information where many people get benefit from it access but still they are being attacked by computer malware and various other threats which distract their normal work flow to be carried out in an efficient manner. New anti-malware technologies are introduced to the world by the clock, but at the same time new malware techniques have also emerged to misuse these technologies.

Malware is a generic term used to describe all kinds of malicious software; viruses, worms, spyware and trojan horses are all examples of malicious software. It is created by attackers to not only cause major threat to the security and privacy of computer users and their sensitive information, but most of the time it is also responsible for a significant amount of financial loss. As the complexity of modern computing systems is growing, various vulnerabilities are unavoidable in software systems and online services; this increases the possibility of the malware attack that usually exploits such vulnerabilities in order to damage the systems [139].

Taking " Stuxnet " malware as an example, which was a very complex threat in 2010. This large piece of malware was developed to target Iranian Nuclear Control Systems in order to take control of the system. Stuxnet is a programmable logic controller rootkit, which has applied complex instruction injection, anti-virus avoidance techniques, network infection routines, peer to peer updates, and a command and control interface [38].

Furthermore, Dell SecureWorks Counter Threat Unit™ published on june 2015 a full analysis of the Stegoloader malware which is a Stealthy Information Stealer and they explained that the malware authors are evolving their techniques to evade network and host-based detection mechanisms. Stegoloader could represent an emerging trend in malware by the use of digital image steganography to hide malicious code. In fact, "Stegoloader has a modular design and uses digital steganography to hide its main module's code inside a Portable Network Graphics (PNG) image downloaded from a legitimate website" [33].

However, further the increasing number of vulnerabilities as well as the high number of new malware variants every day will increase the emergency need for anti-malware systems. This is happening while every day an extensive amount of new, unique samples of malware are appearing (based on VirusTotal [131] daily report) Figure 3.1 and Figure 3.2 using different/new carriers to travel Figure 3.3.

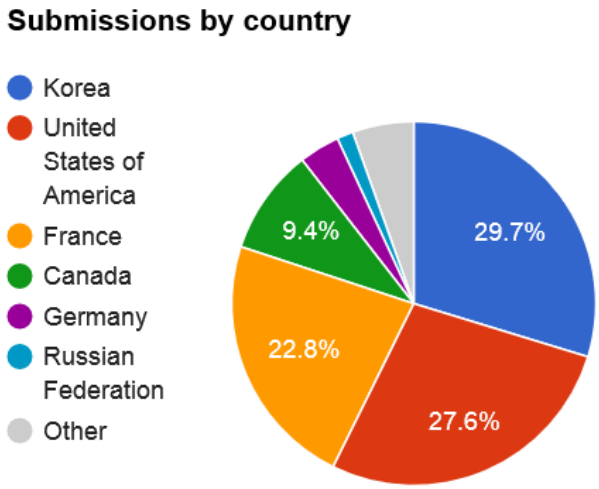


Fig. 3.1 Statistics of Malware Files by Country (during the last 7 days of last access 7-2-2016 via www.virustotal.com)

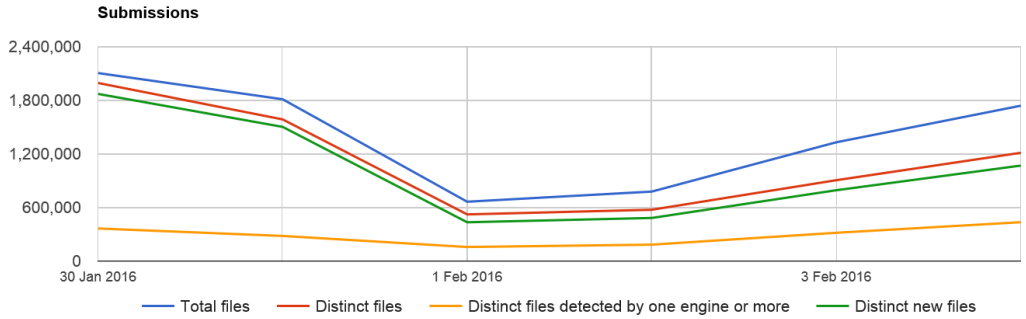


Fig. 3.2 Number of new, unique samples of Malware(during the last 7 days of last access 7-2-2016 via www.virustotal.com)

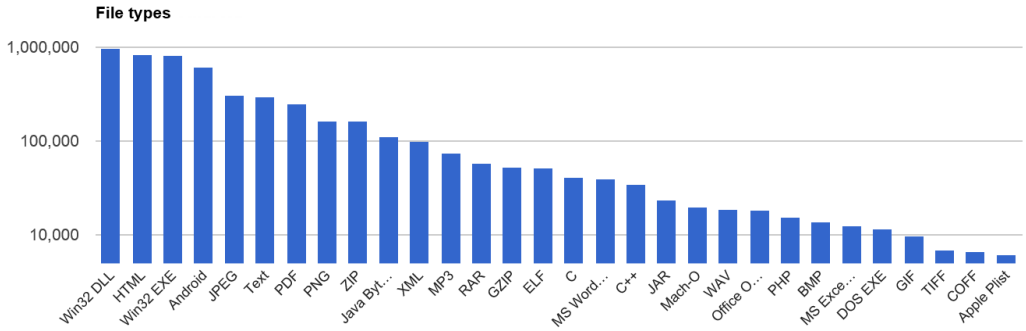


Fig. 3.3 Statistics of Malware using Different Types of Files(during the last 7 days of last access 7-2-2016 via www.virustotal.com)

3.2 Motivation

Although many studies performed towards malware detection, it is always a constant race between malicious code writers and anti-malware researchers. In 2014, the Center for Strategic and International Studies (CSIS) estimated in their report (Net Losses: Estimating the Global Cost of Cybercrime Economic impact of cybercrime) that annual cybercrime cost to the global economy damage is more than $400 billion [31] adding:

> The cost of cybercrime will continue to increase as more business functions move online and as more companies and consumers around the world connect to the Internet.

In this chapter, using a biology tool, a dynamic computer malware detection model has been presented that can detect the malwares to prevent attacks which might cause damaging or stealing sensitive information. This model is inspired by REAL [42] which is an efficient read aligner for next generation sequencing for processing biological data. Experimental results of this study shows that the proposed system is efficient and it is a novel way for detecting malware code embedded in different types of computer files, using bioinformatics tools with consistency and accuracy in detecting the malware and it was able to complete the assignment in high speed without excessive memory usages.

Furthermore, in this chapter, a novel detection approach has been proposed that will concentrate on detecting any kind of hidden URL (innocent or malicious) in most types of images and extract the hidden URL from the carrier image as it is.

3.3 Malware Detection using Computational Biology Tools

3.3.1 Introduction

Considering the daily reports on the huge number of malware variants detected by different honeypots, anti-malware companies and research labs, it is so difficult to manually analyse each variant to understand the purpose of the malware authors. To tackle this problem, malware analysts apply different methodologies which mainly are grouped into three main categories: static, dynamic and a hybrid approach which is a combination of the both static and dynamic analysis.

Static analysis of a malware, focus on the structure of the program and provide a thorough understanding of the malicious software to the analyst without the need to execute the malware. On the other hand, another group of researchers apply dynamic analysis to understand the malware behaviour during the running time. However, in the dynamic analysis the focus of the analyst is on actions of the malware such as the system calls, the read/ write into memory, and etc. by observing the malware behaviour. Apart from the time consuming nature of dynamic analysis, malware authors apply different anti-analysis techniques to obfuscate the analysis process, such as anti-debugging, anti-virtualisation, anti-emulation techniques...etc.

Accordingly, there are many approaches for malware detections which can be classified into two categories. First, is the anomaly-based detection approach which uses its knowledge to monitor the program's behaviour to decide the maliciousness of a program under inspection. Second approach, is the signature-based which is considered as the most popular one [119], it attempts to model the malicious behaviour of malware and uses this model in the malware detection [53]. Both

of the detection approaches can employ one of three different analysis: static, dynamic, or hybrid. Static approach describes the structure of the malicious code in the program that is under inspection before execution. Dynamic approach tries to detect the malicious code during or after the program execution. Hybrid approach is a combination of both previous approaches as shown in Figure 3.4.

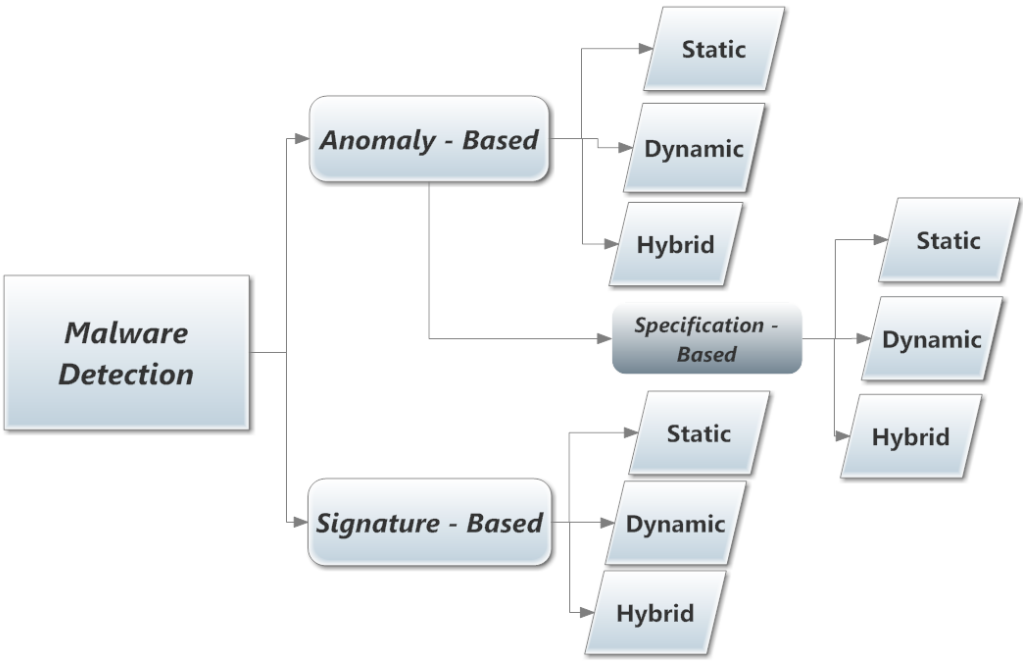


Fig. 3.4 A classification of malware detection techniques

The rest of this section is structured as follows. In subsection 3.3.2, the basic malware analysis and approaches are presented. In subsection 3.3.3, briefly review some of the related work on string matching malware detection approach. In subsection 3.3.4, the basic definitions that are used throughout the section are presented. We give an overview of REAL is subsection 3.3.5. In subsection 3.3.6, formally defining the solved problem. Subsection 3.3.7, the experiment and results are discussed. Finally, briefly concluding the section with some future proposals in subsection 3.3.8.

3.3.2 Background

Malware writers keep improving their obfuscation techniques to make the programs harder to understand and to evade the malware detectors. Encryption is one of the malware techniques that are used widely to evade signature-based detectors. In this technique, an encrypted malware is typically composed of the decryptor and encryptor.

The decryptor recovers the main body whenever the infected file is running. By using a different key for each infection, the malware makes the encrypted part unique, thus hiding its signature [137]. Yet, the main problem of the encryption is that the decryptor remains constant and in such case detector will be able to detect the malware based on the descriptor's code pattern.

However, malware writers always create and develop new techniques in writing malware scripts or codes in order to make it hard to detect. They have reached a point where the virus can modify its code and appearance after each infection in order to avoid the detective and the generic scanning. One of the techniques called *"Polymorphic Malware"* is capable of changing its decryptor slightly, to avoid the problem in the previous technique.

Another more advanced technique is the *"Metamorphic Malware"*. It is considered as one of the best approaches in using obfuscation techniques. It is basically evolves its body into new generations, which changes the total look of the malware while keeping the same functionality. It should be able to recognize, parse and mutate its own body whenever it propagates. It is important that the metamorphic malware never reveals its constant body in memory due to not using encryption or packing, thus making it so difficult for the anti-malware scanners to detect this malware [137].

Nevertheless, there are many obfuscation techniques that are specifically used by

the malware writers in the polymorphic and metamorphic malware approaches for example (Dead-Code Insertion, Register Reassignment, Subroutine Reordering, Instruction Substitution, Code Transposition and Code Integration).
However, most of the malware writers use an old version of a malware to create a new one by reordering the malware instructions. The majority of malwares that appears today is a simple repacked version of an old malware [92]. Even after changing or reordering the instructions of the malware they will still share some behaviours. Different obfuscated versions of the same malware have to share (at least) the malicious intent, namely the maliciousness of their semantics, even if they might express it through different syntactic forms. Therefore, addressing the malware detection problem from a semantic point of view can lead to a more robust detection system [96] which will help in detecting them since the detectors are familiar with the old malware code.

Executable packing in malwares is basically the approach of using the executable packing technique which is popular nowadays among the malware writers to obfuscate malicious code and evade detection by signature-based anti-virus software. This later technique is the most common one. In general, it is believed that nearly 80% of malware are packed and 50% of existing malware are packed versions of old malware [83] and that is due to the accessible effortless open-source and commercial executable packers that help these writers to generate an encrypted version of their malware. Since it has been packed, the signature-based anti-malware will not detect the malicious code as it will not be able to match the signature with the packed malware. As soon as the malware is executed it will be decrypted and do the harm to the computer.
On the other hand, anti-malware providers try their best to follow up with the latest developments in order to be able to detect and remove these new malwares and

overcome their threats. For example, there have been universal unpackers that can help in detecting and extracting encrypted code from packed executables, but these unpackers are expensive and time consuming as it might take hours or even days to scan large collections of executables looking for malware infections.
However, [92] has devised a new approach by applying pattern recognition techniques for fast detection of packed executables. The objective behind fast detection is to efficiently and accurately distinguish between packed and non-packed executables, so that only executables detected as packed will be sent to a universal unpacker, thus saving a significant amount of processing time as descried below in Figure 3.10.

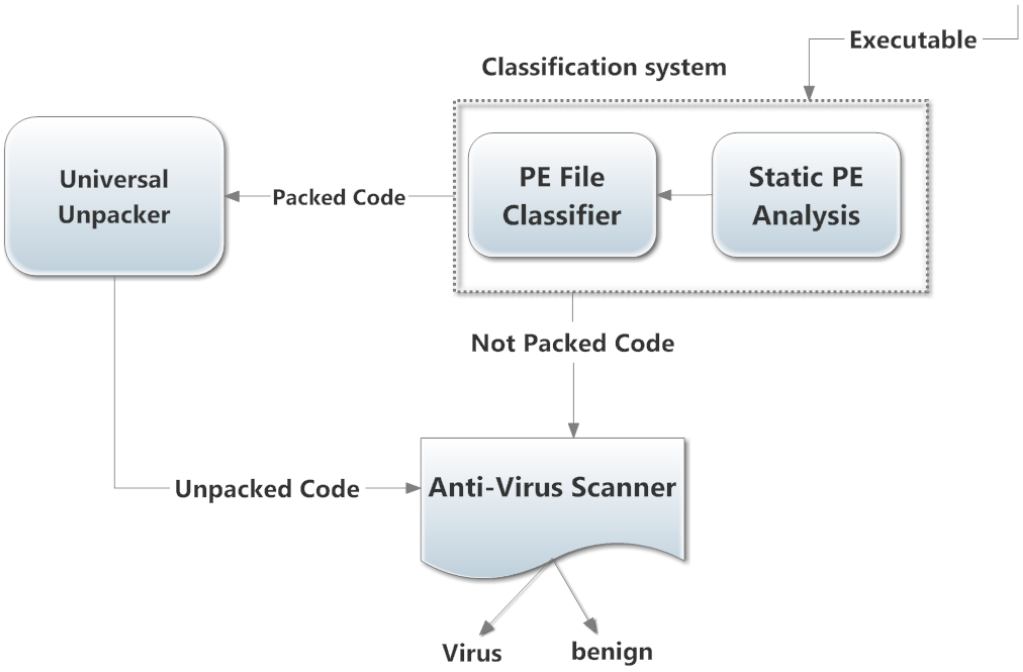


Fig. 3.5 Classification system produced by [92] to distinguish between packed and non-packed executables

This work of classification system extracts a number of features from executable files in PE format through static analysis, which means that they will be able to identify the packed executables without the need of running them. Yet, this technique will only improve the processing time of malware detecting since it will

save time when it distinguish between packed and non-packed executables, and then rely on the unpacker and signature based anti-malware software for detecting malicious code.

Other approach of distinguishing packed from non-packed executables is based on raw binary data which was introduced by [83]. They only used the raw binary information to extract features that can effectively distinguish between packed and unpacked executables without the need of decoding the instructions of the executable. Their algorithm can quickly tell which samples are packed or encrypted [83] and according to that, these packed executables will be sent to the unpacker to unpack them.

Meanwhile, there are no proposed solutions or algorithms for detecting viruses and malwares in packed files without unpacking them first. Results prove that no algorithm can detect packed executables and computer viruses with absolute precision, detection may still be performed with high accuracy [92].

One of the most popular malware detection techniques is the pattern matching algorithm. There is a great demand for high speed and scalable pattern matching algorithms that deal will massive size of databases [143], specifically in the signature-based malware detection approach which we will be adopted and discussed in this section.

Malware Signature

In the malware scanner database, each malware is unique by its signature like a fingerprint for humans; a signature is a sequence of bytes that can be used to identify and detect specific malware by anti-malware scanners [32]. Usually, it's a hash value (a number derived from a string of text) and this unique value indicates the presence of a specific malware (malicious code) and that is what the anti-malware software is designed to detect [45]. The process of identifying and

extracting this unique signature is done through malware researchers in security labs, which first identify and analyse the new malware and then extract and create the signature to add it to the signatures database of the anti-malware software. Therefore, the malware signatures database should be in a constant update phase [106].

The process of malware analysis is a reverse engineering process that will study every new malware by security researchers to identify the threat from that malware, through studying the malware code structure to create a signature specifically for it so the anti-malware scanners can detect that specific malware using that signature. There are several attributes that will help in creating the malware signature that is used in the anti-malware database:

1. Signatures can be generated through hashing parts of the malware code, where then scanners uses the hashing value as the signature to detect that malware. However, since some of the malwares change their code every time to evade malware scanners, the regular hashing technique will not work once the text is changed, therefore, some anti-malware scanners are using fuzzy hashing which generates signatures that will detect and identify files even if they were modified [118].

2. The unique parts in the malware can be identified in different forms, and these parts can help in generating the signature. For example, some malware will have specific steps to perform and the signature corresponding to that unique functions is generated accordingly [11]. Alternatively, there would be a unique text embedded inside the malware, for example, the malware will need to know if the target host was already infected "otherwise the size of an infected file could grow without bounds through repeated infection." [63]. Therefore, malware typically place a signature (such as a string) at a

specific location in the file for this purpose [63]. For instance, the summer of 2001 saw the CodeRed worm which caused possibly billions in damage. The worm ran entirely in memory with no traces of its existence in storage. It worked by exploiting a buffer overflow vulnerability of Microsoft IIS's indexing tool. However, The CodeRed signature below, happen to appears in access logs:

```
GET
  /default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
  NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
  NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090
  %u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u0003%u8b00%u531b%u53f
  f%u0078%u0000%u00=a  HTTP/1.0
```

Fig. 3.6 The CodeRed signature

Or sometimes unique text can be found embedded in the malware such as names of malware authors, special pop-up messages, infected IP address, quotes …etc [45] since malawre authors main purpose sometime is to grab attentions. Another example, is the Alureon, or TDSS, which was a trojan used to intercept network traffic, and obtain sensitive user information. In 2010, it held second place as the most active botnet. Though it originally appeared in 2007, it evolved enough by 2010 to get noticed. Authorities arrested its authors (Russian and Estonian nationals) by 2011. Its designers attempted to evade analysis through obfuscating and encrypting the rootkit body. Interestingly, they also quoted Hamlet in the malware structure. Examine a file fragment containing a Hamlet quote below (Figure 3.7):

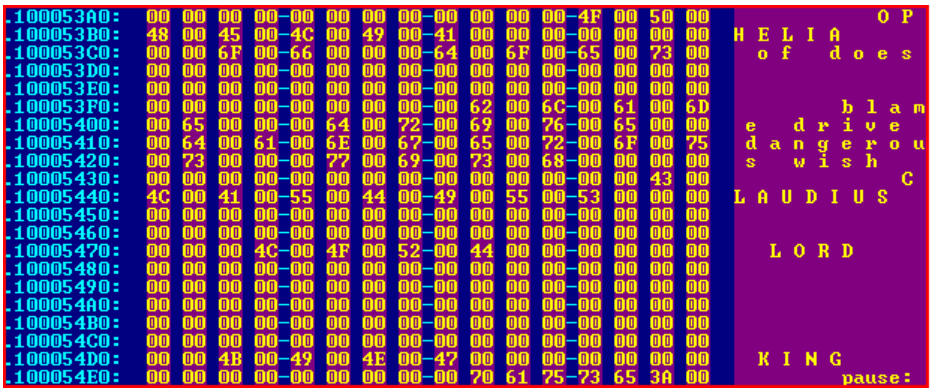


Fig. 3.7 Hamlet words embedded in the malwre code

3. In some cases, the same signature can detect more than one malware since many malwares uses the same code of a previous once or only slightly modify it. They may also preform similar steps as another malware, or generate the malware with the help of virus generation kits and that increases the chances of detecting new malwares (zero-day-malware), since the malware will end up matching a previous malware signature in the anti-malware database [11]. For example, about 27 variants of the virus W32/Bagle have been structured and produced from the original malware code [39].

4. Generic signatures is the wider definition of a malware signature, where security researchers try to identify whole families of malwares instead of one specific malware, e.g. by designing a signature for a code segment shared by multiple pieces of malware [24].

5. According to the CA lab security research report “Every day thousands of new malware variants emerge that rely on common types of previously classified malicious code” [102].

6. Keeping in mind, “The security researches have to choose and create the malware signatures carefully, so that they are not found in innocent files by coincidence" [11].

Finally, the signature-based detection technique is the most reliable and widely used technique over the heuristic detection technique and the behavioral detection technique [24]. The latter two will either examine the malware code after running it or examine the malware behavior also after running the malware. Whereas, the signature based detecting technique will detect the malware using the malware signature before running the malware, and since these signatures are mostly unique, the false-positives are usually rare [24]. However, the challenge in this technique is that is it a time consuming process to extract the malware signature manually on a daily basis. According to the security company "Symantec" Research, traditionally, the malware signatures are created manually, which is slow. Therefore, creating efficient malware signatures has become a major challenge for anti-malware companies where they have to handle the time consuming process and the exponential growth of unique malware signatures databases.

Additionally, some signature-based malware scanners use the "Wildcards" technique in detecting the malware signature and some uses the "Mismatches" technique which is adopted in the proposed solution [11] as the scope of the solution will be dealing with the signatures after they are saved in the anti-malware database, as will be explained in the next sections.

3.3.3 Related Work

Nowadays, the number of virus signatures and the network bandwidth are growing significantly and constantly, thus anti-malware vendors have to work very hard to develop solutions and algorithms that are able to deal with these growing threats. However, researchers have produced a number of solutions to deal with this problem specifically in the pattern matching technique. Thus many pattern matching algorithms have been proposed to solve the problem of intrusion detection system (IDS) [143].

The majority of these algorithms are "Shift based" which are fundamentally relying on the classic single pattern matching algorithm Boyer-Moore algorithm (BM). The core idea of (BM) is to utilize information from the pattern itself to quickly shift the text during searching to reduce number of compares as many as possible. (BM) introduces a bad character heuristic to effectively capture such information [17].

For example, "Clam-AV" is one of the anti-virus pattern matching solutions which has been used widely in UNIX platforms lately [25]. It has been implemented in an extended version of BM (BMEXT) as a core pattern matching algorithm for scanning basic signatures along with other algorithms AC [2]. The down side of this algorithm is that its performance will decrease whenever the number of signatures increases. Another anti-virus pattern matching solution known as (MRSI: A Fast Pattern Matching Algorithm for Anti-virus Applications) introduced in [143] is used to improve the previous solution, after analysing the different types of signatures (Basic, MD5, Regular Expression) and few other signatures types [as shown in Table 3.1].

	Basic	*MD5*	*Regex*	*Other*	*Total*
Time (in Seconds)	78501	64758	4844	233	148270
Percentage %	52.9%	43.7%	3.3%	0.16%	100%

Table 3.1 Different types of signatures that has been analysed by MRSI

And after studying the time processing for each signature type, [143] decided to concentrate their work on matching the (basic) signatures since it is the most popular one and the most time consuming in order to improve the virus scanning speed on Clam-AV. (results in Table 3.2) therefore, they managed to achieve an 80% 100% faster virus scanning speed.

	Basic	*MD5*	*Regex*	*Total*
Time (in Seconds)	6.200	0.054	2.190	8.444
Percentage %	73.4%	0.64%	25.9%	100%

Table 3.2 The processing time for each signature in Clam-AV

In this section we will study the similarities between the malware detection problem and the biological molecules sequencing providing the possibilities of using short reads aligning algorithm REAL in the malware signatures-based detection problem.

Currently, human genome sequence mapping has been completed. Typical applications of bioinformatics are: searching one or a set of gene occurrence in a gene sequence, to compare similarity relationship; or matching unknown protein sequence according to known protein sample. As the protein and gene could be represented as sets of strings, traditional pattern matching technology could be used to solve such matching problems in the malware detection area [37].

3.3.4 Algorithm Preliminaries

Let Σ be a finite alphabet which consists of a set of characters (or symbols). The cardinality of an alphabet, denoted by $|\Sigma|$. The set of all non-empty strings over the alphabet Σ is denoted by Σ^+. The empty string is the empty sequence (of zero length) and is denoted by ε; we write $\Sigma^* = \Sigma^+ \cup \varepsilon$. A string is a sequence of zero or more characters (or symbols) in an alphabet Σ.

The rest of the algorithm preliminaries were defined clearly in Chapter 2.

3.3.5 Real Overview

REAL is a read aligner, which addresses the problem of efficiently mapping the reads (reads are defined in chapter 2) $p_1, p_2, ...p_r$ to a reference genome t with at most k-mismatches. In order for the procedure to be efficient, they made use of word-level parallelism by transforming each factor of length ℓ of t into a *signature* (as shown, in table 3.4) by transferring the reads to its binary equivalent and then computing the decimal value.

In addition, the idea of using the pigeonhole principle to split each read into v fragments is adopted. The general idea for the k-mismatches problem is that inside any match of the pattern of length m, with at most k errors, there must be at least $m-k$ letters belonging to the pattern [42]. Therefore, by fragmenting the pattern into number of fragments it will then require $v-k$ of the fragments (instead of all of them) to be perfectly matched on t, the non-candidates can be filtered out very quickly.

For example, if we split the pattern into 4 fragments and the number of allowed mismatch is 2,

Then at least $(4-2) = 2$ fragments will perfectly match with the text, since the two mismatches can exist in at most two of the fragments (at the same time) as shown in the image next in Figure 3.8.

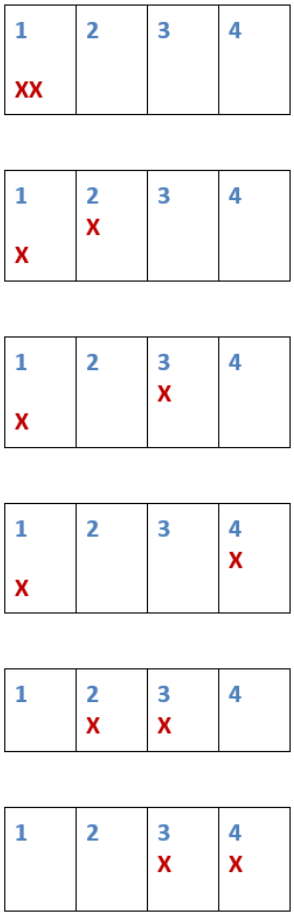


Fig. 3.8 Example where the mismatch can exist in the fragments

By trying the six combinations of the two possible fragments as the seed, it can catch all hits with two mismatches, for example (if the fragments 1 and 2 matches exactly, the the remaining fragments 3 and 4 will have the possible allowed mismatches) as shown in the figure next (The possible six combinations, Figure 3.9).

Matches Exactly	
1	2
1	3
1	4
2	3
2	4
3	4

Possible Mismatches	
3	4
2	4
2	3
1	4
1	3
1	2

Fig. 3.9 (The possible six combinations

Therefore, by combining the word-level parallelism technique to create fragment signatures and then create lists for these signatures depending on the pigeonhole principle to help in filtering out the none candidates fragments, and then sorting these lists according to the signature value will help in performing binary search to find exiting values which will mean the pattern occurs in the text with the most k-mismatch allowed. an outline of the REAL algorithm will be explained next.

An outline of the REAL algorithm is as follows:

Step 1.

The firs step is to partition the text (the reference genome) t into a set of substrings (factors) $z_1, z_2..z_{n\ell+1}$ where $z_i = t[i...i+\ell-1]$, for all $1 \leqslant i \leqslant \ell+1$, and then compute the signature for each factor $\sigma\ (z_i)$ by transforming the factor to its binary equivalent using 2-bits-per-base encoding of the DNA alphabet (as shown in Table 3.3):

A	00
C	01
G	10
T	11

Table 3.3 Binary Encoding of DNA Alphabet

And then storing its decimal value as shown below in Table 3.4 in a list the positions.

Sting x	A	G	C	A	T
Binary Form	00	10	01	00	11
Signature σ	147				

Table 3.4 Signature of String x= AGCTA [10]

And then fragment the factors to compute the signature for each fragment and store them in lists.

Step 2.

The second step is to sort the signatures list using radix-sort algorithm (defined in Chapter 2).

Step 3.

The third step is to compute the fragments and signatures for the patterns (reads)

and then perform the binary search operation to return wither the patten occurs in t, if the exact signature occurs then the matching can be extended to the remaining fragments to match with at most k-mismatches.

REAL algorithm has been presented recently to the gene and DNA sequencing area and so far it performed very well with positive results. It is worthy of trying to use this pattern matching algorithm in signature-based pattern matching for anti-malware and malware detection area.

Full details on how the REAL algorithm work can be found in [42].

3.3.6 Problem Definition

This work considers two areas, signature-based malware detection and bioinformatics sequencing pattern matching. REAL algorithm could solve the problem described as:
The problem of mapping tens of millions of short sequences to a reference genome as follows:

Find whether the pattern $\rho_i = \rho_i[1...\ell]$, for all $1 \leq i \leq r$, with $\rho_i \in \sum^*$, $\sum = \{A,C,G,T\}$, occurs with at most k-mismatches in $t = t[1..n]$, with $t \in \sum^*$

In particular, we are interested in reporting a pattern, for all $1 \leq i \leq r$, in a case that occurs with the least possible number of allowed mismatches, exactly once in t [42],[10].
The malware detection problem is defined as follows:
Problem 1. Given a set of patterns $\{\rho_1, \rho_2, ..., \rho_r\}$ of length ℓ, with $\rho_i \in \sum^*$, $\sum$ is a bounded alphabet, and an integer threshold $h > 0$, find whether ρ_i, for all $1 \leqslant i \leqslant r$, occurs in text t of length n and/or in text $\hat{t}$, where $t, \hat{t} \in \sum^*$ and $\delta_E(t,\hat{t}) \leqslant h$.

Similarity between two problems:

Both Problems are dealing with massive amounts of data that need to be processed everyday with as minimum space, time and cost as possible. The DNA sequencing technologies are improving by the clock and producing tens of millions of short sequences (reads) in a single experiment that needs to be mapped back to reference sequences. Likewise, according to recent security reports more than 317 million new pieces of malware were created in 2014 only. That means around 1 million new threats were released each day that the malware scanners have to cope with and detect everyday (new malware signatures).

3.3.7 The Experiment

The experiment was done with different types of files including Portable Executables (".exe" and ".dll"), email files, Graphics (".jpg" and ".gif"), OLE2 component (eg: VBA script), normalized Web files (HTML, PHP, Jave Script) and normalized ASCII text file.

Generating signatures is out of the problems' scope, since as mentioned before creating malwre signatures will require an intensive analysis of the malware structure and code. In fact, the proposed solution is focusing on detecting the malwre using signatures already saved in the database. However, in order to test the solution two different types of signatures MD5 hash and body-based signature were generated, either by using Sigtool, a tool for generating MD5 hash or body-based signature which is a tool we developed as a signature generator component in Microsoft C# programming language.

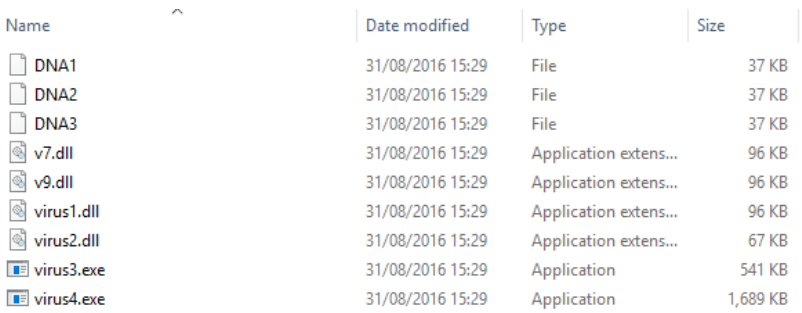


Fig. 3.10 Example of some of the generated infected files

The following steps are for generating the signatures given an infected file

Step 1: For body-based signature we started by loading the infected file content as byte array to the memory (for larger file we only read small segments of the file (2KB, 2048 Bytes), the selected segment could be taken arbitrary from any part of the file. For MD5 hash we added an extra step to the process by passing the byte array extracted from the file to the MD5 hash generator function. Note that

different signatures can be created from different parts of the infected file body by selecting different offsets, (the beginning, middle or the end of the file) finding informative areas in the file body will improve the detection process.

Step 2: Convert the selected byte array to Hexadecimal signature and write the output to the virus signature library file

For example to create a body-based signature for the file "program.exe" using the CALMAV signature tool.

$$root@localhost : /tmp/\$sigtool\ program.exe > test.hdb \tag{3.1}$$

To create MD5 hash signature use the "–md5" option of sigtool as follow:

$$root@localhost : /tmp/\$sigtool\ --hex-dump$$
$$dumpprogram.exe > test.hdb (7.2)$$

The virus library contains list of the signatures stored one signature per line, as shown in table 2 .

	Malware-name	Offset	Seg-length	Sig-type	Hex-Sig
0	90				
1	Trojan-428	0	2048	BBS	de 76 2f f0 ...
2	Worm.Mydoom.I	0	2048	BBS	de 76 cd 51 ...
3	HTML.Phish.B	0	2048	MD5	9 77 9d ef ...
	Win.spam.ts	0	2048	BBS	40 00 68 c4 ...
	...				
	...				
90	Heuristics.Safe	0	2048	BBS	40 00 68 c4 ...

Table 3.5 The Structure of the Virus library File

The first line in the file contains the total number of signature in the library

As a first testing stage, the viruses were distributed in one folder with 93 different files, total folder size 207MB, the scanning time was 1 second detecting all the embedded viruses as shown in the figure below (Figure 3.11).

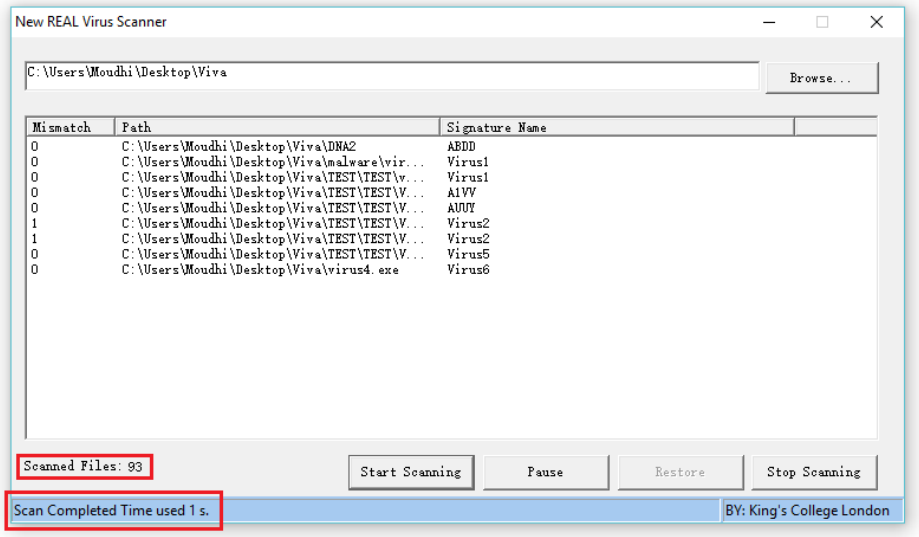


Fig. 3.11 Detecting the infected files with high accuracy and speed A

Then, the viruses were distributed in a larger folder with 1039 different files, total folder size 819MB, the scanning time was 11 seconds) as shown in the figure below (Figure 3.12).

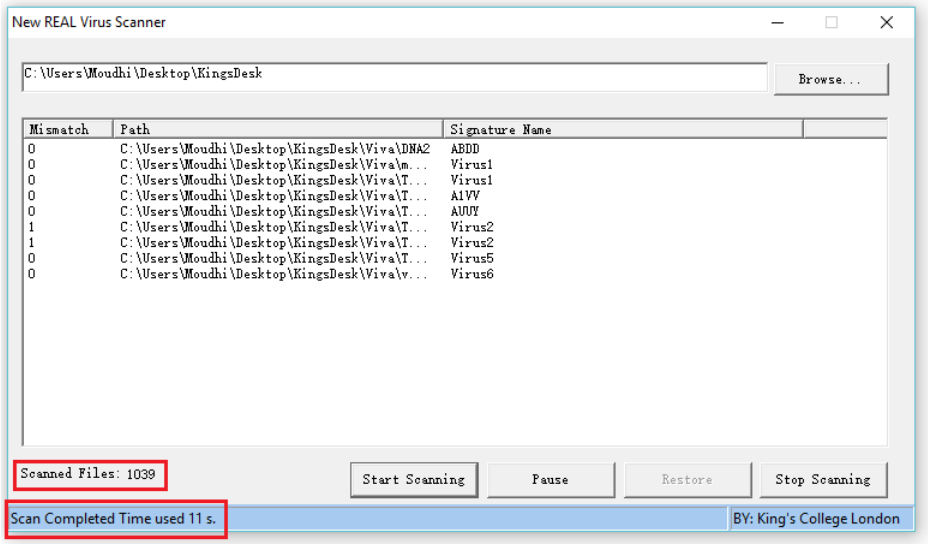


Fig. 3.12 Detecting the infected files with high accuracy and speed B

Detection Accuracy

The detection accuracy of the proposed solution according to the experiment is very efficient and promising, in fact, after testing the solution on number and types of different datasets with different number of allowed mismatches, the solution was able to detect all the infected files embedded in the datasets with high accuracy, as shown in Table 3.6.

File Type	Signature format	Dataset			
		Dataset Size	Number of mismatches	infected files	detected files
Portable Executable	MD5	50	0	30	30
	Body-based	100	1	65	65
	MD5	50	0	30	30
	Body-based	100	2	60	60
Mail File	MD5	50	0	30	30
	Body-based	100	1	65	65
	MD5	50	0	30	30
	Body-based	100	2	60	60
Graphics	MD5	50	0	30	30
	Body-based	100	1	65	65
	MD5	50	0	30	30
	Body-based	100	2	60	60
OLE2 component	MD5	50	0	30	30
	Body-based	100	1	65	65
	MD5	50	0	30	30
	Body-based	100	2	60	60
HTML	MD5	50	0	30	30
	Body-based	100	1	65	65
	MD5	50	0	30	30
	Body-based	100	2	60	60

Table 3.6 The accuracy Experiment Results

Detection Performance Evaluation

The proposed solution was tested against the previous two algorithms that were mentioned in the related work section 3.3.3, the first algorithm is the BMEXT which is an extended version of BM which is a fast string searching algorithm [17], and the second algorithm is the MRSI (Multi-block Recursive Shift Indexing) algorithm [143], both algorithms were implemented in Clam-AV [25] to evaluate their performance. The time cost in the scanning stage was recorded then translated to the scanning speed in Mbps (Megabits per second). Two datasets were used in the experiment, the first dataset contain randomly generated files of 100 Mbytes and the second dataset was a several executable files of 80 Mbytes in total.

As shown in Figure 3.13 the REAL detection tool outperform both algorithms in both datasets.

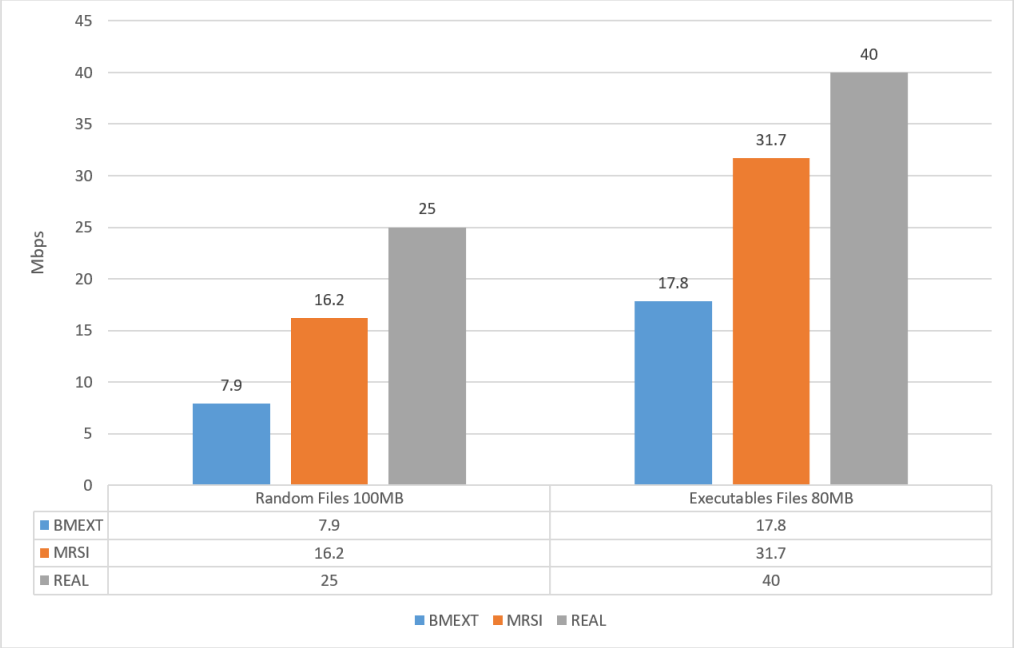


Fig. 3.13 The detection performance evaluation

3.3.8 Discussion

We have introduced a novel way for detecting malware code embedded in different types of computer files, using bioinformatics tool, namely REAL (short read aligner for next generation sequencing), which uses approximate string matching. One of the benefits of this approach is that REAL is implemented in such a way that it does not necessarily load the whole file in memory. Instead, it loads blocks of the file depending on the physical memory of the individual machine. Concerning the storage used for indexing, no additional hard disk space is necessary for REAL, as it does not store an index of the file data. The presented experimental results are very promising, in terms of efficiency and sensitivity on the detection process.

As it is shown by the results in [Table 3.6], REAL showed a consistency and accuracy in the detection process and it was able to complete the assignment much faster, despite not using a stored preprocessed index of the scanned files. REAL outperform classic pattern matching such as Knuth-Morris-Pratt and Boyer-Moore

[42].

The proposed future work is to focus on two parts. First of all, using some heuristic algorithm for optimizing segmentation and selection of the signature region, utilized very well (e.g. level 8 memories). Hence, designing additional algorithms to further optimize the memory cost.

Second, provide support for signature based container meta-data by allowing matching for signatures in files stored inside different container types such as compressed and encrypted files. In Addition, Future work will focus on studying the capability of perm-term analysis instead of segmentation, experiments with different and larger malware collections, and a combination of this technique with machine learning analysis of malicious code.

3.4 Detection of URL in Image Steganography

3.4.1 Introduction

The word steganography is derived from the Greek words (stegos) meaning (cover) and (grafia) meaning (writing) [49]. However, Steganography and Cryptography considered to be complementing each other rather than replacing each other. Cryptography is the art of scrambling messages to make it difficult to understand, whereas, Steganography is the art of hiding it to make it difficult to find.

Therefore, steganography is an extra layer that will support transferring secret information in a secure way (secrete communication) whereas, cryptography in this case is data protection. Besides, when steganography fails and the message can be detected, it is still of no use as it is encrypted using cryptography techniques [49].

Moreover, Steganographic techniques started ages ago back to ancient Greece. Starting by writing text on wax-covered tablets to shaving the head of a messenger and tattoo a message or image on the messenger's head. And when the hair grow back, he will be sent to the receiver where the message will be undetected until the head is shaved again [61].

Since then, the science of steganography has developed significantly to more sophisticated techniques far more than their ancient predecessors, allowing a user to hide large amount of information within image, audio files and even networks. In fact, in reality the main difference between the modern steganographic techniques and the previous once is only the form of (carrier) for the secret information. For instance, instead of using human skin and wooden tables they use media files like images and audio.

Although, within the daily discovery of a message hidden with an existing application, a new steganographic applications are being devised. And an old methods are

given new twists [61]. Therefore, there are so are many types of steganography methods to hide secret data wither with new carriers, new hiding techniques or new type of secret data.

3.4.2 Motivation

Steganography is the science of hiding data within data, either for the purpose of secret communication or leaking sensitive confidential data, or even embedding malicious code or malicious URL. Many different carrier file formats can be used to hide this data (network, audio, image.. etc) but the most common steganography carrier is through embedding secret data within images, as it is considered to be the best and easiest way to hide all types of files (secret files) within an image using different formats (another image, text, video, virus ,URL..etc).

To the human eye, the changes in the image appearance with the hidden data can be imperceptible. In fact, images can be a lot more than what we see with our eyes. Therefore, many solutions were proposed to help in detecting this hidden data, with each solution having strong and weak points either by the limitation of resolving one type of image along with specific hiding technique or most likely without extracting the hidden data.

The detecting process is called “Steganalysis” and most of the current steganalysis techniques (if the original image that was used to hide the message was unknown to them) they can only suspect the presence of a hidden message as it is very difficult to either detect or extract the hidden data if they don’t have the original image to compare to.

Recently, hiding URL in images are becoming more popular because embedding URL in an image instead of the whole secret text will occupy much less space in the carrier [32] and that will help in hiding it, and prevent the chance of it being

corrupted by image manipulations. These URLs can belong to malware that will harm the image's receiver, or they can also be used for leaking secret information or spying.

Since hiding URLs in images is a relatively new technique and an open problem in the security and privacy field, the first part of this section proposes a first attempt of a novel detection approach that will concentrate on detecting any kind of hidden URL in most images and extract the hidden URL as it is from the carrier image that used the LSB least significant bit hiding technique in a very fast and accurate way. The novelty of this approach is that it can extract the hidden text by depending only on the hidden technique and the message format.

The second part of this section builds on the previous approach through detecting hidden and encrypted URL, and extracts the URL as it is, in a very accurate and fast technique.

The rest of this section is structured as follows. In subsection 3.4.3 and subsection 3.4.4, the basic steganography concept and applications are presented. In subsection 3.4.5 and in subsection 3.4.6, defining the image steganography and briefly review some of the current image steganography techniques which is the focus of this section. An overview of the steganalysis is in subsection 3.4.7. In subsection 3.4.8, formally defining the solved problem. Subsection 3.4.9, the URL detection algorithm overview with full analysis of the time and space complexity for each step. Then an improvement of the solution is presented in 3.4.10 to detect and extract encrypted ULR followed by and experiment results in 3.4.11. Finally, briefly concluding the section with some discussion and future proposals in subsection 3.4.13.

3.4.3 The Concept of Steganography

The concept of steganography is to embed data, which is to be hidden, however, this process will require three files:

First, is the secret message which is the information to be hidden and as mentioned before with the new steganography techniques almost any kind of data can be hidden.

Second, is the cover file (carrier) that will hold the hidden information and as well, almost any kind of files can be used as a carrier.

Finally, is the key file to find the hidden message and extract it from the cover file (if needed), the result of these three files is a file called (Stego File) as shown in Figure 3.14.

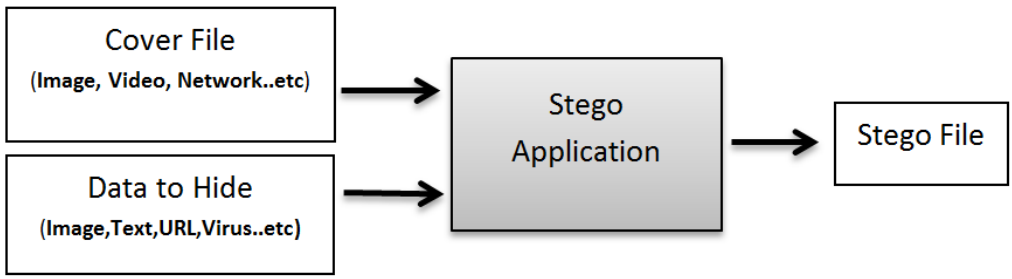


Fig. 3.14 Stego application Scenario

However, the most common steganography technique is embedding messages within images as its considered to be the best carrier (Cover File) to hide all types of files within it. For example, hiding (another image, Virus, URL, text, exe file, audio..etc) without changing its visible properties [98].

For this reason, many companies are finding it difficult to detect the stego files specially images even after scanning all their employees out going emails.

3.4.4 Steganography Applications

Steganography can be used for good intentions in many useful ways for example to help in transferring secret data, copy rights control of materials and smart IDs (identity cards) where individuals' details are embedded in their photographs [20]. Also, it can be used in printed images where the data will be embedded before printing, and after printing the user can scan the printed image with a smart device and the embedded information will appear on the device, this can be useful in so many fields especially exhibitions and as a marketing tool to display the products information.

Nevertheless, hospitals are using Steganography also to keep their patient's confidential data such as DNA sequences in a secret safe place where the access to it is highly restricted unlike other data.

On the other hand, like any other science, cyber-crime is believed to benefit from it in transferring illegal data or embedding viruses and malicious URLs in carries and other harmful actions. Therefore, due to the rapid development of steganography methods and techniques the steganography research area has gained so much attention during the last decade.

Besides that, nowadays any person without any technical background can do harmful cyper-crime actions with the support of the cyper-crime ready made sophisticated softwares which are available online to everyone with no cost and easy to use, for example, there are alot of steganography tools for instance, Xiao Steganography [115] which any user can use to leak his/her company's confidential information with basically 3 clicks (selecting the cover image, selecting the secret file or typing the secret message, and finally clicking on the embedding button) and the software will do the rest.

3.4.5 Image Steganography

Using an image as a cover file is considered to be one of the most useful and cost effective technique [79], all the image steganographic techniques to hide data based on the structure of the most commonly used images format on the internet (GIF-graphics interchange format), (JPEG-joint photographic expert group), (PNG-portable network groups)and (BMP- Bit Map Picture).
The image steganography process contain three files as follows:

- ***Cover Image***: In steganography, the original image that was chosen as a carrier for the secret data is called a cover image.
- ***Stego Image***: Is the result image of choosing the right cover image and embedding the secret data inside it.
- ***Stego Key***: The sender should have an algorithm for creating the stego image to embed the data, and the receiver should have the matching algorithm to extract the hidden data from that particular stego image and sometimes they use a key to extract the secret message which is called a stego key.

To Formally Define the Image Steganography Embedding Process:

Let C be the chosen *Cover Image*, and C' is the *Stego Image*, the *Stego Key* will be denoted as K (if needed), and the hidden message as M and the hidden technique will be denoted as (+) then:

$C + M + K \rightarrow C'$

as shown in Figure 3.15.

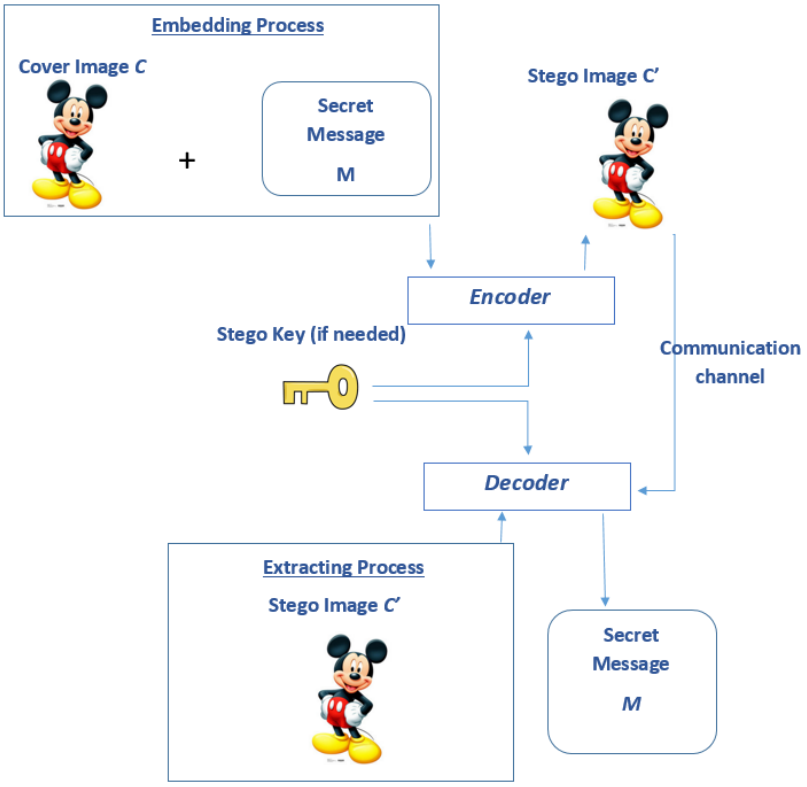


Fig. 3.15 Image Steganography Embedding process [20]

However, the main challenge in image Steganography is that many image manipulation techniques might destroy the hidden message on any image, since it will change the feature of the (stego-image) it might as well change the feature of the hidden message inside it, such as cropping might crop the hidden message if it was long and located in one section of the image. Or corrupt the message, where rotation might give the receiver difficulty in finding the hidden message, filtering might destroy the hidden message completely and so on.

3.4.6 Current Image Steganography Techniques

There are some naive implementation of image steganography for example by feeding windows OS command some code to embed the text file which contain the secret message into a specific image and produce the stego-Image.

```
C:> Copy Cover.jpg /b + Message.txt /b Stego.jpg
```

Fig. 3.16 Stegocode

However, when displaying the image structure the message reveals itself and also will not survive any kind of image manipulation [20].

Steganography embedding techniques can be divided into two groups either Spatial Domain also known as Image Domain which embed directly the secret data in the intensity of the image pixels, usually in the least significant bit (LSB) in the image, or the Transform Domain which is also known as Frequency Domain, where images are first transformed and then the secret data is embedded in the image [81]. However, there are more two newly introduced techniques which are the Adaptive Method and the Vector Quantisation (VQ-based) method.
According to (A Review on current Methods and application of Digital image Steganography 2015) which presented and studied the major steganography algorithms between the year of 2010 and 2014 stated that the Spatial Domain method is more popular and mostly used by more number of stego authors in comparison to the Frequency Domain (as shown in Figure 3.17). Yet, there is a lot of scope and opportunities to develop more and effective methods for steganography [79].

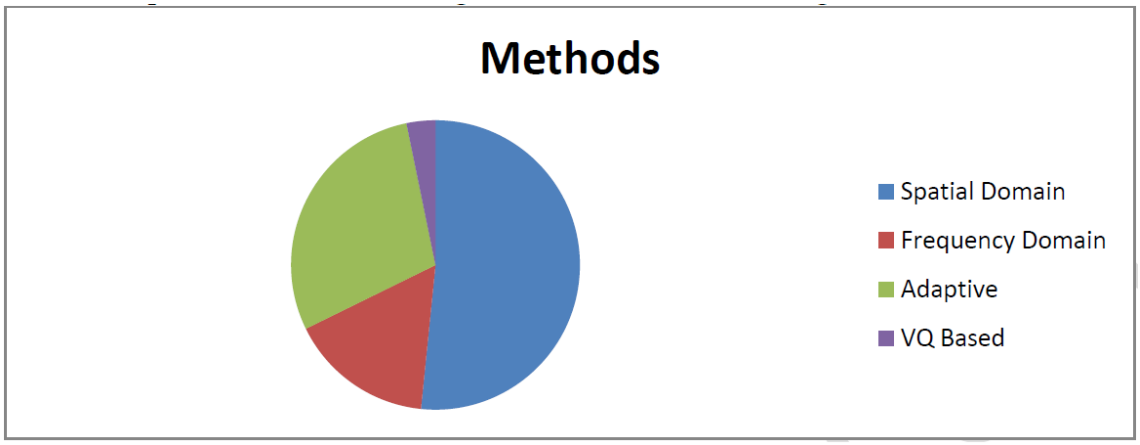


Fig. 3.17 Mostly used Steganography Methods

The focus of this section will be on Spatial Domain the most popular technique. Moreover, in spatial domain, the steganographer modifies the secret data and the cover image, which involves usually re-encoding the least significance bits (LSBs) in the carrier image. To the human eye these changes in the image value of the least significant bits (LSB) are imperceptible [19] the strength of the presented detection approach is that it will preform on most types of images and will extract the hidden URL as it is.

Least Significant Bits (LSB) Hiding Technique

Its the steganography technique of embedding data at the least significant bits (LSB) in the cover image. It is considered to be one of the simplest techniques of embedding data in a cover image. Yet, it's one of the most difficult techniques to be beaten. This technique embeds the bits of the secret data directly into the least significant bit plane of the cover image [20].

The changes will be only made on three bits, on average, only half of the bits in an image will need to be modified to hide a secret data using the maximal cover size. In fact, the result of these changes are too small to be recognized by the human visual system (HVS), so the message is effectively hidden [49].

This technique will be the focus of the presented detection approach since its widely used and on all most of images.

3.4.7 Stegaanalysis

Steganalysis is the main step in the steganography detecting technique to discover the hidden messages. It's the way of identifying the suspected medium, determine whether or not they have an embedded data into it, and, if possible, recovering that data. in other words 'Steganalysis is the science of attacking steganography in a battle that never ends' [20].
Steganlysis can be challenging sometimes more than cryptanalysis, to explain, the steganalyst have first to identify the suspected cover file, then locate the hidden message where sometimes it can be scattered into more than one place in the cover file, finally, most likely the secret message will be encrypted as an extra level of hiding it. Whereas, the cryptanalyst main mission usually will be to decrypt the encrypted message if the encrypted text was provided.

Stegnalysis can be done according to different attacks:

1. **If the steganography attack is known to the steganalyst**: the cover file, the hidden message and the steganography tool (algorithm) are all known to the steganalyst, then its shouldn't be difficult to locate and identify the hidden message.

2. **Only the original file (before embedding the message) and the cover file (after embedding the message) are known to steganalyst**: then the mission will be to compare the two files and identify the hidden message according to the pattern differences that are detected between the two files.

3. **If the secret message only is known to the steganalyst**: then the mission will be looking for a known pattern in all the files, and that might be difficult to achieve if the hidden technique is unknown to the steganalyst.

4. **Only the cover file is known to the steganalyst**: similarly to the previous point it will be challenging to identify the hidden message location since it may be scattered to more than one place or to understand it, since it might be encrypted.

However, image processing are usually the main technique used in building steganalysis programs to study different image manipulations such as translating, filtering, cropping and rotation. Also by examining the cover image structure for first order statistics (histograms) or second order statistics (correlations between pixels, distance, direction). JPEG double compression and the distribution of DCT (discrete cosine transform) coefficients can give hints on the use of DCT-based image steganography [20].

The focus of this section will be under a new kind of attacks where the type of the hidden message is known (URL) and the used technique in hiding it is the (LSB) the least significant bit is know as well in most types and Images.

URL in Image Steganography

Embedding data in images is not a new technique. However, improving this method in image steganography is getting better and more sophisticated by the day, one of these recent improvement is embedding a URL (Uniform Resource Locator) in the image least significant bits either to direct the receiver to a web page that include the secret data. Or the embedded URL can belong to a virus that will harm the image receiver either by destroying data or stealing data. The main reason behind embedding URL in an image instead of the whole secret data is that

the URL will occupied much less space in the carrier [36] and that will help in hiding it and prevent the chance of being corrupted by the image manipulations (as shown in Figure 3.18).

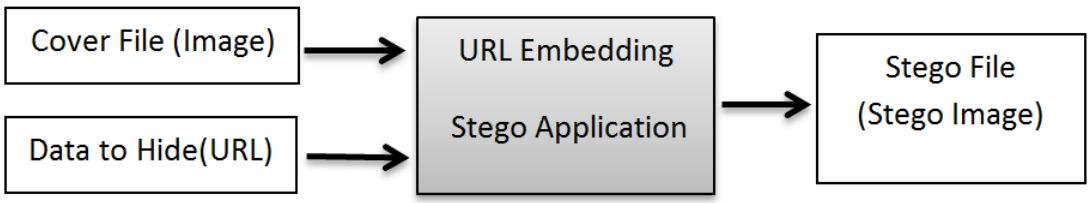


Fig. 3.18 URL Stego Embedding Scenario

Regarding viruses, as mentioned before, Dell SecureWorks Counter Threat Unit™ published on june 2015 a full analysis of the (Stegoloader malware) which is a stealthy information stealer. And they explained

> Malware authors are evolving their techniques to evade network and host-based detection mechanisms. Stegoloader could represent an emerging trend in malware by the use of digital image steganography to hide malicious code [33].

In fact, Stegoloader has a modular design and uses digital steganography to hide its main module's code inside a Portable Network Graphics (PNG) image downloaded from a legitimate website [33].

Furthermore, another analysis by the same source Dell SecureWorks on another malware (Lurk Downloader) whereas, this malware specifically embed a URLs into an image file by inconspicuously manipulating individual pixels , the explained:

> The resulting image contains additional data that is virtually invisible to an observer. Lurk's primary purpose is to download and execute secondary malware payloads [117].

Hidden URL in an image can be for the good cause of secret communication but unfortunately most likely that is not the case in the past 2 years since viruses started using this technique. In fact, hidden URLs now are used by malware authors to download an extra modular as shown in the previous example where the Lurk malware adopted the hidden URL in image technique in 2014, specifically it uses an algorithm that can embed URLs into an image file so the stego image contains additional data that virtually invisible to the human eye. the malwre then downloads and execute secondary payloads using the hidden URL [117]. However, the proposed solution in this section will be able to detect any URL regardless of it's intention.

3.4.8 The Problem

In this section, we are dealing with (URLs) hidden inside images. As described previously, for example, any malicious code can be embedded using least significant bit (LSB) technique. Modifying LSB means modifying the colour by changing the least significant bits of an image. We have different type of colour formats such as 8 bits, 24 bits..etc. They have both colour and grey scale.
8 bits colour means that each pixel can have any of 256 (2^8) colour. The same calculation is applicable on the 8 bits grey scale or 24 bit colors. So, modifying the least significant bit in an array of huge combination of colors does not make much difference in human eye. This makes the use of the least significant bits (LSB) URL attack very effective and undetectable.

For example: an url http://exampleattack.com has got 24 characters, each character on this url takes 8 bits in ASCII format, therefore, the URL will require 192 significant bits from an image.

For the simplicity of example, lets see how first character '*h*' of our example url http://exampleattack.com can be added by using least significant bits of an image.

The ASCII value for '*h*' is decimal 104 and binary 01101000
Before the least significant bits (LSB) insertion lets assume that 8 consecutive bytes of an images is below.
10000010 10100110 11110101 10110101
10110011 10010111 10000100 10110001

After inserting '*h*' (01101000) in the least significant bits the result is below.
1000001**0** 1010011**1** 1111010**1** 1011010**0**
1011001**1** 1001011**0** 1000010**0** 1011000**0**

In this way by using more least significant bit of images we can embed the rest of the characters of the intended URL.

3.4.9 URL detection Algorithm

This subsection will present the proposed detection algorithm overview, and detailed algorithm in pseocode to detect a hidden URL from the least significant bits of an image, and the detailed complexity analyses are done as well.

Algorithm Overview

- **Step 1:**

  Create a sorted list, DOMAIN[], from the static official top level domain list web site (as shown part of it in Table 3.7).

- **Step 2:**

  Create an array calleded BITMAP[], from an image taking each bit in array.

- **Step 3:**

  Make a character array called, LSBCHARARRAY[] from an Intermediate array of LSBARRAY[] by converting each 8 bits to an ASCII character.

- **Step 4:**

  Loop through the LSBCHARARRAY[], find out possible hidden url forming by http or https, www, domain name and top level domain (TLD).

Table 3.7 List of Top-Level Domains by the ICANN - for full list please refer to [52]

AAA	**AARP**	**ABB**	**ABBOTT**	**ABOGADO**
AC	**ACADEMY**	**ACCENTURE**	**ACCOUNTANT**	**ACCOUNTANTS**
ACO	**ACTIVE**	**ACTOR**	**AD**	**ADS**
ADULT	**AE**	**AEG**	**AERO**	**AF**
AFL	**AG**	**AGENCY**	**AI**	**AIG**
AIRFORCE	**AIRTEL**	**AL**	**ALLFINANZ**	**ALSACE**
AM	**AMICA**	**AMSTERDAM**	**ANALYTICS**	**ANDROID**
AO	**APARTMENTS**	**APP**	**APPLE**	**AQ**
AQUARELLE	**AR**	**ARAMCO**	**ARCHI**	**ARMY**
ARPA	**ARTE**	**AS**	**ASIA**	**ASSOCIATES**
AT	**ATTORNEY**	**AU**	**AUCTION**	**AUDI**
AUDIO	**AUTHOR**	**AUTO**	**AUTOS**	**AW**
AX	**AXA**	**AZ**	**AZURE**	**..etc**

The Algorithm in Pseudocode

The 4 steps given previously are presented in the appendix of this thesis with detail pseudocode so that users can convert any programming language easily with little efforts.

Space and Time Complexity analyses:

Step 1 (Create a sorted list from the static official top level domain):

Space complexity: We have a known List of the Top-Level Domains (TLD) (as shown in Table 3.7) from [52]. So, in preprocessing stage, we create an indexed array, DOMIAN[] considering each TLD as a string. Space complexity is linear to the size of all characters plus the index of each string position in a sorted order. Also, we create a separate index list with just starting position of TLDs with a specific character. For example, if .co and .com both starts with c, so if we know where the 'c' starts in the whole sorted list, we just can look in the block starts with 'c'. The overall space complexity for the sorted list is $O(m) + O(t) + O(i)$ where m is the total number of characters, t is the index on each TLD string which is limited to the official static list.

Time complexity: This can be the step of computation of preprocessing, so complexity is not a major issue. However, it is possible to build up sorted list by radix sort [109] where the LSD radix sort operates in $O(nk)$ in all cases, where n is the number of keys, and k is the average key length.

Step 2 (Create a sorted list from the static official top level domain list):

Space complexity: $O(m)$ where m is the number of bits.

Time complexity: $O(n)$ where n is the number of bits. This means in just a single iteration the array is built.

Step 3 (Make a character array by converting each 8 bits to an ASCII character):

Space complexity: The complexity is $O(n)$ here where $n = m/8$ and m is the number of bits in BitMap and only one in each 8 bits are placed in character array by converting 8 such Least Significant bits into character. So, the complexity here is sub linear. Although an intermediate LSBARRAY has been introduced in Step 3 for clarity purpose of the flow, it is possible to calculate the LSBCHARARRAY directly from BITMAP[] array. So LSBARRAY[] is not required in implementation.

Time complexity: This is looping through the BitMap array just once and producing character array by taking each 8 significant bit together and converting to ASCII. So, the time complexity is linear here with $O(n)$ where looping n bits just once produce the result.

Converting to ASCII and character is happened just 1 in 1/64 where 1 byte (8 consecutive LSB) comes from 64 bits. This operation produces time complexity of $O(n+n/64)$ which is linear.

Step 4 (Loop through the array, find out possible hidden url forming by http or https, www, domain name and top level domain(TLD))

Space complexity: The space complexity holds the linearity here with $O(n)$ where n is the number of characters in the array.

Time complexity: This is a loop through the characters array. Finding first 3 parts of an URL (http/https and/or www, domaion name) are done in one go in the single loop. There are nested loops used to find the position and for the calculation purpose for http, https and www. The actual counter of characters array is incremented in each go whether it is inner loop or outer loop. The complexity holds linear for the operations because the whole characters array are traversed just

once. Looking up the 4th part, Top Level Domain (TLD) requires a short lookup in a sorted array described in Step 1. For the whole character array, this lookup is just done to complete the search in a sorted and indexed Top Level Domain array which we called in step 1 as DOMAIN[]. In a sorted list, the binary search works as $log(n)$ complexity in worst case where n is the number of items in an array. But in our case, n is narrowed down by index of each character. So, each block of searched area is n/m where m is the number characters in the alphabet. So, the search takes $log(n/m)$ time complexity because we know the starting character to lookup in DOMAIN[] array. The overall complexity stays linear for step 4.

3.4.10 Next Level Detection (Detecting and Extracting Encrypted URL)

The previous tool can be considered as one of the first tools to detect the hidden text in images and extract these hidden messages as it is, therefore, we considered taking it to the next level and improve the detection algorithm to detect and extract encrypted URLs.

Assuming that the sender encrypted the URL using the (NOT) encryption technique (as will be explained next) which will help in encrypting and hiding the plain text at the least significant bits (LSB) of the image.

The following proposed algorithm is a linear time algorithm so, it terms of time and space it does not add to any more complexity of the previous algorithm.

The (NOT) Encryption Technique

This level of text encryption will not be detectable using the previous algorithm, since it will evade the URL detection through using the binary operation NOT to encrypt the plain text.

To explain, continuing on the example that was mentioned in the Problem Definition in section 3.4.8:

The ASCII value for '*h*' is decimal 104 and binary 01101000
Before the least significant bits (LSB) insertion lets assume that 8 consecutive bytes of an images is below.
10000010 10100110 11110101 10110101
10110011 10010111 10000100 10110001

To add the extra encryption level on the plain text before embedding it in the image, the '*h*' binary will transform from 01101000 to 10010111

Therefore, after inserting the encrypted '*h*' (10010111) in the least significant bits the result is below.
1000000**1** 1010011**0** 1111010**0** 1011010**1**
1011001**0** 1001011**1** 1000010**1** 1011000**1**

The strength of this technique that it will encrypt the URL, which is a very short text embedded in a very large number of pixels, giving the sender the advantage of hiding the text without any key for the receiver to use to extract the text, in fact, the receiver will only need to know the hiding technique and the text location to extract the hidden text.

3.4.11 Experiments

The solution was implemented using Visual Studio 2015 Studio, ASP.Net 4.5 and javascript. The solution is available on http://tanvera-001-site2.htempurl.com. it was tested using bmp, png and gif images of different sizes, colour depth, colour palletes and compression types. furthermore, the solution has been tested using IE11, Firefox 4 and Chrome Ver 50.0.

It uses javascript as a client side scripting language and it will work only on browser where javascript is enabled. It also needs to access files from client machines or folders, so if there are restrictions on accessing images files, the browser will not be able to read the images files.

It cannot accept compressed and lossy images as there is a possibility that the URL data will be lost or corrupted when the images are compressed and the solution will not be able to extract the URL from the stego image [81]. Furthermore, for monochrome images, changing the least significant bits (LSB) technique might alter the image in such a way that the changes are visible to the viewers and raise suspicion that the image have been altered, therefore, it will be eliminated.

Extracting URL Images: The users needs to select the image using the 'Browse' button, which will open a file selector window. Then, the user will needs to select an image. Once the image file is selected then the user needs to click 'Extract URL' button. The website will then upload the image to the server, extract the bitmap of the image and apply the algorithm. If it identifies the URL then it will display the output on the screen as shown in Figure 3.19. However, If a URL is not present, then an appropriate message is displayed.

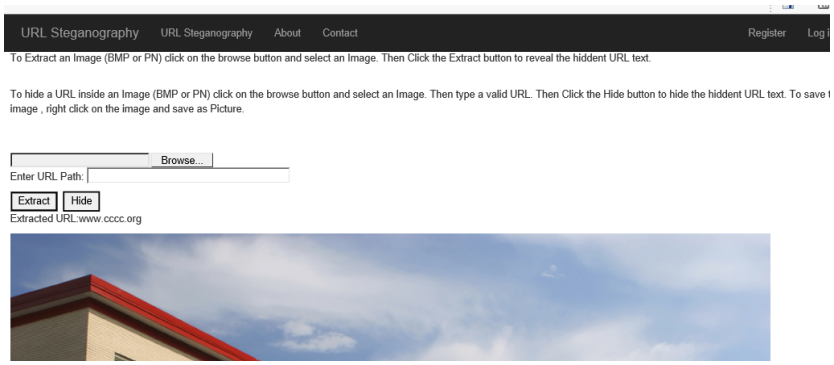


Fig. 3.19 Extracting URL from Image

Hiding URL Images: The user need to select the image where they want to hide the URL using the 'Browse' button, which will open a file selector window. Then, the user needs to select an image. Once the image file is selected then the URL needs to be entered in the given textbox. After clicking on the 'Hide URL' button, the system will first verify the URL to make sure that it is a valid URL, otherwise it will display an error message. The website will then upload the image to the server, extract the bitmap of the image and hide the URL inside the image. The modified image will be then displayed on the browser as shown in Figure 3.20. The user can right click on the image and save the image in a desired location.

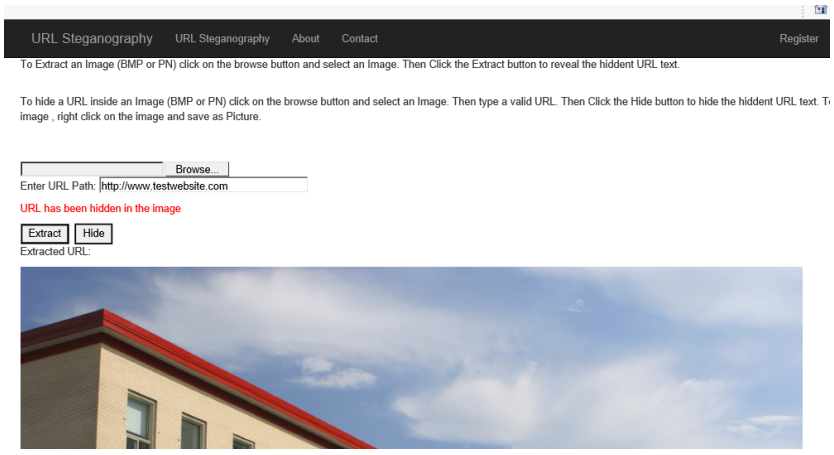


Fig. 3.20 Hidding URL using Stego Process

3.4.12 Checking Experiment Results

Image Difference

We tested the generated images (Stego Images) with the original images (Cover Images) using a free image comparison website [30]. The website found no difference between the original and image containing the hidden URL as shown in Figure 3.21. It compares the pixel value and colour between the images, there is a threshold (3 points) which the pixel must exceed in order to register as a difference. And that confirms that the statistics steganalysis techniques will not be effective in detecting and extracting the hidden encrypted URLs since they are very short and the changes that they do to the images are imperceptible.

Fig. 3.21 Image Difference between original image and stego image

Histograms Analysis

We have analyzed the histograms of the image and the generated stego image. There was no difference between the histograms of both these original image 3.22 and the stego image 3.23. And that again confirm that the steganalysis depending on the histograms wont detect the hidden URL even if the original image is known and the Stego image is known as well.

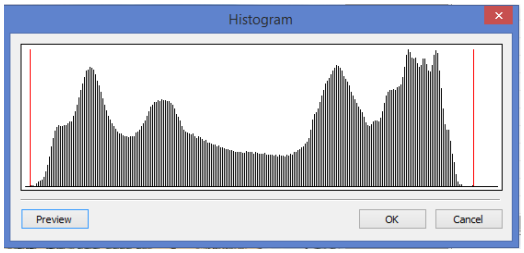


Fig. 3.22 Histograms Analysis

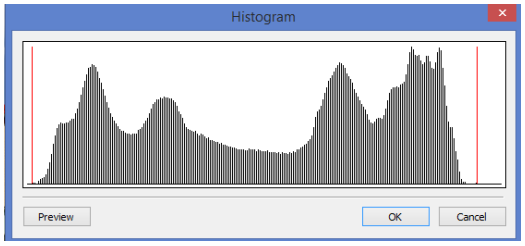


Fig. 3.23 Histograms Analysis

3.4.13 Discussion and Future Work

This section described in details from the existing research how data can be hidden in an image. Also, have dealt with hidden URL detection in images and explained the approach as well as providing the proposed algorithm in pseodocode, which can be implemented in a programming language of any choice with little efforts. Furthermore, the URL detection problem in images was simplified with respect to string matching approach which can be used in other kind of string matching problem in an image. For example, users may be interested to search for malicious

commands or other kind of strings hidden in the image using least significant bits (LSB) of the image. However, the proposed solution was taken to the next level to consider detecting and extracting encrypted hidden URLs. The experiments showed that the solution is very effective in detecting and extracting the URLs comparing to the statistical detecting technique which it will not even realised the differences after embedding the URL and that is due to URL short length.

Detecting and extracting URL as it is, specifically in images is a novel approach in image Steganography analysis. However, the reason of concentrating on this problem is a response to the introduction of the new technique of embedding malicious URLs in images recently, and that is relatively a new technique for hiding/spreading viruses. The approach time and space complexity are promising, therefore, as a future work the detection tool can be improved to cover more encrypting techniques.

Chapter 4

Fingerprint Recognition Techniques

4.1 Introduction

In Biometrics, fingerprint is still one of the most reliable and used technique to identify individuals. Recently, the need for automatic person identification has increased more and more in our daily activities, in general, and in the world of business and industry specifically.

To this end, the use of biometrics has become ubiquitous [108, 128]. Biometrics refers to metrics related to human characteristics and traits. Since biometric identifiers are unique to individuals, automatic person identification systems based on biometrics offer more reliable means of identification than the classical knowledge-based schemes such as password and personal identification number (PIN) and token based schemes such as magnetic card, passport and driving license.
Among all the various forms of biometrics including face, hand and finger geometry, eye, voice, speech and fingerprint [119], the fingerprint-based identification is the most reliable and popular personal identification method. Fingerprints offer an infallible means of personal identification and has been used for person authentication since long. Possibly, the earliest cataloguing of fingerprints dates back to 1891

when the fingerprints of criminals were collected in Argentina [85]. Now, it is used not only by police for law enforcement, but also in commercial applications, such as access control and financial transactions; and in recent times in mobile phones and computers.

In terms of applications, there are two kinds of fingerprint recognition systems, namely, verification and identification. In the former, the input is a query fingerprint with an identity (ID) and the system verifies whether the ID is consistent with the fingerprint and then outputs either a positive or a negative answer depending on the result. On the contrary, in identification, the input is only a query fingerprint and the system computes a list of fingerprints from the database that resemble the query fingerprint. Therefore, the output is a short (and possibly empty) list of fingerprints.

The majority of research in recent times has focused mostly on the fingerprint authentication, but not on the rotation of fingerprints. The lion share of the state-of-the-art in the fingerprint literature review assumes that the fingerprint is aligned in the same direction of the stored fingerprint images. This is an important aspect of fingerprint matching, which various techniques have ignored, and only very few, in the literature [1], have considered. As computers and mobile devices adopt fingerprint recognition as a way to authenticate user, this apparent tension gains more popularity, becoming an integral research area which must be addressed.

4.1.1 Motivation

The performance of Automated Fingerprint Identification System (AFIS) heavily relies on how efficiently minutiae are extracted. Most, if not all, AFIS compare minutiae information (such as ridge endings and bifurcation position) in form of sets of coordinates for verification or identification. Surprisingly, research on alternative minutiae extraction schemes is scarce. This section, proposes the

approach and implementation of a novel approach for fingerprint recognition based on the extraction of minutiae in form of circular strings, which are suitable for approximate circular string matching. In addition to that, the proposed solution is able to detect the exact location and rotation of the input fingerprint regardless of its location on the scan surface.

The organization of the rest of this chapter is as follows. In Subsection 4.2.1, we present some background related to fingerprints. Subsection 4.2.2 presents a brief literature review. We present our approach in Subsection 4.3.1 after discussing some preliminaries in Section 4.3. The experiment and the result analysis will be presented in Section 4.4. Finally, we briefly conclude in Section 4.5.

4.2 A Novel Pattern Matching Approach for Fingerprint-Based Authentication

4.2.1 Background

Fingerprint pattern can be simply defined as the combination of ridges and grooves on the surface of a fingertip. The inside surfaces of the fingers contain minute ridges of skin with furrows between each ridge. The ridges and furrows run in parallel lines and curves to each other forming complicated patterns. The basic fingerprint (FP) patterns are whorl, loop, and arch [99]. However, the most common and widely used classification method is based on Henry's classification [50] [69] which contain 8 classes: Plain Arch, Tented Arch, Left Slant Loop, Right Slant Loop, Plain Whorl, Double-Loop Whorl Central-Pocket Whorl, and Accidental Whorl (as shown in Figure 4.1).

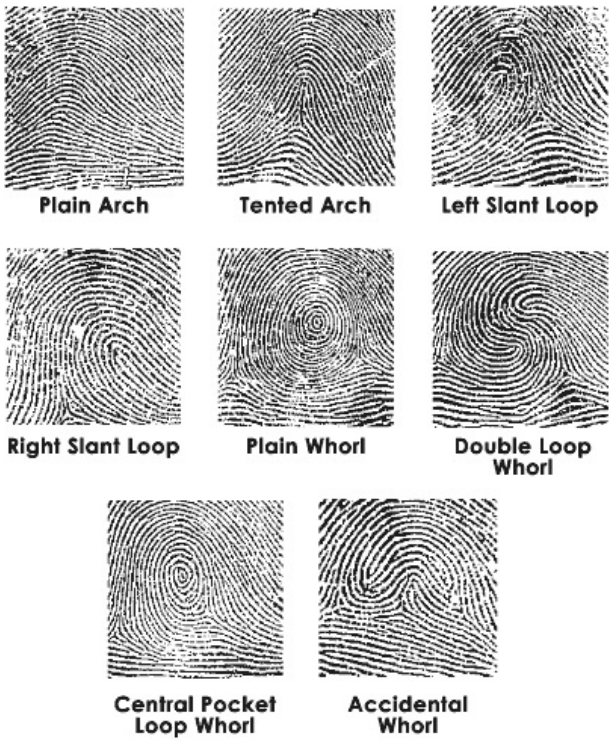


Fig. 4.1 Classification of Fingerprint Patterns

Each fingerprint is highly stable and unique. This uniqueness is determined by global features like valleys and ridges, and by local features like ridge endings and ridge bifurcations, which are called minutiae. According to recent studies, the probability of two individuals having the same fingerprint is less than one in a billion [107].

Fingerprinting has been used historically to identify individuals using the so-called ink-technique [77], where the fingers are dabbed with ink to get an imprint on paper cards which are then scanned to produce the digital image. In this off-line fingerprint acquisition technique, the fingerprints are matched by using the scanned images produced above. This method is still very important and popular especially in the forensics field, where fingerprints are captured from crime scenes. However, this type of off-line methods are not feasible for biometric systems [44]. The other approach is to scan and match fingerprints in real time via scanners.

4.2.2 Related works

Fingerprint recognition has been the centre of studies for a long time and as a result, many algorithms/approaches have been proposed to improve the accuracy and performance of fingerprint recognition systems. In the fingerprint recognition literature, a large body of work has been done based on the minutiae of fingerprints [122, 56, 66, 121]. These works consider various issues including, but not limited to, compensating for some of the non-linear deformations and real word distortion in the fingerprint image. As a trade off with accuracy, the issue of memory and processor intensive computation has also been discussed in some of these works.

The minutiae-based matching are the most popular approach due to the popular belief that minutiae are the most discriminating and reliable features [62]. Nevertheless, this approach faces some serious challenges related to the large distortions caused by matching fingerprints with different rotation (as shown in Figure 4.2).

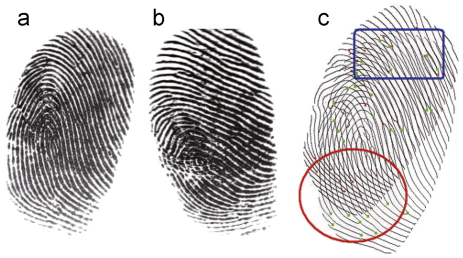


Fig. 4.2 An example of large distortion from FVC2004 DB1 [135]

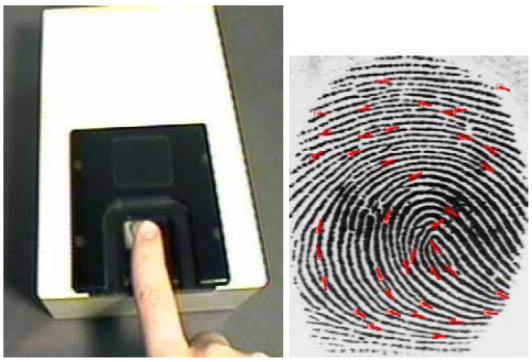
As a result, researchers have also used other features for fingerprint matching. For example, the algorithm in [104] works on a sequence of points in the angle-curvature domain after transforming the fingerprint images into these points. The actual matching is performed by computing the least-squares error of the Euclidean distance between corresponding points of the query fingerprint and the original stored in the database.

A filter-based algorithm using a bank of Gabor filters to capture both local and global details in a fingerprint as a compact fixed-length finger code is presented in [58]. The combinations of different kind of features have also been studied in the literature [55],[18]. There exist various other works in the literature proposing different techniques for fingerprint detection based on different feature sets of fingerprints [57],[43],[66].

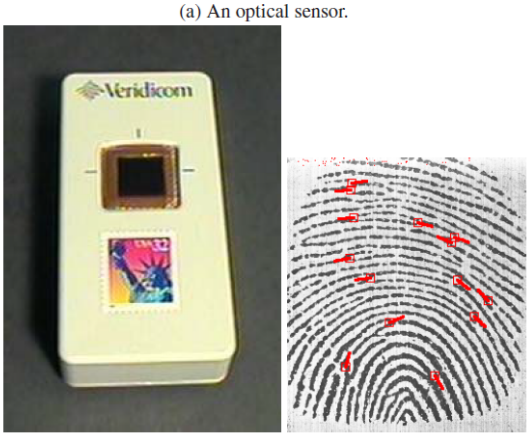
Note that, in addition to the large body of scientific literature, a number of commercial and propitiatory systems are also in existence. In the related industry, such systems are popularly termed as Automatic Fingerprint Identification System (AFIS). One issue with the AFIS available in the market relates to the sensor used to capture the fingerprint image.

In particular, the unrealistic assumption of the most biometric systems that the fingerprint images to be compared are obtained using the same sensor, restricts their ability to match or compare biometric data originating from different sensors

[103] (as show in Figure 4.3) .



(a) An optical sensor.



(b) A solid-state sensor.

Fig. 4.3 An example from [103] of the same fingerprint using 2 diffrent AFI scanners, the number of detected minutiae points in the corresponding images are 39 for image (a) and 14 for image (b)

Another major challenge of commercially available AFISs is to increase the speed of the matching process without substantially compromising accuracy in the application context of identification, especially, when the database is large [48]. This is why the quest for even better fingerprint recognition algorithms is still on particularly in the high-performance computing context [48].

4.2.3 Contribution

This section, will revisit the fingerprint recognition problem that is the basis of any fingerprint based identification system. Despite a plethora of fingerprint matching algorithms there is still room for improvement [48]. Interestingly enough, in spite of similarities between the two domains, there has not been much work at the intersection of algorithms on strings and the study of fingerprint recognition. To the best of our knowledge, here we make a first attempt to employ circular string matching techniques to solve fingerprint recognition problem efficiently and accurately. Converting the fingerprint image into string results in a small string. Matching these strings against other fingerprint images stored as strings and it can be done in linear time with respect to the total length of the strings. In this approach, we have formulated an algorithm to detect and verify a fingerprint regardless of its position and rotation in wide scanning surface area inefficient way.

4.3 Preliminaries

Let a string x of length n considered as an array $x[0..n-1]$, where every $x[i]$, $0 \leq i < n$, is a letter drawn from some fixed alphabet Σ of size $\sigma = |\Sigma|$.
The empty string of length 0 is denoted by ε. A string x is a factor of a string y if there exist two strings u and v, such that $y = uxv$.

The rest of the definitions that are used in the overview of the results and the algorithm were defined clearly in chapter 2.

Furthermore, the circular string matching problem also was clearly defined in Chapter 2, the exact and the approximate matching, however we will provide

further explanation to the approximate circular string matching since it will be used as part of our solution.

Approximate Circular String Matching

The Approximate Circular String Matching via Filtering (ACSMF) algorithm [14] is used here in order to identify the orientation of the fingerprint. The basic principle of the algorithm ACSMF is the partitioning scheme that splits the concatenation of the circular pattern string into $2d+4$ fragments, where d is the maximum edit distance allowed. The Aho-Corasick automaton [2] is then used to search for the fragments against the text. Once a fragment is identified, the fragment is extended on both left and right directions to determine a valid match. In fact, algorithm ACSMF requires average-case time $O(n)$.

4.3.1 The Approach

This section presents the main contribution, i.e. a novel pattern matching approach to solve the fingerprint recognition problem. As has been discussed before, there are two main difficulties related to the fingerprint recognition problem. First, the lack of a fixed orientation and second, the presence of errors in the scanned image due to various reasons (e.g. the presence of dust, oil and other impurities on the finger and on the scanning surface).

In particular, the following issues will be addressed:

First: Placement and orientation of the input fingerprint on the capturing surface (Rotation Problem).

Second: Consideration of the noises of the input fingerprint, i.e. error tolerant verification.

Third: Efficiency of identification, i.e. employment of a fast accurate algorithm.

The proposed solution is a first attempt to address the issue of using circular string matching in fingerprint recognition systems through proposing a solution based on the comparison of circular string representation of the fingerprint images. To explain, instead of collecting information about ridge endings and bifurcations from each fingerprint, the proposed algorithm extracts minutiae information in the form of circular strings using the proposed extracting algorithm (will be explained in section 4.3.2). Then, the Approximate Circular String Matching via Filtering (ACSMF) algorithm [14] is applied to the circular strings, in order to find all occurrences of the rotations of a pattern of length m in a text of length n, in other words, to match the extracted circular strings with the string representations of the fingerprints stored in the database regardless of the different rotation. It follows that the complexity of this approach is $O(n)$. Therefore, we will be employing a

two-stage algorithm. We start with a brief overview of the proposed algorithm below.

Algorithmic Overview

The proposed solution is divided into two main stages, namely the *Orientation Identification* stage and the *Matching and Verification* stage.

Stage 1: *Orientation Identification:* After extracting the circular strings from the input fingerprint image using the extracting algorithm (section 4.3.2) the solution will identify the proper rotation for the input image using the Approximate Circular String Matching with k-mismatches via filtering (ACSMF) algorithm; output is the matched input fingerprint correctly reoriented, and formatted according to the database standards.

To explain, in the fingerprint scanning process the finger can be placed on the scanning device at different angles. It could be aligned to the left, right or upside down (as shown in Figure 4.4).

Fig. 4.4 Different orientation for the same finger print

In fact, the position of the finger can be placed anywhere on the scanning surface and that is one of the challenging problems in fingerprint recognition systems. Without identifying the proper orientation, we can not properly compare it with the fingerprint image in the database and the recognition will not be possible

nor accurate. Therefore, the task of this stage (Stage 1) is to extract the circular strings and use them to identify and locate the fingerprint with it correct orientation.

Stage 2: *Verification and Matching:* This stage will authenticate the input fingerprint by applying standard string matching techniques to match the input fingerprint with the fingerprint image stored in the database; output is a yes/no answer to indicate whether the input image matches the image in the database or not.

To explain, like all other fingerprint recognition systems a database is maintained with fingerprint information against which the input fingerprint will be matched. In the database, we will store a black and white image. Once the orientation of the input fingerprint has been identified, we can easily reorient the fingerprint impression and then the matching algorithm runs.

Since fingerprints can contain dust, smudges, etc., the scanned information may contain errors which means that an exact match with the existing data is highly unlikely. So, in this stage (Stage 2) we perform an error tolerant matching in an effort to recognize the input fingerprint against the database of the system.

Stage 1 is in fact where the novel approach takes place. Whereas, Stage 2 makes use of standard techniques for verification, such as edit distance matching. The first part of Stage 1 (the novel minutiae extraction algorithm) will produce circular strings for the circular matching operations. This is described in depth in the next section followed by the description of the (ACSMF) algorithm. Such an algorithm represents the main contribution of this study, as it replaces the traditional approach in which fingerprints are compared according to the position of their ridges and bifurcations.

4.3.2 Details of Stage 1: Orientation Identification

The goal of this stage is to extract the circular strings in order to find the exact location and orientation of the input fingerprint in a new and efficient way. In other words, the result of this stage will be the location of the fingerprint (if there was a match), and the rotation to be applied in order to re-align the input fingerprint image according to the matched image in the database.

This **first** step in this stage, is to extract sets of circular strings at regular intervals (center points) from the scanned image using the Minutiae Extraction Algorithm.

The **second** step in this stage, is that each set of strings is then matched against the fingerprint image stored in the database using the Approximate Circular String Matching with k-mismatches via Filtering (ACSMF) algorithm. The best matched set of circular strings identify the location of the fingerprint which needs to be re-oriented, by rotating the image around the found center point p.

This stage well help in finding a set of candidate matches on the scan surface and return the minutiae (in the form of circular strings) of those matches along with their position on the scanning surface and their best alignment, i.e. the rotation to apply for re-alignment of the fingerprint. The target is to return as few candidate matches as possible, so the verification in Stage 2 will only need to perform a few steps to confirm these match results.

The Minutiae Extraction Algorithm

A preprocessing step is required to enhance the fingerprint image quality to have a better matching process. To be more specific, the main concept of the minutiae extraction algorithm in terms of speed and accuracy heavily depends on elimination

of false minutiae detection [105]. Therefore, a thinning step will help in enhancing the image quality and reduce the image noise.

For the proposed solution, different thinning techniques have been analysed, including [116], [130], [134] and [142]. However, the latter has been implemented in a C++ implementation of the Guo-Hall image thinning algorithm [90] (the following figure shows an example of the thinning result, Figure 4.5).

Fig. 4.5 Before and after performing the thinning preprocessing step on the fingerprint images

After the preprocessing step, comes the first stage which starts with the Minutiae Extraction Algorithm that uses a novel approach in finding unique combinations of minutiae for fingerprint comparison. Most, if not all, available approaches traditionally store information about ridge endings and bifurcations as sets of coordinates. The proposed algorithm in this section stores ridge and furrow information in the form of circular strings and binary strings. The rationale behind this novel technique is that such an algorithm is fast in practice, and its output is suitable for circular approximate string matching techniques.

The circular strings are extracted by drawing concentric circles over an input image, and collecting all pixels at the intersection between each circle and the fingerprint. Each input image is converted into a $2d$ matrix of pixels, for example, a $2d$ char array; to find the intersection of a circle C with radius r, having center at (cx, cy) it is sufficient to find all (x, y) coordinates which satisfy the equation of

the circumference of a circle:

$$r^2 = (x-cx)^2 + (y\check{}cy)^2. \tag{4.1}$$

4.3.3 The Algorithm in pseudo-code

```
1:  procedure NOVEL MINUTIAE EXTRACTION
2:      Input: "image", a 2d char array representing the scan fingerprint
3:      Input: "r", the radius of the current circular scan
4:      Input: (cx,cy) the coordinates of the center of the current circular scan
5:      Output:"Pattern", a circular binary string
6:      for  i From cx − r To cx + r do
7:          for  j From cy − r To cy + r do
8:              if r^2 = (i − x)^2 + (j − y)^2 then
9:                  pixel image[i][j] is at the intersection of the Fingerprint with
    the circle
10:                 if x<cx then
11:                     Left half of scan circle
12:                     if pixel at image[i][j] < 125  then
13:                         append 0 to the left of pattern (end of the pattern)
14:                     else
15:                         append 1 to the left of pattern (end of the pattern)
16:                     end if
17:                 else
18:                     Right half of scan circle
```

Algorithm 1 Novel Minutiae Extraction

```
                    if pixel at img[i][j] < 125  then
                        append 0 to the right of pattern (end of the pattern)
                    else
                        append 1 to the right of pattern (end of the pattern)
                    end if
                end if
            end if
        end for
        return pattern
    end for
end procedure
```

To explain in detail, the proposed solution will require creating at each center point a set of circular strings (inner circles) with different radius (as shown in the figure below, Figure 4.6).

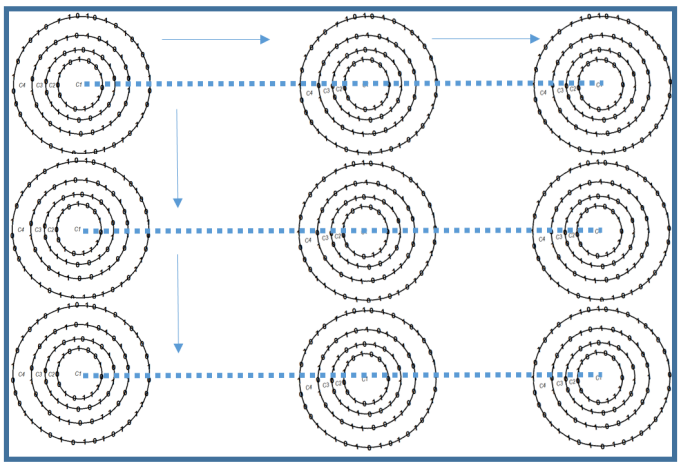


Fig. 4.6 Extracting Set of Circles with Interval Center Points

However, the extraction algorithm will extract only one circular string at a time, therefore, the algorithm will be repeated S times, where S is the number of

inner circles for each set (each center point). In fact, the maximum radius is an input parameter which will then be used to calculate the number of inner circles for each set (since the minimum radius and the distance between each circle at the same center point are fixed, so, the number of inner circles can be calculated accordingly).

Furthermore, the output of this algorithm is one circular binary string extracted utilizing a circular template having radius r and center point (cx, cy). Note that in this case the resulting string has alphabet $\Sigma = \{0, 1\}$.

To explain the algorithm in detail, line 6 and 7 loop through each pixel of a square region of $2r$ by $2r$ having center at (cx, cy); while the nested loops make it evident that the Novel Minutiae Extraction algorithm has $O(n2)$ processing time, this will give the option to search only a specific region where the scan circles are extracted, instead of searching the whole fingerprint scan. As a result, efficiency of the algorithm is increased by orders of magnitude. To explain, for an image of 300x300 pixels, extracting 45 circular strings with radius of 60 pixels takes on average 0.2200 seconds for interval center points.
Then, the previous equation (equation 4.1) is applied (line 8) in order to find pixels that belong to the circular string. In other words, all (x, y) coordinates that satisfy the equation identify pixels that lie on the circumference of radius r.

In lines 10 to 21, the algorithm stores in the output string as 0 whenever the selected pixel intersects with a ridge, and 1 otherwise (i.e. it intersects with a furrow). Since the pixels are stored as integers with values between 0 and 255, in order to know if the pixel intersects with a ridge or a furrow, it is sufficient to check if this value is greater than 125 (lines 12 and 18).

Afterwards, the final step in the algorithm, is to add an additional condition in order to avoid scrambling the output string chars. This is because for each iteration of the inner loop (i.e. for each value of j) one pixel belongs to the left and one to

the right half of the circle. This is the reason why 0s and 1s are added either at the left hand side or right hand side of the output string (lines 10 and 16).

Example 1: Let us use f_i to denote the image of the input fingerprint. Let us assume for now that we know the appropriate center point, p of f_i. We then can convert f_i to a representation consisting of multiple circular bit streams by extracting circular segments of the image. This is achieved by constructing s concentric circles C_j of radius $r_j, 1 \leq j \leq s$, with center at point p [as shown in Figure 4.7].

For each circle as mentioned previously we obtain minutiae features of the image by storing 0 wherever the edge of a circle intersects with a ridge and a 1 if it intersects with a furrow.

Fig. 4.7 Fingerprint with multiple circle scans

So, in this way, for the fingerprint image f_i, we get s concentric circles, which can be transformed into s circular binary strings [as shown in Figure 4.8] with p as assumed center point.

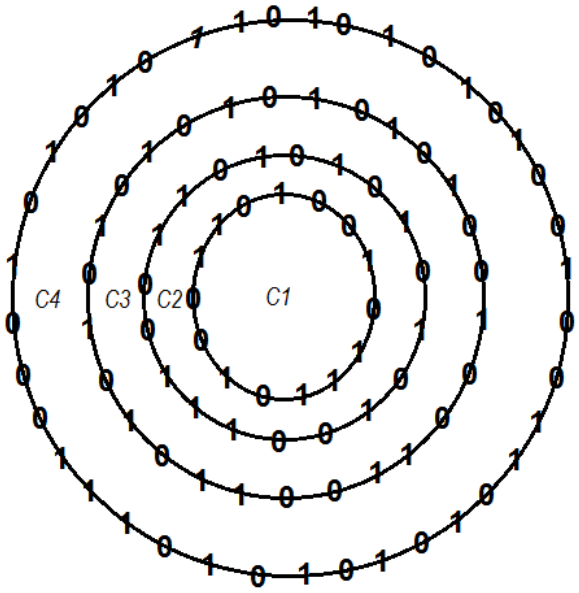


Fig. 4.8 Intersection of a circle with the fingerprint

Clearly this procedure can be easily applied on the fingerprint image stored in the database. In what follows, we will use $Y_j, 1 \leq j \leq S$ to denote the s circular strings set obtained after applying the above procedure on a fingerprint image stored in the database. In what follows, we may slightly abuse the notation and say that Y_j corresponds to the circle of radius r_j so it can be matched against the scanned fingerprint with the same radius.

Now, to identify the location and orientation of the input fingerprint we generalize the above approach to extract the minutiae features and apply the approximate circular string matching algorithm of [14] as will be described in the next section.

In fact, for the input fingerprint, we cannot assume a particular center point to draw the concentric circle which is actually the main reason for difficulty in the process. So instead, we take reference points at regular intervals across rows and columns of the entire frame of the image (i.e. the input scanning area) and at each point p_ℓ, concentric circles $C_{j\ell}$ of radius r_j are constructed. Like before, s is the

number of circles at each reference point p_ℓ. So from the above procedure, for each point p_ℓ we get s circular strings denoted by $X_{j\ell}, 1 \leq j \leq s$.
At this point the problem comes down to identifying the best match across the set of circles with the same radius. To do this, we make use of the Approximate Circular String Matching via Filtering (ACSMF) algorithm, presented in [14], which is accurate and extremely fast in practice, and will be explained in detail in the next section.

The Approximate Circular String Matching with k-Mismatches via Filtering Algorithm (ACSMF)

The proposed solution will make use of the (ACSMF) algorithm as a second step in the first stage. As it is a fast average-case algorithm for approximate circular string matching with k-mismatches, under the Hamming distance model, requiring time $O(n)$.

The (ACSMF) algorithm solves the following problem:

Given a pattern x of length m, a text t of length n, and an integer threshold $k < m$, find all factors u of t such that $u =_k x^i, 0 \leqslant i < m$.

An outline of algorithm ACSMF for solving the previous problem is as follows:

- The first step to apply the (ACSMF) algorithm is to construct the string $x' = x[0..m-1]\, x[0...m-2]$ of length $2m-1$.

  Fact 1: Any rotation of x is a factor of x'.

To explain, any rotation of $x = x[0..m-1]$ is a factor of $x' = x[0..m-1]$ $x[0..m-2]$; and any factor of length m of x' is a rotation of x (as shown in the Figure 4.9 below).

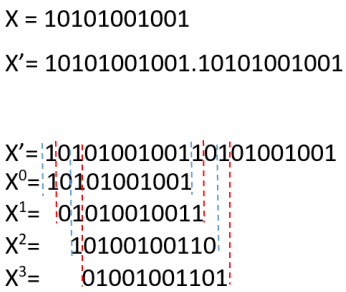


Fig. 4.9 Example of Fact 1

- The pattern x' is then partitioned in $2k+4$ fragments of length $(2m-1)/(2k+4)$ and $(2m-1)/(2k+4)$ where k is the number of allowed mismatches.

Lemma 1: at least $k+1$ of the $2k+4$ fragments are factors of any rotation of x.

To explain, if we partition $x' = x[0\cdots m-1]x[0\cdots m-2]$ in $2k+4$ fragments of length $(2m-1)/(2m+4)$ and $(2m-1)/(2m+4)$, at least $k+1$ of the $2k+4$ fragments are factors of any factor of length m of x' (as shown in the Figure 4.10 below).

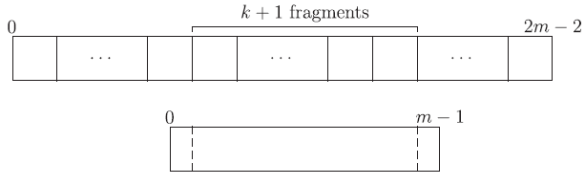


Fig. 4.10 Example of Lemma 1 reference ACSMF

- Then, it matches the $2k+4$ fragments against the text t using an Aho Corasick automaton for exact matching [2] and constructs a list of tuples named $\mathscr{L}$ to be the list of size Occ of tuples, where $< p_{x'}, \ell, p_t > \in \mathscr{L}$ is a 3-tuple such that $0 \leqslant p_{x'} < 2m-1$ is the position where the fragment occurs in $x^{'}, \ell$ is the length of the corresponding fragment, and $0 \leqslant p_t < n$ is the position where the fragment occurs in t.

- Finally, it simply perform letter comparisons and count the number of mismatches that occurred. The extension to the right side and the left side of the factor will stop right before the $k+1th$ mismatch, as illustrated in the figure below (Figure 4.11) and will report a non-match, or stop at the end of the pattern and will report a match since the allowed number of mismatches was not exceeded.

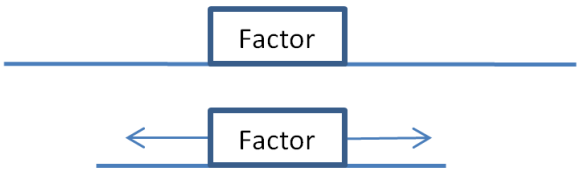


Fig. 4.11 Illustration of the final step in ACSMF

To continue the previous example (example 1), we take a particular extracted circular string $X_{j\ell}$, and construct $X_{j\ell}.X_{j\ell}$ (to ensure that all conjugates of $X_{j\ell}$ are considered) and apply algorithm ACSMF on $X_{j\ell}.X_{j\ell}$ and Y_j. In other words, we try to match the circular string Y_j (corresponding to the circle of radius r_j) to all circular strings $X_{j\ell}$ (corresponding to the circle of radius r_j) generated at each point p_ℓ. Thus we can identify the best matched circular string, i.e. the best matched circles and thereby locate and identify the fingerprint impression with the correct orientation. Once the orientation has been identified, we can apply standard

techniques to reorient the image to match with the image from the database in the next stage.

Approximate circular string matching is a rather undeveloped area. ACSMF is the best algorithm to be applied in this stage. Full details on how the ACSMF algorithm work can be found in [95].

4.3.4 Details of Stage 2: Verification and Matching

Once Stage 1 of the algorithm is complete, we can assume that we have two images of the same size and orientation which we need to match and verify. We call this a verification process since in (Stage 1) as well we have done a sort of matching already. However, we need to be certain, and hence we proceed with the current stage as follows. Each image can now be seen as a two dimensional matrix of zero/one values, which can be easily converted to a (one dimensional) binary string. Now, it simply comes down to pattern matching between two strings of the same length. However, note that, here as well we need to consider possibilities of errors. So, we simply compute the edit distance between the two binary strings and if the distance is within the tolerance level, we consider the fingerprint to be recognized. Otherwise, the authentication fails.

4.4 The Experiment

As opposed to gathering information about ridge endings and bifurcations from each fingerprint, the proposed algorithm extracts minutiae information in form of circular strings. Thereafter, the Approximate Circular String Matching via Filtering (ACSMF) algorithm [95] is applied to the circular strings, to find all occurrences of the rotations of a pattern of length m in a text of length n [14], where n is the concatenation of all string representations of the fingerprints in the database, and m is the string representation of the fingerprint to identify. It follows that complexity of this approach is $O(n)$. The solution proposed in the previous section is divided into two main stages:

- Stage 1 – Orientation Identification
- Stage 2 – Verification and Matching

4.4.1 The Implementation

The proposed solution has been developed in ANSI C/C++ using the external library OpenCV (freely available for academic use, under the BSD licensed, at (http://opencv.org) for standard image processing.
Different inputs have been tested by running the function `fp_auth` several times against the Fingerprint Special Database of the National Institute of Standards (NIST) [89] which is a database containing fingerprint images scanned from the same scanner.

"The NIST database of fingerprint images contains 2000 8-bit gray scale fingerprint image pairs. Each image is 512-by-512 pixels with 32 rows of white space at the bottom and classified using one of the five following classes: A=Arch, L=Left Loop, R=Right Loop, T=Tented Arch, W=Whorl." (NIST) [89]

All external sources are open/free for academic purposes under (BSD licence). The experiment has been tested with black and white Tiff images. These images have been preprocessed to be thinned fingerprints using a C++ implementation of the Guo-Hall image thinning algorithm [90]. The experiment results shows over-enhanced images with different parameters for the maximum radius.

The steps that the proposed solution will preform in order to compare an input fingerprint against another fingerprint stored in the database are the following:

- It will start by loading two enhanced (thinned) fingerprint images in two $2d$ arrays of pixels. The first image is the input fingerprint from the scanner to be verified, the second one is the fingerprint stored in the database (assuming both images uses same scanner standards).
- Then it will extract $1..n$ circular binary string set and each set contains s inner circular strings as a representation of both fingerprints taken at regular intervals (center points) from a center (px_n, py_n) using the minutiae extraction algorithm described in section 4.3.2.
- Store each set of circular strings in a $2d$ array.
- Then, loop through each row and column of the $2d$ array; for each set of strings, apply the Approximate Circular String Matching with k-mismatches via filtering (ACSMF) algorithm to a corresponding set of circular strings in the database, using a specific tolerance threshold (the allowed k mismatch), under which the input fingerprint is to be considered as a candidate match; discard all non-matching circular scans.
- Then, the second stage of the solution will be to confirm and verify that this is the best match and rotation. The proposed solution will uses a simple

Levenshtein distance implementation in order to return the best alignment of that candidate match and if the results equal or below the allowed distance then the matching was successfully done. Figure 4.12 is the call graph of the proposed solution.

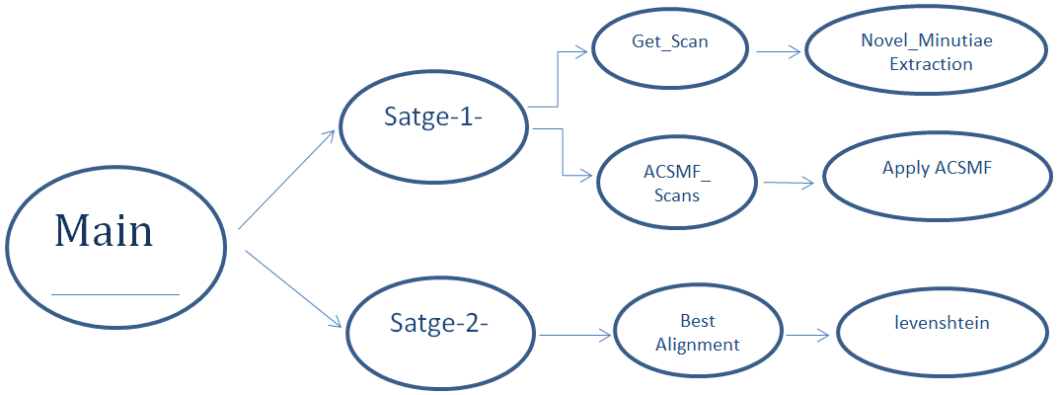


Fig. 4.12 Call graph of the proposed solution.

The following experiment results (in Table 4.1) shows over-enhanced images with different parameters for the maximum radius.

Table 4.1 Experiment Results

Mated Image	No. Mismatch allowed	Max Radius	Radius distance	Elapsed time of get scans	No. Matches	Rotation in pixels
Y	10	60	2	0.7197	0	0
Y	30	50	2	0.7308	1	10
Y	30	60	10	0.077261	6	<10
Y	30	60	5	0.292098	5	<10
Y	50	60	2	0.6865	>30	10
Y	60	60	10	0.074452	>30	<10
N	80	60	2	0.7823	2	137
N	20	60	10	0.054203	0	0

The data entries in the table are explained as follows:

Mated image: Refers to the input image whether it is related to the compared image or not.

No. of mismatch allowed: This is the tolerance threshold under which the input fingerprint is to be considered as candidate match corresponding to the set of circular strings.

Max radius: The radius for the maximum circle in pixels that can be scanned per image.

Radius distance: The interval size in pixels between each center point in the scanned image.

Elapsed time to get scans: The time in seconds to get the total circular scans per image according to the maximum radius and radius distance (the interval distance between the center points of circles).

No. matches: The number of candidate matches after applying the (ACSMF) algorithm for the circles with maximum radius.

Finally, **Rotation in pixels** is the rotation to be applied on the input fingerprint image in pixels.

In particular, the table shows that the time to get scans for each image is less than a second. To explain, since the exact center point for the scanned image is unknown, the process of getting the scans or extracting the strings, will be through considering multiple center points according to the (radius distance) parameter starting from the bottom left of the scanned image and moving on (as explained previously in the extraction section). The higher radius distance will result in extracting less circular strings in less time.

Essentially, it shows that the increasing number of allowed mismatches, will result in increasing the number of matched candidates returned by (ACSMF). For

instance, when mated images are scanned and compared when the number of allowed mismatch is very low, almost equal to 10, it returns a negative result.
In contrast, when the number of mismatches allowed is very high, for example 80, the number of returned matches is 2 even though the input image is different to the stored image.

However, the correct match is shown when the number of mismatches is equal to or around 30. Keeping in mind, extending the string length through having a bigger radius with the same number of allowed mismatches will result in the chance of decreasing the accuracy of the results, therefore, will have more results and more matches (for example the circular string with radius 50 will give more accurate and less number of matches than a circular string with radius 60 with the same number of allowed mismatches 30). However, having a small radius or bigger one will mean extracting circular strings from different positions in the scanned image with different length. Yet, the maximum radius size will have a great effect on the results in deciding the number of inner circles as it will be explained in the next section.
In general, the previous results are for extracting the maximum radius and matching it with the maximum radius of the stored image in the database. These results shows clearly the effect of choosing the number of mismatches allowed, which should not be very high to avoid false positive returns, nor very low to prevent the false negative rate.

The main advantage of this approach is regardless of the fingerprint rotation degree, the accuracy of the result will not be affected, whereas most of the other fingerprint detection algorithms accuracy results are affected by the rotation degree. Furthermore, as mentioned before, the rotation problem is still an open problem and requires a lot of improvements "In the fingerprint there is a need of a fast and

accurate method which can withstand the challenges like rotation" [65].

Moreover, according to the Fingerprint Matching and Non-Matching Analysis for Different Tolerance Rotation Degrees study in [93] where they evaluated three biometric systems (Neurotechnology Verifinger 6.0 Extended, Innovatrics IDKit SDK and Griaule Fingerprint SDK 2007) and the influence of the fingerprint rotation degrees on false match rate (FMR) and on the false non-match rate (FNMR), their results showed that these values are affected by the rotation degree as the (FMR) values increase as rotation degrees increase too. Additionally, it was stated that one of the factors that affect the performance of the matching algorithm is the fingerprint rotation.

For example, the result table (table 4.13) from [93] shows the affect of the rotation degree on the (FMR) value using the biometric system (Innovatrics IDKit SDK) with overall (FMR) value of mean over 30%.

Degree	FMR			
	Group 1	Group 2	Group 3	Group 4
15°	29.51%	32.54%	34.84%	25.51%
30°	30.57%	33.37%	35.89%	28.17%
45°	30.88%	33.53%	36.14%	30.20%
60°	30.73%	33.25%	36.03%	31.36%
Mean	*30.42%*	*33.17%*	*35.73%*	*28.81%*

Fig. 4.13 FMR mean of IDKit SDK in [93]

However, this is not the case in the proposed solution, as shown in the results next table (Table 4.2) extracting two images (with two inner circular strings) and matching them with different rotation degree, the extracting time and matching time was not affected with the rotation degree nor the accuracy of the results.

Table 4.2 Scanning and Matching same Fingerprint with different rotation with extracting 2 inner circles

Image Name	**Rotation Degree**	**Scanning Time**	**Matching Time**
Image -1-	90% to the R	0.80 sec	4.15 sec
Image -1-	180%	0.88 sec	4.14
Image -2-	90% to the R	0.89	2.56
Image -2-	90% to the L	0.88	2.57

The FMR and FNMR rates are the error rates that define the performance of biometric systems, so, the final table (table 4.3) will compare the false match rate (FMR) and false non-match rate (FNMR) mean values for the proposed solution among the three solutions in in [93] and also among the solution in [65] in regrade of different rotation degrees.

	FMR	FNMR
Griaule Fingerprint SDK 2007	0.02	60.46
Innovatrics IDKit SDK	32.00	13
Neurotechnology Verifinger 6.0	8.33	> 7
Khan 2015	6.80	14.20
Proposed Approach	8.60	0.00

Table 4.3 FMR and FNMR of The Proposed Algorithm and other Approaches

It's clear that the proposed solution will give more accurate results than the other solutions and faster, in average time 4.5 seconds.

4.4.2 Accuracy and Speed

This approach has two values that determine the accuracy and speed. In Stage 1, the accuracy depends on the number of concentric circles (inner circles) which was

denoted by s. The larger the value of s, the higher the accuracy of pinpointing the location with the correct orientation. However, as s increases the computational requirement and time also increases as well.

In Stage 1 and 2, there is another value (k-mismatch) which is the tolerance level, i.e. the edit distance allowed between the two strings. Note that, the bulk of the computational processing in this approach is required in Stage 1, where we extract the circular strings, then we apply the (ACSMF) algorithm to identify the best matched circles. As has been shown in [95], on average, ACSMF works in linear time to the size of the input and is extremely fast. The size of the circles and hence the corresponding circular strings are very small and can be assumed to be constant for all practical purposes.

As a result the running time of Stage 1 would be extremely fast. Again, since the size of the fingerprint image is very small, any efficient approximate string matching algorithm in Stage 2 would give us a very quick result.

As shown in table 4.3 the accuracy of the algorithm is high comparing to other solutions and considering different rotation degrees.

As mentioned previously, the proposed solution concentrated on two challenges in the fingerprint identifications systems (Image noise and Rotation degree) and were tested several times against the Fingerprint Special Database of the National Institute of Standards (NIST) [89].

4.4.3 Cross Matching

Furthermore, the proposed solution also overcomes the challenge that was mentioned in the related work (section 4.2.2) which was the restriction on ability to match nor compare biometric data originating from different sensors or as it called (Cross matching). Since each sensor detects different numbers of minutiae points at different positions in the fingerprint images (figure 4.3) which will affect the

accuracy of the matching results since most, if not all, available approaches traditionally store information about detected minutiae points as sets of coordinates, and that is not the case in the proposed solution which uses a different approach of approximate circular string matching.

However, there is another challenge in the (cross-matching) that the proposed solution will not overcome yet, which is the size of the Region of interest (ROI) that is to be matched against.

As we assumed all the fingerprint images are within the same size, since we targeted the systems that use the same scanner to store the fingerprints in the database and to capture the input fingerprint image, so theses fingerprints from the same sensors have the same resolution, therefore there is no need to do the scaling. For example the employees attendees system will capture the employees fingerprint for the first time and store them in the database, then will match it daily using the same sensor (overcoming the challenge of the presence of dust, oil and other impurities on the finger or on the scanning surface and the finger position on the scanning surface). And that is the scenario in many applications, however, it will not consider the image size differences, in fact this problem (image size) does not exist in traditional automatic fingerprint identifications (regular-matching), rotation and noise are the main two main considerations [101].

However, to target wider applications we have to consider the current situation where the market is full of Automatic Fingerprint Identification devices and each device captures the fingerprint image with different sizes. This introduces the second challenge with the cross matching process (fingerprint image size) to be considered.

The fingerprint image size or as it called zooming or scaling problem is an important issue for fingerprints from different capture sensors [138]. The proposed solution can deal with the fingerprint image captured using any device (sensor)

as long as the (ROI) is within the same size. In fact, it is still a limitation and a challenge in most of the fingerprint verification systems, since matching two fingerprints with two different (ROI) sizes will definitely affect the (FMR) and (FNMR) rate in a negative way.

However, this challenge is related to the image enhancement preprocessing stage, where the fingerprint image is firstly enhanced in different ways before the features contained in it could be detected or extracted [12] and there are several different image enhancement techniques that are used to improve the image quality before the matching process [12] such as segmentation, local normalization, filtering and binarisation/thinning.

Nevertheless, the proposed solution considered one step in the preprocessing stage for image enhancement which was the image binarisation and thinning step to improve the fingerprint image before applying the novel pattern matching proposed solution. However, overcoming the scaling or (ROI) challenge can be done through adding an extra step to the prepossessing stage for scaling or re-sizing the fingerprint image and detecting the best (ROI) to enhance the image quality and therefore, enhance the matching quality.

The fingerprint scaling problem refers to the adjustment of the fingerprint images to solve the problem of sensor interoperability [100]. Different methods were introduced to solve the scaling problem for example in [59] where they improved the interoperability of the fingerprint recognition through using resolution compensation based on sensor evaluation. They directly calculated the scale parameter as the quotient resolution of two compared sensors which are known (e.g. scale between URU sensor and UPEK sensor is 700dpi/508dpi) which means the sensors values will be added manually and the method will be limited only to the sensors included in the study. Another method [138] developed a coarse-fine technique to estimate the optimal scale between the input fingerprint

and the template fingerprints, they calculate the global scale based on the ridge distance map and determined by the histogram of local refined scale between all the matchable minutiae pairs. However, this method does not perform equally on all types of sensors (capacitive, optical, thermal). For example, the accuracy of scaling between the same kind of sensors, optical sensors (URU vs. UPEK) outperforms that between different kinds of sensors, optical vs. Capacitive (URU vs. AES).

After analyzing many scaling methods some of them were limited to the sensors type and some were limited to the fingerprint detecting methods. And since the used technique in the proposed solution is a novel new technique using circular strings, the best suitable scaling method to be used, is through estimating the average inter-ridge distance of the fingerprint image. To explain, the average inter-ridge distance is an important characteristic in the fingerprint image enhancement process [140] and the experimental results in [100] showed that the estimation method for the scaling parameter is appropriate, and its robustness allows the typical fingerprint image to be re-scaled accurately. They added, "For the same finger, average inter-ridge distance is stable correspondingly, so we could calculate the scaling parameter through making the average inter-ridge distances of two fingerprint images unified."

Therefore, this can be an extra step in the preprocessing stage to improve the proposed solution to target wider applications not just in the (regular-matching) process but in the (cross-matching) process as well.

4.5 Discussion AND Future Work

This chapter proposes yet a new pattern matching based approach for fast and accurate recognition of fingerprints. A notable challenge in fingerprint matching is that the rotation of the fingerprint is assumed to be in sync with the stored image; in this chapter we have tackled this issue. The novel element of this approach is the process of using a series of circles to transform minutiae information into string information consisting of 0s and 1s, and then using the approximate circular string matching algorithm to identify the orientation. This technique has improved the performance and accuracy of the fingerprint verification system.

Although the matching algorithm produces nearly accurate results at high speed, However this approach have only considered the mode where the input is a query fingerprint with an identity (ID) and the system verifies whether the ID is consistent with the fingerprint (verification mode). As a future work we can consider the problem of (identification mode) where the need to match the query fingerprint against a list of fingerprints in the database and usually its a very big volume of data with high accuracy and speed similarly to the current approach.

Furthermore, implementing extra processing layers to the database in the current approach will improve the accuracy and speed for big volume data. For example, as an improvement for the proposed solution we can make use of the center of gravity of the fingerprint scan image to use it as the center point p in the image f_i for the constructed circular. An on going experiment is under development to provide more details.

Chapter 5

Smart Meter Data Analysis Technique

5.1 Introduction

In response to increasing demands to solve energy problems, the EU has chosen 2020 to be the target of fully implementing its 20-20-20 plan (20% increase in energy efficiency, 20% reduction of CO2 emissions and 20% renewables). One way of achieving that goal is revamping the way electricity usage is measured and communicated through the invention of smart electricity meters [64].

Smart meters are an integral part of a bigger and more complicated system, which, in addition to smart meters, encompasses communication infrastructure and control devices [35].

As opposed to the current power grids, smart meter systems have a two-way communication capability that allows real-time data transfer. These systems are called Advanced Metering Infrastructure (AMI) [64].

Smart meters are currently rolled out in many countries as a plan to cover the need of conserving energy by reducing the electricity use in houses along with increasing

users' energy literacy and improving the user's consumption behaviours. Since the smart meters will proved the users with real electricity readings, they will be able to decide and identify which devices are consuming energy in that specific moment and how much it will cost. Smart meters read and send electricity consumption information such as the values of voltage, phase angle and the frequency in real time to electricity companies [35].

Smart meters are designed with a bidirectional communication capability [as shown in Figure 5.1, that allows suppliers/electricity companies to send back consumption data and its corresponding cost to consumers utilizing in-home-display (IHC) capability [23].

Fig. 5.1 Smart Meters

5.1.1 Motivation

Providing a global understanding of online privacy is crucial, because everything is connected. Nowadays, companies are providing their customers with more services that will give them more access to their data and daily activity; electricity companies are marketing the new smart meters as a new service with great benefit

to reduce the electricity usage by monitoring the electricity reading in real time. Although the users might benefit from this extra service, it will compromise the privacy of the users by giving others constant access to the readings. Since the smart meters will provide the companies with real electricity readings, they will be able to decide and identify which devices are consuming energy in that specific moment and how much it will cost. This kind of information can be exploited by numerous types of people. Unauthorized use of this information is an invasion of privacy and may lead to severe consequences.

The advantages of this new technology is clearly defined and introduced to the clients in a very attractive marking campaigns by the electricity companies, however, the downside of this technology is not yet exposed to the people clearly, therefore providing a proof of how analysing the smart meter reading can lead to identifying exactly the private activates at each house is important, in other words, these data readings can act like a magnifier at each house, and this can be considered as a major privacy breach.

This new technology has been recently the focus of many researches who have been studying the techniques and performance of this technology. This chapter intends to prove that gaining access to the smart meter readings will give a clear vision of the exact in house activates through proposing an algorithmic approach to compare and analyze smart meter data readings, considering the time and temperature factors at each second to identify the used patterns at each house by identifying the appliances activities at each second with time complexity $O(log(m))$.

The organization of the rest of this chapter is as follows. Smart meters background is presented in Subsection 5.1.2 and some related privacy issues are presented in Subsection 5.1.3. In Subsection 5.1.4 some related work to smart meter data analysis. Subsection 5.1.5 presents the functionally of the new smart meters

and how they work. We define the problem in the Section 5.2 and present the approach in Subsection 5.2.1 with some definitions. The Algorithm description and complexity analysis will be presented in Subsection 5.3.3. Finally, we briefly conclude and state the future work in Section 5.4.

5.1.2 Background

The use of smart meters provides us with a lot of advantages such as eliminate the need for manual meter reading. Also, providing real time consumption readings, will help in managing electric loads to prevent outages and blackouts. More accurate bills; avoiding to overpay, underpay, or estimating the bill amount. Promote efficient consumption by helping consumers to reduce their electricity usage. It will also, provide the users with the ability to schedule preventive maintenance. Defining customer segments which will be used for achieving higher returns in energy programs pricing and also program marketing [67]. Finally, The ability to detect unwanted or harmful components in current, which helps in rectifying the problem [35].

On the other hand, like any other newly introduced technology, implementing smart meters systems bears its risks and challenges, for example, system deployment and implementation is extremely costly;according to the UK government (Smart Metering Implementation Program) report it will cost around £11bn [34]. Also, the lack of the adequate infrastructure would make the process of replacing the current system to a smart one cumbersome. The method of which smart meters work present a serious privacy and security issues, the data collected reveals what appliances the consumers use at their premises, when they use them, their presence or absence, and their usage habits [35].

5.1.3 Privacy Issues

Breach of privacy has been the number one problem discussed in the media regarding smart meters. The methodology by which the smart meter system works is actually what makes it vulnerable and compromises the privacy of its customers. Any weakness in the Advanced Metering Infrastructure (AMI) might

give the privacy hackers the chance to expose customers' privacy through analysing electricity consumption information. Having the ability to collect, send, and analyse electricity usage, it would be possible to detect private in-house activities, and identify exactly what home appliances were used and when [64].

Each electrical device has its unique method of consuming electricity. Therefore, analysing consumption data would allow suppliers, and possible hackers, to know what time customers watched TV, if they used the washing machine, the time they showered and much more. As opposed to the current metering system that collects data once a month, smart meters collect data at finer time scales that reveals private information such as the number of people in a household as well as their sleeping and eating patterns [16].

Despite its smart capability of data collection and transfer, analysing these data does not require advanced technology or prior training. In fact, there are numerous off-the-shelf statistical methods that could decipher the tremendous amounts of consumption data. These methods can detect several activities in the household though analysing the level of power consumption, its intermittency, and its duration [16].

5.1.4 Related Work

Having real time readings or every 5 min, 15 min, or even 30-minutes reading points will put the electricity supplier companies in a great challenge on how they are going to manage and deal with the massive data collected from these smart meters from every house. How they are going to decode these readings into meaningful information that will help in improving the companies electricity marketing programs after understating the user consumption behavioural.

Furthermore, data analytic will help them to introduce customized packages to each customer for energy saving. many data analytic approaches where introduce

for that purpose trying to transfer the meter reading into meaningful information that will give an understanding of the user electricity daily usage pattern.
Smart meter data analysis is relatively new concept just like the technology it self, therefore, the related work to this concept is not much and most of these approaches tried to provide the users with appliance-specific consumption break down, focusing on implementing sensors on each appliance in the house as an identifier for that appliance when receiving the electricity reading.
However, these approaches are costly and not convenient since it requires a sensor implementation or using smart power outlets on every home appliance. Moreover, another rule-based approach concentrated on grouping the appliance with sillier power usage and identifying them according to their frequency of use [78].

In 2012 [132] presented a single sensor approach which tries to link between smart meters and smart phones (via a gateway) to produce an energy efficiency services (such as itemized electricity bills or Targeted energy saving advice) but even with this improvement it is still considered as costly method.

5.1.5 Functionality

By using it bidirectional communication capability, smart meters enable local and remote execution of control demands such as monitoring home appliances and devices. In addition, smart meters can communicate with other meters in their reach, control heat, light, and air conditioning, keep operation schedule for home appliances and devices, and use data collected to bill consumers accordingly [35].
The two-way communication capability, is beyond just receiving readings from customer's meters, but it also goes to sending commands, and get a response back. For instance, if the electricity provider cute the power off the client due to unpaid bills, with smart meters systems; it will only take a click to turn the power back on the client side, whereas, with the classic meters it will take up to days where the

electricity provider have to arrange for a technician to go to the client's house to turn the power on.

Furthermore, since the smart meter will provide a real time electricity readings on the customer side via the smart meter monitor the electricity provider can access the reading data every hour which will mean that they will gather 24 reading point a day, or every 30 minutes which is 48 reading point a day, or every 15 minutes and get 96 reading points a day and for 5 minuets they will get a 288 reading points day, and so on, that means the more reading points they can retrieve the more efficient their data analytic will result.

x_i is the reading in kWh at the i^{th} second of the daily reading points $X = [x_1, x_2 \ldots x_i \ldots x_{86,400}]$

Hourly reading : $[\sum_{k=1}^{3,600} x_i \ldots \sum_{k=86,399}^{86,400} x_i]$

Daily readings : $E = \sum_{k=1}^{86,400} x_i$

Therefore it is a fact that smart meters will record much more detailed consumption data and have more power than the classic electricity meters can ever provide.

5.2 The Problem

In each house, the electricity consumption is influenced by some factors such as environmental factors (weather, day time, seasons, house size...etc) along with user's social behaviour factors and the number of residents in each house which will definitely effect the electricity consumption.

In this chapter we developed an algorithm to analyse the energy reading in real time points (every second). Along with some external environmental factors such as daytime or night-time usage, and outside temperature.

5.2.1 The Approach

The dataset that are used in this approach and later in the experiment is the (Aggregate Electrical Data) which is part of the Smart* project is to optimize home energy consumption [127]. The dataset contain the smart meter readings for one house every second for 4 months. by studying the changes in the power reading at each second it will be possible to identify almost exact used appliances and weather it was turned ON or OFF according to the appliances database.

However, if the changes in the power readings (in one second) retrieved more than one home appliances then we will use the probability approach and identify the higher probability between the 2 appliances. Even If there was no matches then we will take the appliances with the higher probability at that second and then the next higher probability and so on.

The positive change will be considered as (appliances ON) the negative change as (appliances OFF). If there was an (appliances ON) then it can't take another same (appliances ON) again without the OFF action (unless it was specified in the database), in that case, we consider the highest appliances probability at that second along with the other factors.

5.2.2 Definitions

Problem : Smart Meter on Every Second Data

Input : $D[1..n]$ is an array of string which holds data for watt
+ second + temperature in each index of this array.
For example, $D[i]$="watt, second, temperature", $1 \leq i \leq n$

$A[1..m]$ is an array of string which holds data for watt
+ increment + time with percentage of probability
+ temperature with percentage of probability
+ appliance name in each index of this array.
For example, $A[j]$="watt, increment,
time: the percentage of possibility,
temperature: the percentage of possibility,
appliance name", $1 \leq j \leq m$

Note 1: The time is divided into 24 units a day, for example:
00:00:00 - 00:59:59; 01:00:00 - 01:59:59; 02:00:00 - 02:59:59;
....; 23:00:00 - 23:59:59;
Note 2: The temperature is divided into 12 units, for example:
under 0; 0-4; 5-9; 10-14; 15-19; 20-24;
25-29; 30-34; 35-39; 40-44; 45-49;
and the final unit equal or over 50 centigrade.

Output : $O[1..x]$ is an array of string which holds data for each second
+ open appliances or nothing in each index of this array.
For example, $O[y]$="second data, open appliances or nothing"

5.2.3 Example

Example : Smart Meter on Every Second Data

Input : $D[1] =$ "$2500,\ 00:00:00,\ 5$"
$D[2] =$ "$2500,\ 00:00:01,\ 5$"
$D[3] =$ "$2740,\ 00:00:02,\ 5$"
$D[4] =$ "$2740,\ 00:00:03,\ 5$"
(First number "2500" means currently watt is 2500 watt)
(Middle number "00 : 00 : 00" means time is 0 o'clock)
(Last number "5" means temperature is 5 Centigrade)

$A[1] =$ "$30, 10$
$00:00:00-0:59:59\ \ 50\%,$
$01:00:00-01:59:59\ \ 50\%,$
....
$23:00:00-23:59:59\ \ 90\%,$
$under\ 0\ \ 90\%,$
$0-4\ \ 90\%,$
$5-9\ \ 90\%,$
....
$equal\ or\ over\ 50\ \ 90\%,$
$Light$"

$A[2] =$ "$240, 20$
$00:00:00-00:59:59\ \ 50\%,$
$01:00:00-01:59:59\ \ 50\%,$
....
$23:00:00-23:59:59\ \ 70\%,$
$under\ 0\ \ 90\%,$
$0-4\ \ 90\%,$
$5-9\ \ 90\%,$
....
$equal\ or\ over\ 50\ \ 70\%,$
$Computer$"

$A[3] =$ "$2500, 100$
$00:00:00-00:59:59\ \ 70\%,$
$01:00:00-01:59:59\ \ 70\%,$
....
$23:00:00-23:59:59\ \ 70\%,$
$under\ 0\ \ 90\%,$
$0-4\ \ 90\%,$
$5-9\ \ 90\%,$
....
$equal\ or\ over\ 50\ \ 00\%,$
$Heater$"

Output : $O[1] =$ "$2500,\ 00:00:00,\ 5; Open : Heater$"
$O[2] =$ "$2500,\ 00:00:01,\ 5;$"
$O[3] =$ "$2740,\ 00:00:02,\ 5; Open : Computer$"
$O[4] =$ "$2740,\ 00:00:03,\ 5;$"

5.3 Algorithm

5.3.1 The Algorithm in pseudocode

Algorithm 2 Smart Meter Reading at Every Second Data

```
1:  procedure Smart Meter Reading at Every Second Data
2:      for integer w=1 to n on D[1..n] do do
3:          newwatt = watt of D[1]
4:      end for
5:      if w==1 then
6:          oldwatt = newwatt
7:      end if
8:      if (newwatt-oldwatt)>3 then
9:          countwatt = countwatt+(newwatt-oldwatt);
10:         remembertimes[countremembertimes+1] = w;
11:         countremembertimes++;
12:     end if
13:     if ((newwatt-oldwatt)<-3&&countwatt!=0) then
14:         ———find one device———
15:         if (highestpoint[countwatt][0]==1) then
16:             printf("Time: %d",remembertimes[1]);
17:             printf("Increase watt: %d", countwatt);
18:             printf("Open device name: ");
19:             for(int i=0; i<8; i++)
20:             printf("%c", devicesname[highestpoint[countwatt][1]][i]);
21:         end if
```

```
22:         ————find more than one device————
23:         if (highestpoint[countwatt][0]>1) then
24:             compare[0]=0;
25:             for (int s=1; s<=(highestpoint[countwatt][0]); s++) do
26:                 compare[s]=(newhours[highestpoint [countwatt] [s]]
    [int(remembertimes[1]/3600)] + newtemperatures[highestpoint [countwatt]
    [s]] [int(AT/5)+1])/2;
27:             end for
28:             for (int ss=1; ss<=(highestpoint[countwatt][0]); ss++) do
29:                 if (compare[ss]>compare[0]) then
30:                     compare[0]=compare[ss];
31:                 end if
32:             end for
33:             for (int sss=1; sss<=(highestpoint[countwatt][0]); sss++) do
34:                 if (compare[sss]==compare[0]) then
35:                     printf("Time: %d",remembertimes[1]);
36:                     printf("Increase watt: %d", countwatt);
37:                     printf("Open device name: ");
38:                     for (int i=0; i<8; i++) do
39:                         printf("%c", devices-
    name[highestpoint[countwatt][sss]][i]);
40:                     end for
41:                 end if
42:             end for
43:         end if
```

```
44:         ——not find device, using greedy algorithm——
45:         if (highestpoint[countwatt][0]==0) then
46:             int tempremain=countwatt;
47:             int tempwatt=countwatt;
48:             while (tempremain!=0&&tempwatt!=0) do
49:                 tempwatt=tempremain;
50:                 while (highestpoint[tempwatt][0]==0&&(tempwatt!=0)) do
51:                     tempwatt=tempwatt-1;
52:                 end while
53:                 ———greedy find one device———
54:                 if (highestpoint[tempwatt][0]==1) then
55:                     if (savegreedy[0]==0) then
56:                         savegreedy[++savenum]=highestpoint[tempwatt][1];
57:                         savegreedy[0]=savenum;
58:                         printf("Time: %d",remembertimes[1]);
59:                         printf("Increase watt: %d", countwatt);
60:                         printf("Greedy algorithm find device");
61:                         printf("Increase watt: %d", tempwatt);
62:                         printf("Open device name: ");
63:                         for (int i=0; i<8; i++) do
64:                             printf("%c",                    devices-
    name[highestpoint[tempwatt][1]][i]);
65:                         end for
66:                     end if
67:                     if (savenum==1) then
68:                         if (highestpoint[tempwatt][1]!=savegreedy[1]) then
69:                             savegreedy[++savenum]=highestpoint[tempwatt][1];
70:                             savegreedy[0]=savenum;
71:                             printf("Time: %d",remembertimes[1]);
```

```
72:                         printf("Increase watt: %d", countwatt);
73:                         printf("Greedy algorithm find device");
74:                         printf("Increase watt: %d", tempwatt);
75:                         printf("Open device name: ");
76:                         for (int i=0; i<8; i++) do
77:                             printf("%c", devices-
    name[highestpoint[tempwatt][1]][i]);
78:                         end for
79:                     end if
80:                 end if
81:                 if (savenum>1) then
82:                     int j=0;
83:                     for (int i=1; i<=savenum; i++) do
84:                         if (highestpoint[tempwatt][1]!=savegreedy[i]) then
85:                             j++;
86:                         end if
87:                     end for
88:                     if (j==savenum) then
89:                         savegreedy[++savenum]=highestpoint[tempwatt][1];
90:                         savegreedy[0]=savenum;
91:                         printf("Time: %d",remembertimes[1]);
92:                         printf("Increase watt: %d", countwatt);
93:                         printf("Greedy algorithm find device");
94:                         printf("Increase watt: %d", tempwatt);
95:                         printf("Open device name: ");
96:                         for (int i=0; i<8; i++) do
97:                             printf("%c", devices-
    name[highestpoint[tempwatt][1]][i]);
98:                         end for
```

```
99:                         end if
100:                    end if
101:                end if
102:                —greedy find more than one device—
103:                if (highestpoint[tempwatt][0]>1) then
104:                    compare[0]=0;
105:                    for (int s=1; s<=(highestpoint[tempwatt][0]); s++) do
106:                        compare[s]=(newhours[highestpoint [tempwatt] [s]
    [int(remembertimes[1]/3600)] + newtemperatures[highestpoint [tempwatt]
    [s]] [int(AT/5)+1])/2;
107:                    end for
108:                    for (int ss=1; ss<=(highestpoint[tempwatt][0]); ss++) do
109:                        if (compare[ss]>compare[0]) then
110:                            compare[0]=compare[ss];
111:                        end if
112:                    end for
113:                    for (int sss=1; sss<=(highestpoint[tempwatt][0]); sss++)
    do
114:                        if (compare[sss]==compare[0]) then
115:                            if (savegreedy[0]==0) then
116:                                savegreedy[++savenum] = highest-
    point[tempwatt][sss];
117:                                savegreedy[0]=savenum;
118:                                printf("Time: %d",remembertimes[1]);
119:                                printf("Increase watt: %d", countwatt);
120:                                printf("Greedy algorithm find device");
121:                                printf("Increase watt: %d", tempwatt);
122:                                printf("Open device name: ");
123:                                for (int i=0; i<8; i++) do
```

```
124:                                printf("%c", devices-
    name[highestpoint[tempwatt][sss]][i]);
125:                            end for
126:                        end if
127:                        if (savenum==1) then
128:                            if (highestpoint[tempwatt][sss]!=savegreedy[1])
    then
129:                                savegreedy[++savenum]=highestpoint[tempwatt][sss];
130:                                savegreedy[0]=savenum;
131:                                printf("Time: %d",remembertimes[1]);
132:                                printf("Increase watt: %d", countwatt);
133:                                printf("Greedy algorithm find device");
134:                                printf("Increase watt: %d", tempwatt);
135:                                printf("Open device name: ");
136:                                for (int i=0; i<8; i++) do
137:                                    printf("%c", devices-
    name[highestpoint[tempwatt][sss]][i]);
138:                                end for
139:                            end if
140:                        end if
141:                        if (savenum>1) then
142:                            int j=0;
143:                            for (int i=1; i<=savenum; i++) do
144:                                if (highest-
    point[tempwatt][sss]!=savegreedy[i]) then
145:                                    j++;
146:                                end if
147:                            end for
148:                            if (j==savenum) then
```

```
149:                                    savegreedy[++savenum]=highestpoint[tempwatt][sss];
150:                                    savegreedy[0]=savenum;
151:                                    printf("Time: %d",remembertimes[1]);
152:                                    printf("Increase watt: %d", countwatt);
153:                                    printf("Greedy algorithm find device");
154:                                    printf("Increase watt: %d", tempwatt);
155:                                    printf("Open device name: ");
156:                                    for (int i=0; i<8; i++) do
157:                                        printf("%c", devices-
    name[highestpoint[tempwatt][sss]][i]);
158:                                    end for
159:                                end if
160:                            end if
161:                        end if
162:                    end for
163:                end if
164:                tempremain=tempremain-tempwatt;
165:            end while
166:            savegreedy[0]=0;
167:            savenum=0;
168:        end if
169:        countwatt=0;
170:        countremembertimes=0;
171:        remembertimes[0]=0;
172:    end if
173:    oldwatt=newwatt;
174: end procedure
```

5.3.2 Algorithm Description

Step 1:

Create the array $D[1..n]$ of smart meter data for every second. Each record $D[i]$ consists of three values (watt value, time of the day and the temperature).

Step 2:

According to the input array $A[1..m]$, we create several arrays for appliances which holds the data for the appliances consumption in watt, time of the day, the temperature, the of probability percentage of the appliance being switched ON and the appliance name. For example, the watt array $highestpoint[][]$ holds the appearance peak watt, the first index number was indexed by the watt value and the second is the index position which stores corresponding devices.

Step 3:

Loop through each (second) data in $D[1..n]$. For each record inside $D[1..n]$, look for a match appliances data. Either the value in watt can be directly matched through the appliance array of $highestpoint[][]$ or there is a possibility of multiple appliance switched ON at that particular time. Therefore, if it was a direct match then output the appliance name and time. Otherwise, for no match go to Step 4.

Step 4:

Look for the percentage of probability of each appliance to be switched ON at that particular time using the temperature factor and resolve this by the greedy approach whichever matches first based on given probability.

5.3.3 Algorithm Complexity

Space complexity:

For input array $D[1..n]$ with the every second data, the space complexity is $O(n)$, where n is the total number of seconds. For input array $A[1..m]$ with appliances, the space complexity is $O(m)$, where each index of this array holds the data for watt, increment, time with percentage of portabilities, temperature with percentage of probability and appliance name, using common separated storage.

Time complexity:

This approach uses binary search in $A[1..m]$ for each index in $D[1..n]$, for 1 iteration the time complexity is $O(log(m))$ because both Binary search and probabilistic approach inside appliance array takes $O(log(m))$ time. So For n iteration the time complexity is $O(nlog(m))$.

5.4 Discussion and Future Work

This chapter described an algorithmic approach in terms of probabilistic conditions to detect private in-house activities, and identify almost exactly what home appliances were used. We have defined the problem at each second dataset and solved it in an algorithmic manner. The approach and the complexity are promising, an on going experiment is under development to provide more details.

As an improvement to the first step in creating the array it can be an array of watt of applications, WATT[ ][ ], from A[1..m], each index of array means the value of watt sorted, second index of array stored corresponding application's code and that should increase the speed significantly. Also, as a future work the output data can be trained to predict the exact user daily pattern in general on the long term.

Chapter 6

Concluding Remarks

In this thesis we introduced novel solutions to overcome the weakness of existing solutions and solve new open problems in information security.

- The **Malware Detection Techniques** chapter discusses the malware detection which is a well-known problem introduced to computer security research a long time ago, but the rapid development of malware requires more extended solutions since malware writers continually improving their obfuscation techniques to makes the malicious code harder to understand and to evade the malware detectors.

  Therefore, due to the amount of malware signatures generated daily, an efficient algorithm was introduced for detecting these signatures at high speed without excessive memory use in that chapter. Nevertheless, improvements can be made to continue to overcome the compressed malware problem.

  The second part of this chapter introduced a novel solution for a relativity new problem in information security. Recently, malware writers have started to use image steganography to embed either the whole malicious code or a malicious URL in the images. A hidden URL detection in an image approach was introduced and presented in a detailed algorithmic matter in this chapter.

And then the algorithm was taken to the next level to consider detecting and extracting encrypted hidden URLs. The approach analysis and experimental results are very promising since this tool can be consider as one of the first tools to extract the hidden URL as it is from the image, therefore improving this algorithm as well as considering variations to detect malicious attacks by hidden data can be considered as guidelines for future work.

- The **Fingerprint Recognition Techniques** chapter proposed a new fingerprint matching technique, which matches the fingerprint information through using algorithms for approximate circular string matching. The minutiae information is transformed into string information using a series of circles, which intercepts the minutiae of that information into a string. This string fingerprint information is then matched against a database using approximate string matching techniques.

  This chapter proposed a novel fingerprint pattern matching approach for quick and accurate recognition of fingerprints. One overlooked feature in this approach is that the rotation of the fingerprint is assumed to be in sync with the stored image; in this chapter we tackled this issue.

  Also, the novel element is the process of using a series of circles to transform minutiae information into string information consisting of 0s and 1s, and then using the approximate circular string matching algorithm to identify the orientation. This technique is expected to improve the performance and accuracy of the fingerprint verification system.

  For future work we can consider the problem of identification mode where there is a need to match the query fingerprint against a list of fingerprints

in a large database and improve the current solution to cope with a large volume of data.

- In the **Smart Meter Data Analysis Technique** chapter a new privacy problem was introduced to the research area along with the new technology of a smart metre. This technology will bring a great benefit to electricity companies and will have an impact on users' privacy due to constant access by the electricity companies to home electricity readings in real time.

  In this manner, we proposed an algorithmic approach to compare and analyse smart metre data readings, considering time and temperature factors at each second to identify usage patterns in the house by identifying appliance activity.

  The approach time and space complexity are very promising, and an experiment is under development to provide more details. Also, an improvement to the implementation of the algorithm was proposed to improve the speed of the algorithm. Furthermore, as future work we can consider the security aspect of smart metres by analysing the weakness of the device and the possibility of hackers having full control of home electricity – just as the electricity companies do – to control the home electricity through these devices remotely.

References

[1] AGARWAL, A., SHARMA, A. K., AND KHANDELWAL, S. Study of rotation oriented fingerprint authentication. *International Journal of Emerging Engineering Research and Technology 2*, 7 (2014), 211–214.

[2] AHO, A. V., AND CORASICK, M. J. Efficient string matching: an aid to bibliographic search. *Communication ACM 18* (1975), 333–340.

[3] ALATABBI, A. *Advances in Stringology and Applications*. PhD thesis, Natural and Mathematical Sciences, King's College London, 2014.

[4] ALATABBI, A., AL-JAMEA, M., AND ILIOPOULOS, C. S. Malware detection using computational biology tools. *International Journal of Engineering and Technology 5*, 2 (2013), 315.

[5] ALJAMEA, M.AND ILIOPOULOS, C., AND SAMIRUZZAMAN, M. Detection of url in image steganography. In *Proceedings of the 2016 ACM International Conference on Internet of things and Cloud Computing* (2016), ACM. accepted.

[6] ALJAMEA, M.AND BRANKOVIC, L., GAO, J., ILIOPOULOS, C., AND SAMIRUZZAMAN, M. Smart meter data analysis. In *Proceedings of the 2016 ACM International Conference on Internet of things and Cloud Computing* (2016), ACM. accepted.

[7] ALJAMEA, M., AJALA, O., ILIOPOULOS, C. S., AND M, A. Fast fingerprint recognition using circular string pattern matching techniques. In *PATTERNS 2016: The Eighth International Conferences on Pervasive Patterns and Applications* (2016), IARIA.

[8] ALJAMEA, M., ATHAR, T., ILIOPOULOS, C. S., PISSIS, S. P., AND RAHMAN, M. S. A novel pattern matching approach for fingerprint-based authentication. In *PATTERNS 2015: The Seventh International Conferences on Pervasive Patterns and Applications* (2015), IARIA, pp. 45–49.

[9] ALJAMEA, M., GHANAEI, V., ILIOPOULOS, C. S., AND OVERILL, R. E. Static analysis and clustering of malware applying text based search. In *The International Conference on Digital Information Processing, E-Business and Cloud Computing (DIPECC2013)* (2013), The Society of Digital Information and Wireless Communication, pp. 188–193.

[10] ANTONIOU, P., DAYKIN, J., ILIOPOULOS, C., KOURIE, D., MOUCHARD, L., AND PISSIS, S. Mapping uniquely occurring short sequences derived from high throughput technologies to a reference genome. In *Informa-*

tion Technology and Applications in Biomedicine, 2009. ITAB 2009. 9th International Conference on (2009), IEEE, pp. 1–4.

[11] ASK, K. *Automatic malware signature generation*. PhD thesis, Master's thesis, KTH Royal, 2006.

[12] BABATUNDE, I. G., KAYODE, A. B., CHARLES, A. O., AND OLATUBOSUN, O. Fingerprint image enhancement: Segmentation to thinning.

[13] BARTON, C., ILIOPOULOS, C. S., KUNDU, R., PISSIS, S. P., RETHA, A., AND VAYANI, F. Accurate and efficient methods to improve multiple circular sequence alignment. In *Experimental Algorithms*. Springer, 2015, pp. 247–258.

[14] BARTON, C., ILIOPOULOS, C. S., AND PISSIS, S. P. Fast algorithms for approximate circular string matching. *Algorithms for Molecular Biology 9*, 9 (2014).

[15] BIOINFORMATICS ORGANIZATION. Bioinformatics. https://www.bioinformatics.org/. [Last accessed: 10.2015].

[16] BOHLI, J.-M., SORGE, C., AND UGUS, O. A privacy model for smart metering. In *Communications Workshops (ICC), 2010 IEEE International Conference on* (2010), IEEE, pp. 1–5.

[17] BOYER, R., AND MOORE, J. A fast string searching algorithm. *Communications of the ACM 20*, 10 (1977), 762–772.

[18] CEGUERRA, A. V., AND KOPRINSKA, I. Integrating local and global features in automatic fingerprint verification. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on* (2002), vol. 3, IEEE, pp. 347–350.

[19] CHANU, Y. J., TUITHUNG, T., AND MANGLEM SINGH, K. A short survey on image steganography and steganalysis techniques. In *Emerging Trends and Applications in Computer Science (NCETACS), 2012 3rd National Conference on* (2012), IEEE, pp. 52–55.

[20] CHEDDAD, A., CONDELL, J., CURRAN, K., AND MC KEVITT, P. Digital image steganography: Survey and analysis of current methods. *Signal processing 90*, 3 (2010), 727–752.

[21] CHEN, K.-H., HUANG, G.-S., AND LEE, R. C.-T. Bit-Parallel Algorithms for Exact Circular String Matching. *Computer Journal* (2013).

[22] CHEN, X., TIAN, J., AND YANG, X. A new algorithm for distorted fingerprints matching based on normalized fuzzy similarity measure. *Image Processing, IEEE Transactions on 15*, 3 (2006), 767–776.

[23] CHO, H. S., YAMAZAKI, T., AND HAHN, M. Determining location of appliances from multi-hop tree structures of power strip type smart meters. *Consumer Electronics, IEEE Transactions on 55*, 4 (2009), 2314–2322.

[24] CHRISTIANSEN, M. Bypassing malware defenses. *SANS Institute InfoSec Reading Room* (2010), 3–4.

[25] CISCO. Clamav. http://www.clamav.net. [Last accessed: 01.2016].

[26] CLIFFORD, R., AND ILIOPOULOS, C. Approximate string matching for music analysis. *Soft Computing 8*, 9 (2004), 597–603.

[27] COLORNI, A., DORIGO, M., MANIEZZO, V., ET AL. Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life* (1991), vol. 142, Paris, France, pp. 134–142.

[28] CORMEN, T. H. *Introduction to algorithms*. MIT press, 2009.

[29] CROCHEMORE, M., HANCART, C., AND LECROQ, T. *Algorithms on strings*. Cambridge University Press, 2007.

[30] CRYER, J. Image analysis and comparison, 2015.

[31] CSIS. Net losses: Estimating the global cost of cybercrime economic impact of cybercrime. Tech. rep., The Center for Strategic and International Studies, 2014.

[32] DAMODARAN, A., DI TROIA, F., VISAGGIO, C. A., AUSTIN, T. H., AND STAMP, M. A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques* (2015), 1–12.

[33] DELL SECUREWORKS COUNTER THREAT UNIT™ THREAT INTELLIGENCE. Stegoloader: A stealthy information stealer.

http://www.secureworks.com/cyber-threat-intelligence/threats/stegoloader-a-stealthy-information-stealer/. [Last accessed: 11.2015].

[34] DEPARTMENT OF ENERGY AND CLIMATE CHANGE UK GOV. Smart metering implementation programme third annual report on the roll-out of smart meters. [Last accessed: 11.2015].

[35] DEPURU, S. S. S. R., WANG, L., AND DEVABHAKTUNI, V. Smart meters for power grid: Challenges, issues, advantages and status. *Renewable and sustainable energy reviews 15*, 6 (2011), 2736–2742.

[36] E. SATIR, O. K. A distortionless image steganography method via url. In *The 7th International Conference Information Security and Cryptology* (2014).

[37] ELLOUMI, M., HAYATI, P., ILIOPOULOS, C., PISSIS, S., AND SHAH, A. Detection of fixed length web spambot using real (read aligner). In *Proceedings of the CUBE International Information Technology Conference* (2012), ACM, pp. 820–825.

[38] FALLIERE, N., MURCHU, L. O., AND CHIEN, E. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response 5* (2011).

[39] FILIOL, E. Malware pattern scanning schemes secure against black-box analysis. *Journal in Computer Virology 2*, 1 (2006), 35–50.

[40] FITZ, A., AND GREEN, R. Fingerprint classification using a hexagonal fast Fourier transform. *Pattern recognition 29*, 10 (1996), 1587–1597.

[41] FREDRIKSSON, K., AND GRABOWSKI, S. Average-optimal string matching. *Journal of Discrete Algorithms 7*, 4 (2009), 579–594.

[42] FROUSIOS, K., ILIOPOULOS, C. S., MOUCHARD, L., PISSIS, S. P., AND TISCHLER, G. Real: an efficient read aligner for next generation sequencing reads. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology* (New York, NY, USA, 2010), BCB '10, ACM, pp. 154–159.

[43] GIRGIS, M. R., SEWISY, A. A., AND MANSOUR, R. F. A robust method for partial deformed fingerprints verification using genetic algorithm. *Expert Systems with Applications 36*, 2 (2009), 2008–2016.

[44] GRIAULE BIOMETRICS. Online and offline acquisition. http://www.griaulebiometrics.com/en-us/book/. [Last accessed: 01.2016].

[45] GRIFFIN, K., SCHNEIDER, S., HU, X., AND CHIUEH, T.-C. Automatic generation of string signatures for malware detection. In *Recent advances in intrusion detection* (2009), Springer, pp. 101–120.

[46] GUO, J.-M., LIU, Y.-F., CHANG, J.-Y., AND LEE, J.-D. Fingerprint classification based on decision tree from singular points and orientation field. *Expert Systems with Applications 41*, 2 (2014), 752–764.

[47] GUSFIELD, D. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press, 1997.

[48] GUTIERREZ, P., LASTRA, M., HERRERA, F., AND BENITEZ, J. A high performance fingerprint matching system for large databases based on gpu. *IEEE Transactions on Information Forensics and Security 9*, 1 (2014), 62–71.

[49] HARIRI, M., KARIMI, R., AND NOSRATI, M. An introduction to steganography methods. *World Applied Programming 1*, 3 (2011), 191–195.

[50] HENRY, E. R. *Classification and Uses of Finger Prints*. Routledge, 1900.

[51] IANCU, I., AND CONSTANTINESCU, N. Intuitionistic fuzzy system for fingerprints authentication. *Applied Soft Computing 13*, 4 (2013), 2136–2142.

[52] ICANN. List of top-level domains. https://www.icann.org/resources/pages/tlds-2012-02-25-en. [Last accessed: 11.2015].

[53] IDIKA, N., AND MATHUR, A. A survey of malware detection techniques. *Purdue University* (2007), 48.

[54] ISENOR, D., AND ZAKY, S. G. Fingerprint identification using graph matching. *Pattern Recognition 19*, 2 (1986), 113–122.

[55] JAIN, A., ROSS, A., AND PRABHAKAR, S. Fingerprint matching using minutiae and texture features. In *Image Processing, 2001. Proceedings. 2001 International Conference on* (2001), vol. 3, IEEE, pp. 282–285.

[56] JAIN, A. K., HONG, L., PANKANTI, S., AND BOLLE, R. An identity-authentication system using fingerprints. *Proceedings of the IEEE 85*, 9 (1997), 1365–1388.

[57] JAIN, A. K., PRABHAKAR, S., AND HONG, L. A multichannel approach to fingerprint classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence 21*, 4 (1999), 348–359.

[58] JAIN, A. K., PRABHAKAR, S., HONG, L., AND PANKANTI, S. Filterbank-based fingerprint matching. *Image Processing, IEEE Transactions on 9*, 5 (2000), 846–859.

[59] JANG, J., ELLIOTT, S. J., AND KIM, H. On improving interoperability of fingerprint recognition using resolution compensation based on sensor evaluation. In *International Conference on Biometrics* (2007), Springer, pp. 455–463.

[60] JIANG, X., AND YAU, W.-Y. Fingerprint minutiae matching based on the local and global structures. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on* (2000), vol. 2, IEEE, pp. 1038–1041.

[61] JOHNSON, N. F., AND JAJODIA, S. Exploring steganography: Seeing the unseen. *Computer 31*, 2 (1998), 26–34.

[62] KAI, C., XIN, Y., XINJIAN, C., YALI, Z., JIMIN, L., AND JIE, T. A novel ant colony optimization algorithm for large-distorted fingerprint matching. *Pattern Recognition 45*, 1 (2012), 151–161.

[63] KAK, A. Lecture notes on "computer and network security", 2013.

[64] KALOGRIDIS, G., EFTHYMIOU, C., DENIC, S. Z., LEWIS, T., CEPEDA, R., ET AL. Privacy for smart meters: Towards undetectable appliance load signatures. In *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on* (2010), IEEE, pp. 232–237.

[65] KHAN, A. I., AND WANI, M. A. Efficient and rotation invariant fingerprint matching algorithm using adjustment factor. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)* (2015), IEEE, pp. 1103–1110.

[66] KOVACS-VAJNA, Z. M. A fingerprint verification system based on triangular matching and dynamic time warping. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 22*, 11 (2000), 1266–1276.

[67] KWAC, J., TAN, C.-W., SINTOV, N., FLORA, J., AND RAJAGOPAL, R. Utility customer segmentation based on smart meter data: Empirical study. In *Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on* (2013), IEEE, pp. 720–725.

[68] KWAC, J., TAN, C.-W., SINTOV, N., FLORA, J., AND RAJAGOPAL, R. Utility customer segmentation based on smart meter data: Empirical

study. In *Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on* (2013), IEEE, pp. 720–725.

[69] LEE, H. C., RAMOTOWSKI, R., AND GAENSSLEN, R. E., Eds. *Advances in Fingerprint Technology, Second Edition*. CRC Press, 2002.

[70] LI, J., YAU, W.-Y., AND WANG, H. Combining singular points and orientation image information for fingerprint classification. *Pattern Recognition 41*, 1 (2008), 353–366.

[71] LIN, H., JAIN, A., PANKANTI, S., AND BOLLE, R. Identity authentication using fingerprints. In *Audio and Video based Biometric Person Authentication* (1997), Springer, pp. 103–110.

[72] LIU, M. Fingerprint classification based on adaboost learning from singularity features. *Pattern Recognition 43*, 3 (2010), 1062–1070.

[73] LOTHAIRE, M., Ed. *Algebraic Combinatorics on Words*. Cambridge University Press, 2001.

[74] LOTHAIRE, M. *Applied Combinatorics on Words*. Cambridge University Press, 2005.

[75] LUSCOMBE, N. M., GREENBAUM, D., AND GERSTEIN, M. What is bioinformatics? an introduction and overview. *Yearbook of Medical Informatics 1* (2001), 83–99.

[76] MAIO, D., AND MALTONI, D. A structural approach to fingerprint classification. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on* (1996), vol. 3, IEEE, pp. 578–585.

[77] MALTONI, D., MAIO, D., JAIN, A. K., AND PRABHAKAR, S. *Handbook of Fingerprint Recognition*. Springer-Verlag, 2009.

[78] MARCEAU, M. L., AND ZMEUREANU, R. Nonintrusive load disaggregation computer program to estimate the energy consumption of major end uses in residential buildings. *Energy Conversion and Management 41*, 13 (2000), 1389–1403.

[79] MOHAPATRA, C., AND PANDEY, M. A review on current methods and application of digital image steganography. *International Journal of Multidisciplinary Approach & Studies 2*, 2 (2015).

[80] MOLINA-MARKHAM, A., SHENOY, P., FU, K., CECCHET, E., AND IRWIN, D. Private memoirs of a smart meter. In *Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building* (2010), ACM, pp. 61–66.

[81] MORKEL, T., ELOFF, J. H., AND OLIVIER, M. S. An overview of image steganography. In *ISSA* (2005), pp. 1–11.

[82] NAGATY, K. A. Fingerprints classification using artificial neural networks: a combined structural and statistical approach. *Neural Networks 14*, 9 (2001), 1293–1305.

[83] NATARAJ, L., JACOB, G., AND B.S.MANJUNATH. Detecting packed executables based on raw binary data. Tech. rep., University of California, Santa Barbara, Jun 2010.

[84] NATIONAL CENTER FOR BIOTECHNOLOGY INFORMATION (NCBI). Bioinformatics. http://www.ncbi.nlm.nih.gov/Class/MLACourse/Modules/MolBioReview/bioinformatics.html. [Accessed Feb 1, 2016].

[85] NATIONAL CRIMINAL JUSTICE REFERENCE SERVICE. *The Fingerprint Sourcebook*. CreateSpace Independent Publishing Platform, 2014.

[86] NATIONAL HUMAN GENOM INSITIUTE (NHGRI). Deoxyribonucleic acid (DNA). https://www.genome.gov/. [Last accessed: 01.2016].

[87] NATURE EDUCATION. DNA sequencing. http://www.nature.com/scitable/content/dna-sequencing-6656663. [Last accessed: 01.2016].

[88] NAVARRO, G. A guided tour to approximate string matching. *ACM Comput. Surv. 33*, 1 (2001), 31–88.

[89] NIST. Biometric special databases and software. http://www.nist.gov/itl/iad/igspecial_dbases.cfm, 2015. [retrieved: 11.2015].

[90] OPENCV-CODE. Implementation of guo-hall thinning algorithm, 2015. [retrieved: 11.2015].

[91] PATTICHIS, M. S., PANAYI, G., BOVIK, A. C., AND HSU, S.-P. Fingerprint classification using an am-fm model. *IEEE Transactions on Image Processing 10*, 6 (2001), 951–954.

[92] PERDISCI, R., LANZI, A., AND LEE, W. Classification of packed executables for accurate computer virus detection. *Pattern Recognition Letters 29*, 14 (2008), 1941–1946.

[93] PEREZ-DIAZ, A., AND ARRONTE-LOPEZ, I. Fingerprint matching and non-matching analysis for different tolerance rotation degrees in commercial matching algorithms. *Journal of applied research and technology 8*, 2 (2010), 186–199.

[94] PISSIS, S. Lecture notes in algorithms for computational molecular biology, February 2016.

[95] PISSIS, S. P. Approximate string matching via filtering - implementation code. https://github.com/solonas13/asmf/, 2015. [retrieved: 10.2015].

[96] PREDA, M., CHRISTODORESCU, M., JHA, S., AND DEBRAY, S. A semantics-based approach to malware detection. *ACM Transactions on Programming Languages and Systems (TOPLAS) 30*, 5 (2008), 25.

[97] PROBER, J. M., TRAINOR, G. L., DAM, R. J., HOBBS, F. W., ROBERTSON, C. W., ZAGURSKY, R. J., COCUZZA, A. J., JENSEN, M. A., AND BAUMEISTER, K. A system for rapid dna sequencing with fluorescent chain-terminating dideoxynucleotides. *Science 238*, 4825 (1987), 336–341.

[98] PROVOS, N., AND HONEYMAN, P. Hide and seek: An introduction to steganography. *Security & Privacy, IEEE 1*, 3 (2003), 32–44.

[99] Q. ZHANG, K. H., AND YAN, H. Fingerprint classification based on extraction and analysis of singularities and pseudoridges. In *Fingerprint Classification Based on Extraction and Analysis of Singularities and Pseudoridges* (Sydney, Australia, 2001), VIP 2001, VIP.

[100] REN, C., GUO, J., QIU, D., CHANG, G., AND WU, Y. A framework of fingerprint scaling. *Indonesian Journal of Electrical Engineering and Computer Science 11*, 3 (2013), 1547–1559.

[101] REN, C.-X., YIN, Y.-L., MA, J., AND LI, H. Fingerprint scaling. In *International Conference on Intelligent Computing* (2008), Springer, pp. 474–481.

[102] RESEARCHERS, C. L. Malware detection and classification.

[103] ROSS, A., AND JAIN, A. Biometric sensor interoperability: A case study in fingerprints. In *Biometric Authentication*. Springer, 2004, pp. 134–145.

[104] SALEH, A. A., AND ADHAMI, R. R. Curvature-based matching approach for automatic fingerprint identification. In *System Theory, 2001. Proceedings of the 33rd Southeastern Symposium on* (2001), IEEE, pp. 171–175.

[105] SALEH, A. M., ELDIN, A. M. B., AND WAHDAN, A.-M. A. A modified thinning algorithm for fingerprint identification systems. In *Computer*

Engineering & Systems, 2009. ICCES 2009. International Conference on (2009), IEEE, pp. 371–376.

[106] SATHYANARAYAN, V. S., KOHLI, P., AND BRUHADESHWAR, B. Signature generation and detection of malware families. *Lecture Notes in Computer Science 5107* (2008), 336–349.

[107] SEBASTIAN, S. Literature survey on automated person identification techniques. *International Journal of Computer Science and Mobile Computing 2*, 5 (2013), 232–237.

[108] SEBASTIAN, S. Literature survey on automated person identification techniques. *International Journal of Computer Science and Mobile Computing 2*, 5 (May 2013), 232–237.

[109] SEDGEWICK, R., AND WAYNE, K. Radix sorts. https://www.cs.princeton.edu/~rs/AlgsDS07/18RadixSort.pdf, 2014. [Last accessed: 01.2016].

[110] SENIOR, A. A combination fingerprint classifier. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 10 (2001), 1165–1174.

[111] SHAFFER, C. A. *Data Structures & Algorithm Analysis in Java*. Courier Corporation, 2011.

[112] SHAFFER, C. A. Data structures and algorithm analysis. *Update 3* (2012), 0–3.

[113] SKIENA, S. S. *The algorithm design manual*, vol. 2. Springer Science & Business Media, 2008.

[114] SMYTH, B. *Computing Patterns in Strings*. Pearson Addison-Wesley, 2003.

[115] SOFTONIC. Xiao steganography. http://xiao-steganography.en.softonic.com/. [Last accessed: 10.2015].

[116] SPEIR, J. A., AND HIETPAS, J. Frequency filtering to suppress background noise in fingerprint evidence: Quantifying the fidelity of digitally enhanced fingerprint images. *Forensic science international 242* (2014), 94–102.

[117] STONE-GROS, B. Malware analysis of the lurk downloader. http://www.secureworks.com/cyber-threat-intelligence/threats/malware-analysis-of-the-lurk-downloader/?view=Standard, 2014. Dell SecureWorks Counter Threat Unit, [Last accessed: 11.2015].

[118] SWEENEY, A. M. Malware analysis and antivirus signature creation.

[119] SZOR, P. *The art of computer virus research and defense*. Addison-Wesley Professional, 2005.

[120] TAN, X., AND BHANU, B. Fingerprint verification using genetic algorithms. In *Sixth IEEE Workshop on Applications of Computer Vision* (2002), IEEE, pp. 79–83.

[121] TAN, X., AND BHANU, B. Robust fingerprint identification. In *International Conference on Image Processing 2002* (2002), vol. 1, IEEE, pp. I–277.

[122] TAN, X., AND BHANU, B. Fingerprint matching by genetic algorithms. *Pattern Recognition 39*, 3 (2006), 465–477.

[123] TAN, X., BHANU, B., AND LIN, Y. Fingerprint classification based on learned features. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 35*, 3 (2005), 287–300.

[124] THE PRAGUE STRINGOLOGY CLUB. Introduction to stringology. http://www.stringology.org/, 2016. [Accessed Feb 2, 2016].

[125] THERMO FISHER SCIENTIFIC INC. appliedbiosystems. http://www.thermofisher.com/uk/en/home/brands/applied-biosystems.html, 2016. Accessed Feb 2, 2016.

[126] UKKONEN, E. Finding approximate patterns in strings. *Journal of algorithms 6*, 1 (1985), 132–137.

[127] UMASS TRACE REPOSITORY. Smart* data set for sustainability project. http://traces.cs.umass.edu/index.php/Smart/Smart. [Last accessed: 01.2016].

[128] UNARA, J., SENGA, W. C., AND ABBASI, A. A review of biometric technology along with trends and prospects. *Pattern Recognition 47*, 8 (August 2014), 2673—2688.

[129] USA DEPARTMENT OF STATE FREEDOM OF INFORMATION ACT (FOIA). The privacy act. https://foia.state.gov/learn/privacyact.aspx. [Last accessed: 01.2016].

[130] UZ, T., BEBIS, G., EROL, A., AND PRABHAKAR, S. Minutiae-based template synthesis and matching for fingerprint authentication. *Computer Vision and Image Understanding 113*, 9 (2009), 979–992.

[131] VIRUSTOTAL. File statistics during last 7 days. Tech. rep., Rotarua Limited (d.b.a. VirusTotal), 2016.

[132] WEISS, M., HELFENSTEIN, A., MATTERN, F., AND STAAKE, T. Leveraging smart meter data to recognize home appliances. In *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on* (2012), IEEE, pp. 190–197.

[133] WILLIAMS, R. Cyber crime costs global economy $445 bn annually. http://www.telegraph.co.uk/technology/internet-security/10886640/Cyber-crime-costs-global-economy-445-bn-annually.html, 2014. The Telegraph , [Last accessed: 01.2016].

[134] WILLIS, A. J., AND MYERS, L. A cost-effective fingerprint recognition system for use with low-quality prints and damaged fingertips. *Pattern recognition 34*, 2 (2001), 255–270.

[135] XINJIAN, C., JIE, T., XIN, Y., AND YANGYANG, Z. An algorithm for distorted fingerprint matching based on local triangle feature set. *Information Forensics and Security, IEEE Transactions on 1*, 2 (2006), 169–177.

[136] YAO, Y., MARCIALIS, G. L., PONTIL, M., FRASCONI, P., AND ROLI, F. Combining flat and structured representations for fingerprint classification with recursive neural networks and support vector machines. *Pattern Recognition 36*, 2 (2003), 397–406.

[137] YOU, I., AND YIM, K. Malware obfuscation techniques: A brief survey. In *Int. Conf. on Broadband, Wireless Computing, Communication and Applications* (2010), pp. 297–300.

[138] ZANG, Y., YANG, X., JIA, X., ZHANG, N., TIAN, J., AND ZHU, X. A coarse-fine fingerprint scaling method. In *2013 International Conference on Biometrics (ICB)* (2013), IEEE, pp. 1–6.

[139] ZENG, Y., LIU, F., LUO, X., AND YANG, C. Formal description and analysis of malware detection algorithm mom a. In *International Symposium on Computer Science and Computational Technology* (2009), pp. 139–142.

[140] ZHAN, X., SUN, Z., YIN, Y., AND CHU, Y. Fingerprint ridge distance estimation: algorithms and the performance. In *International Conference on Biometrics* (2006), Springer, pp. 294–301.

[141] ZHANG, Q., AND YAN, H. Fingerprint classification based on extraction and analysis of singularities and pseudo ridges. *Pattern Recognition 37*, 11 (2004), 2233–2243.

[142] ZHANG, T., AND SUEN, C. Y. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM 27*, 3 (1984), 236–239.

[143] ZHOU, X., XI, B., QI, Y., AND LI, J. Mrsi: A fast pattern matching algorithm for anti-virus applications. In *Networking, 2008. ICN 2008. Seventh International Conference on* (2008), IEEE, pp. 256–261.

Appendix

.1 URL Detection Algorithm

.1.1 Detecting and Extracting Hidden URL

```
1:  procedure FindURLInImage
2:      CREATE a sorted indexed array DOMAIN[] from the official top level
    domain list
3:      CREATE a BITMAP[] array from the image taking each bit
4:      *Comment: Loop through the BITMAP[] and create an array LSBAR-
    RAY[] with the least significant bits
5:      Integer i, j
6:      i=0
7:      j=0
8:      for  i = 0 to BITMAP[] do
9:          if (i != 0) AND (i+1) MOD 8 = 0 then
10:             return LSBARRAY[j++] = BITMAP[i]
11:         end if
12:     end for
13:     *Comment: Loop through LSBARRAY[] and convert to a LSBCHARAR-
    RAY[] character array
14:     i=0
15:     j=0
16:     String t=""
```

```
17:     for  i = 0 to LSBARRAY[] do t = STRING((t) + LSBARRAY[i])
18:         if (i!=0) and (i+1) MOD 8 = 0 then
19:             return LSBCHARARRAY[j++] = ConvertToCharacter(t)
20:             t = ""
21:         end if
22:     end for
23:     *Comment: Loop through the LSBCHARARRAY[] to detect URL by
    using the DOMAIN[] array Integer temp, l,s
24:     ' Initialize i and s outside the loop
25:     i=0
26:     s=0
27:     Boolean httpOrHttpsExists
28:     Boolean wwwExists
29:     Boolean urlFound
30:     String URL = ""
31:     String OutPutURLArray[]
32:     for i=0 to LSBCHARARRAY[] do
33:         ' Initializeat start of loop
34:         httpOrHttpsExists = False
35:         wwwExists = False
36:         urlFound = False
37:         URL = ""
38:         J=0
39:         t=""
40:         temp = 0
41:         if LSBCHARARRAY[i] = ":" then
42:             *Comment:Check Possibility of having an http:\\
43:             t = ConvertToString(LSBCHARARRAY[i-4] to LSBCHARAR-
    RAY[i+2])
44:             if LowerCase(t) = "http:\\" then
```

```
45:                 httpOrHttpsExists = True
46:                  temp = i + 3
47:                 URL= "http:\\"
48:             end if
49:             t = ""
50:             if httpOrHttpsExists = False then
51:                 *Comment:Possibility of having an https:\\
52:                   t = ConvertToString(LSBCHARARRAY[i-5] to LSBCHARARRAY[i+2])
53:                 if LowerCase(t) = "https:
54: " then
55:                     httpOrHttpsExists = True
56:                     temp = i + 3
57:                     URL= "https:\\"
58:                 end if
59:             end if
60:              t = ""
61:             t = ConvertToString(LSBCHARARRAY[i+3] to LSBCHARARRAY[i+6])
62:             if LowerCase(t) = "www." then
63:                 temp = temp+ i + 5
64:                 wwwExists = True
65:                 URL= Concat(URL,"www.")
66:             end if
67:         end if
68:         if (httpOrHttpsExists = False Or wwwExists = False) AND LSBCHARARRAY[i] = "." then
69:             *Comment: Check for . to find www because at this point we know that http or www trap inside the condition for ":" failed.
```

```
70:             t = ""
71:             t = ConvertToString(LSBCHARARRAY[i-3] to LSBCHARAR-
    RAY[i])
72:             if LowerCase(t) = "www."  then
73:                 *Comment: jump the i to the new position and save in a tem-
    morary variable
74:                 temp = i + 5
75:                 wwwExists = True
76:                 URL= "www."
77:             end if
78:         end if
79:         if httpOrHttpsExists = True Or wwwExists = True then
80:             *Comment: Assign the position of i to find the URL
81:             i = temp
82:             t = ""
83:             'At this point the existence of http of www is found. Now look for
    the rest of the url
84:             for j = i to LSBCHARARRAY[] do
85:                 if LSBCHARARRAY[j] = "." then
86:                     URL = Concat(URL, ConvertToString(LSBCHARARRAY[j-
    i+1] to LSBCHARARRAY[j]))
87:                     i = j + 1
88:                     urlFound = True
89:                     Exit FOR
90:                 end if
91:             end for
92:             if urlFound then
93:                 urlFound = False
```

Algorithm 3 Procedure FindURLInImage

```
94:                 for j = i to LSBCHARARRAY[] do
95:                     t = Concat(t,LSBCHARARRAY[j])
96:                     *Comment: Now check t in sorted top level domain list
    DOMAIN
97:                     if t EXISTS in DOMAIN[] then
98:                         URL = Concat(URL,
    ConvertToString(LSBCHARARRAY[j-i+1] to LSBCHARARRAY[j]))
99:                         urlFound = True
100:                        *Comment: Reinitialize the value of i for the next
    iteration
101:                        i = j + 1
102:                        EXIT FOR
103:                    end if
104:                end for
105:            end if
106:        end if
107:        if urlFound then
108:            *Comment: It is possible to have multiple URL in different posi-
    tion
109:            OutPutURLArray[s] = URL
110:            s = s + 1
111:        end if
112:    end for
113:    if s > 0 THEN then
114:        *Comment: URL has been found and OutPutURLArray[] contains
    the urls
115:        return OutPutURLArray[]
116:    end if
117: end procedure
```

.1.2 Detecting and Extracting Encrypted URL

```
1:  procedure FindURLInImage
2:      CREATE a sorted indexed array DOMAIN[] from the official top level
    domain list
3:      CREATE a BITMAP[] array from the image taking each bit
4:      *Comment: Loop through the BITMAP[] and create an array LSBAR-
    RAY[] with the Inverted least significant bits
5:      Integer i, j
6:      i=0
7:      j=0
8:      for  i = 0 to BITMAP[] do
9:          if (i != 0) AND (i+1) MOD 8 = 0 then
10:             return LSBARRAY[j++] = BIT WISE NOT BITMAP[i]
11:         end if
12:     end for
13:     *Comment: Loop through LSBARRAY[] and convert to a LSBCHARAR-
    RAY[] character array
14:     i=0
15:     j=0
16:     String t=""
17:     for  i = 0 to LSBARRAY[] do t = STRING((t) + LSBARRAY[i])
18:         if (i!=0) and (i+1) MOD 8 = 0 then
19:             return LSBCHARARRAY[j++] = ConvertToCharacter(t)
20:             t = ""
21:         end if
22:     end for
```

```
23:     *Comment: Loop through the LSBCHARARRAY[] to detect URL by
    using the DOMAIN[] array Integer temp, l,s
24:     ' Initialize i and s outrside the loop
25:     i=0
26:     s=0
27:     Boolean httpOrHttpsExists
28:     Boolean wwwExists
29:     Boolean urlFound
30:     String URL = ""
31:     String OutPutURLArray[]
32:     for i=0 to LSBCHARARRAY[] do
33:         ' Initializeat start of loop
34:         httpOrHttpsExists = False
35:         wwwExists = False
36:         urlFound = False
37:         URL = ""
38:         J=0
39:         t=""
40:         temp = 0
41:         if LSBCHARARRAY[i] = ":" then
42:             *Comment:Check Possibility of having an http:\\
43:             t = ConvertToString(LSBCHARARRAY[i-4] to LSBCHARAR-
    RAY[i+2])
44:             if LowerCase(t) = "http:\\" then
45:                 httpOrHttpsExists = True
46:                  temp = i + 3
47:                 URL= "http:\\"
48:             end if
```

```
49:             t = ""
50:             if httpOrHttpsExists = False then
51:                 *Comment:Possibility of having an https:\\
52:                   t = ConvertToString(LSBCHARARRAY[i-5] to LSBCHARARRAY[i+2])
53:                 if LowerCase(t) = "https:
54: " then
55:                     httpOrHttpsExists = True
56:                     temp = i + 3
57:                     URL= "https:\\"
58:                 end if
59:             end if
60:              t = ""
61:             t = ConvertToString(LSBCHARARRAY[i+3] to LSBCHARARRAY[i+6])
62:             if LowerCase(t) = "www." then
63:                 temp = temp+ i + 5
64:                 wwwExists = True
65:                 URL= Concat(URL,"www.")
66:             end if
67:         end if
68:         if httpOrHttpsExists = False Or wwwExists = False) AND LSBCHARARRAY[i] = "." then
69:             *Comment: Check for . to find www because at this point we know that http or www trap inside the condition for ":" failed.
```

```
70:             t = ""
71:             t = ConvertToString(LSBCHARARRAY[i-3] to LSBCHARAR-
    RAY[i])
72:             if LowerCase(t) = "www."  then
73:                 *Comment: jump the i to the new position and save in a tem-
    morary variable
74:                 temp = i + 5
75:                 wwwExists = True
76:                 URL= "www."
77:             end if
78:         end if
79:         if httpOrHttpsExists = True Or wwwExists = True then
80:             *Comment: Assign the position of i to find the URL
81:             i = temp
82:             t = ""
83:             'At his point the existence of http of www is found. Now look for
    the rest of the url
84:             for j = i to LSBCHARARRAY[] do
85:                 if LSBCHARARRAY[j] = "." then
86:                     URL = Concat(URL, ConvertToString(LSBCHARARRAY[j-
    i+1] to LSBCHARARRAY[j]))
87:                     i = j + 1
88:                     urlFound = True
89:                     Exit FOR
90:                 end if
91:             end for
92:             if urlFound then
93:                 urlFound = False
94:                 for j = i to LSBCHARARRAY[] do
95:                     t = Concat(t,LSBCHARARRAY[j])
```

Algorithm 4 Procedure FindURLInImage

```
96:                     *Comment: Now check t in sorted top level domain list
    DOMAIN
97:                     if t EXISTS in DOMAIN[] then
98:                         URL = Concat(URL,
    ConvertToString(LSBCHARARRAY[j-i+1] to LSBCHARARRAY[j]))
99:                         urlFound = True
100:                        *Comment: Reinitialize the value of i for the next
    iteration
101:                        i = j + 1
102:                        EXIT FOR
103:                    end if
104:                end for
105:            end if
106:        end if
107:        if urlFound then
108:            *Comment: It is possible to have multiple URL in different posi-
    tion
109:            OutPutURLArray[s] = URL
110:            s = s + 1
111:        end if
112:    end for
113:    if s > 0 THEN then
114:        *Comment: URL has been found and OutPutURLArray[] contains
    the urls
115:        return OutPutURLArray[]
116:    end if
117: end procedure
```

.1.3 URL Detection code

The first solution is on the following link:

http://tanvera-001-site1.htempurl.com/

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.IO;
using System.Drawing;
using Steganography;

public partial class _Default : Page
{
     private string extractedText = string.Empty;


    protected void Page_Load(object sender, EventArgs e)
    {



    }

    private bool IsValidFile(string filePath)
    {
        bool isValid = false;
        string[] fileExtensions = { ".bmp", ".png", ".BMP", ".PNG", ".gif",".GIF" };
        for (int i = 0; i < fileExtensions.Length; i++)
        {
            if (filePath.Contains(fileExtensions[i]))
            {
                isValid = true;
            }

        }
        return isValid;
    }



    protected void ExtractURL_Click(object sender, EventArgs e)
    {
        Bitmap bmp = null;

        ExtractedText.Text = "";
        ErrorTxt.Text = "";
        Image1.ImageUrl = "";
        //string fullPath = ImageFileUpload.HttpPostedFile.FileName;
```

```
    try
    {

        if (ImageFileUpload.HasFile)
        {
            if (IsValidFile(Convert.ToString(ImageFileUpload.PostedFile.FileName)))
            {
                string dirUrl = "uploads" + this.Page.User.Identity.Name;
                string dirPath = Server.MapPath(dirUrl);
                string fileUrl = dirUrl + "/" + Path.GetFileName(ImageFileUpload.PostedFile.FileName);
                if (File.Exists(Server.MapPath(fileUrl)))
                {
                    File.Delete(Server.MapPath(fileUrl));
                }
                ImageFileUpload.PostedFile.SaveAs(Server.MapPath(fileUrl));
                Image1.ImageUrl = fileUrl;
                using (bmp = new Bitmap(Server.MapPath(fileUrl).ToString()))
                {
                    string extractedText = SteganographyURL.extractURL(bmp);
                    if (extractedText.Equals(""))
                    {
                        ErrorTxt.Text = "No URL Found";
                    }
                    else
                    {
                        ExtractedText.Text = extractedText;
                    }
                }
            }
            else
            {
                ErrorTxt.Text = "this file type is not supported";
            }
        }
        else
        {
            ErrorTxt.Text = "Select a bmp or png or gif iamge";
        }
     }
     catch (Exception Exc)
    {

        ErrorTxt.Text = Exc.Message;
        if (bmp != null) bmp.Dispose();
        bmp = null;

    }
    bmp = null;
}

private static byte[] BitmapToBytes(Bitmap img)
{
    using (MemoryStream stream = new MemoryStream())
    {
        img.Save(stream, System.Drawing.Imaging.ImageFormat.Png);
```

```
        return stream.ToArray();
    }
}

protected void HideURL_Click(object sender, EventArgs e)
{
    Bitmap bmp = null;
    Random rnd = new Random();
    string rndString = rnd.Next().ToString() ;

    ExtractedText.Text = "";
    ErrorTxt.Text = "";
    Image1.ImageUrl = "";

    try
    {
        if (URLText.Text == "")
        {
            ErrorTxt.Text = "URL Text cannot be blank";

        }
        else
        if (ImageFileUpload.HasFile)
        {
            if (SteganographyURL.ValidURL(URLText.Text))
            {
                Image1.ImageUrl = "";
                if (IsValidFile(Convert.ToString(ImageFileUpload.PostedFile.FileName)))
                {
                    if (Image1.ImageUrl == "")
                    {
                        string dirUrl = "uploads" + this.Page.User.Identity.Name;
                        string dirPath = Server.MapPath(dirUrl);
                        string fileUrl = dirUrl + "/" + Path.GetFileName(ImageFileUpload.PostedFile.FileName);
                        fileUrl = fileUrl.Replace(".", rndString + "_.");
                        if (File.Exists(Server.MapPath(fileUrl)))
                        {
                            File.Delete(Server.MapPath(fileUrl));
                        }
                        ImageFileUpload.PostedFile.SaveAs(Server.MapPath(fileUrl));
                        Image1.ImageUrl = fileUrl;

                        using (bmp = new Bitmap(Server.MapPath(fileUrl).ToString()))
                        {
                            if (bmp.PixelFormat.ToString() == "Format1bppIndexed")
                            {
                                ErrorTxt.Text = "This image format is not supported, the
                                 image file must have at least 8 bits per pixel";
                                bmp = null;
                            }
                            else
                            if (bmp != null)
                            {
                                bmp = SteganographyURL.embedText(URLText.Text, bmp);
                                byte[] bytes =BitmapToBytes(bmp);
```

```
                                string base64String = Convert.ToBase64String(bytes, 0, bytes.Length);
                                Image1.ImageUrl= "" + base64String;
                                ErrorTxt.Text = "URL has been hidden in the image";
                                ExtractedText.Text = "";
                            }
                            else
                            {
                                ErrorTxt.Text = "Error saving image";
                            }
                        }
                    }


                }
                else
                {
                    ErrorTxt.Text = "This image format is not supported";

                }
            }
            else
            {
                ErrorTxt.Text = "The Text is not a valid URL";
            }
        }
    }
    catch (Exception ex)
    {
        if (ex.Message == "SetPixel is not supported for images with indexed pixel formats.")
        {
            ErrorTxt.Text = "This image does not support URL hidding";
        }
        else
        {
            ErrorTxt.Text = "Error Saving file " + ex.Message + " " + ex.Data.ToString();
        }
        bmp = null;
        if (bmp != null) bmp.Dispose();
    }
    bmp = null;

  }
}
```

.2 Fingerprint Implementation code

.2.1 Main Function

```
/**
 * fp_auth.cc: novel fingerprint auth implementation
```

```
 *
 */

#include "fp_auth.h"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <vector>

// read input img and to_match img
// get scan circles from to_match_img
// get scan circles at regular intervals from img
// match each circle with to_match_img
// check for best match

int main(int argc, char** argv){

        int match_result=0,match=0;
        int m=0,n=0;
        int inner_circles;
        int MIN_RADIUS, RAD_DIST;

        cv::Mat im = imread("/home/k1505471/Downloads/src/db/104_7_thin.tif", 0);
         // input image, ex. "img_bw/104_7_t_.tif"

                  //Image pre-processing
        //resize(im,im,im.size(),150,150,INTER_LINEAR );

                        //  cv::cvtColor(im, im, CV_BGR2GRAY);
                        // cv::adaptiveThreshold(im,im,255,ADAPTIVE_THRESH_GAUSSIAN_C,THRESH_BINARY,15,-11);

                        thinning(im);
                        //Converting to black and white
                          //cv::threshold(im, im, 0,255,THRESH_BINARY_INV);



        // load image to match
        cv::Mat im_match = imread("/home/k1505471/Downloads/src/db/104_7_thin.tif", 0);
         // image in DB to be matched against


                  //Image pre-processing
                            //cv::resize(im_match,im_match,im_match.size(),150,150,INTER_LINEAR );
                         // cv::cvtColor(im_match, im_match, CV_BGR2GRAY);
                         //cv::adaptiveThreshold(im_match,im_match,255,ADAPTIVE_THRESH_GAUSSIAN_C,THRESH_BINARY,15,-11);

                         //Fingerprint Thinning
                        thinning(im_match);
                        //Converting to black and white
                          //cv::threshold(im_match, im_match, 0,255,THRESH_BINARY_INV);



    /*////////////////////////////////////////////////////////*/
```

```
        cout<<"Enter the Min Raduis ";
        cin>>MIN_RADIUS;
        cout<<"Enter the  Raduis Dist ";
    cin>>RAD_DIST;
        cout<<"Enter the number of inner cirlces ";
        cin>>inner_circles;
        //get one circle per pixel in a fixed raduis for the input image

        vector< vector< vector<string> > > scans = get_scans(im, MIN_RADIUS, 1,1);
         // 2d array of scan circles
        vector< vector< vector<string> > > scans_match = get_scans(im_match, MIN_RADIUS,1,1);
         std::vector< vector< vector<int> > >x_y_axis;


x_y_axis.resize(scans.size());

for(int i=0;i<scans.size();i++){
        x_y_axis[i].resize(scans.size());
}
for(int i=0;i<scans.size();i++){
        for(int j=0;j<scans.size();j++){
                x_y_axis[i][j].resize(2);
        }

}

/*for(int i=0;i<scans.size();i++){
        for(int j=0;j<scans[i].size();j++){
                cout<<i; cout<<" "; cout<<j;
                cout<<endl;
cout<<x_y_axis[i][j][0];
        cout<<x_y_axis[i][j][1];
        cout<<endl;

        }

}*/

/*for(int i=0;i<scans.size();i++){
        for(int j=0;j<scans.size();j++){
                cout<<scans[i][j][0];
                cout<<endl;
        }

        }*/

string lexstring=lex(scans_match[scans.size()/2][scans.size()/2][0]);
string lexstring1;


for(int i=0;i<scans.size();i++){
for(int j=0;j<scans.size();j++){
```

```
        lexstring1=lex(scans[i][j][0]);
                match_result=needlemanWunschMatching(lexstring1,lexstring,lexstring.length());

        if(match_result>=50){
         cout<<match_result;
                cout<<endl;}}}
        //      match++;
//x_y_axis[m][n][0]=i;
//x_y_axis[m][n][1]=j;
        //      cout<<"match ";
                //cout<<match_result;
                //cout<<endl;

         //n++;}
//}
//if(match_result>=10)
//m++;
//}
//cout<<match;cout<<endl;
//cout<<m;
//cout<<" ";
//cout<<n;
//cout<<m; cout<<" "; cout<<n;*/

        ////////////////////////////////////////////////////////////////////////////////
/*if(match_result>=80){
        vector< vector< vector<string> > > extend_scans ;
        vector< vector< vector<string> > > extend_scans_match ;
        for(int i=0;i<scans.size();i++){
                for(int j=0;j<scans.size();j++){
         novel_minutiae_extraction(im, extend_scans[i][j], MIN_RADIUS, x_axis[i],
          y_axis[j],inner_circles);

                }
        }
        }*/

        return EXIT_SUCCESS;
}
```

.2.2 Calling Functions

```
/**
 * fp_func.c: novel fingerprint auth functions
 *
 */

#include "fp_func.h"
#include "iostream"
#include "string"
double gettime( void )
{
    struct timeval ttime;
    gettimeofday( &ttime , 0 );
```

```
    return ttime.tv_sec + ttime.tv_usec * 0.000001;
}

/*
 * Novel Minutiae Extraction procedure.
 *
 * This function returns a set of circular strings having center at (cx, cy) as a 2d string vector.
 * For now it returns only the outer circle as a pointer, however
 * at least 5 or 7 should be returned for better comparison.
 * In addition the length of the circular string obtained is returned as well.
 */
int novel_minutiae_extraction(Mat im, vector<string> &scans, double r, double cx, double cy,int inner_circles)
{
        string str_l = ""; // left part of circular string
        string str_r = ""; // right part of circular string
        string circular_str = "";
    int circ_str_len = 0;


    scans.resize(inner_circles);

    // for each radius
    for (int i = 0; i < inner_circles; ++i, r += RAD_DELTA)
    {

                for ( size_t x = cx - r; x <= cx + r; x++ )
                {
                    for ( size_t y = cy - r; y <= cy + r; y++ )
                    {

                        // all pixels in the circle of radius r
                        // having center in (cx, cy) satisfy the formula:
                        // r^2 = (x - cx)^2 + (y - cy)^2
                        double a = (x - cx);
                        double b = (y - cy);
                        double d = a * a + b * b;

                        // 1 more pixel and 1 less pixel allowed
                        if (d < r * (r + 1) && d > r * (r - 1))
                        {
                                int pixel = (int) im.at<uchar>(x, y);

                                if (y < cy) {
                                        // left emi-cicle
                                        str_l = (pixel < 180 ? "0" : "1") + str_l;
                                }
                                else {
                                        // right emi-cicle
                                        str_r = str_r + (pixel < 180 ? "0" : "1");
                                }

                                circ_str_len++;
                        }

                    }
```

```
                }

                scans[i] = str_l + str_r;
                str_l = "";
                str_r = "";
    }

    return circ_str_len;
}

/* get_scans extracts circular string representations from an input fingerprint */
vector< vector< vector<string> > > get_scans(Mat im, int MIN_RADIUS, int RAD_DIST,int inner_circles)
{
        /* start center of input image */

    double r = MIN_RADIUS;// - 2; // radius

        // calc n. scans per row
        int n_scans = 0;
        for (size_t x = MIN_RADIUS; x <= im.rows - MIN_RADIUS; x += RAD_DIST) n_scans++;
        vector< vector< vector<string> > > scans; // 2d vector of vector scan circles, i.e. 3d vector
        scans.resize(n_scans);

    /* get scan circles at regular intervals */
        for ( size_t x = MIN_RADIUS, i = 0; x <= im.rows - MIN_RADIUS; x += RAD_DIST, i++ )
        {
                scans[i].resize(n_scans);

            for ( size_t y = MIN_RADIUS, j = 0; y <= im.cols - MIN_RADIUS; y += RAD_DIST, j++ )
            {


                    novel_minutiae_extraction(im, scans[i][j], r, x, y,inner_circles);
            }
        }

        return scans;
}









int levenshtein(char *s1, char *s2)
{
    unsigned int s1len, s2len, x, y, lastdiag, olddiag;
    s1len = strlen(s1);
    s2len = strlen(s2);
    unsigned int column[s1len+1];
    for (y = 1; y <= s1len; y++)
        column[y] = y;
    for (x = 1; x <= s2len; x++) {
```

```
        column[0] = x;
        for (y = 1, lastdiag = x-1; y <= s1len; y++) {
            olddiag = column[y];
            column[y] = MIN3(column[y] + 1, column[y-1] + 1, lastdiag + (s1[y-1] == s2[x-1] ? 0 : 1));
            lastdiag = olddiag;
        }
    }
    return(column[s1len]);
}

int get_distance(string scans, string scans_match)
{
        char * c_text = new char [scans_match.length()+1];
        strcpy(c_text, scans_match.c_str());
        char * c_ptrn = new char [scans.length()+1];
        strcpy(c_ptrn, scans.c_str());
        int dist = levenshtein(c_ptrn, c_text); // match it at about the center of the image to match
        return dist;
}

int best_alignment(string scans, string scans_match)
{
        int dist = 0;
        int min = 0;
        int rotation_pos = 0;
        string best_alignment = "";

        best_alignment = scans;
        min = get_distance(scans, scans_match);

        for (int i = 0; i < scans.size(); i++)
        {
            rotate(scans.begin(), scans.begin()+1, scans.end());
                dist = get_distance(scans, scans_match);
                if (dist < min)
                {
                        min = dist;
                        best_alignment = scans;
                        rotation_pos = i;
                }
        }
        cout << "best_alignment: " << best_alignment << endl;
        cout << "Min distance: " << min << ", rotation_pos: " << rotation_pos << endl;

        return dist;
}


/**
 * Perform one thinning iteration.
 * Normally you wouldn't call this function directly from your code.
 *
 * @param  im    Binary image with range = 0-1
 * @param  iter  0=even, 1=odd
 */
```

```
void thinningGuoHallIteration(cv::Mat& im, int iter)
{
    cv::Mat marker = cv::Mat::zeros(im.size(), CV_8UC1);

    for (int i = 1; i < im.rows; i++)
    {
        for (int j = 1; j < im.cols; j++)
        {
            uchar p2 = im.at<uchar>(i-1, j);
            uchar p3 = im.at<uchar>(i-1, j+1);
            uchar p4 = im.at<uchar>(i, j+1);
            uchar p5 = im.at<uchar>(i+1, j+1);
            uchar p6 = im.at<uchar>(i+1, j);
            uchar p7 = im.at<uchar>(i+1, j-1);
            uchar p8 = im.at<uchar>(i, j-1);
            uchar p9 = im.at<uchar>(i-1, j-1);

            int C  = (!p2 & (p3 | p4)) + (!p4 & (p5 | p6)) +
                     (!p6 & (p7 | p8)) + (!p8 & (p9 | p2));
            int N1 = (p9 | p2) + (p3 | p4) + (p5 | p6) + (p7 | p8);
            int N2 = (p2 | p3) + (p4 | p5) + (p6 | p7) + (p8 | p9);
            int N  = N1 < N2 ? N1 : N2;
            int m  = iter == 0 ? ((p6 | p7 | !p9) & p8) : ((p2 | p3 | !p5) & p4);

            if (C == 1 && (N >= 2 && N <= 3) & m == 0)
                marker.at<uchar>(i,j) = 1;
        }
    }

    im &= ~marker;
}

/**
 * Function for thinning the given binary image
 *
 * @param  im  Binary image with range = 0-255
 */
void thinningGuoHall(cv::Mat& im)
{
    im /= 255;

    cv::Mat prev = cv::Mat::zeros(im.size(), CV_8UC1);
    cv::Mat diff;

    do {
        thinningGuoHallIteration(im, 0);
        thinningGuoHallIteration(im, 1);
        cv::absdiff(im, prev, diff);
        im.copyTo(prev);
    }
    while (cv::countNonZero(diff) > 0);

    im *= 255;
}
```

```
/**
 * Code for thinning a binary image using Zhang-Suen algorithm.
/**
 * Perform one thinning iteration.
 * Normally you wouldn't call this function directly from your code.
 *
 * @param  im    Binary image with range = 0-1
 * @param  iter  0=even, 1=odd
 */
void thinningIteration(cv::Mat& im, int iter)
{
    cv::Mat marker = cv::Mat::zeros(im.size(), CV_8UC1);

    for (int i = 1; i < im.rows-1; i++)
    {
        for (int j = 1; j < im.cols-1; j++)
        {
            uchar p2 = im.at<uchar>(i-1, j);
            uchar p3 = im.at<uchar>(i-1, j+1);
            uchar p4 = im.at<uchar>(i, j+1);
            uchar p5 = im.at<uchar>(i+1, j+1);
            uchar p6 = im.at<uchar>(i+1, j);
            uchar p7 = im.at<uchar>(i+1, j-1);
            uchar p8 = im.at<uchar>(i, j-1);
            uchar p9 = im.at<uchar>(i-1, j-1);

            int A  = (p2 == 0 && p3 == 1) + (p3 == 0 && p4 == 1) +
                     (p4 == 0 && p5 == 1) + (p5 == 0 && p6 == 1) +
                     (p6 == 0 && p7 == 1) + (p7 == 0 && p8 == 1) +
                     (p8 == 0 && p9 == 1) + (p9 == 0 && p2 == 1);
            int B  = p2 + p3 + p4 + p5 + p6 + p7 + p8 + p9;
            int m1 = iter == 0 ? (p2 * p4 * p6) : (p2 * p4 * p8);
            int m2 = iter == 0 ? (p4 * p6 * p8) : (p2 * p6 * p8);

            if (A == 1 && (B >= 2 && B <= 6) && m1 == 0 && m2 == 0)
                marker.at<uchar>(i,j) = 5;
        }
    }

    im &= ~marker;
}

/**
 * Function for thinning the given binary image
 *
 * @param  im  Binary image with range = 0-255
 */
void thinning(cv::Mat& im)
{
    im /= 255;

    cv::Mat prev = cv::Mat::zeros(im.size(), CV_8UC1);
    cv::Mat diff;
```

```
    do {
        thinningIteration(im, 0);
        thinningIteration(im, 1);
        cv::absdiff(im, prev, diff);
        im.copyTo(prev);
    }
    while (cv::countNonZero(diff) > 0);

    im *= 255;
}
////////////////////////////////////////////////////////////////////////////////////////////////


//Function to update the Direction string for the Direction table
string tableUpdateFuncDirec(int a, int b, int c)
{
        if(a == b && a == c)
        {
            string result = "123";
            return result;
        }
        else if ( a==b && a > c)
        {
            string result = "12";
            return result;
        }
        else if ( a==c && a > b)
        {
            string result = "13";
            return result;
        }
        else if ( b==c && b > a)
        {
            string result = "23";
            return result;
        }
        else
        {
            int f = max(a,b);
            f = max(f, c);
            string result;
            if (f==a)
            {
                result = "1";
            }

            else if  (f==b)
            {
                result = "2";
            }
            else
            {
                result = "3";
            }
            return result;
```

```
        }
}

int tableUpdateFuncMain(int a, int b, int c)
{
        int f = max(a,b);
        f = max(f, c);
        return f;
}

//Function to match the sequence when updating the MainTable
int matchingFunc(int x, int y)
{
        if (x == y)
            return 1;
        else return -1;
}

    //function to interpret the direction mapping when tracing the path
    // 1 --> diagonal (top left)
    // 2 --> left
    // 3 --> top
    // x = direction, (y,z) = present position
IntVec traceCoord(char x, int y, int z)
{
        IntVec nextCoord;
        if (x == '1')
        {
            nextCoord.clear();
            nextCoord.push_back(y-1);
            nextCoord.push_back(z-1);
            return nextCoord;
        }
        else if (x == '2')
        {
            nextCoord.clear();
            nextCoord.push_back(y);
            nextCoord.push_back(z-1);
            return nextCoord;
        }
        else
        {
            nextCoord.clear();
            nextCoord.push_back(y-1);
            nextCoord.push_back(z);
            return nextCoord;
        }
}
int needlemanWunschMatching(string t,string u,int SeqLength ){

    // variable declaration
    //int SeqLength;
    int R;
    string s;
```

```
    // Obtain length of binary sequence
   // cout<<"Enter the length of sequence"<<endl;
    //cin>> SeqLength;

    int Sequence1[SeqLength];
    int Sequence2[SeqLength];

    // Obtain the first binary sequence
//    cout<<"Enter the binary sequence 1"<<endl;
   // cin>>t;

    for (unsigned uv = 0; uv < t.length(); uv++){
        int conv = (int)t[uv] - 48;
        Sequence1[uv] = conv;
    }

    // Obtain the second binary sequence
   // cout<<"Enter the binary sequence 2"<<endl;
    //cin >> u;
    for (unsigned uv = 0; uv < u.length(); uv++){
        int conv = (int)u[uv] - 48;
        Sequence2[uv] = conv;
    }

    int n = SeqLength + 2;
    int rowcount1, columncount1;

    // MainTable holds the completed table according to the Needleman Wunsch's algorithm using the following score system:
    // +1 for match
    // -1 for no match
    // -1 for indel
    // DirectionTable holds the directions computed according to the following representation
    // 1 --> diagonal (top left)
    // 2 --> left
    // 3 --> top

    int MainTable[n ][n];
    string DirectionTable[n-2][n-2];

    //initialise Maintable
    MainTable[0][0] = 0;
    MainTable[0][1] = 0;
    MainTable[1][0] = 0;
    MainTable[1][1] = 0;

    // Enter sequence 1
    for (columncount1 = 2; columncount1 < n; columncount1++){
        MainTable[0][columncount1] = Sequence1[columncount1 - 2];
    }

    // Enter sequence 2
    for (rowcount1 = 2; rowcount1 < n; rowcount1++){
        MainTable[rowcount1][0] = Sequence2[rowcount1 - 2];
    }
```

```
//Updating the table by matching with the 'gap' with sequence 1
for (columncount1 = 2; columncount1 < n; columncount1++){
    MainTable[1][columncount1] = MainTable[1][columncount1 - 1] - 1;
}

//Updating the table by matching with the 'gap' with sequence 2
for (rowcount1 = 2; rowcount1 < n; rowcount1++){
    MainTable[rowcount1][1] = MainTable[rowcount1 - 1][1] - 1;
}

//Complete both tables updates
//number mapping

for(rowcount1 = 2; rowcount1 < n; rowcount1++){
    for (columncount1 = 2; columncount1 < n; columncount1++){
        //MainTable update
        MainTable[rowcount1][columncount1] = tableUpdateFuncMain(matchingFunc(Sequence1[columncount1 - 2],
         Sequence2[rowcount1 - 2]) +
         MainTable[rowcount1 - 1][columncount1 - 1],
          MainTable[rowcount1][columncount1 - 1]
          - 1, MainTable[rowcount1 - 1][columncount1] - 1);
        //Direction Table update
        DirectionTable[rowcount1 - 2][columncount1 - 2] = tableUpdateFuncDirec(matchingFunc(Sequence1[columncount1 - 2],
        Sequence2[rowcount1 - 2]) +
         MainTable[rowcount1 - 1][columncount1 - 1],
          MainTable[rowcount1][columncount1 - 1]
          - 1, MainTable[rowcount1 - 1][columncount1] - 1);
    }
}

//obtain the coordinates for one of the best alignments
vector<IntVec> Align1;

Align1.clear();
int pp, qq;
IntVec Fill;
Fill.clear();
Fill.push_back(SeqLength-1);
Fill.push_back(SeqLength-1);
int jj = SeqLength-1;
int kk = SeqLength-1;
Align1.push_back(Fill);

// randomly obtain a path according to the algorithm

do{
    srand (time(NULL));
    int ran =  rand() % DirectionTable[jj][kk].length();
    pp = traceCoord(DirectionTable[jj][kk][ran], jj, kk)[0];
    qq = traceCoord(DirectionTable[jj][kk][ran], jj, kk)[1];
    jj = pp;
    kk = qq;
    IntVec Fi;
    Fi.clear();
    Fi.push_back(pp);
```

```
    Fi.push_back(qq);
    Align1.push_back(Fi);

}while(pp >=0 && qq>=0);

if (pp <0 && qq < 0){
    Align1.erase(Align1.begin() + (Align1.size()-1));
}

IntVec sq1;
IntVec sq2;

int MatchScore= 0;
double PercentMatch;

//output the obtained match/alignment and percentage match
int yyy = 0;
int xxx = 0;
int smart1 = 100;
int smart2 = 100;




for (int fcount = Align1.size(); fcount > 0; fcount--){
    if (Align1[fcount-1][1] < 0){
        //cout<<"-";
        sq1.push_back(2);
    }
    else if (Align1[fcount-1][1] == smart1){
       // cout<<"-";
        sq1.push_back(2);
    }
    else{
     //    cout<<Sequence1[xxx];
        sq1.push_back(Sequence1[xxx]);
        xxx++;
    }
    smart1 = Align1[fcount-1][1];
}


for (int fcount = Align1.size(); fcount > 0; fcount--){
    if (Align1[fcount-1][0] < 0){
       // cout<<"-";
        sq2.push_back(2);
    }
    else if (Align1[fcount-1][0] == smart2){
       // cout<<"-";
        sq2.push_back(2);
    }
    else{
        //cout<<Sequence2[yyy];
        sq2.push_back(Sequence2[yyy]);
        yyy++;
```

```
        }
        smart2 = Align1[fcount-1][0];
    }

    //Computing and Printing the Percentage Match calculated as:
    //                                Total match score
    //       Percentage Match =     ---------------------- * 100
    //                                Length of sequence
    //
    for(unsigned score = 0; score < sq1.size(); score++){
        if(sq1[score] == 2 || sq2[score] == 2){
            MatchScore--;
        }
        else if (sq1[score] != sq2[score]){
            MatchScore--;
        }
        else{
            MatchScore++;
        }
    }

    if (MatchScore >= 0){
        PercentMatch = ((double)MatchScore / (double)SeqLength) *100;
    }
    else{
        PercentMatch = 0;
    }

    return PercentMatch ;
}




int zeroDistance(int c, map<int, int> d)
{
    int x;
    int y = 0;
    map<int, int>::iterator it;
    it = d.find(c);
    if(it == d.end()) return -1;
    else if((*it).second != 0) return -1;
    else
    {
        for(map<int, int>::iterator itx=it; itx!=d.end(); itx++){
            if(itx->first != it->first)
            {
                y++;
                if(itx->second == 1) break;
            }
            x = -1;
        }
        if(x == -1 ) return x;
        else return y;
    }
```

```
}

long zeroDist(int c, vector<int> d)
{
    int f = 0;
    vector<int> nn;
    for(unsigned u = c; u < d.size(); u++){
        if(d[u] == 1) f =1;
        if(f == 1) nn.push_back(d[u]);
    }
    for(int u = 0; u < c; u++){
        nn.push_back(d[u]);
    }
    std::ostringstream oss;
    for(unsigned u = 0; u < nn.size(); u++){
        oss<<nn[u];
    }
    string num = oss.str();
    long number = atol(num.c_str());
    return number;

}

 string lex (string input_data){

            vector<int> input, inputx, strout;
            for(unsigned p =0; p< input_data.length(); p++){
                input.push_back( (int)input_data[p] - 48);
            }
            map<int, int> labstrg;
            map<int, int>::iterator it1, it2;
            for(unsigned p =0; p< input_data.length(); p++){
                labstrg[p] = input[p];
                inputx.push_back(p);
            }
            int iterations = 0;
            for(int i =1; i <input_data.length(); i++){
                if(unsigned(pow(2.0,i)) >= input_data.length())
                {
                    iterations = i;
                    break;
                }
            }
            for(int v =0; v< iterations; v++){
                vector<int> stgres;
                int ttt = (int)(inputx.size() / 2);
                //cout<<ttt <<",";
                for(int t =0; t < (int)(inputx.size() / 2); t++){
                    it1 = labstrg.find(inputx[2 * t]);
                    it2 = labstrg.find(inputx[(2 * t)+1]);
                    if((*it1).second != (*it2).second)
                    {
                        //different values; so select the smaller one
                        if((*it1).second < (*it2).second) stgres.push_back((*it1).first);
                        else stgres.push_back((*it2).first);
```

```
            }
            else if((*it1).second == 1)
            {
                //Both values are 1
                if((*it1).first < (*it2).first) stgres.push_back((*it1).first);
                else stgres.push_back((*it2).first);
            }
            else
            {
                //Both values are zero
                if( v == 0)
                {
                    stgres.push_back((*it1).first);
                }
                else
                {
                    if(zeroDist((*it1).first, input) < zeroDist((*it2).first, input))stgres.push_back((*it1).first);
                    else if(zeroDist((*it1).first, input) > zeroDist((*it2).first, input)) stgres.push_back((*it2).first);
                    else stgres.push_back((*it1).first);
                }
            }
        }
        if(inputx.size() % 2 > 0)
        {
            stgres.push_back(inputx[inputx.size() - 1]);
        }
        inputx.clear();
        for(unsigned u = 0; u < stgres.size(); u++){
            inputx.push_back(stgres[u]);
         //    cout<<stgres[u]<<", ";
        }
        cout<<endl;
    }
    int result = inputx[0];
    //cout<<"This digit selected as the least lexicographical rotation of the string is:"<<endl;
    //cout<<result<<endl;

    //Correct string output rotation is
    for(unsigned u = result; u < input.size(); u++){
        strout.push_back(input[u]);
    }
    for(int u = 0; u < result; u++){
        strout.push_back(input[u]);
    }

    stringstream ss;

    string str1;
   // cout<<"The starting point of string information to be used for matching within the database is: "<<endl;
    for(unsigned u = 0; u < strout.size(); u++){
        //cout<<strout[u];
        //cout<<endl;
       ss<<strout[u];

       str1=ss.str();
```

```
            }
           //  cout<<str;
            //cout<<endl;
//cout<<str.length();
//cout<<endl;
//cout<<strout.size();
return str1;
        }
```