جـامـعـة الـشـرق الأوسط
MIDDLE EAST UNIVERSITY

Enhancing Open Space Method in Data Hiding Technique – Text under Text

تحسين في استخدام الفراغ المتاح في تقنية اخفاء البيانات – نص خلال نص

By

Yaman Fakhri Issa Abdallah

Supervisor

Dr. Hebah H. O. Nasereddin

Submitted in Partial Fulfillment of the Requirements of the Master's Degree in Computer Information System

Computer Information System department

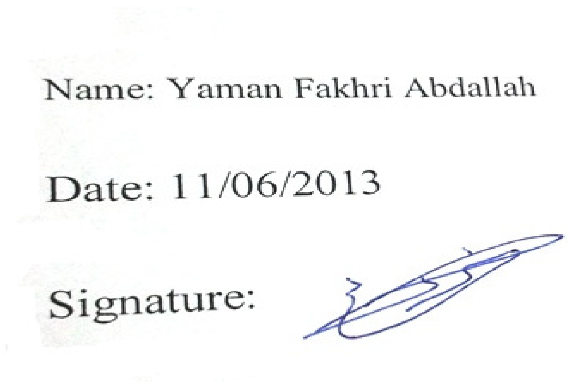
Faculty of Information System and Technology

Middle East University

May, 2013

Authorization Statement

I'm Yaman Fakhri Abdallah, authorize Middle East University to supply hardcopies and electronic copies of my thesis to libraries, establishments, or bodies and institutions concerned with research and scientific studies upon request, according to the university regulations.



Name: Yaman Fakhri Abdallah

Date: 11/06/2013

Signature: [illegible]

Examination Committee Decision

This is to certify that the thesis entitled **"Enhancing Open Space Method in Data Hiding Technique – Text under Text"** was successfully defended and approved on.

Examination Committee Member	**Signature**
1- Dr. Heba Nasser Al-deen	[illegible]
2- Dr. Oleg Viktorov	[illegible]
3- Dr. Asim Al Sheikh	[illegible] Prof. Asim El Sheikh

Acknowledgements

At the mid of my academic path pursuing my Master's degree, and when I was hesitant about selecting the subject of my thesis; Dr. Hebah Nasereddin accompanied me in this path providing full guidance and supervision starting with the subject selection ending with this complete academic work. Also, she played a significant role in publishing my first paper in one of the most accredited journals in the IT studies' field, and introduced me as a participant - by presenting my work-in the international IT conference.

After finishing my thesis technically, I was fortunate to have my friend Mohammad Jihad editing my paper and assisting me to present it in a scientific and proper language.

Sharing this moment of success with my dear mother –May Allah's mercy be upon her- who would have been delighted to witness my proved academic development is my only wish.

To my wife I'm grateful; who gracefully dedicated her days and nights ensuring I got all it takes to achieve this work.

For the constant support, I can only thank my brothers and sisters.

Finally, I am thankful to the educational board in the MEU University; and to the Information Technology department in particular.

Dedication

To **my father's** soul who taught me how to find my way through man's most challenging hardships

To **my mother's** soul who dedicated her life to watch over me

To my brother, my mentor, and my teacher, **Dr. Al-Hareth**

To my **brothers** and **sisters** with whom I shared with the joy of life

To my precious wife, To **Safa'a**

To the one who guided me toward success in my academic advancement **Dr. Hebah Nasereddin**

To my friends, my lifetime companions; **Mohammed Jihad and Hazem Khalid**

Table of Contents

List of Tables

List of Figures

Abbreviations

DH:	Data Hiding
CT:	Cover Text
ST:	Secret Text
DHDD:	Data Hiding for Documents based on Dots
ASCII:	American Standard Character Interchange
RTF:	Rich Text File

تحسين في استخدام الفراغ المتاح في تقنية اخفاء البيانات – نص خلال نص

الطالب

يمان فخري عبدالله

المشرف

الدكتورة هبة ناصر الدين

الملخص

لقد تم في هذا البحث دراسة منهجية اخفاء البيانات داخل نصوص مختلفة؛ حيث ان عملية اخفاء البيانات هي احدى مجالات امن المعلومات المتمثلة بإخفاء البيانات وتشفيرها وغيرها من الفروع التي تمثل امن المعلومات.

ان لإخفاء البيانات طرق متعددة كإخفاء البيانات داخل صورة او داخل ملف صوتي او داخل نص، حيث ان إخفاء البيانات داخل هذه الوسائل تحقق الفائدة من حفظ حقوق نشر والطبع أو عدم وصول غير المخولين الى هذه البيانات بسهولة، كون هذه البيانات المخفية تتصف بالسرية والخصوصية.

إن عملية إخفاء البيانات داخل نص تختلف كليا عن باقي طرق الإخفاء، فهي تكون من خلال التغيير في المساحات الفارغة في النص او تغيير النص نفسه او تغيير في بعض خصائص اخرى غير النص والمساحات الفارغة؛ وهذه التغييرات تتم بطريقة يمكن من خلالها إخفاء بيانات داخل النص.

لقد تم في هذا البحث يطرح تحسين لطريقة المساحات الفارغة في اخفاء البيانات؛ وهي عبارة عن اجراء تغيير بالنص السري باستخراج الرقم التسلسلي للأحرف من النص الغطاء او من الرمز القياسي الأمريكي لتبادل المعلومات ومن ثم تحويل هذه الارقام من نظام العد العشري الى نظام العد الثماني وذلك لاستخدام الارقام ثمانية وتسعة للفصل بين باقي الارقام بحيث اذا كان الرقم يؤشر على الرقم التسلسلي للحرف فيتم وضع رقم تسعة قبله اما اذا كان الرقم يؤشر على الرمز القياسي فيتم وضع رقم ثمانية قبله ومن ثم دمج هذه الارقام في المساحات الفارغة بين الكلمات في النص الغطاء وذلك بتغيير حجم النص للأرقام الى واحد ولون الخط مثل لون خلفية النص الغطاء.

ولقد تم في هذا البحث التحقق من العملية المقترحة في تحسين استخدام تقنية الفراغ المتاح في إخفاء البيانات – نص خلال نص، ولقد تم التحقق من ثلاثة امور اساسية وهي: سرعة العملية، حجم النص الناتج من هذه العملية وبالإضافة الى التحقق من أن النص الناتج من هذه العملية مطابق للنص الاصلي الذي قد تم استخدامه لإخفاء البيانات خلاله.

Enhancing Open Space Method in Data Hiding Technique – Text under Text

By

Yaman Fakhri Issa Abdallah

Supervisor

Dr. Hebah H. O. Nasereddin

Abstract

I’ve been in this research study methodology for hide data within different texts. That's where the data masking process is one of the areas of information security to hide and encrypt data and other branches that represent information security.

That to hide the data multiple ways as hide data within an image or within an audio file or within the text, where the hide data within these means check the interest of protecting the rights of publication and copyright or non-arrival of unauthorized to this data easily, the fact that these hidden data is characterized by secrecy and privacy.

The process of hiding data within the text are completely different from the rest of methods of concealment, it will be through a change in the empty spaces in the text or change the text itself or change in some of the properties other than text and empty spaces. These changes are a way by which to hide data within the text.

This thesis proposes an enhancement of white space method for hiding data; which is processed by changing the secret text through extracting the indexes of the characters from the cover text or ASCII code , then converting these numbers from the decimal numeral system into the octal numeral system in order to use the number 8 and 9 as indicators against the remaining numbers. As placing the number 9 before the extracted number indicates that it is an index of a character. and placing the number 8 before the extracted number indicates that it is an ASCII code. Then, merging these outcomes with the white spaces between the words in the cover text by changing the font size of these numbers to 1pt, and changing the font color to match background color of the cover text.

I’ve been in this research verification of the proposed process in enhancing the technique of open space methods to hide the data - text under text, and have been verified three basic things: the speed of the hiding data process, the size of the output text of this process, as well as to verify that the text resulting from these process is identical to the original text, which has been used to hide data under it.

Chapter One
Introduction

1.1 Preface

Information security has two branches; data encryption and data hiding (DH). Data encryption is masking the data to become meaningless, while data hiding is concerned with concealing the data to become unreadable.

Data hiding is: (Bender, Gruhl, Morimoto, & Lu, 1996) “Data hiding represents a class of processes used to embed data, such as copyright information, into various forms of media such as image, audio, or text with a minimum amount of perceivable degradation to the “host” signal”.

(YILMAZ, 2003) "Famous examples of steganography go back to antiquity. According to a story from Herodotus, a slave’s head was shaved by his master, Histiæus, and tattooed with a secret message around 440 B.C. After growing the hair back, the message disappeared and then the slave journeyed to carry the message. When shaved his head upon arriving, message was revealed”. (Nasereddin & Al Farzaeai, 2010) “In 1860, the major problems had been solved to make a small image by "Darjun", who is a French photographer worked in the war. Frank and France in 1870- 1861 when Paris was besieged, by writing messages on photographic films which were sent by the carrier pigeon. The purpose of this was to invoke disobedience against his antagonist Persinas. Steganography is the ability of hiding data in redundant bits of any cover media. Its target is to keep the secret information unreadable without damaging the cover media environment.”. (Ibrahim & Zabian, 2009) “A security issue must be used in each step. Information hiding can be used in different applications include military, E- commerce, confidential communication, copyright protection, copy control, authentication, digital elections. In these fields, hiding information is better than ciphering. Because in the former, nobody can notice that there is a message hiding behind an image”.

1.2 Objectives and Problem Definition

Data hiding is a process used to embed the data into media such as image, audio or text. Digital media has become more prevalent and expanding. Also, the security of information transmission has become more vulnerable to theft and unauthorized access. One of the processes to preserve the security of the information is data hiding. There are many researchers are talking about hiding data in image, audio and some of researchers are talking about data hiding text under text.

Open space is part of data hiding text into text, some of open space problems are:

1. To hide two words like “Top Secret” requires text size cover of more than 80 words; because each character size is 8 bit “1 byte” and each bit requires one space. That means “T+o+p+ +S+e+c+r+e+t” equal 10 characters, 10 characters multiply by 8 bits equal 80 bits.
2. To be able to hide a large secret message; the result will be a very large message.
3. In a properly *justified format* of text, not all spaces are available to be used to hide the required data.

Based on what had been said, data hiding techniques are obviously suffering some major issues and in some certain cases may become inefficient. Thus, **the aim of this research is to develop data hiding via text under text while taking into consideration Open Space method problems.**

1.3 Significance

The importance of this research is to enhance open space method in data hiding – text under text. (Most of researchers talk about hiding data under image, audio, and less researchers talk about text under text).

This research enhances a technique for data hiding – text under text using open space method in efficient and effective way, more secure and faster process of data hiding text under text, the proposed enhancement take advantage from the unused white space in the text and another techniques to support the proposed enhanced technique in a way to make the result matches the cover text which will be used to hide the secret text within the cover text.

1.4 Limitations

The limitations of this research are:

1. The size of the cover text must be compatible with the secret text to be able to hide the secret text within the cover text.
2. It is must to use soft copy for the cover text and the secret text.
3. The result text must be transmitted via e-media.

1.5 Thesis Outline

Chapter two represents the theoretical background and related works of data hiding concepts, data hiding techniques and types including data hiding in image, audio and video.

Chapter three represents the proposed model, the proposed solution and the supported techniques used in the proposed solution and the specification of this supported techniques and the effectiveness of this supported techniques in the proposed solution.

Chapter four represents the experimental results and evaluates the proposed solution in many cases, also compare it to another technology of hide text under text by using open space method.

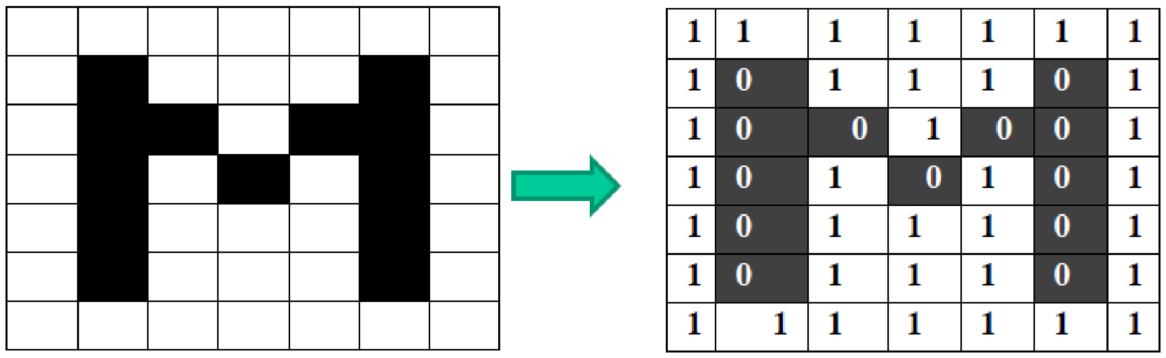
Finally chapter five will discuss the results, draw the conclusions and the future works.

Chapter Two
Literature Survey

This chapter represents knowledge and theoretical background about the data hiding techniques, some of the methods followed by researchers in their researches will be reviewed, discusses more specifically data hiding - text under text techniques and represents the related research about the presentation of data hiding techniques – text under text.

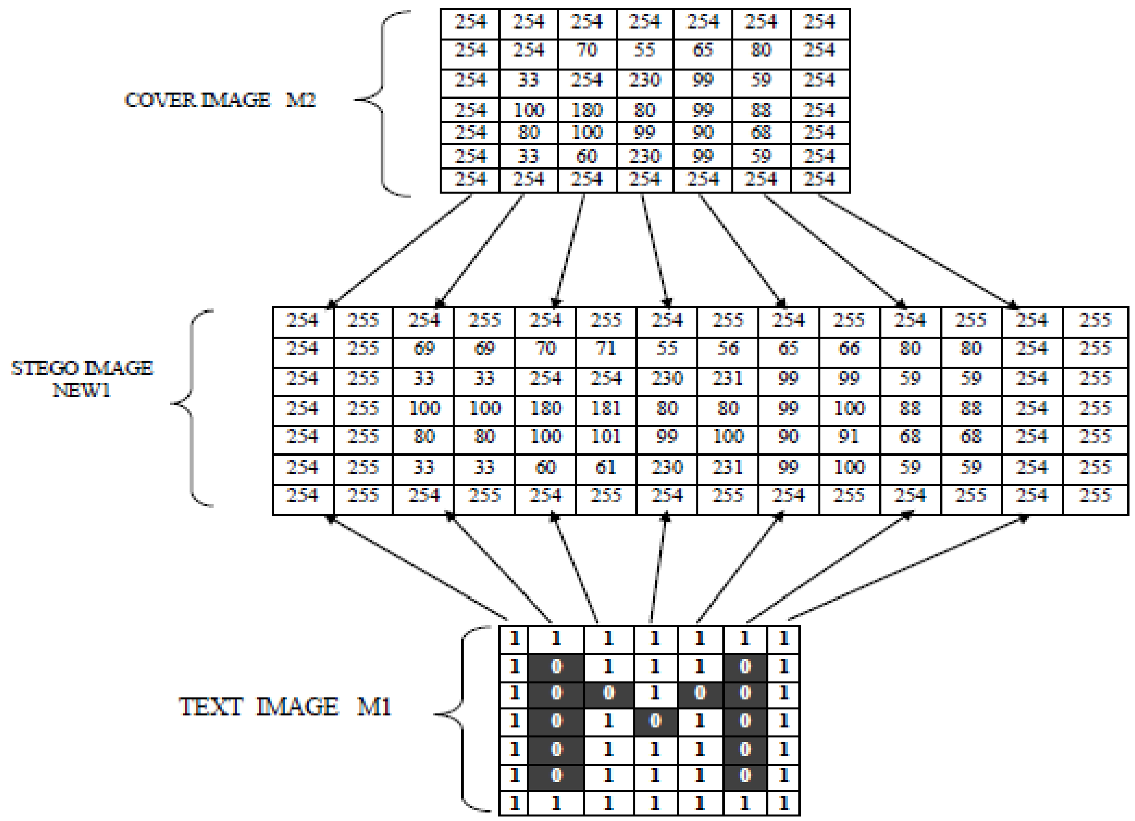
2.1 Theoretical Background

(Nasereddin & Al Farzaeai, 2010) **Proposed data hiding technique text image inside image (TIII)**. A new technique to hide text inside digital image by the concept of the visual representation of the text within image. The proposed technique TIII is based on the existence of the text in the form of an image in black and white, by representing white as "1" and black as "0". The idea of the proposed is merging text image with cover image by integrating text image bits with cover image bits. The proposed technique TIII didn't address the difference of the size between the cover image and the stego image, because it can be solved by using cover images with the same size of the stego image and doing some resizing methods to the cover image to make it as the same size of the text image. The font size of the message in the text image will be an important factor in the integrity of the message during the transmission process and the size of the hidden message, so that when the font size is large the safety of the transmission will be strong, and when the size of the message is small, the safety becomes more vulnerable.



1	1	1	1	1	1	1
1	0	1	1	1	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	1
1	0	1	1	1	0	1
1	0	1	1	1	0	1
1	1	1	1	1	1	1

(a)



COVER IMAGE M2

254	254	254	254	254	254	254
254	254	70	55	65	80	254
254	33	254	230	99	59	254
254	100	180	80	99	88	254
254	80	100	99	90	68	254
254	33	60	230	99	59	254
254	254	254	254	254	254	254

STEGO IMAGE NEW1

254	255	254	255	254	255	254	255	254	255	254	255	254	255
254	255	69	69	70	71	55	56	65	66	80	80	254	255
254	255	33	33	254	254	230	231	99	99	59	59	254	255
254	255	100	100	180	181	80	80	99	100	88	88	254	255
254	255	80	80	100	101	99	100	90	91	68	68	254	255
254	255	33	33	60	61	230	231	99	100	59	59	254	255
254	255	254	255	254	255	254	255	254	255	254	255	254	255

TEXT IMAGE M1

1	1	1	1	1	1	1
1	0	1	1	1	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	1
1	0	1	1	1	0	1
1	0	1	1	1	0	1
1	1	1	1	1	1	1

(b)

Figure 2-1 (a): The binary of the M character as TIII technique

(b): The result of merging the M character within the cover image

(Nasereddin & Al Farzaeai, 2010)

(Zou & Shi, 2005) **A novel formatted text document data hiding algorithm**. Called Inter-word Space Modulation (ISM) scheme, is proposed in which the spaces between neighboring words are modulated to hide data. In contrast to prior arts, this method does not require original documents for hidden data extraction. The hidden data are robust to printing, copying and scanning. The experiments show that after printing, ten times of repeated copying, followed by scanning, the hidden data can still be extracted without a single bit error. It is expected that it can find wide applications for secure document processing, including digital notarization. Three different methods for formatted text document data hiding: line shift coding, word shift coding and feature coding. Line shift coding and word shift coding are robust to printing, copying and scanning to some extent. The major drawback is that the original intact document is needed for hidden data extraction which may not be available in many cases. Also the paper mention a baseline detection method for line shift coding which did not require the original document. However, as pointed out by the authors themselves, it is not reliable to printing, copying and scanning. Besides, the embedding capacity is about one bit per two lines.

(Asif, Shaikh, Manza, & Ramteke, 2010) **The comparison of original text in the form of bitmap image and extracted image by using various fonts of text as a bitmap image**. In image the basic objective of data hiding is to store as much as data in the host image without degrading the quality of the host image and which will be reconstructed again without compromising the loss of source image data and the actual hided information. Out of which the most emerging area is hiding the data into different media files such as image, audio, video, etc. In these media files the image is considered as the most suitable file format for the data processing. The study has done the preparation of the text data set of size 20 characters in single font type, variable font sizes and the color of text as black. The bitmap image can be hided into any color image source which will acts as a medium of carrier of the text data. This color image is further decoded to get the actual data of 20 characters without any loss if possible. The experimental steps used for text data hiding in images is done with above data set and image processing functions of MATLAB. The data set of a single color source image and the data set of 2 to 3 sentence each of 24 characters with above specification. The result is found to be most satisfactory and prominent in the font VERDANA in the font size of 26 to 30 resulted into 85% to 90% of reconstruction rate of actual hided text data.

(Dutta, Bhattacharyya, & Kim, 2009) **Data hiding in Audio Signal: a review**. This paper introduced a robust method of imperceptible audio data hiding. This system is to provide a good and efficient method for hiding the data from hackers and sent to the destination in a safe manner. This proposed system will not change the size of the file even after encoding and also suitable for any type of audio file format. The proposed idea is to hide secret message within audio signal using with a stego key, to retrieve the embedded message should be using the extractor with the same stego key. This paper conclude that audio data hiding techniques can be used for a number of purposes other than covert communication or deniable data storage, information tracing and finger printing, tamper detection. As the sky is not limit so is not for the development. Man is now pushing away its own boundaries to make every thought possible. So similarly these operations described above can be further modified as it is in the world of Information Technology. After designing any operation every developer has a thought in his mind that he could develop it by adding more features to it. Figure 2-2 describes hiding data within audio signal.

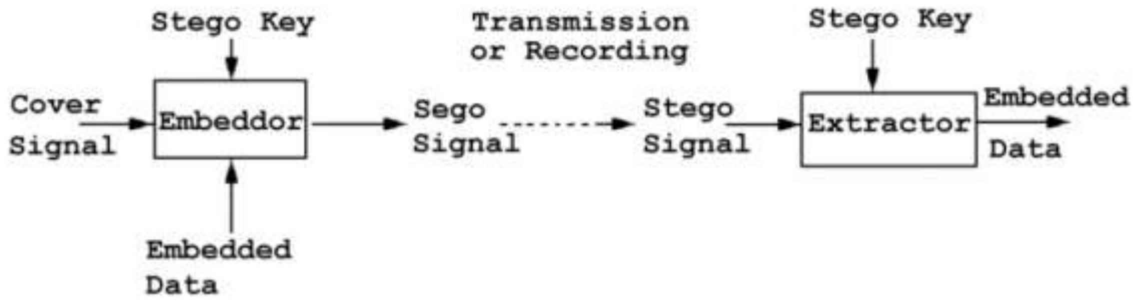


Figure 2-2: describes hiding data within audio signal

(Dutta, Bhattacharyya, & Kim, 2009)

(Abdul Qadir & Ahmad, 2006) **Digital Text Watermarking: Secure content delivery and data hiding in digital documents**. This developed a novel encoding scheme which can be used to insert information in plain text without changing the text. A system has been developed based upon this encoding scheme. This paper suggested a novel idea based upon an intelligent encoding scheme in the world of text watermarking which has no effect on the alteration of the syntax of the document as well as the layout. Thus providing a layout/format independent technique in which information within the text is manipulated to hide certain information. This paper encodes the information in the existing characters of the text in an intelligent way that does not change the document. Moreover, the hidden information is being preserved by the document. The system has two parts: insertion of watermark; and detection of watermark.

(Deguillaume, Rytsar, Voloshynovskiy, & Pun, 2005) **Protocols for data hiding based text document security and automatic processing**. Propose for the following to use graphical features modulation based text data hiding scheme, which encodes the data into one or several features of individual characters or groups of characters, without changing the textual content itself. It is presented generic text data-hiding based protocols for documents, in both electronic and hardcopy format. The presented protocols are suitable for document authentication and tamper proofing, content self-recovery, and automatic document processing. Applications can be among others authentication documents (such as passports and ID cards), payment documents, contracts, letters, and technical reports. They provide cheap and convenient solutions for many practical scenarios, requiring only standard printers and scanners. Moreover these frameworks can be integrated directly into common text document editing/publication tools. Figures 2-3 represents the protocol of embedding the data within document while figure 2-4 represents the protocol to extract the data from the document

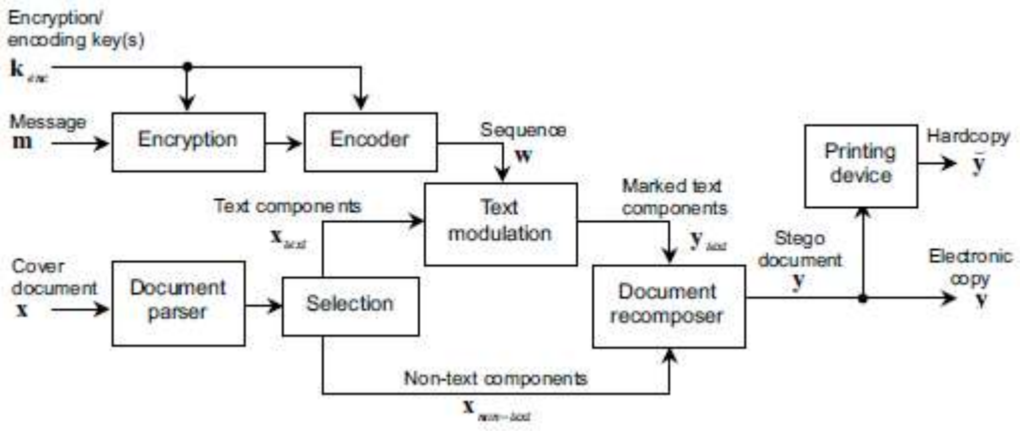


Figure 2-3: Protocol for document data-hiding: embedding

(Deguillaume, Rytsar, Voloshynovskiy, & Pun, 2005)

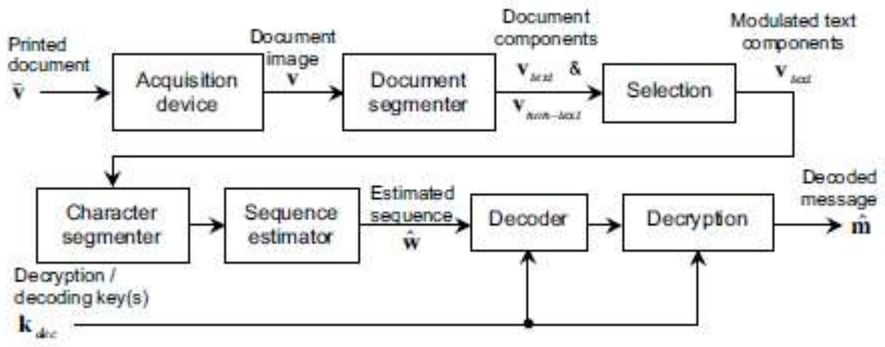


Figure 2-4: Protocol for document data-hiding: extraction

(Deguillaume, Rytsar, Voloshynovskiy, & Pun, 2005)

(Yang & Kot, 2005) **Data hiding for text document image authentication by connectivity preserving**. Propose a data hiding technique which is based on the connectivity-preserving in 3 × 3 neighborhood. The “uneven embed ability” of the host image is considered by embedding the watermark only in those “embeddable” blocks. A small block size, e.g., 4 × 4 is employed in order to achieve the larger capacity. The proposed scheme can be used for document authentication, e.g., e-Certificate authentication. The odd-even enforcement is employed for the watermark embedding, which is vulnerable to “parity attack”, i.e., an adversary can carefully flip two pixels while keeping the odd-even feature of the block unchanged. This paper proposes to adopt a hard authenticator watermark to tackle this problem in order to generate the hard authenticator watermark; the key issue is how to locate the flipped pixel given the watermarked image. For the fixed 3 × 3 block, the flipped location is always the center pixel of the block; therefore it is easy to locate the flipped pixel. The fixed 3 × 3 block, non-interlaced and interlaced block are employed and the capacity of using different types of blocks are compared.

(Kim & Mayer, 2007) **Data Hiding for Binary Documents Robust to Print-Scan, Photocopy and Geometric Distortions**. This paper presents a data hiding technique (steganography) for embedding information into documents printed in high-resolution bicolor printers, e.g., conventional laser and inkjet printers. In the literature, there are several data hiding techniques designed for binary images. These techniques can be applied to copy control, annotation, and authentication. However, most of them are designed only for binary images in digital form and cannot be applied for printed documents. Data hiding for binary images can be divided into three basic classes:

1- Component-wise: Change the characteristics of some pixel groups (connected components, character, words, etc.).

2- Pixel-wise: Change the values of individual pixels.

3- Block-wise: Divide the cover image into blocks and modify some characteristic of each block to hide the data.

Also the paper proposed a technique named DHDD (Data Hiding for Documents based on Dots). Current laser/inkjet printers can print tiny dots hardly noticeable at normal reading distance. This implementation is able to embed up to 1370 bits in an A4-sized document printed at 600 dpi. As the printing technology evolves, it is expected that future printers will be able to impress even smaller dots, resulting in more visually imperceptible watermarking with more data hiding capacity. Propose to use the entire binary document for embedding the watermark with a high robustness to print-scan, photocopy and geometric attacks.

(Dutta, Bhattacharyya, & Kim, 2009) **Current Steganography Tools and Methods.** Provide a review and analysis of several freeware tools that employ some of the more common methods of hiding information in digital files, demonstrating how one can easily embed secret messages in some of the more commonly exchanged image, audio and text file formats. Steganography is by no means a modern practice. Literally meaning “covered writing” it is the practice of hiding messages within other messages in order to conceal the existence of the original. For the security professional, this means that data you are paid to protect could be leaving your control without your knowledge. Some specific terminology in the field of steganography has developed to make clear the differences between files to be hidden, those that they get hidden in and the resulting combinations of the two.

Data Hiding in Text: Steganography in text files can be accomplished through various techniques. Methods that can be applied to both the soft and hard copies of a document include line-shift coding, word-shift coding and feature coding as well as syntactic semantic methods.

The first three of these systems rely on visually changing the formatting or look of the file, by modifying spacing between lines, spacing between words, or modifying features of certain letters respectively.

Syntactic and semantic methods of steganography in text files utilize modification of “diction and structure of text without significantly altering meaning or tone”.

Data hiding in image and audio files: Typically, using image files as hosts for stenographic messages takes advantage of the limited capabilities of the human visual system. Some of the more common method for embedding messages in image files can be categorized into two main groups, image domain methods and transform domain methods. The image domain methods modify their host files at the bit level, changing the file bit by bit to encode their message. The transform domain methods manipulate the algorithms and transformations inherent in the creation of the image itself, like the transformation used in JPEG compression.

2.2 Related Works

(Abdullah & Nasereddin, 2013) **Proposed Data Hiding Technique – Text under Text.** The technique is based on two texts; secret text and cover text. By taking advantage of the unused white spaces in the cover text, this paper proposed to change the format of the secret text by changing the secret text font size and font color to the font size 1pt and secret text font color white, then to separate the secret text into many parts “characters”, then to hide these parts within the cover text’s unused white spaces. The outlined that the proposed data hiding technique as Image, audio, and text are used for data hiding. Data hiding in text is to embed text within another text to be unreadable. Open space methods used for data hiding in text and white space method is one of these methods; the paper takes advantages of the unused white space from the text “Cover Text” to hide the data “Secret Data” on the cover text. Changing the format of the secret by setting the text size to 1px, setting the font color to white as the back color of cover text, and then merging the secret text with the cover using white space method to generate the result text hiding the secret message within it. Figure 2-5 shows a brief description of the proposed technique.

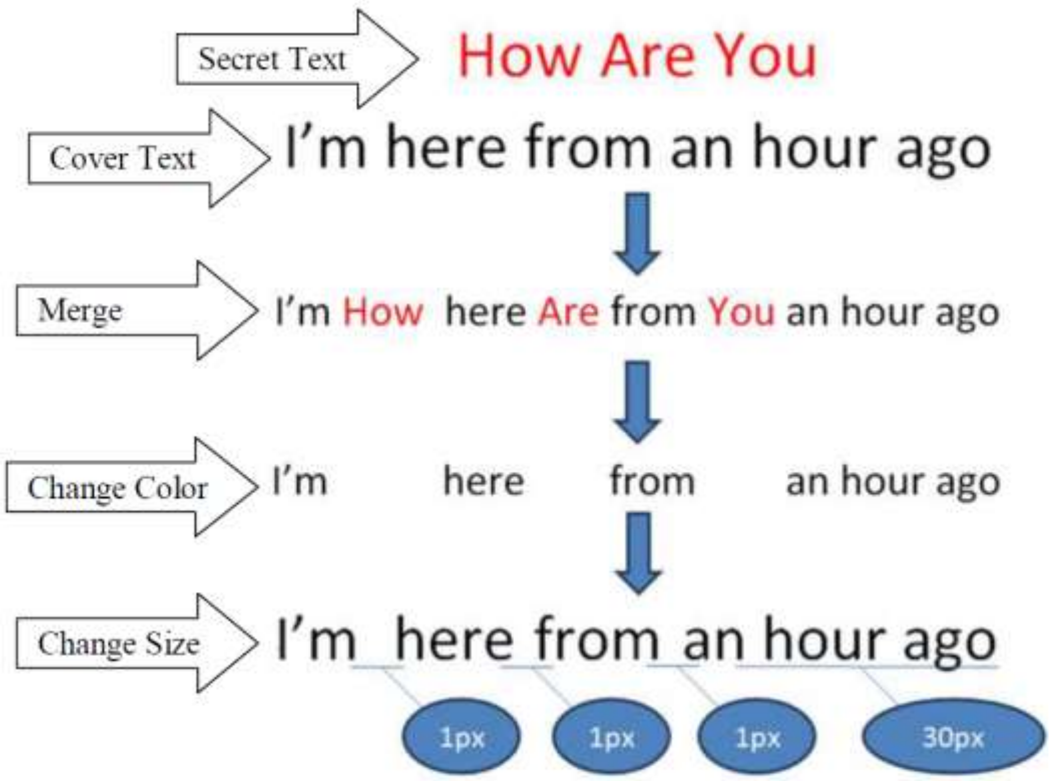

Secret Text
How Are You
Cover Text
I'm here from an hour ago
Merge
I'm How here Are from You an hour ago
Change Color
I'm here from an hour ago
Change Size
I'm here from an hour ago
1px
1px
1px
30px

Figure 2-5: Proposed data hiding technique – Text under Text

(Abdullah & Nasereddin, 2013)

(Brassil, Low, Maxemchuk, & O'Gorman, 1995) **Electronic Marking and Identification Techniques to Discourage Document Copying**. It is defined as "unauthorized dissemination" as distribution of documents without the knowledge of any payment to the publisher; this contrast legitimate document distribution by the publisher or the publisher's electronic document distributor. This paper describes a means of discouraging unauthorized copying and dissemination. The described techniques here are complementary to the security practices that can be applied to the legitimate distribution of documents. For example; a document can be encrypted prior to transmission across a computer network. Then even if the document file is intercepted or stolen from a database, it remains unreadable to those not possessing the decrypting key. The described techniques in this paper provide security after a document has been decrypted, and is thus readable to all. In addition its proposed encoding techniques can also make paper copies of documents traceable. In particular, the code word embedded in each document survives plain paper copying. Hence, these techniques can also be applied to "closely held" documents, such as confidential, limited distribution correspondence. The paper describe this both as a potential application of the methods and an illustration of their robustness in noise. Document marking can be achieved by altering the text formatting, or by altering certain characteristics of textual elements (e.g., characters). The goal in the design of coding methods is to develop alterations that are reliably decodable (even in the presence of noise) yet largely indiscernible to the reader. These criteria, reliable decoding and minimum visible change, are somewhat conflicting; herein lies the challenge in designing document marking techniques. Common to each technique is that a code word is embedded in the document by altering particular textual features. This paper describes these features for each method below and gives a simple comparison of the relative advantages and disadvantages of each technique. The three coding techniques that proposed:

1- Line-Shift Coding: This is a method of altering a document by vertically shifting the locations of text lines to encode the document uniquely.

2- Word-Shift Coding: This is a method of altering a document by horizontally shifting the locations of words within text lines to encode the document uniquely. This encoding can be applied to either the format file or to the bitmap of a page image. Decoding may be performed from the format file or bitmap.

3- Feature Coding: This is a coding method that is applied either to a format tile or to a bitmap image of a document. The image is examined for chosen text features, and those features are altered, or not altered, depending on the code word. Decoding requires the original image, or more specifically, a specification of the change in pixels at a feature.

Among the proposed encoding techniques, line-shifting is likely to be the most easily discernible by readers. Paper also expect line-shifting to be the most robust type of encoding in the presence of noise. This is because the long lengths of text lines provide a relatively easily detectable feature. For this reason, line shifting is particularly well suited to marking documents to be distributed in paper form, where noise can be introduced in printing and photocopying. This paper expects that word-shifting will be less discernible to the reader than line-shifting, since the spacing between adjacent words on a line is often varied to support text justification. Feature encoding can accommodate a particularly large number of sanctioned document recipients, since there are frequently two or more features available for encoding in each word. Feature alterations are also largely indiscernible to readers. A technically sophisticated “attacker” can detect that a document has been encoded by any of the three techniques paper have introduced. Such an attacker can also attempt to remove the encoding (e.g., produce a uuencoded document copy). The goal in the design of encoding techniques is to make successful attacks extremely difficult or costly.

(Bender, Gruhl, Morimoto, & Lu, 1996) **Data hiding in text.** Soft copy text is in many ways the most difficult place to hide data. (Hard-copy text can be treated as a highly structured image and is readily amenable to a variety of techniques such as slight variations in letter forms, kerning, baseline, etc.) This is due largely to the relative lack of redundant information in a text file as compared with a picture or a sound bite. While it is often possible to make imperceptible modifications to a picture, even an extra letter or period in text may be noticed by a casual reader. Data hiding in text is an exercise in the discovery of modifications that are not noticed by readers. Paper considered three major methods of encoding data: open space methods that encode through manipulation of white space (unused space on the printed page), syntactic methods that utilize punctuation, and semantic methods that encode using manipulation of the words themselves.

Open space methods. There are two reasons why the manipulation of white space in particular yields useful results. First, changing the number of trailing spaces has little chance of changing the meaning of a phrase or sentence. Second, a casual reader is unlikely to take notice of slight modifications to white space. Paper describes three methods of using white space to encode data. The methods exploit inter-sentence spacing, end-of-line spaces, and inter-word spacing in justified text.

The first method encodes a binary message into a text by placing either one or two spaces after each terminating character.

A second method of exploiting white space to encode data is to insert spaces at the end of lines. The data are encoded allowing for a predetermined number of spaces at the end of each line (Figure 2-6). Two spaces encode one bit per line, four encode two, and eight encode three.

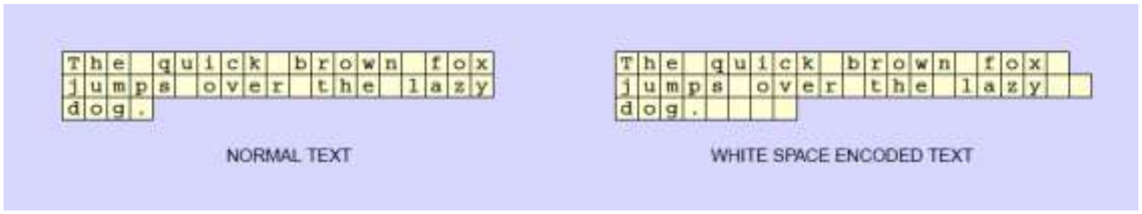
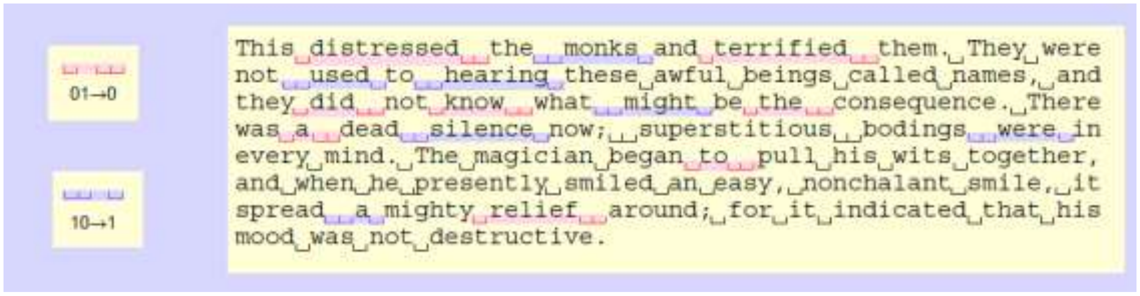


Figure 2-6: Example of data hidden using white space

(Bender, Gruhl, Morimoto, & Lu, 1996)

A problem unique to this method is that the hidden data cannot be retrieved from hard copy.

A third method of using white space to encode data involves justify format of text. Data are encoded by controlling where the extra spaces are placed. One space between words is interpreted as a "0." Two spaces are interpreted as a "1." This method results in several bits encoded on each line (Figure 2-7).



This distressed the monks and terrified them. They were
not used to hearing these awful beings called names, and
they did not know what might be the consequence. There
was a dead silence now; superstitious bodings were in
every mind. The magician began to pull his wits together,
and when he presently smiled an easy, nonchalant smile, it
spread a mighty relief around; for it indicated that his
mood was not destructive.

Figure 2-7: Data hidden through justification (text from A Connecticut Yankee in King Arthur's Court by Mark Twain)

(Bender, Gruhl, Morimoto, & Lu, 1996)

Open space methods are useful as long as the text remains in an ASCII (American Standard Character Interchange) format. As mentioned above, some data may be lost when the text is printed. Printed documents present opportunities for data hiding far beyond the capability of an ASCII text file. Data hiding in hard copy is accomplished by making slight variations in word and letter spacing, changes to the baseline position of letters or punctuation, changes to the letter forms themselves, etc. Also, image data-hiding techniques such as those used by Patchwork can be modified to work with printed text".

(Por, Wong, & Chee, 2012) **A text-based data hiding method using Unicode space characters**. This paper proposes a text-based data hiding method to insert external information into Microsoft Word document. The drawback of low embedding efficiency in the existing text-based data hiding methods is addressed, and a simple attack, DASH, is proposed to reveal the information inserted by the existing text-based data hiding methods. Then, a new data hiding method, UniSpaCh, is proposed to counter DASH. The characteristics of Unicode space characters with respect to embedding efficiency and DASH are analyzed, and the selected Unicode space characters are inserted into inter-sentence, inter-word, end-of-line and inter-paragraph spacing's to encode external information while improving embedding efficiency and imperceptivity of the embedded information. UniSpaCh is also reversible where the embedded information can be removed to completely reconstruct the original Microsoft Word document. Experiments were carried out to verify the performance of UniSpaCh as well as comparing it to the existing space manipulating data hiding methods. Results suggest that UniSpaCh offers higher embedding efficiency while exhibiting higher imperceptivity of white space manipulation when compared to the existing methods considered. In the best case scenario, UniSpaCh produces output document of size almost 9times smaller than that of the existing method.

(Vill´an, et al., 2006) **Text Data-Hiding for Digital and Printed Documents.** Proposed a new theoretical framework for the data-hiding problem of digital and printed text documents. Explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc. And also introduce color quantization, a new semi-fragile text data-hiding method that is fully automatable, has high information embedding rate, and can be applied to both digital and printed text documents. The main idea of this method is to quantize the color or luminance intensity of each character in such a manner that the human visual system is not able to distinguish between the original and quantized characters, but it can be easily performed by a specialized reader machine. The paper describes halftone quantization, a related method that applies mainly to printed text documents. Since these methods may not be completely robust to printing and scanning, an outer coding layer is proposed to solve this issue. Finally, describe a practical implementation of the color quantization method and present experimental results for comparison with other existing methods.

(Rahma, AbdulWahab, & Al-Noori, 2011) **Physical characteristics of computer system.** Proposed a method of data hiding by taking advantage of the physical characteristics of computer system and how it stores document file and treating it as a compound file. The unused block in Microsoft Compound Document File Format

(MCDFF) is used to hide data. The possibilities provided by Microsoft Word Processor program have also been utilized, such as Tools, to generate cover for hiding. The proposed system embeds steganography text in structure (Binary File Format) of digital and printed text document file which is a file of Microsoft Word Document file (Doc.) using two Processes: Cover Generation and Embedding Processes. Cover Generation Process: where the cover is a document of Microsoft Word Document file format 2003 (doc.) and will appear to be the product of a collaborative writing effort among authors using Track Changes tool. Embedding Process hides text string in unused block of binary file format of that document cover. This paper proposed a new technique, which gives good results, such that the user can hide 63byte in 34KB document cover size with informed about size of empty document=10/11KB, in addition, using Track Changes tool does not effect on hidden data and no problem was detect on hidden data at stego-document mailing or copying.

(Jebran, 2007) **Text 2Text Steganography**. Proposed a technique to build a simple application that is able to send and receive encrypted messages embedded in Rich Text Format: **.DOC*, **.RTF*, EMAIL /Message Body/, etc. The user has the ability to choose the fake text he wants and the program must be able to tell whether or not this fake text will suit the real text.

The user can set a different password for every message he sends. This will enable the manager to transmit to two groups two different messages with two different passwords using the same fake text. Thus, you will be able to send encrypted and hidden messages in any source code that you choose.

The proposed technique will not change the text itself, but it will change the unseen attributes of the text. These attributes are many and it is impossible for web servers to track them all. There are lots of Steganographic methods and tracking them will waste huge amounts of processing for uncertain results. Be aware that Steganography is more effective than encryption when used in the right way. The deletion of all attributes is not an option, so will choose the size and the color. Figure 2-8 will underscore the point.

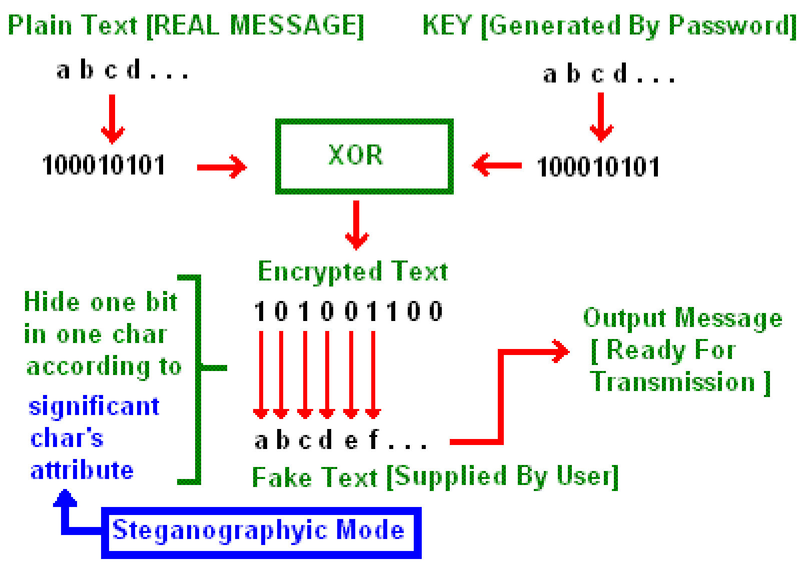

Plain Text [REAL MESSAGE]
KEY [Generated By Password]
a b c d . . .
a b c d . . .
100010101
XOR
100010101
Encrypted Text
1 0 1 0 0 1 1 0 0
Hide one bit in one char according to significant char's attribute
Output Message [Ready For Transmission]
a b c d e f . . .
Fake Text [Supplied By User]
Steganographyic Mode

Figure 2-8: Criteria of encrypt the text

(Jebran, 2007)

In mode of change font size will change the size of the characters in the fake text according to the selected font size and differential factor. Here will use 2 sizes, X1 and X2. X1 is the selected font size and X2 is the selected font size plus the differential factor. 0 bit is represented by the occurrence of the character whose size is X1. 1 bit is represented by the occurrence of the character whose size is X2.

Color changed mode is the more recommended mode for use, as it is very stable and safe. In this mode, will change the color of the chars in the fake text according to the selected color and the program's calculated color. Will use 2 colors, X1 and X2. X1 is the selected color and X2 is the program's calculated color. The proposed technique will search to find the nearest color for which it is impossible to recognize the difference with naked eye. 0 bit is represented by the occurrence of the character whose color is X1. 1 bit is represented by the occurrence of the character whose color is X2. The recipient must know which color you have chosen for decryption. After hiding the real message in the fake message, the rest of the fake message characters will be colored as X1”.

Chapter Three
The Proposed Model

This chapter represents a model for a proposed enhanced technique for data hiding – text under text by using open space methods. It explains the used supported concepts; methodology, enhanced technique procedures and the advantages of this enhanced technique.

Data hiding is important to conceal critical information from unauthorized persons. (Bender, Gruhl, Morimoto, & Lu, 1996) "Data hiding represents a class of processes used to embed data, such as copyright information, into various forms of media such as image, audio, or text with a minimum amount of perceivable degradation to the "host" signal".

Open space method is the first used methods to hide data in white space: between words, lines and paragraph. this method is divided into three methods:

The first, method encodes a binary message into text by placing either one or two spaces after each terminating character.

The second, method is exploiting white space to encode data to insert spaces at the end of lines. The data are encoded, allowing for a predetermined number of spaces at the end of each line (Figure 2-6). Two spaces encode one bit per line, four encode two, and eight encode three.

The third, method of using white space to encode data involves justified format of text. Data are encoded by controlling where the extra spaces are placed. One space between words is interpreted as a “0.” Two spaces are interpreted as a “1.” This method results in several bits encoded on each line (Figure 2-7).

(Bender, Gruhl, Morimoto, & Lu, 1996) There are two reasons why the manipulation of white space in particular yields useful results:

First, changing the number of trailing spaces has little chance of changing the meaning of a phrase or sentence.

Second, a casual reader is unlikely to take notice of slight modifications to white space.

The paper describes three methods of using white space to encode data. The methods exploit **inter-sentence spacing, end-of-line spaces, and inter-word spacing in justified text**.

3.1 Supported Concepts

The supported concepts are Character Indexes, ASCII Code Characters and Octal Numeral System.

Character Indexes

Character Indexes used in the proposed solution to get the index of secret characters from the cover text to be able to retrieve the secret text in the stage of the text show.

Please be informed that the character indexes begin count from 0.

Example, represent character indexes in the text. Suppose the cover text is:

Information hiding techniques

The character indexes for above sentence is appear in the table 3-1:

Table 3-1: Character indexes for cover text

Indexes	Characters set	Indexes	Characters set
0	I	15	i
1	n	16	n
2	f	17	g
3	o	18	
4	r	19	t
5	m	20	e
6	a	21	c
7	t	22	h
8	i	23	n
9	o	24	i
10	n	25	q
11		26	u
12	h	27	e
13	i	28	s
14	d		

So, from above example, suppose the secret text is: **hi man**

Table 3-2 shows the index of the secret text characters:

Table 3-2: Character indexes for secret text

Indexes	Characters set
12	h
0	i
11	(space)
5	m
6	a
1	n

The “h” character is repeated twice in the cover text, in this case, the index of the first one is enough to know the intended character.

ASCII Code Characters

The characters’ indexes are not enough to indicate the intended character from the cover text, so, ASCII code characters give a unique code for each character in the secret text “with sensitive case”.

Table 3-3: ASCII code characters

Description	Symbol	Code	Description	Symbol	Code
Space		32	Uppercase N	N	78
Exclamation mark	!	33	Uppercase O	O	79
Double quotes	“	34	Uppercase P	P	80
Number	#	35	Uppercase Q	Q	81
Dollar	$	36	Uppercase R	R	82
Procenttecken	%	37	Uppercase S	S	83
Ampersand	&	38	Uppercase T	T	84
Single quote	‘	39	Uppercase U	U	85
Open parenthesis	(	40	Uppercase V	V	86
Close parenthesis	)	41	Uppercase W	W	87
Asterisk	*	42	Uppercase X	X	88
Plus	+	43	Uppercase Y	Y	89
Comma	,	44	Uppercase Z	Z	90
Hyphen	-	45	Lowercase a	a	97
Period, dot or full stop	.	46	Lowercase b	b	98
Slash or divide	/	47	Lowercase c	c	99
Zero	0	48	Lowercase d	d	100
One	1	49	Lowercase e	e	101
Two	2	50	Lowercase f	f	102
Three	3	51	Lowercase g	g	103
Four	4	52	Lowercase h	h	104
Five	5	53	Lowercase i	i	105
Six	6	54	Lowercase j	j	106
Seven	7	55	Lowercase k	k	107
Eight	8	56	Lowercase l	l	108
Nine	9	57	Lowercase m	m	109
Uppercase A	A	65	Lowercase n	n	110
Uppercase B	B	66	Lowercase o	o	111
Uppercase C	C	67	Lowercase p	p	112
Uppercase D	D	68	Lowercase q	q	113
Uppercase E	E	69	Lowercase r	r	114
Uppercase F	F	70	Lowercase s	s	115
Uppercase G	G	71	Lowercase t	t	116
Uppercase H	H	72	Lowercase u	u	117
Uppercase I	I	73	Lowercase v	v	118
Uppercase J	J	74	Lowercase w	w	119
Uppercase K	K	75	Lowercase x	x	120
Uppercase L	L	76	Lowercase y	y	121
Uppercase M	M	77	Lowercase	z	122

Any character exists in the secret text; not exists in the cover text; will take the character code from the above table “Table 3-3” ASCII code characters.

Octal Numeral System

The octal numeral system used to convert the decimal codes as octal numbers for:

1- Security reasons, in case anyone finds out the hidden data, the data will be meaningless and unreadable. The secret text will be formatted then merged within the cover text.

2- Identify character, after getting the character index for the secret text as octal numbers; these numbers will be separated by inserting the numbers eight and nine between the octal numbers to dedicate each character number.

3.2 Hide data technique

Section 3.1 discussed the supported concepts Character Indexes, ASCII Code Characters and Octal Numeral System. These concepts will used in the proposed solution by combine them to hide text under text by using open space methods.

The idea is to take advantage of the unused white space from the cover text, but before that some changes should be made on the secret text. Then, hide them within the cover text; the changes on the secret text will be as the following:

3.2.1 Format Secret Text

Formatting secret text by extracting the index of the secret text characters from cover text and convert the index numbers from decimal numeral system into octal numeral system as the following:

Step One: Index of the secret text characters:

As mentioned above; there are two texts used in the proposed solution, secret text and cover text, the proposed solution will hide the secret text within the cover text.

This section will show in detail how to extract the index of the secret text character from the cover text by using the following steps:

First. Taking each character from the secret text.

Second. Looking for character in the cover text.

Third. Taking the index of the character from the cover text.

Fourth. Compilation of the indexes taken from the cover text in the array.

For more clarification; below is an example to get index of the secret text character from the cover text:

Suppose the cover text is: **Information hiding techniques**

And the secret text is: **hi man**

The table 3-4 show the index for each character in the cover text while table 3-5 show the index of the secret text characters in the cover text and it's shown in the highlighted columns.

Table 3-4: Characters index for cover text

I	n	f	o	r	m	a	t	i	o	n		h	i	d	i	n	g		t	e	c	h	n	i	q	u	e	s
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

Table 3-5: Characters index for secret text in the cover text

I	n	f	o	r	m	a	t	i	o	n		h	i	d	i	n	g		t	e	c	h	n	i	q	u	e	s
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

The steps of take these indexes will be like the following:

I. Create an empty array for secret characters:

"**hi man**" is 6 characters so the array length is 6.

II. Take the first character from the secret text "h".

III. Look for "h" character in the cover text; what is needed is the first "h" character of the cover text.

IV. Add the index of "h" character in the array:

V. Take the second character from the secret text "i".

VI. Look for "i" character in the cover text; what is needed is the first "i" character of the cover text.

VII. Add the index of "i" character in the array:

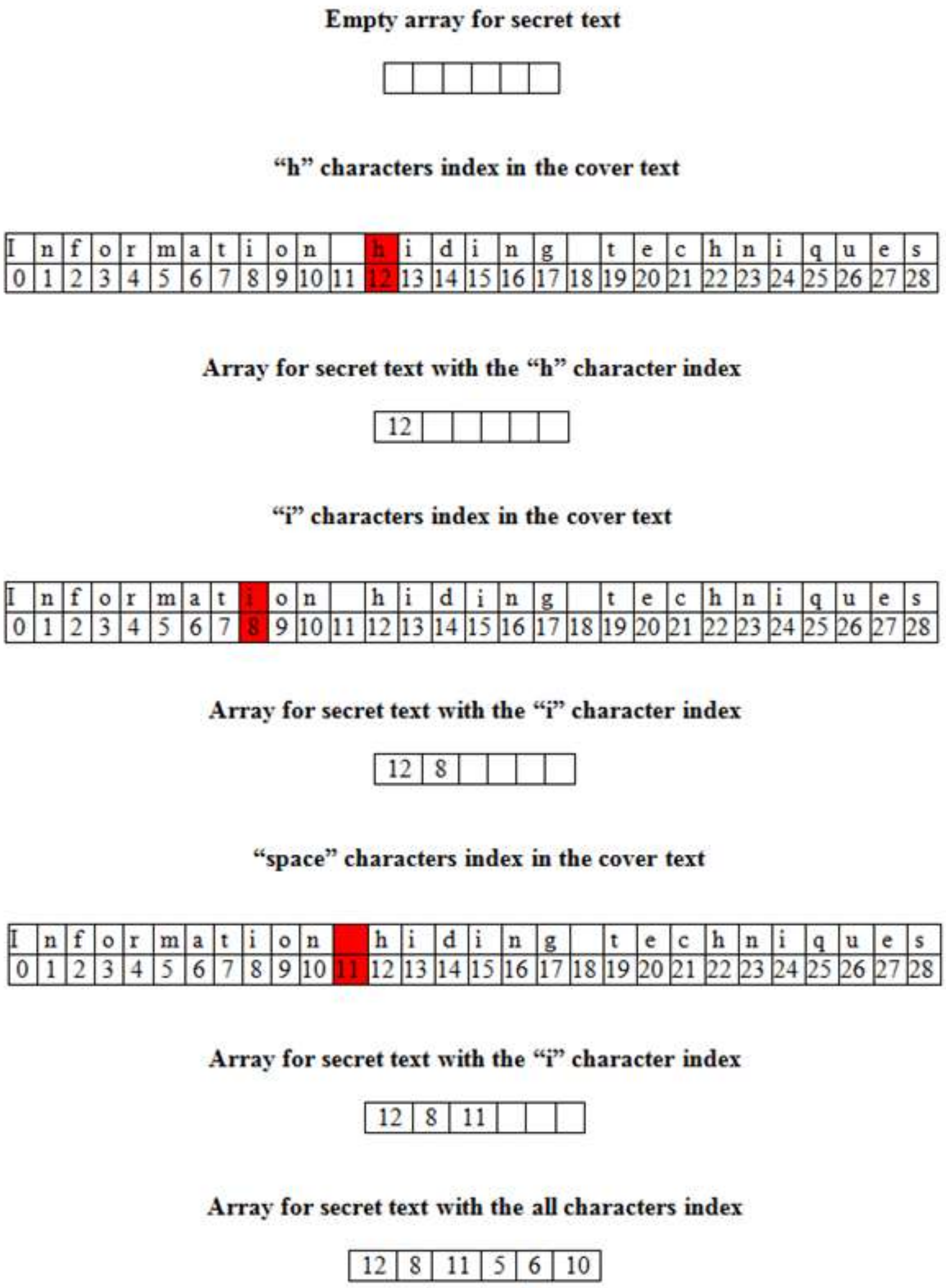
VIII. Take the third character from the secret text " "; it's space.

IX. Look for “space” character in the cover text; what is needed is the first “space” character of the cover text.

X. Add the index of “space” character in the array:

XI. Do the above steps for all secret characters until fill the index array of secret text.

After these steps, the cover text contains all the characters in the secret text and the indexes is taken from the cover text and saved on the array, Figure 3-1 will show the steps of taking the indexes.



Empty array for secret text

"h" characters index in the cover text

I	n	f	o	r	m	a	t	i	o	n		h	i	d	i	n	g		t	e	c	h	n	i	q	u	e	s
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

Array for secret text with the "h" character index

12					

"i" characters index in the cover text

I	n	f	o	r	m	a	t	i	o	n		h	i	d	i	n	g		t	e	c	h	n	i	q	u	e	s
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

Array for secret text with the "i" character index

12	8				

"space" characters index in the cover text

I	n	f	o	r	m	a	t	i	o	n		h	i	d	i	n	g		t	e	c	h	n	i	q	u	e	s
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

Array for secret text with the "i" character index

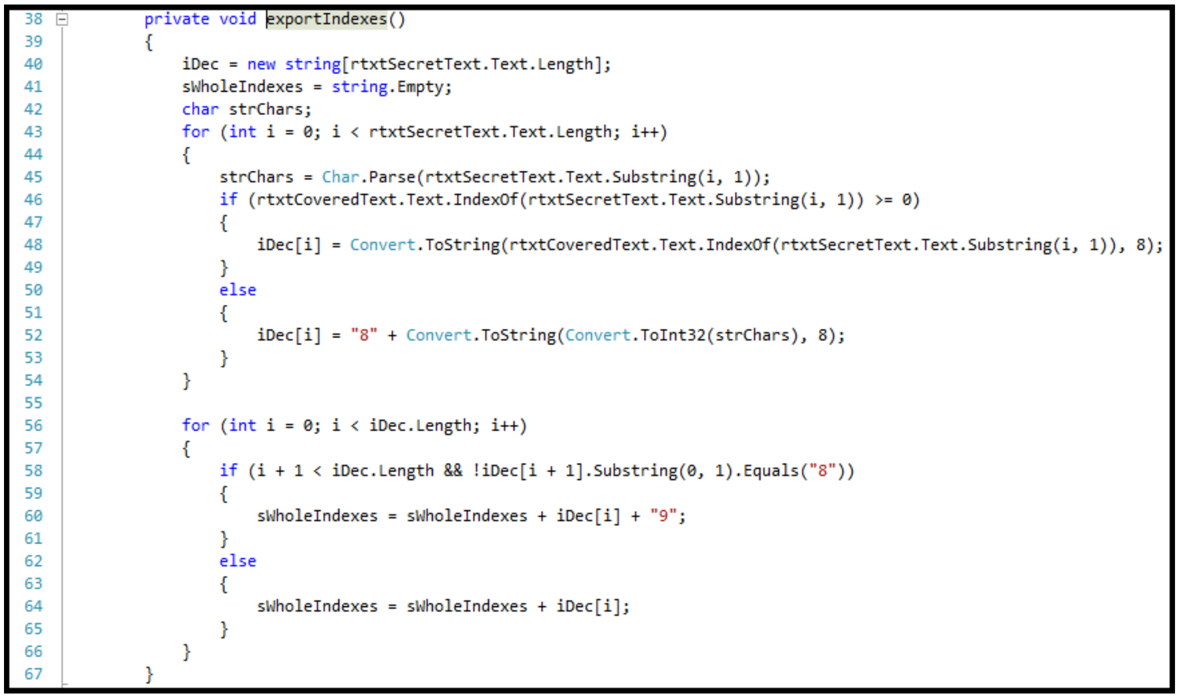
12	8	11			

Array for secret text with the all characters index

12	8	11	5	6	10

Figure 3-1: Steps of taking the indexes

Figure 3-2 show the part of code which for exporting the indexes of secret text characters from the cover text.



```
        private void exportIndexes()
        {
            iDec = new string[rtxtSecretText.Text.Length];
            sWholeIndexes = string.Empty;
            char strChars;
            for (int i = 0; i < rtxtSecretText.Text.Length; i++)
            {
                strChars = Char.Parse(rtxtSecretText.Text.Substring(i, 1));
                if (rtxtCoveredText.Text.IndexOf(rtxtSecretText.Text.Substring(i, 1)) >= 0)
                {
                    iDec[i] = Convert.ToString(rtxtCoveredText.Text.IndexOf(rtxtSecretText.Text.Substring(i, 1)), 8);
                }
                else
                {
                    iDec[i] = "8" + Convert.ToString(Convert.ToInt32(strChars), 8);
                }
            }

            for (int i = 0; i < iDec.Length; i++)
            {
                if (i + 1 < iDec.Length && !iDec[i + 1].Substring(0, 1).Equals("8"))
                {
                    sWholeIndexes = sWholeIndexes + iDec[i] + "9";
                }
                else
                {
                    sWholeIndexes = sWholeIndexes + iDec[i];
                }
            }
        }
```

Figure 3-2: The code of export the indexes of the secret text characters

This section shows how to extract the index of the secret text characters; still, there are problems might be faced: the character in the secret text does not exist in the cover text or the character in the secret text exists in the cover text but not in the same capitalization", solutions are presented in the next section.

Step Two: ASCII Code Characters of the secret text characters:

Get the character code of the secret text characters from the ASCII code characters in case the character in the secret text does not exist in the cover text characters or it does not exist in the cover text characters with the same capitalization.

In these cases, follow the below proposed solution:

First. Take each character from the secret text.

Second. Look for the character in the cover text.

Third. If the character does not exist in the cover text or exists with different capitalization; get the character code from the ASCII code characters as it's listed in Table 3-3.

Fourth. Compile the indexes taken from the cover text in the array.

For more clarification; the below is example to get index of the secret text character from the cover text:

Suppose the cover text is: **Information hiding techniques**

Suppose the secret text is: **Hi man**

The table 3-6 show the index for each character in the cover text while table 3-7 show the index of the secret text characters in the cover text and it's shown in the highlighted columns but "H" character does not exists in the cover text with the same capitalization.

Table 3-6: Characters index for cover text

I	n	f	o	r	m	a	t	i	o	n		h	i	d	i	n	g		t	e	c	h	n	i	q	u	e	s
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

Table 3-7: Characters index for secret text in the cover text except not exists character

I	n	f	o	r	m	a	t	i	o	n		h	i	d	i	n	g		t	e	c	h	n	i	q	u	e	s
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

In this case; the solution is to take the character code for "H" character from the ASCII code characters.

The steps to take these indexes will be similar to the steps in step one, but the "H" character will take its code from ASCII code characters. And the result array of the secret text characters will be as it appears in the Table 3-8:

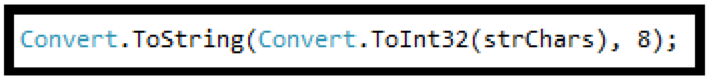
Table 3-8: Array for secret text with the all characters index

72	8	11	5	6	10

The first column "72" is the ASCII code for uppercase "H" character, taken from Table 3-3, but the other columns in the index of the remaining secret text characters are taken from the cover text characters.

Once reaching the phase to retrieve the hidden data from the cover text, indexes should be distinguished from the ASCII codes to be able determine the source of the character of the given number "know the source of the number either it's index or ASCII code"; so; what if there is number 72 in the cover text for another character? the problem is solved in step three.

Figure 3-3 show the part of code which for exporting the ASCII code of secret text characters.



```
Convert.ToString(Convert.ToInt32(strChars), 8);
```

Figure 3-3: The code of export the ASCII code for the secret text characters

Step Three: Octal numeral system:

After collecting the index of the secret character from the cover text and ASCII code for the characters of the secret text that do not exist in the cover text characters with the same capitalization; convert the result array of numbers from decimal numeral system into octal numeral system.

The conversion shall follow the below procedures:

1- Every character of the secret text exists in the cover text characters will be taken from the index of these characters. Then, convert this index from decimal numeral system into octal numeral system.

2- Every character of the secret text that does not exist in the cover text characters will be taken from the ASCII codes, then follow the below steps:

   a. Convert the ASCII code from decimal numeral system into octal numeral system.

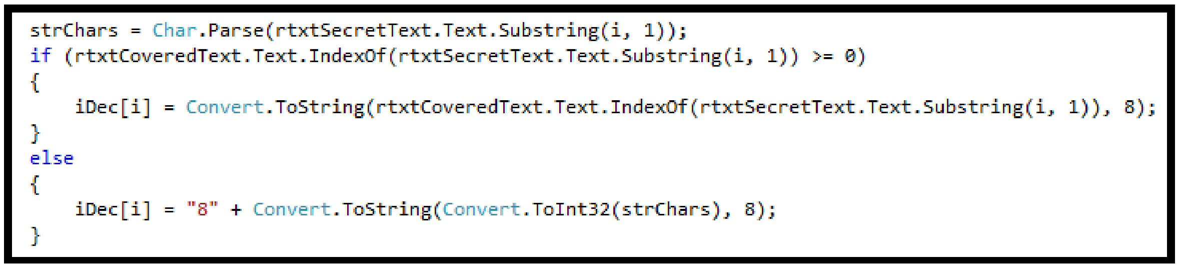
   b. Add number “8” begin of the converted octal number; this step to distinguish the number is from ASCII code characters.

3- Compile the indexes and the ASCII codes for secret text characters in the same order in the text by inserting the number “9” between these numbers to separate the characters indexes and ASCII codes from each other.

For more information; the octal number doesn’t contain the 8 and 9 numbers. This feature enables us to use those numbers to:

1- Separate between characters indexes and ASCII codes.
2- Distinguish that the number is coded from ASCII code characters or index from secret text characters index in the cover text characters.

Figure 3-4 show the part of code which for convert the decimal numeral system into octal numeral system.



```
strChars = Char.Parse(rtxtSecretText.Text.Substring(i, 1));
if (rtxtCoveredText.Text.IndexOf(rtxtSecretText.Text.Substring(i, 1)) >= 0)
{
    iDec[i] = Convert.ToString(rtxtCoveredText.Text.IndexOf(rtxtSecretText.Text.Substring(i, 1)), 8);
}
else
{
    iDec[i] = "8" + Convert.ToString(Convert.ToInt32(strChars), 8);
}
```

Figure 3-4: The code of convert the decimal numeral system into octal numeral system.

3.2.2 Hide the secret text:

The used way to hide the secret text is to merge the result array number of secret text characters index and the ASCII code within the cover text in the unused white spaces between cover text words.

The data will be hidden with font size 1pt and color as cover text back color, by follow these steps:

Step One: Detect the number of digits:

To merge the result array number of secret text characters index and ASCII code within the cover text in the unused white spaces between cover words; you must specify how many digits should be inserted between each word in the unused white spaces.

The number of digits determined depends on the font size; for each font size there is a space size between words, these spaces will have the secret data and the size of this data should be the same of the empty space size in the cover text.

The font size scale is the point; (wikipedia, Point (typography), 2013) “In typography, a point is the smallest unit of measure, being a subdivision of the larger pica. It is commonly abbreviated as pt. The point has long been the usual unit for measuring font size and leading and other minute items on a printed page. The original printer's point, from the era of foundry metal typesetting and letter press printing, varied between 0.18 and 0.4 mm depending on various definitions of the foot. By the end of the 19th Century, it had settled to around 0.35 to 0.38 mm, depending on one’s geographical location.

In the late 1980s to the 1990s, the traditional point was supplanted by the desktop publishing point (also called the PostScript point), which was defined as 72 points to the inch (1 point = 1⁄72 inches = 25.4⁄72 mm = 0.3527 mm). In either system, there are 12 points to the pica. In metal type, the point size of the font described the size (height) of the metal body on which the typeface's characters were cast. In digital type, the body is now an imaginary design space, but is used as the basis from which the type is scaled.

A measurement in picas is usually represented by placing a lower case p after the number, such as "10p" meaning "10 picas." Points are represented by placing the number of points after the p, such as 0p5 for "5 points," 6p2 for "6 picas and 2 points," or 1p1 for "13 points" which is converted to a mixed fraction of 1 pica and 1 point. (An alternate nomenclature is described in the pica article.)”.

The space between words will be different depending on the font size. (wikipedia, Sentence spacing, 2013) “Sentence spacing is the horizontal space between sentences in typeset text. It is a matter of typographical convention. Since the introduction of movable-type printing in Europe, various sentence spacing conventions have been used in languages with a Latin-derived alphabet. These include a normal word space (as between the words in a sentence), a single enlarged space, two full spaces, and, most recently in digital media, no space. Although modern digital fonts can automatically adjust a single word space to create visually pleasing and consistent spacing following terminal punctuation, most debate is about whether to strike a keyboard's spacebar once or twice between sentences. Traditionally, two spaces could distinguish from a mid-sentence abbreviation or initials, as in, "He was faster than I. P. Jones was next."”

According to results shown in chapter four; the best number of digits with size 1pt to hide within the cover text is as appearing in table 3-9:

Table 3-9: The best number of digits with size 1pt to hide within the cover text

Cover text font size	Number of hidden digits
10	1
12	2
14	3
16	4

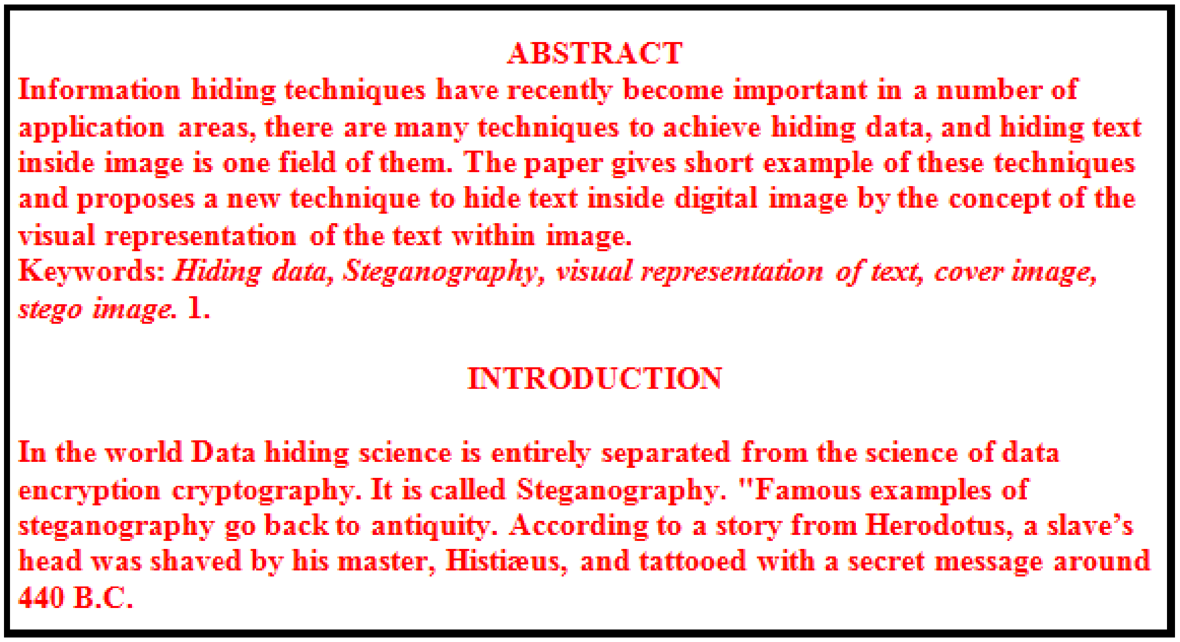
Step Two: Merge the index and ASCII code numbers in the cover text:

After detecting the number of digits according to the cover text font size; merge these digits within the cover text in the white spaces, this operation will be done in iterations.

Before that, the indexes array must be done and ready to work on it and hide it, so; suppose the following: The secret text is:

"**Become important in $ a number of application areas**"

The cover text is shown in figure 3-5



ABSTRACT

Information hiding techniques have recently become important in a number of application areas, there are many techniques to achieve hiding data, and hiding text inside image is one field of them. The paper gives short example of these techniques and proposes a new technique to hide text inside digital image by the concept of the visual representation of the text within image.

Keywords: *Hiding data, Steganography, visual representation of text, cover image, stego image.* 1.

INTRODUCTION

In the world Data hiding science is entirely separated from the science of data encryption cryptography. It is called Steganography. "Famous examples of steganography go back to antiquity. According to a story from Herodotus, a slave’s head was shaved by his master, Histiæus, and tattooed with a secret message around 440 B.C.

Figure 3-5: The cover text

The font size of the cover text is 12pt, so; the best number of digits to hide in each white space is 2 digits “as explained in step one”.

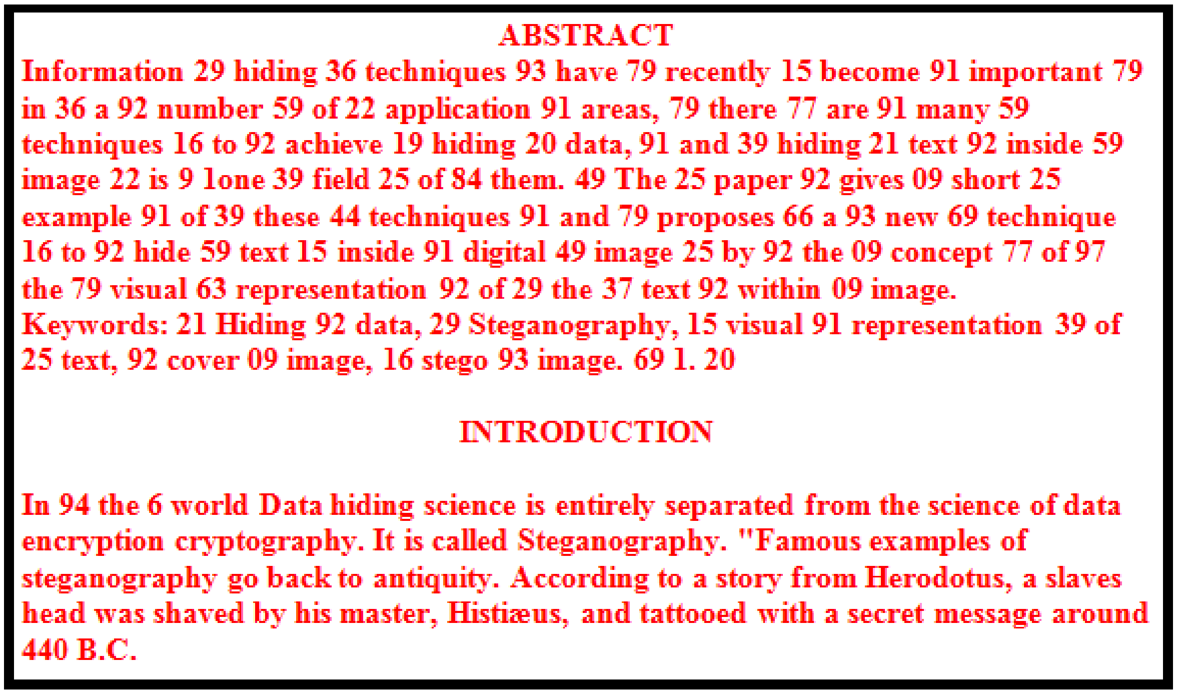
The result array of the indexes and ASCII codes for the secret text characters in the cover text in octal numeral system is:

“29369379159179369259229179779159169219209139219259229139258449259209259139449179669369169259159149259209779779639229379209219229159139259209169369209 46”

This array will be divided into two digits together and then merge it like the following:

"29, 36, 93, 79, 15, 91, 79, 36, 92, 59, 22, 91, 79, 77, 91, 59, 16, 92, 19, 20, 91, 39, 21, 92, 59, 22, 91, 39, 25, 84, 49, 25, 92, 09, 25, 91, 39, 44, 91, 79, 66, 93, 69, 16, 92, 59, 15, 91, 49, 25, 92, 09, 77, 97, 79, 63, 92, 29, 37, 92, 09, 21, 92, 29, 15, 91, 39, 25, 92, 09, 16, 93, 69, 20, 94, 6"

Every two digits will be insert in white spaces of cover text is shown in Figure 3-6:



ABSTRACT

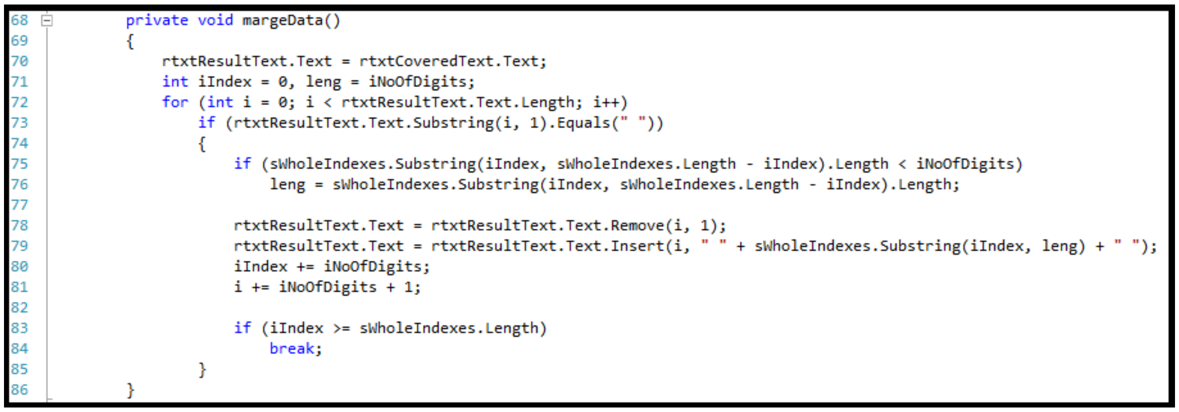
Information 29 hiding 36 techniques 93 have 79 recently 15 become 91 important 79 in 36 a 92 number 59 of 22 application 91 areas, 79 there 77 are 91 many 59 techniques 16 to 92 achieve 19 hiding 20 data, 91 and 39 hiding 21 text 92 inside 59 image 22 is 9 lone 39 field 25 of 84 them. 49 The 25 paper 92 gives 09 short 25 example 91 of 39 these 44 techniques 91 and 79 proposes 66 a 93 new 69 technique 16 to 92 hide 59 text 15 inside 91 digital 49 image 25 by 92 the 09 concept 77 of 97 the 79 visual 63 representation 92 of 29 the 37 text 92 within 09 image. Keywords: 21 Hiding 92 data, 29 Steganography, 15 visual 91 representation 39 of 25 text, 92 cover 09 image, 16 stego 93 image. 69 1. 20

INTRODUCTION

In 94 the 6 world Data hiding science is entirely separated from the science of data encryption cryptography. It is called Steganography. "Famous examples of steganography go back to antiquity. According to a story from Herodotus, a slaves head was shaved by his master, Histiæus, and tattooed with a secret message around 440 B.C.

Figure 3-6: Insert digits in the white space of the cover text

Figure 3-7 show the code which for marge the array numbers within the cover text.



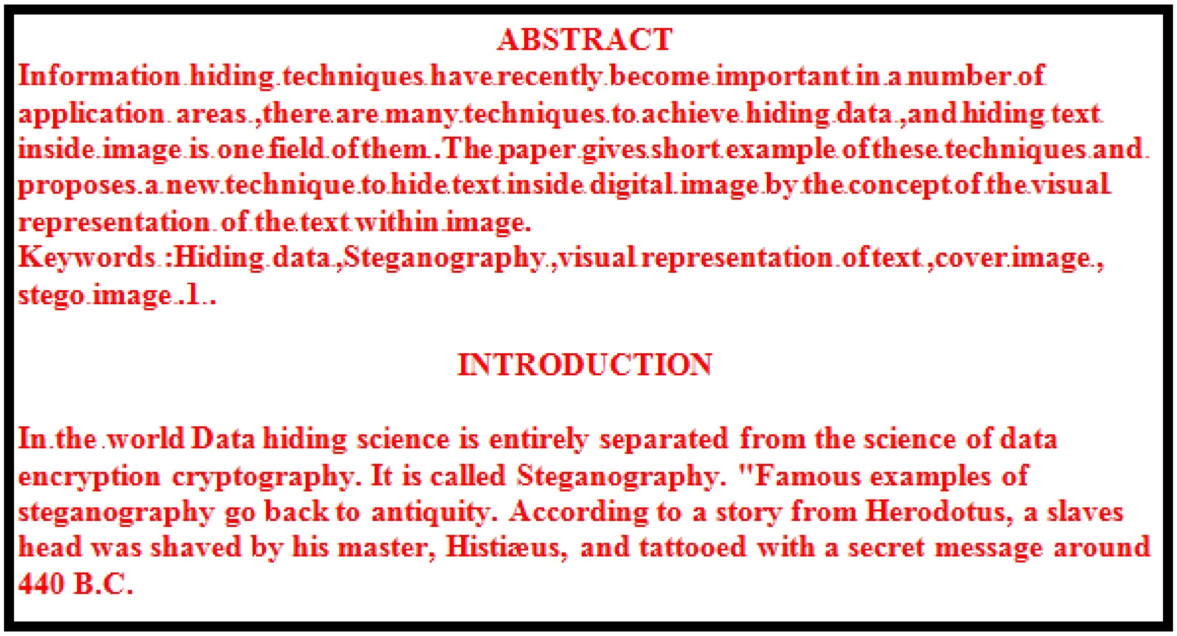
```
        private void margeData()
        {
            rtxtResultText.Text = rtxtCoveredText.Text;
            int iIndex = 0, leng = iNoOfDigits;
            for (int i = 0; i < rtxtResultText.Text.Length; i++)
                if (rtxtResultText.Text.Substring(i, 1).Equals(" "))
                {
                    if (sWholeIndexes.Substring(iIndex, sWholeIndexes.Length - iIndex).Length < iNoOfDigits)
                        leng = sWholeIndexes.Substring(iIndex, sWholeIndexes.Length - iIndex).Length;

                    rtxtResultText.Text = rtxtResultText.Text.Remove(i, 1);
                    rtxtResultText.Text = rtxtResultText.Text.Insert(i, " " + sWholeIndexes.Substring(iIndex, leng) + " ");
                    iIndex += iNoOfDigits;
                    i += iNoOfDigits + 1;

                    if (iIndex >= sWholeIndexes.Length)
                        break;
                }
        }
```

Figure 3-7: The code of marge the array numbers within cover text.

Step Three: Change font size and color for the numbers:

All inserted digits will be format as font size 1pt and color as cover text back color "white color".

The result of change font size to 1pt is shown in figure 3-8:



ABSTRACT

Information.hiding.techniques.have.recently.become.important.in.a.number.of application. areas.,there.are.many.techniques.to.achieve.hiding.data.,and.hiding.text inside.image.is.one.field.of.them..The.paper.gives.short.example.of.these.techniques.and. proposes.a.new.technique.to.hide.text.inside.digital.image.by.the.concept.of.the.visual representation. of.the.text.within.image.

Keywords.:Hiding.data.,Steganography.,visual.representation.of.text.,cover.image., stego.image..1..

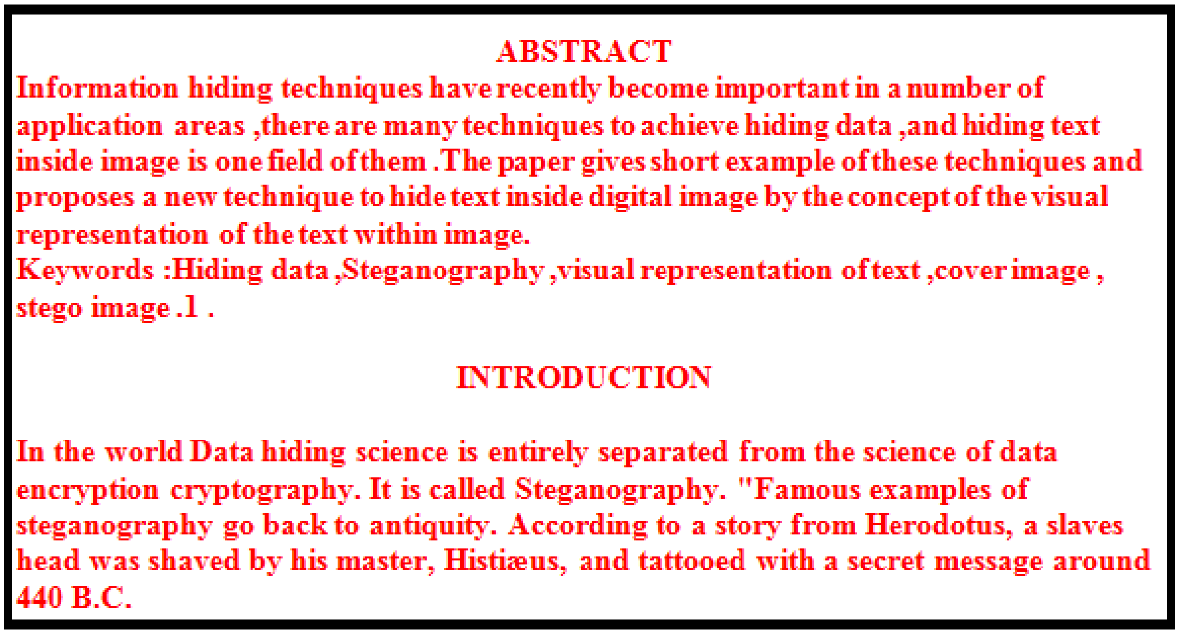
INTRODUCTION

In.the.world Data hiding science is entirely separated from the science of data encryption cryptography. It is called Steganography. "Famous examples of steganography go back to antiquity. According to a story from Herodotus, a slaves head was shaved by his master, Histiæus, and tattooed with a secret message around 440 B.C.

Figure 3-8: The result with hidden data font size 1pt

Then will change the color of the digits as the back color of the cover text “white color”.

The final result of hidden secret text in the cover text is shown in figure 3-9:



ABSTRACT

Information hiding techniques have recently become important in a number of application areas ,there are many techniques to achieve hiding data ,and hiding text inside image is one field of them .The paper gives short example of these techniques and proposes a new technique to hide text inside digital image by the concept of the visual representation of the text within image.

Keywords :Hiding data ,Steganography ,visual representation of text ,cover image , stego image .1 .

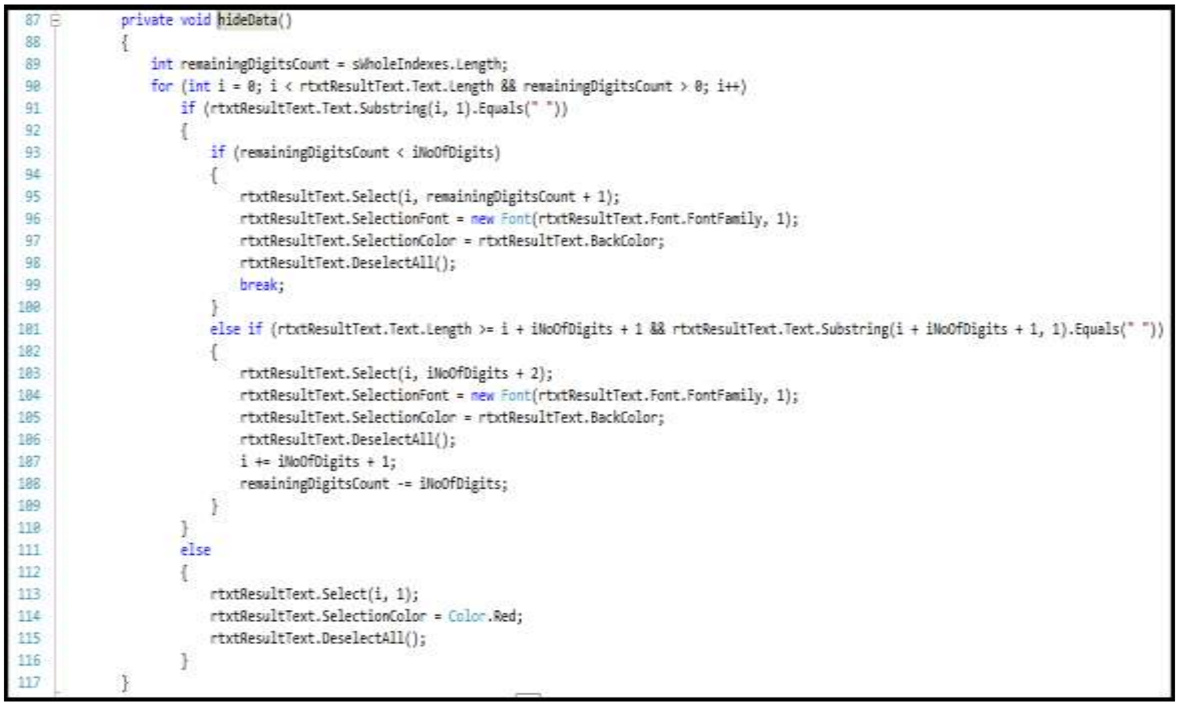
INTRODUCTION

In the world Data hiding science is entirely separated from the science of data encryption cryptography. It is called Steganography. "Famous examples of steganography go back to antiquity. According to a story from Herodotus, a slaves head was shaved by his master, Histiæus, and tattooed with a secret message around 440 B.C.

Figure 3-9: The result with hidden data font size 1pt and white color

In case the cover text back color is not white; the proposed solution will change the secret text color as cover text back color.

Figure 3-10 show the code which for change secret text font size and color.



```
private void hideData()
{
    int remainingDigitsCount = sWholeIndexes.Length;
    for (int i = 0; i < rtxtResultText.Text.Length && remainingDigitsCount > 0; i++)
        if (rtxtResultText.Text.Substring(i, 1).Equals(" "))
        {
            if (remainingDigitsCount < iNoOfDigits)
            {
                rtxtResultText.Select(i, remainingDigitsCount + 1);
                rtxtResultText.SelectionFont = new Font(rtxtResultText.Font.FontFamily, 1);
                rtxtResultText.SelectionColor = rtxtResultText.BackColor;
                rtxtResultText.DeselectAll();
                break;
            }
            else if (rtxtResultText.Text.Length >= i + iNoOfDigits + 1 && rtxtResultText.Text.Substring(i + iNoOfDigits + 1, 1).Equals(" "))
            {
                rtxtResultText.Select(i, iNoOfDigits + 2);
                rtxtResultText.SelectionFont = new Font(rtxtResultText.Font.FontFamily, 1);
                rtxtResultText.SelectionColor = rtxtResultText.BackColor;
                rtxtResultText.DeselectAll();
                i += iNoOfDigits + 1;
                remainingDigitsCount -= iNoOfDigits;
            }
        }
        else
        {
            rtxtResultText.Select(i, 1);
            rtxtResultText.SelectionColor = Color.Red;
            rtxtResultText.DeselectAll();
        }
}
```

Figure 3-10: The code of change secret text font size and color.

Chapter Four
Experimental Results

In order to evaluate the proposed enhanced technique, the implementation of the proposed technique is done as a program to simulate hiding text under text by providing secret text and cover text to the program and hiding the secret text under the cover text. This will compare the proposed solution with the white space method that involves justifying format of a text which is proposed in chapter two “see Figure 2-7”. Also, it will compare the hiding process in many aspects, and check the results data in several cases. The evaluation will check another implementation of hiding text under text by changing text size and color. The other implementation is done by (Jebran, 2007).

Evaluation methods are designed to cover the main factors of data hiding process; process speed, result text size and matching result text with the original cover text. This chapter is divided into sections; each section covers one of the evaluation methods. This chapter starts with process speed evaluation, then evaluating the size of the result text, and the last section evaluates the matching of the result text with the original cover text.

At the end of this chapter; results of the proposed implementation in variant cases are provided: the use of one, two, three, four and five digits in cover text size 10pt, 12pt, 14pt and 16pt.

4.1 Comparing the Proposed Solution with White Space Method:

White space is part of data hiding text into text. Some of white space problems are.

To hide two words like “Top Secret” requires text size cover of more than 80 words; because each character size is 8 bit “1 byte” and each bit requires one space. That means “T+o+p+ +S+e+c+r+e+t” equal 10 characters, 10 characters multiply by 8 bits equal 80 bits.

To be able to hide a large secret message; the result will be a very large message.

In a properly justified format of text, not all spaces are available to be used to hide the required data.

By using the proposed solution; above problems are solved, and the result is:

To hide two words like “Top Secret” in text size 12pt and by using 2 digits; requires text size cover not more than 20 words in case the result of each character index or ASCII code is 4 digits; that means “T+o+p+ +S+e+c+r+e+t” equal 10 characters, each character index 4 digits multiply by 10 characters equal 40 digits, 40 digits divided into two digits equal 20 parts, each part disappears in one white space.

To be able to hide a large secret message; the result will be close to the size of the original text.

In a properly justified format of text, all spaces are available to be used to hide the required data.

4.2 Evaluating the Speed of the Data Hiding Process

The speed process is different from case to another, the cases of evaluating the speed process are: hiding secret text under cover text using cover text size 10pt, hiding secret text under cover text using cover text size 12pt, hiding secret text under cover text using cover text size 14pt and hiding secret text under cover text using cover text size 16pt.

The comparison will be between the proposed implementation and The other implementation is done by (Jebran, 2007).

Hiding data by using size 10pt

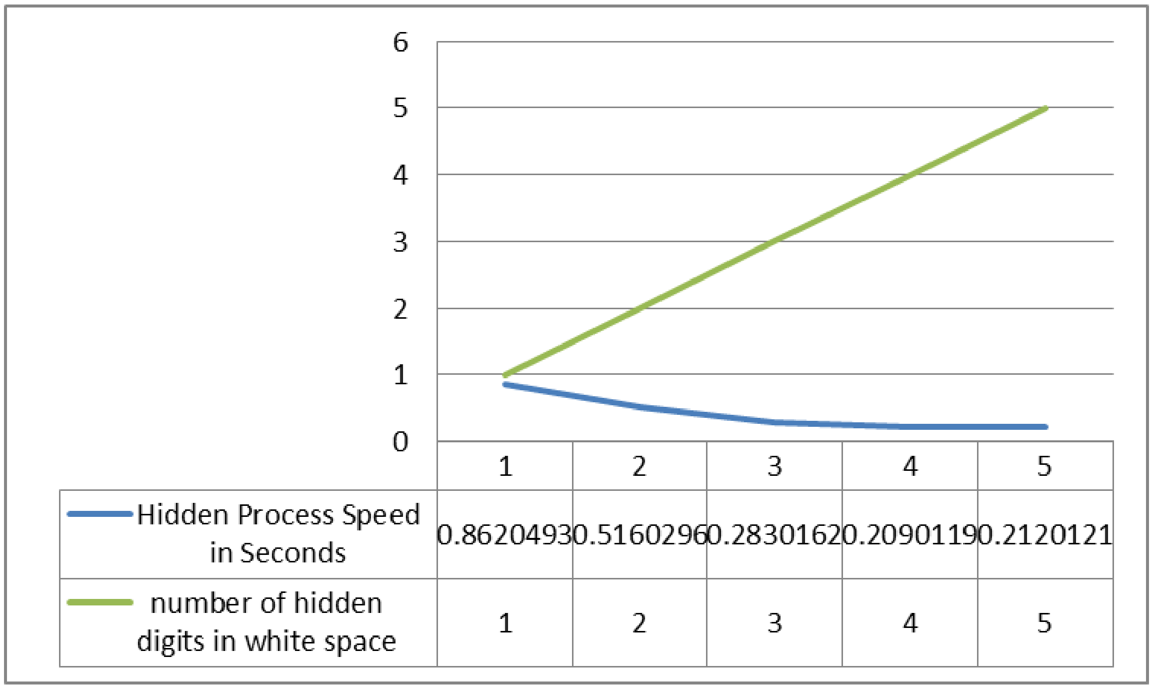
Table 4-1: Speed results of hiding secret text using Cover text size 10pt

Number of hidden digits in white space	Hidden Process Speed in Seconds
1	0.8620493
2	0.5160296
3	0.2830162
4	0.2090119
5	0.2120121

(a) The proposed implementation

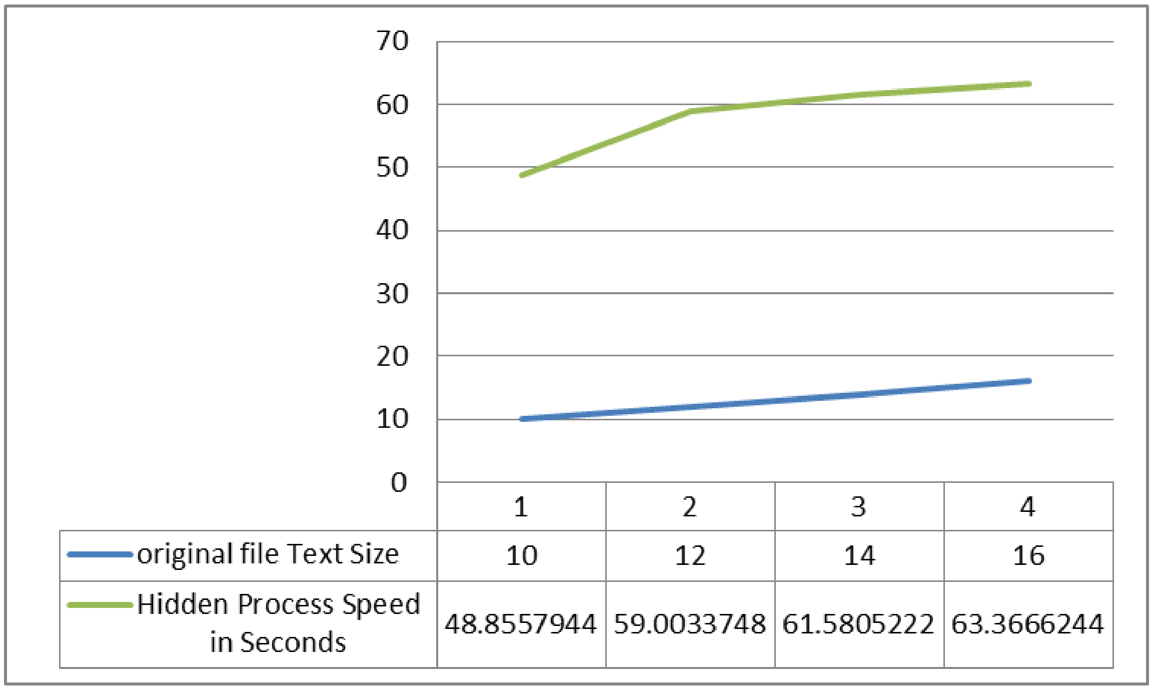
Original file Text Size	Hidden Process Speed in Seconds
10	**48.8557944**
12	59.0033748
14	61.5805222
16	63.3666244

(b) Jebran implementation



	1	2	3	4	5
Hidden Process Speed in Seconds	0.8620493	0.5160296	0.2830162	0.2090119	0.2120121
number of hidden digits in white space	1	2	3	4	5

(a) The proposed implementation



	1	2	3	4
original file Text Size	10	12	14	16
Hidden Process Speed in Seconds	48.8557944	59.0033748	61.5805222	63.3666244

(b) Jebran implementation

Figure 4-1: Speed results of hiding secret text using cover text size 10pt

As it’s shown in the above Tables (4-1) and Figures (4-1), the speed results generated by applying the proposed implementation are different depending on the number of digits are used to hide them within the white space of the cover text “described on 3.2.2.1 Detect the number of digits”, the fastest case occurs when using four digits in the text size 10pt it takes 0.2090119 seconds and the slowest case occurs when using one digit in the text size 10pt it takes 0.8620493 seconds. By applying the other implementation; the speed resulted are the same on variant number of digits used to hide them within the white space of the cover text, the speed of hiding one, two, three, four and five digits in text size 10pt is 48.8557944 seconds.

Hiding data by using size 12pt

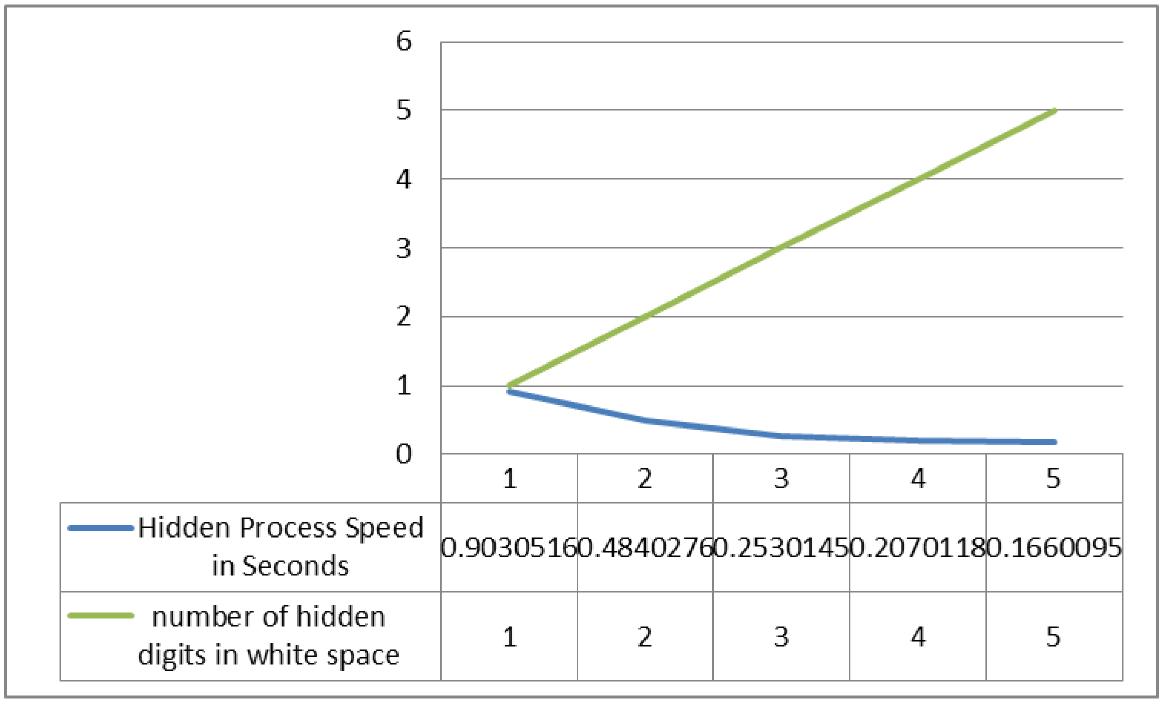
Table 4-2: Speed results of hiding secret text using cover text size 12pt

number of hidden digits in white space	**Hidden Process Speed in Seconds**
1	0.9030516
2	0.4840276
3	0.2530145
4	0.2070118
5	0.1660095

(a) The proposed implementation

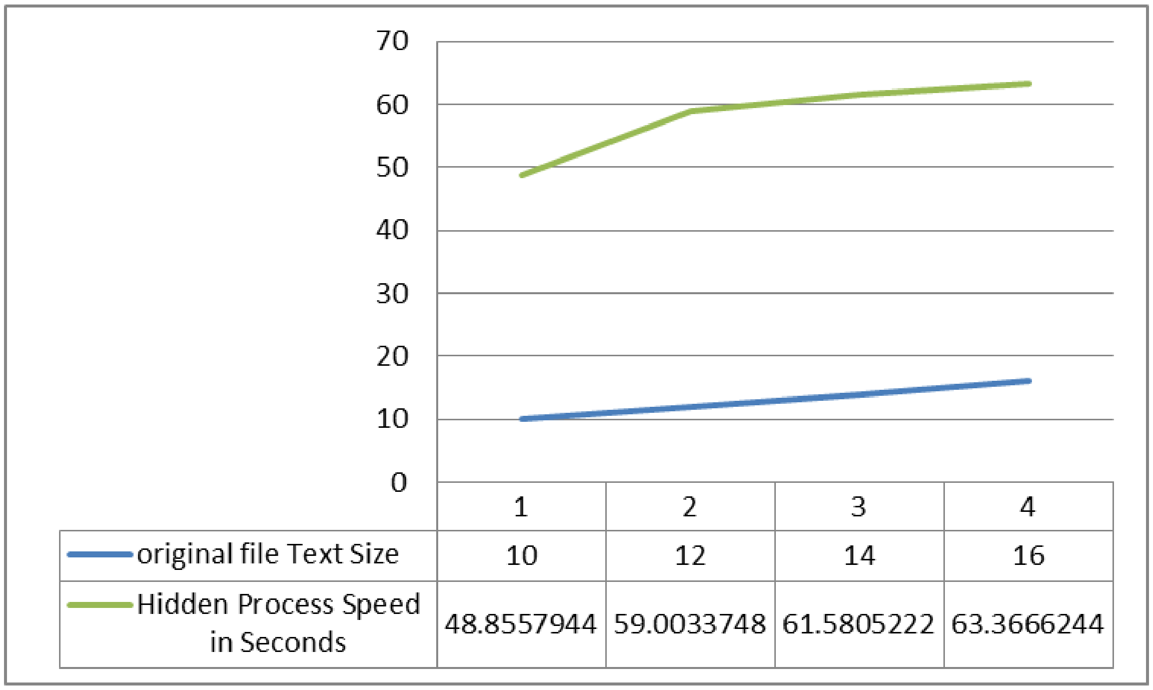
Original file Text Size	**Hidden Process Speed in Seconds**
10	48.8557944
12	**59.0033748**
14	61.5805222
16	63.3666244

(b) Jebran implementation



	1	2	3	4	5
🟦 Hidden Process Speed in Seconds	0.9030516	0.4840276	0.2530145	0.2070118	0.1660095
🟩 number of hidden digits in white space	1	2	3	4	5

(a) The proposed implementation



	1	2	3	4
🟦 original file Text Size	10	12	14	16
🟩 Hidden Process Speed in Seconds	48.8557944	59.0033748	61.5805222	63.3666244

(b) Jebran implementation

Figure 4-2: Speed results of hiding secret text using cover text size 12pt

As it's shown in the above Tables (4-2) and Figures (4-2), the speed results generated by applying the proposed implementation are different depending on the number of digits are used to hide them within the white space of the cover text "described on 3.2.2.1 Detect the number of digits", the fastest case occurs when using five digits in text size 12pt it takes 0.1660095 seconds and the slowest case occurs when using one digit in text size 12pt it takes 0.9030516 seconds. By applying the other implementation; the speed results are the same on variant number of digits used to hide them within the white space of the cover text, the speed of hiding one, two, three, four and five digits in text size 12pt is 59.0033748 seconds.

Hiding data by using size 14pt

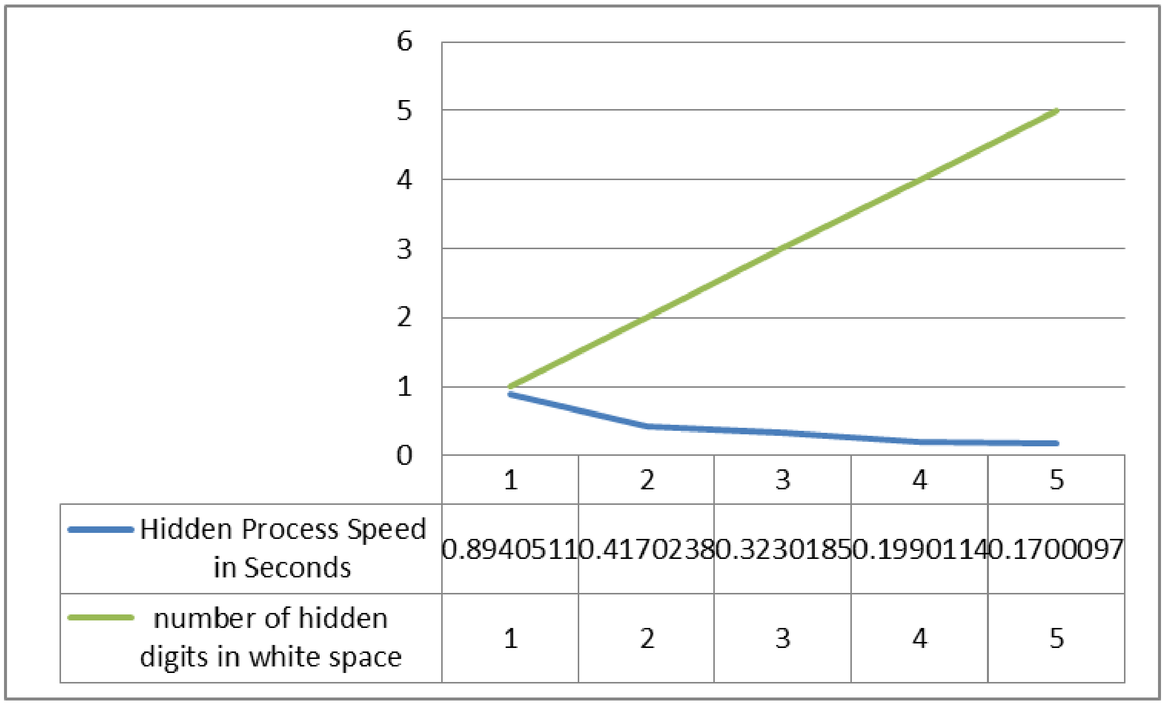
Table 4-3: Speed results of hiding secret text using cover text size 14pt

number of hidden digits in white space	Hidden Process Speed in Seconds
1	0.8940511
2	0.4170238
3	0.3230185
4	0.1990114
5	0.1700097

(a) The proposed Implementation

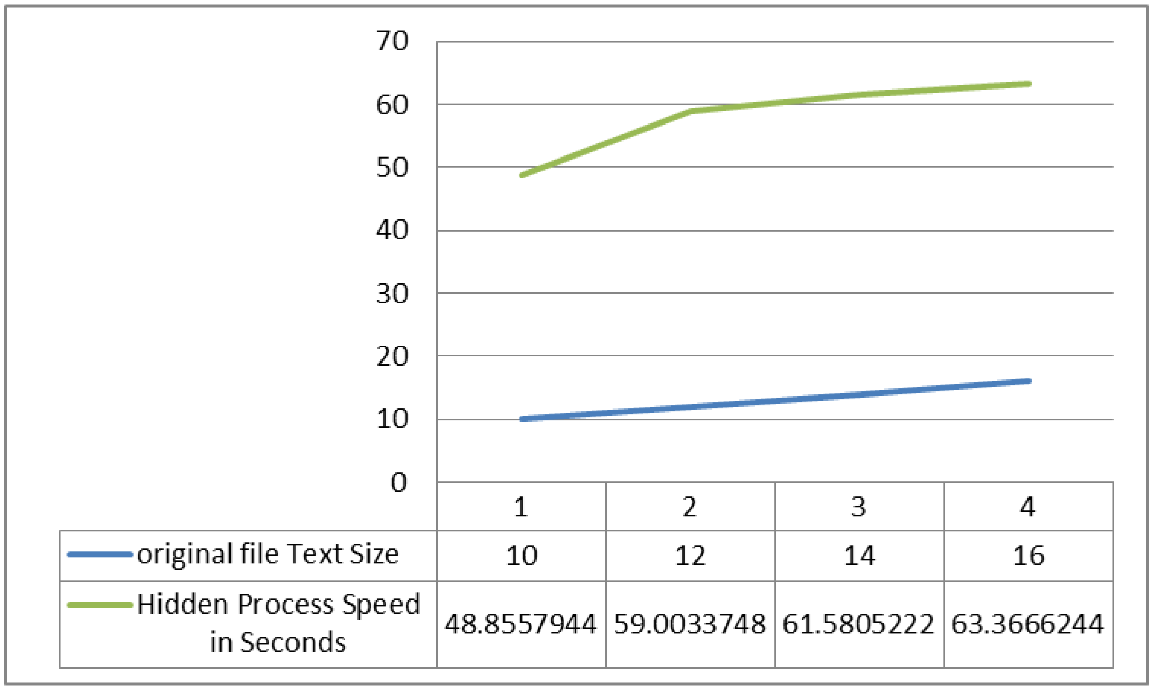
Original file Text Size	Hidden Process Speed in Seconds
10	48.8557944
12	59.0033748
14	**61.5805222**
16	63.3666244

(b) Jebran implementation



	1	2	3	4	5
Hidden Process Speed in Seconds	0.8940511	0.4170238	0.3230185	0.1990114	0.1700097
number of hidden digits in white space	1	2	3	4	5

(a) The proposed implementation



	1	2	3	4
original file Text Size	10	12	14	16
Hidden Process Speed in Seconds	48.8557944	59.0033748	61.5805222	63.3666244

(b) Jebran implementation

Figure 4-3: Speed results of hiding secret text using cover text size 14pt

As it’s shown in above Tables (4-3) and Figures (4-3), the speed results generated by applying the proposed implementation are different depend on the number of digits are used to hide them within the white space of the cover text “described on 3.2.2.1 Detect the number of digits”, the fastest case occurs when using five digits in text size 14pt it takes 0.1700097 seconds and the slowest case occurs when using one digit in text size 14pt it takes 0.8940511 seconds. By applying the other implementation; the result of speed are the same on variant number of digits used to hide them within the white space of the cover text, the speed of hiding one, two, three, four and five digits in text size 14pt is 61.5805222 seconds.

Hiding data by using size 16pt

Table 4-7 and figure 4-7 show the speed results for hiding secret text under cover text using text size 16pt by applying the proposed implementation.

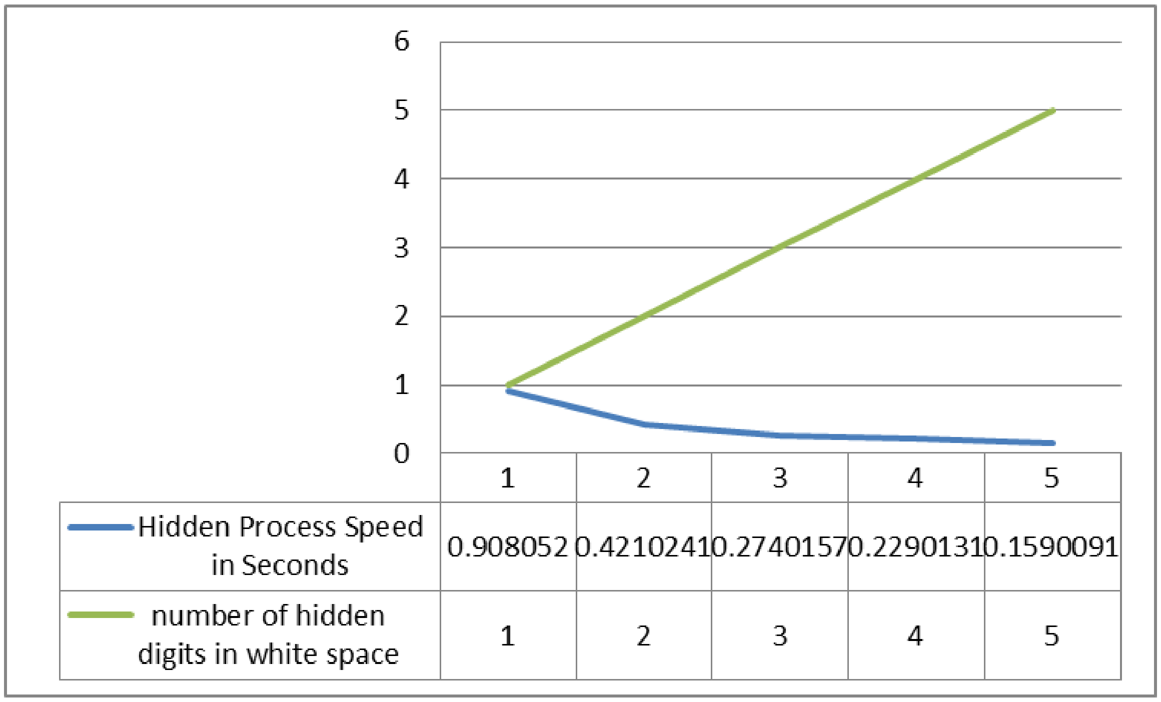
Table 4-4: Speed results of hide secret text using cover text size 16pt

number of hidden digits in white space	Hidden Process Speed in Seconds
1	0.908052
2	0.4210241
3	0.2740157
4	0.2290131
5	0.1590091

(a) The proposed implementation

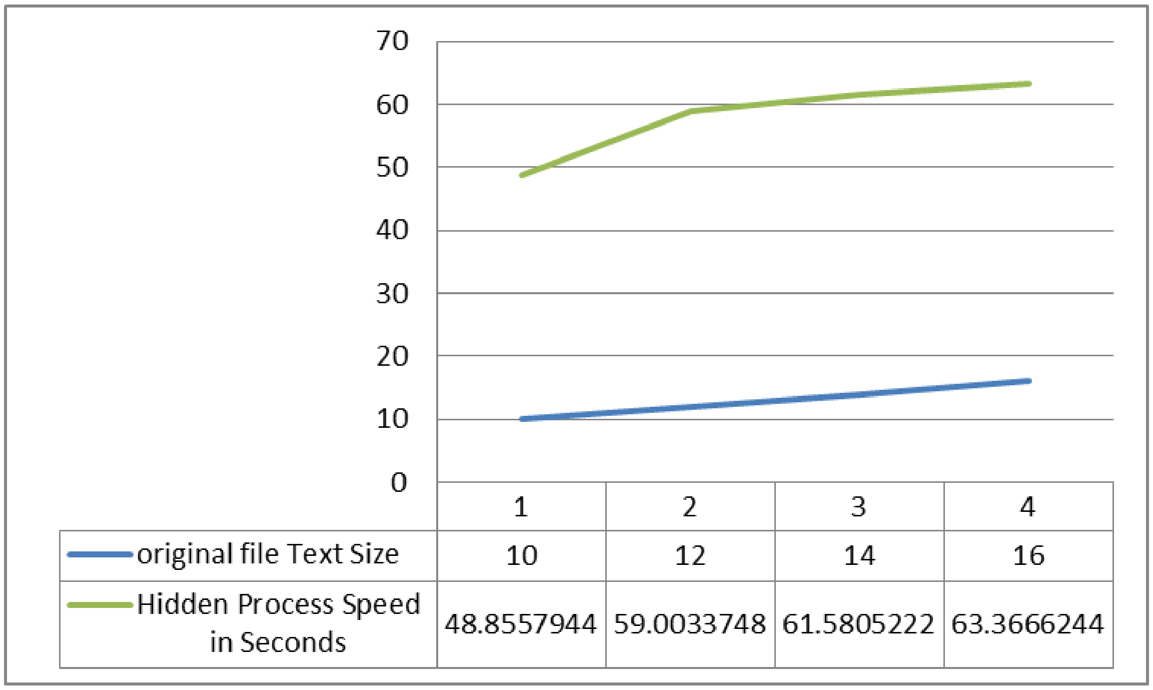
Original file Text Size	Hidden Process Speed in Seconds
10	48.8557944
12	59.0033748
14	61.5805222
16	**63.3666244**

(b) Jebran implementation



	1	2	3	4	5
🟦 Hidden Process Speed in Seconds	0.908052	0.4210241	0.2740157	0.2290131	0.1590091
🟩 number of hidden digits in white space	1	2	3	4	5

(a) The proposed implementation



	1	2	3	4
🟦 original file Text Size	10	12	14	16
🟩 Hidden Process Speed in Seconds	48.8557944	59.0033748	61.5805222	63.3666244

(b) Jebran implementation

Figure 4-4: Speed results of hiding secret text using cover text size 16pt

As it’s shown in above Tables (4-4) and Figures (4-4), the speed results generated by applying the proposed implementation are different depend on the number of digits are used to hide then within the white space of the cover text “described on 3.2.2.1 Detect the number of digits”, the fastest case occurs when using five digits in text size 16pt it takes 0.1590091 seconds and the slowest case occurs when using one digit in text size 16pt it takes 0.908052 seconds. By applying the other implementation; the result of process speed are the same on variant number of digits used to hide then within then white space of the cover text, the speed of hiding one, two, three, four and five digits in text size 16pt is 63.3666244 seconds.

4.3 Evaluating the Size of the Result Text

The size of the result text is different from one case to another, the cases of evaluating the size of the result text are: hiding secret text under cover text using cover text size 10pt, hiding secret text under cover text using cover text size 12pt, hiding secret text under cover text using cover text size 14pt and hiding secret text under cover text using cover text size 16pt.

The comparison will be between the proposed implementation and The other implementation is done by (Jebran, 2007).

Hiding data by using size 10pt

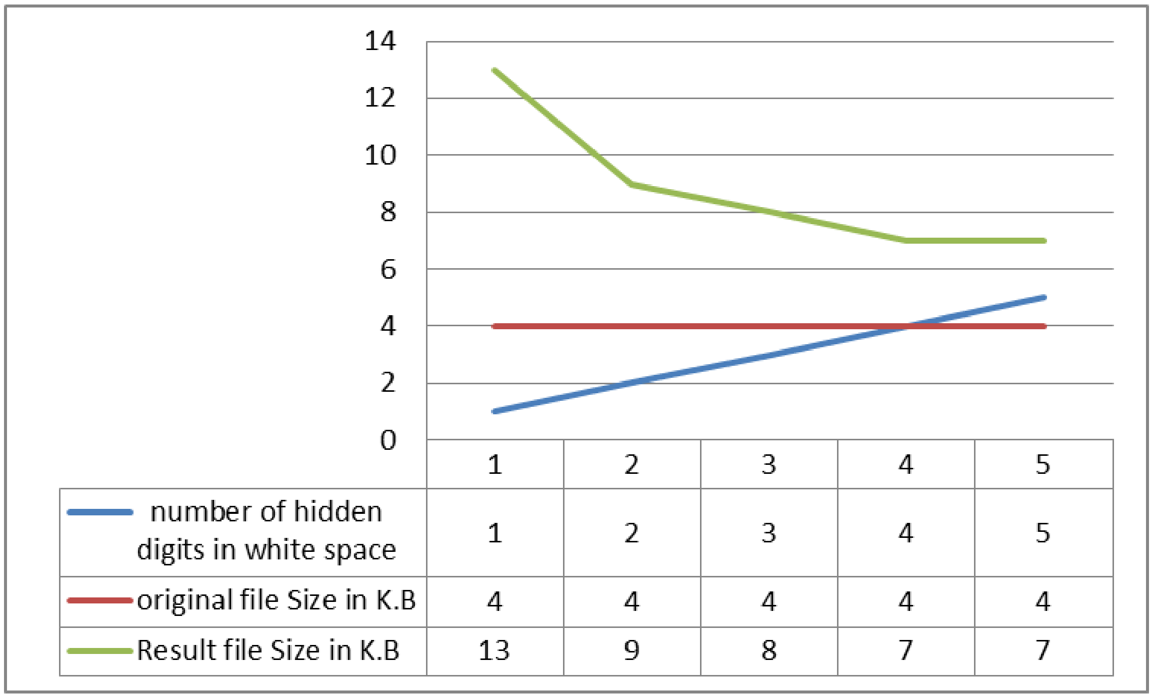
Table 4-5: Size of the result text for hiding secret text using cover text size 10pt

number of hidden digits in white space	original file Size in K.B	Result file Size in K.B
1	4	13
2	4	9
3	4	8
4	4	7
5	4	7

(a) The proposed implementation

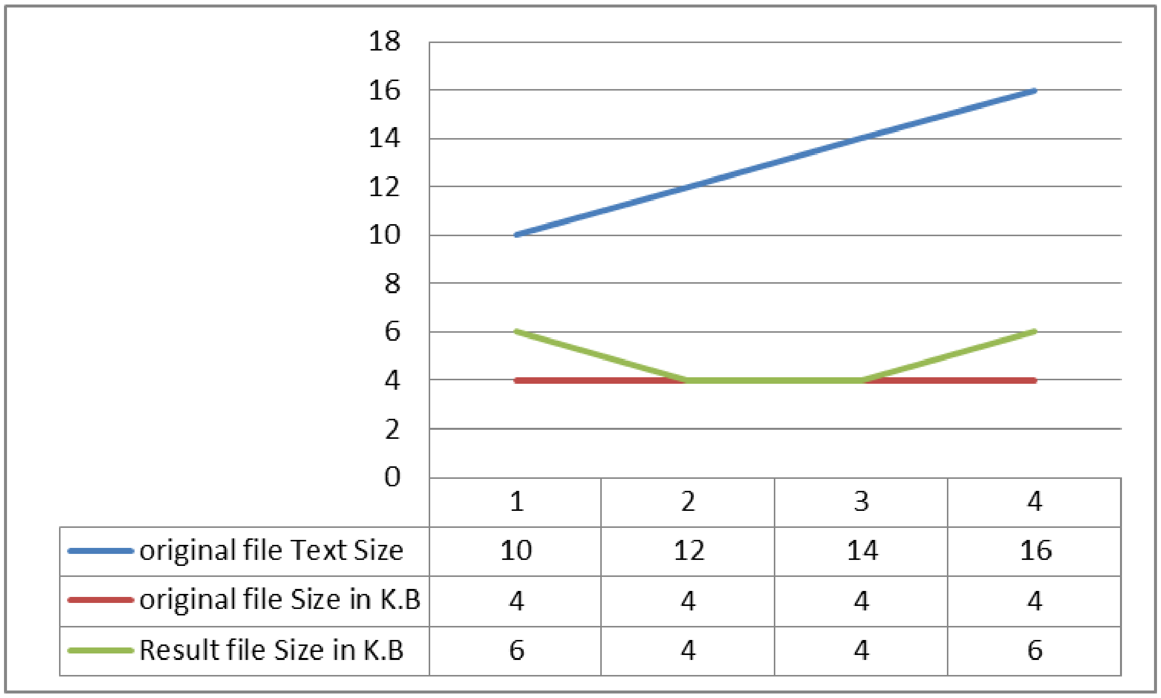
Original file Text Size	Original file Size in K.B	Result file Size in K.B
10	**4**	**6**
12	4	4
14	4	4
16	4	6

(b) Jebran implementation



	1	2	3	4	5
number of hidden digits in white space	1	2	3	4	5
original file Size in K.B	4	4	4	4	4
Result file Size in K.B	13	9	8	7	7

(a) The proposed implementation



	1	2	3	4
original file Text Size	10	12	14	16
original file Size in K.B	4	4	4	4
Result file Size in K.B	6	4	4	6

(b) Jebran implementation

Figure 4-5: Size of the result text for hiding secret text using cover text size 10pt

As it’s shown in above Tables (4-5) and Figures (4-5), the size of the result text generated by applying the proposed implementation is different depending on the number of digits are used to hiding them within then white space of the cover text “described on 3.2.2.1 Detect the number of digits”, the minimum size case occurs when using four and five digits in cover text size 10pt, the size is 7kb and the maximum size case occurs when using one digit in cover text size 10pt, the size is 13kb. By applying the other implementation; the size of the result text is the same on variant number of digits used to hiding them within the white space of the cover text, the size of the result text by hiding one, two, three, four and five digits in cover text size 10pt is 6kb.

Hiding data by using size 12pt

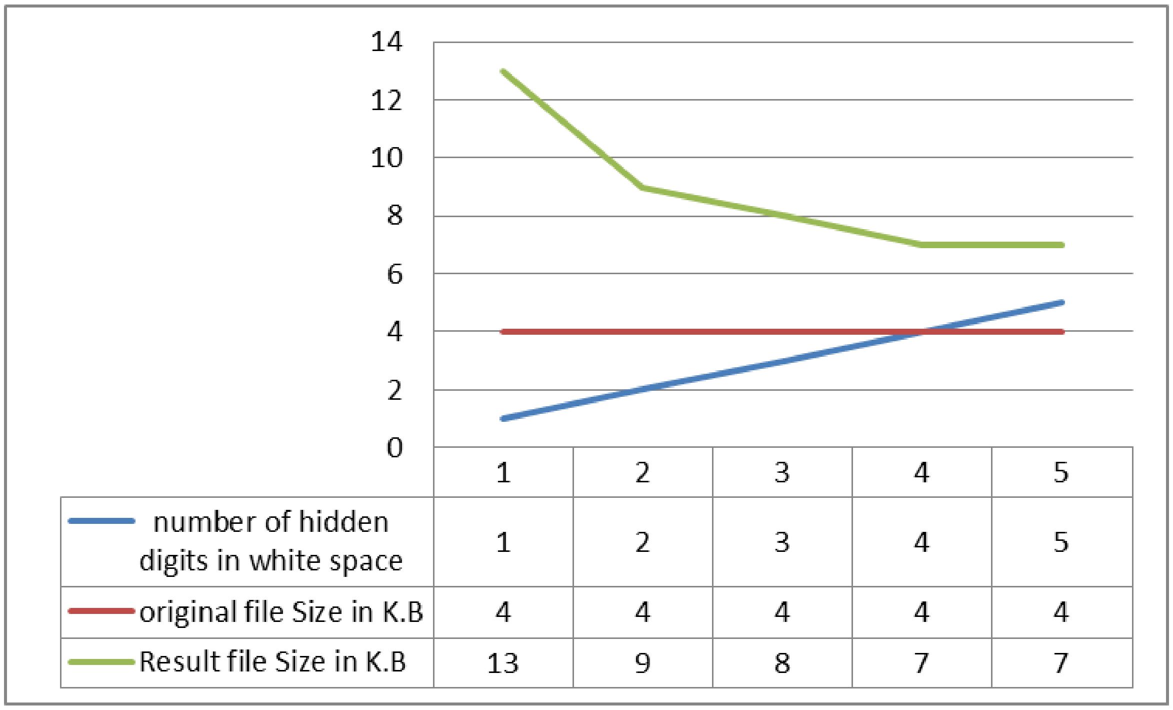
Table 4-6: Size of the result text for hiding the secret text using cover text size 12pt

number of hidden digits in white space	original file Size in K.B	Result file Size in K.B
1	4	13
2	4	9
3	4	8
4	4	7
5	4	7

(a) The proposed implementation

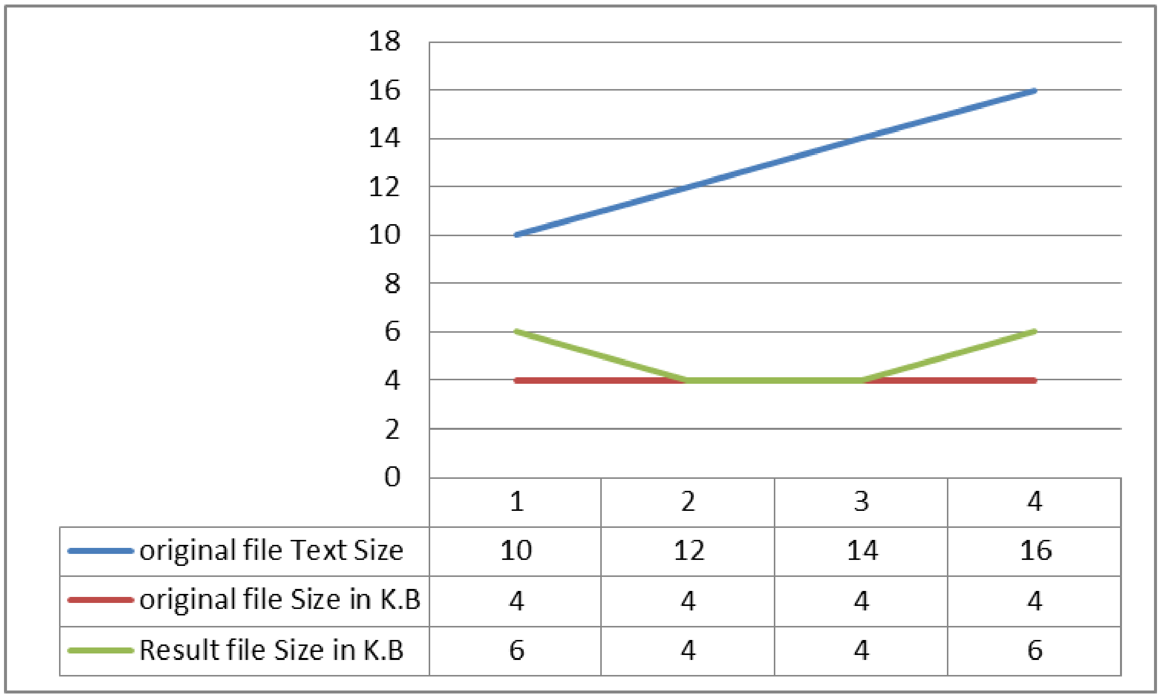
Original file Text Size	Original file Size in K.B	Result file Size in K.B
10	4	6
12	**4**	**4**
14	4	4
16	4	6

(b) Jebran implementation



	1	2	3	4	5
number of hidden digits in white space	1	2	3	4	5
original file Size in K.B	4	4	4	4	4
Result file Size in K.B	13	9	8	7	7

(a) The proposed implementation



	1	2	3	4
original file Text Size	10	12	14	16
original file Size in K.B	4	4	4	4
Result file Size in K.B	6	4	4	6

(b) Jebran implementation

Figure 4-6: Size of the result text for hiding the secret text using cover text size 12pt

As it’s shown in above Tables (4-6) and Figures (4-6), the size of the result text generated by applying the proposed implementation is different depending on the number of digits are used to hide it on the cover text white space “described on 3.2.2.1 Detect the number of digits”, the minimum size case occurs when using four and five digits in cover text size 12pt, the size is 7kb and the maximum size case occurs when using one digit in cover text size 12pt, the size is 13kb. By applying the other implementation; the size of the result text are the same on variant number of digits used to hiding them within the white space of the cover text, the size of the result text by hiding one, two, three, four and five digits in cover text size 12pt is 4kb.

Hiding data by using size 14pt

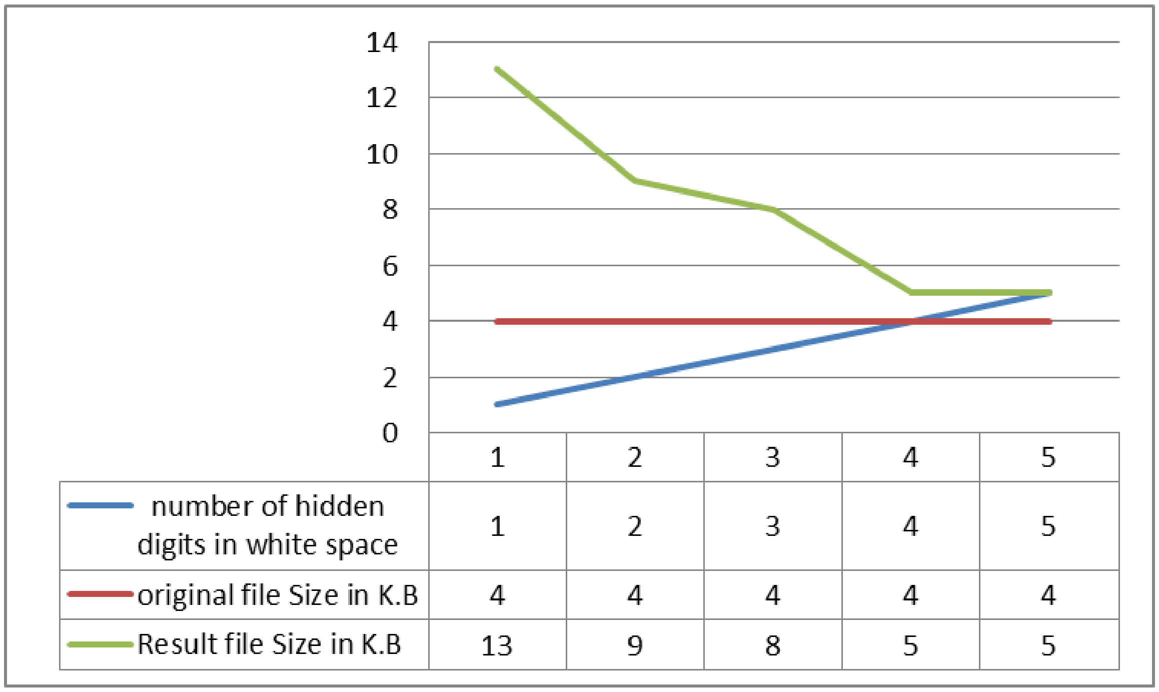
Table 4-7: Size of the result text for hiding secret text using cover text size 14pt

number of hidden digits in white space	original file Size in K.B	Result file Size in K.B
1	4	13
2	4	9
3	4	8
4	4	5
5	4	5

(a) The proposed implementation

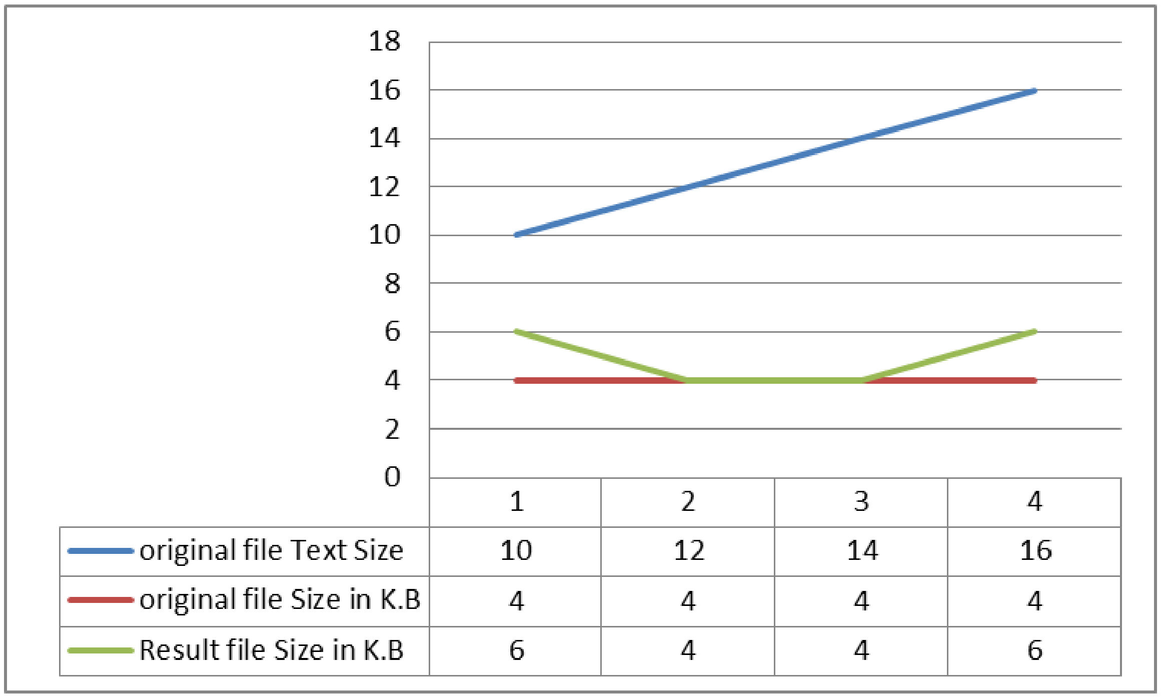
Original file Text Size	Original file Size in K.B	Result file Size in K.B
10	4	6
12	4	4
14	**4**	**4**
16	4	6

(b) Jebran implementation



	1	2	3	4	5
number of hidden digits in white space	1	2	3	4	5
original file Size in K.B	4	4	4	4	4
Result file Size in K.B	13	9	8	5	5

(a) The proposed implementation



	1	2	3	4
original file Text Size	10	12	14	16
original file Size in K.B	4	4	4	4
Result file Size in K.B	6	4	4	6

(b) Jebran implementation

Figure 4-7: Size of the result text for hiding secret text using cover text size 14pt

As it's shown in above Tables (4-7) and Figures (4-7), the size of the result text generated by applying the proposed implementation is different depending on the number of digits are used to hiding them within the white space of the cover text "described on 3.2.2.1 Detect the number of digits", the minimum size case occurs when using four and five digits in cover text size 14pt, the size is 5kb and the maximum size case occurs when using one digit in cover text size 14pt, the size is 13kb. By applying the other implementation; the size of the result text is the same on variant number of digits used to hide them within the white space of the cover text, the size of the result text by hiding one, two, three, four and five digits in cover text size 14pt is 4kb.

Hiding data by using size 16pt

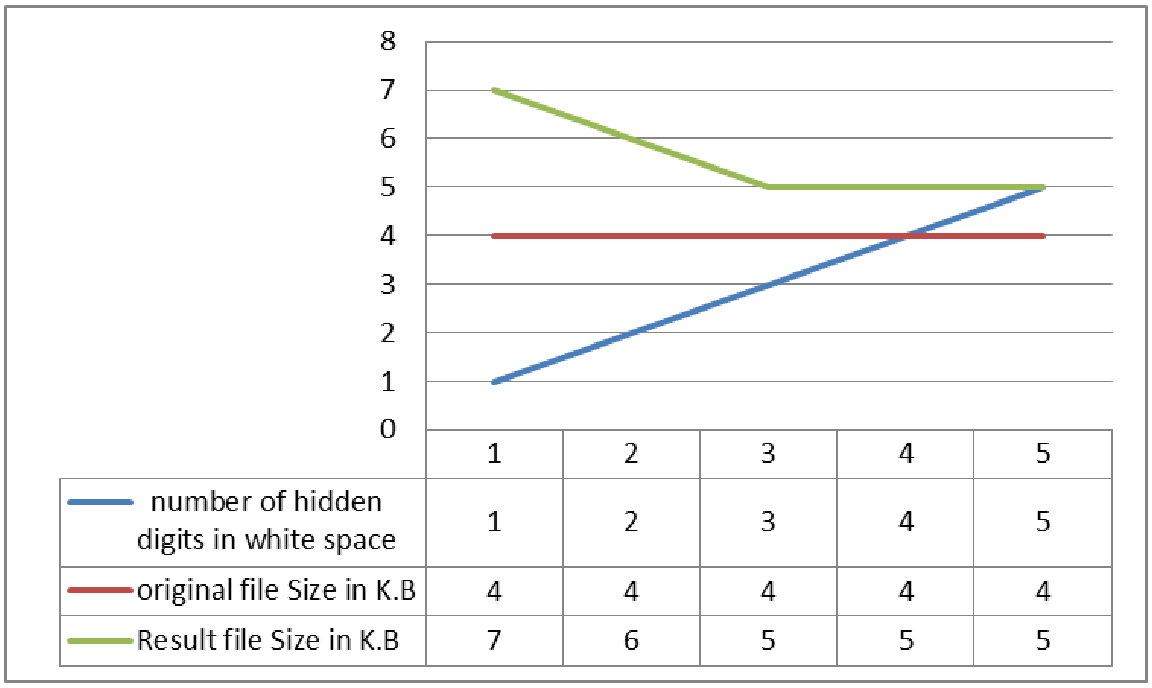
Table 4-8: Size of the result text for hiding the secret text using cover text size 16pt

number of hidden digits in white space	original file Size in K.B	Result file Size in K.B
1	4	7
2	4	6
3	4	5
4	4	5
5	4	5

(a) The proposed implementation

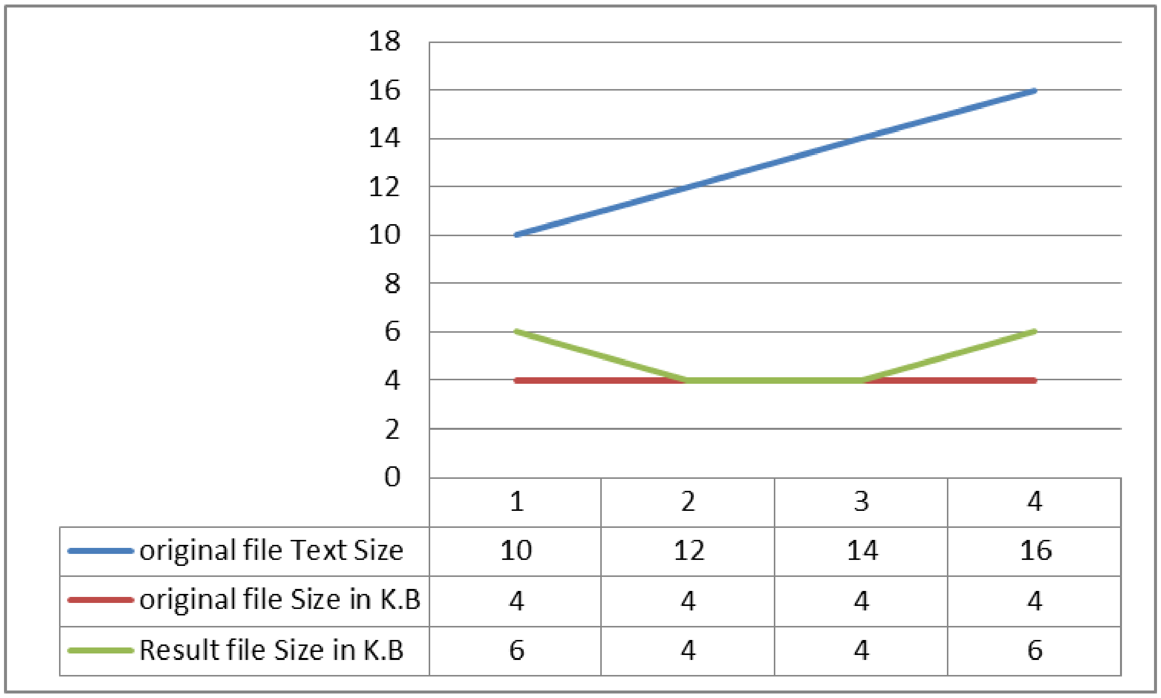
Original file Text Size	Original file Size in K.B	Result file Size in K.B
10	4	6
12	4	4
14	4	4
16	**4**	**6**

(b) Jebran implementation



	1	2	3	4	5
number of hidden digits in white space	1	2	3	4	5
original file Size in K.B	4	4	4	4	4
Result file Size in K.B	7	6	5	5	5

(a) The proposed implementation



	1	2	3	4
🟦 original file Text Size	10	12	14	16
🟥 original file Size in K.B	4	4	4	4
🟩 Result file Size in K.B	6	4	4	6

(b) Jebran implementation

Figure 4-8: Size of the result text for hiding secret text using cover text size 16pt

As it’s shown in above Tables (4-8) and Figures (4-8), the size of the result text generated by applying the proposed implementation is different depending on the number of digits are used to hiding them within the cover text white space “described on 3.2.2.1 Detect the number of digits”, the minimum size case occurs when using four and five digits in cover text size 16pt, the size is 5kb and the maximum size case occurs when using one digit in cover text size 16pt, the size is 7kb. By applying the other implementation; the size of the result text are the same on variant number of digits used to hiding them within the cover text white space, the size of the result text by hide one, two, three, four and five digits in cover text size 16pt is 6kb.

4.4 Evaluating the Result Text Matching with the Original Cover Text

The result text matching with the original cover text are different from one case to another, the cases of evaluating the result text matching with the original cover text are: hiding secret text under cover text using cover text size 10pt, hiding secret text under cover text using cover text size 12pt, hiding secret text under cover text using cover text size 14pt and hiding secret text under cover text using cover text size 16pt.

The comparison will be between the proposed implementation and The other implementation is done by (Jebran, 2007).

Hiding data by using size 10pt

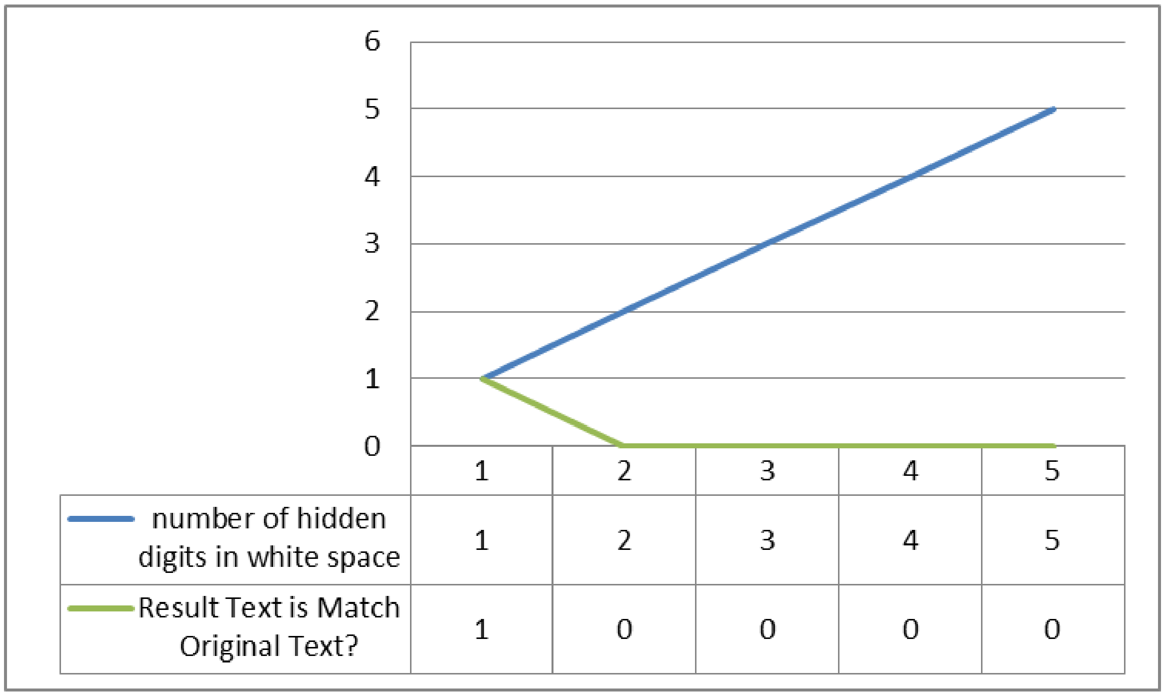
Table 4-9: Result text matching with the original cover text for hiding secret text using cover text size 10pt

number of hidden digits in white space	Result Text is Match Original Text?
1	TRUE
2	FALSE
3	FALSE
4	FALSE
5	FALSE

(a) The proposed implementation

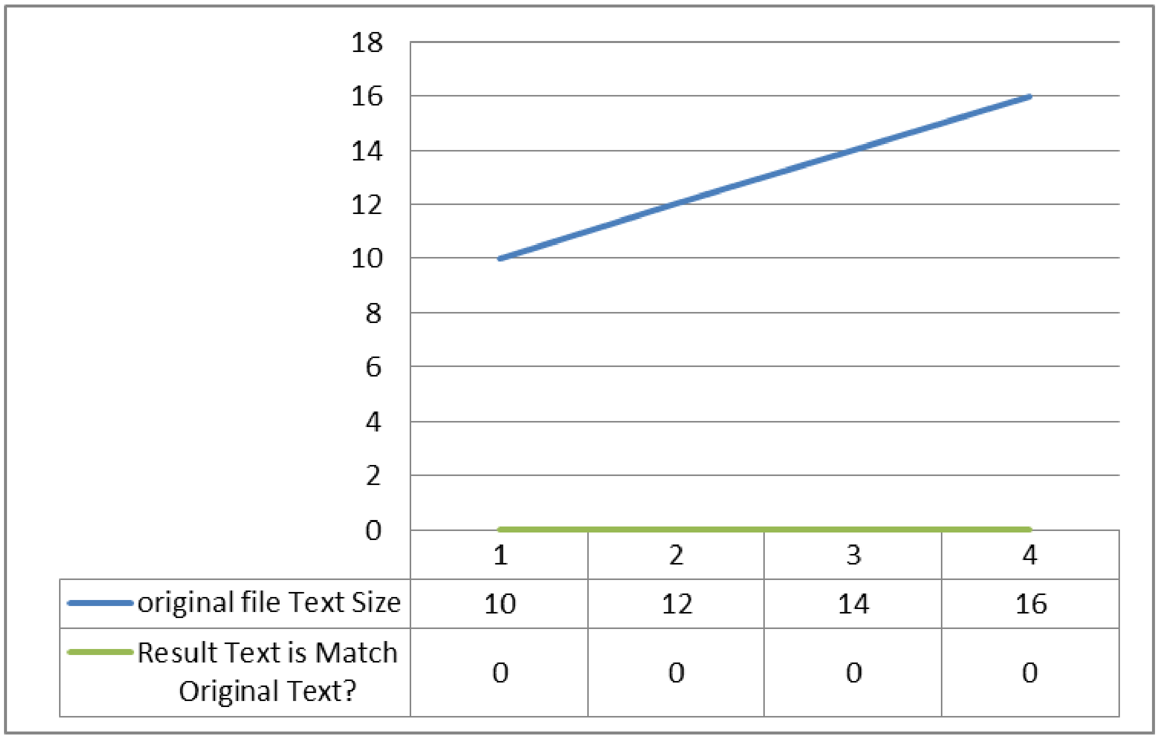
Original file Text Size	Result Text is Match Original Text?
10	**FALSE**
12	FALSE
14	FALSE
16	FALSE

(b) Jebran implementation



	1	2	3	4	5
number of hidden digits in white space	1	2	3	4	5
Result Text is Match Original Text?	1	0	0	0	0

(a) The proposed implementation



	1	2	3	4
original file Text Size	10	12	14	16
Result Text is Match Original Text?	0	0	0	0

(b) Jebran implementation

Figure 4-9: Result text matching with the original cover text for hiding secret text using cover text size 10pt

As it’s shown in above Tables (4-9) and Figures (4-9), the result text matching with the original cover text generated by applying the proposed implementation is different depending on the number of digits are used to hide them within the white space of the cover text “described on 3.2.2.1 Detect the number of digits”, when using one digit in cover text size 10pt the result text matches the original cover text, and when using two, three, four and five digits in cover text size 10pt the result text does not match the original cover text. By applying the other implementation; the result text does not match the original cover text on variant number of digits used to hide them within the white space of the cover text using cover text size 10pt.

Hiding data by using size 12pt

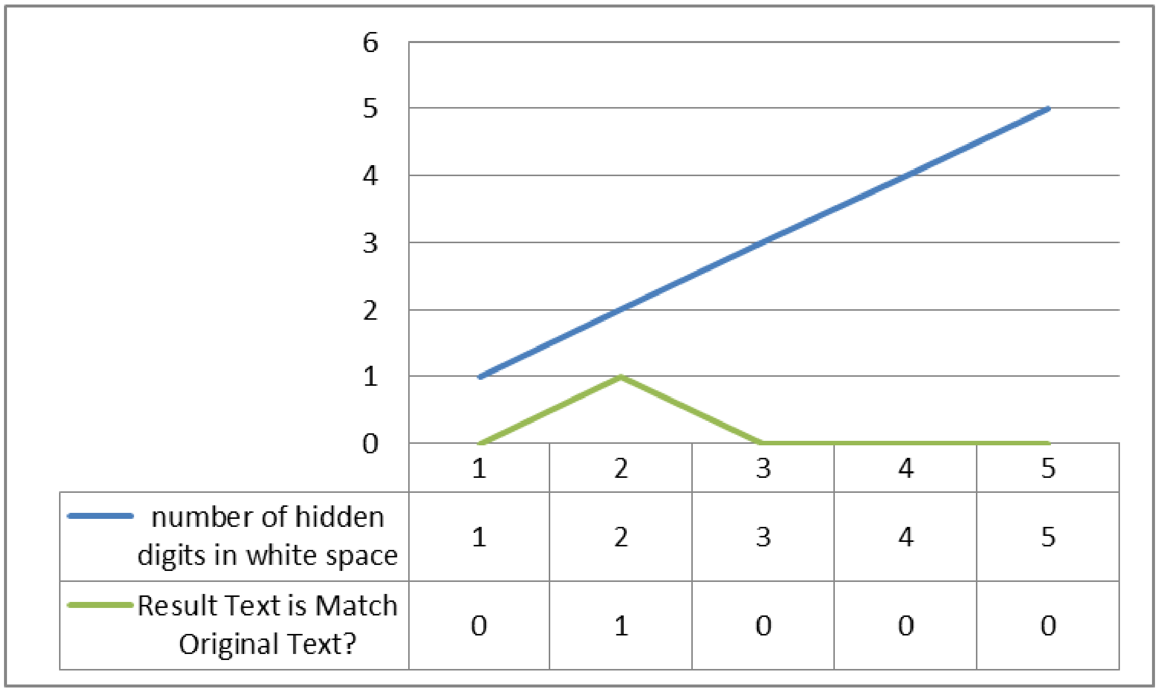
Table 4-10: the result text matching with the original cover text for hide secret text using cover text size 12pt

number of hidden digits in white space	Result Text is Match Original Text?
1	FALSE
2	TRUE
3	FALSE
4	FALSE
5	FALSE

(a) The proposed implementation

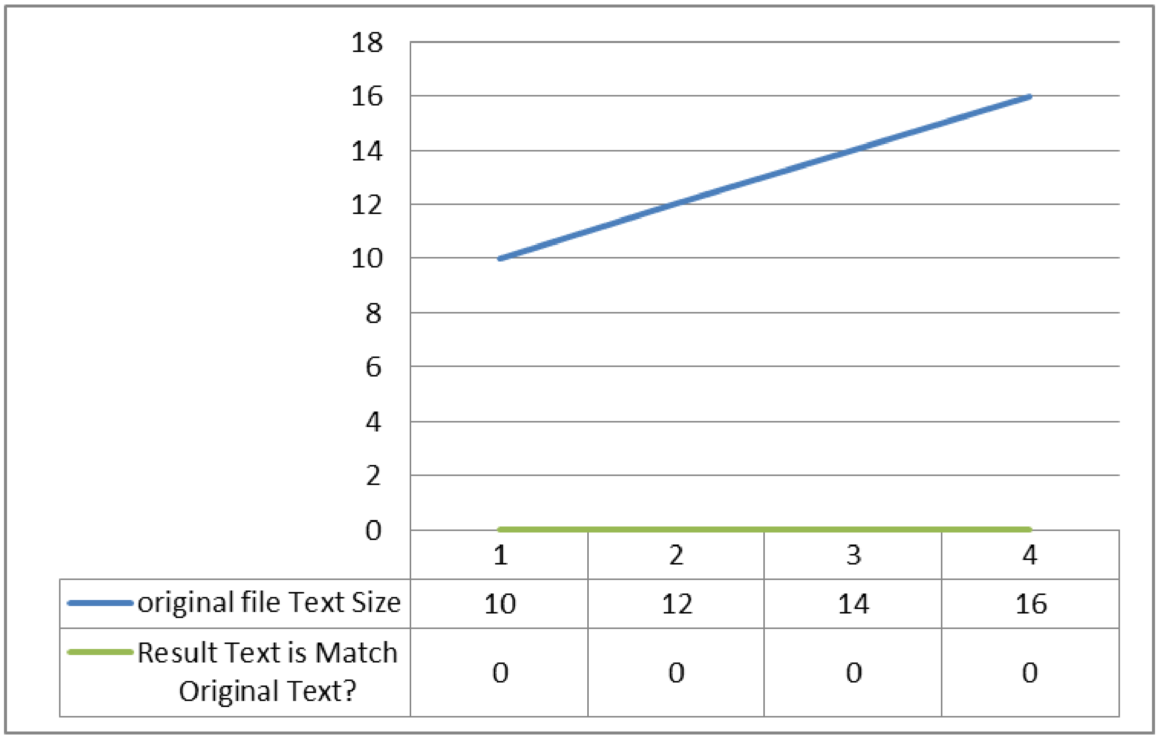
Original file Text Size	Result Text is Match Original Text?
10	FALSE
12	**FALSE**
14	FALSE
16	FALSE

(b) Jebran implementation



	1	2	3	4	5
number of hidden digits in white space	1	2	3	4	5
Result Text is Match Original Text?	0	1	0	0	0

(a) The proposed implementation



	1	2	3	4
original file Text Size	10	12	14	16
Result Text is Match Original Text?	0	0	0	0

(b) Jebran implementation

Figure 4-10: Result text matching with the original cover text for hiding secret text using cover text size 12pt

As it’s shown in above Tables (4-10) and Figures (4-10), the result text matching with the original cover text generated by applying the proposed implementation is different depending on the number of digits are used to hide them within the white space of the cover text “described on 3.2.2.1 Detect the number of digits”, when using two digits in cover text size 12pt the result text matches the original cover text, and when using one, three, four and five digits in cover text size 12pt the result text does not match the original cover text. By applying the other implementation; the result text does not match the original cover text on variant number of digits used to hide them within the white space of the cover text in cover text size 12pt.

Hiding data by using size 14pt

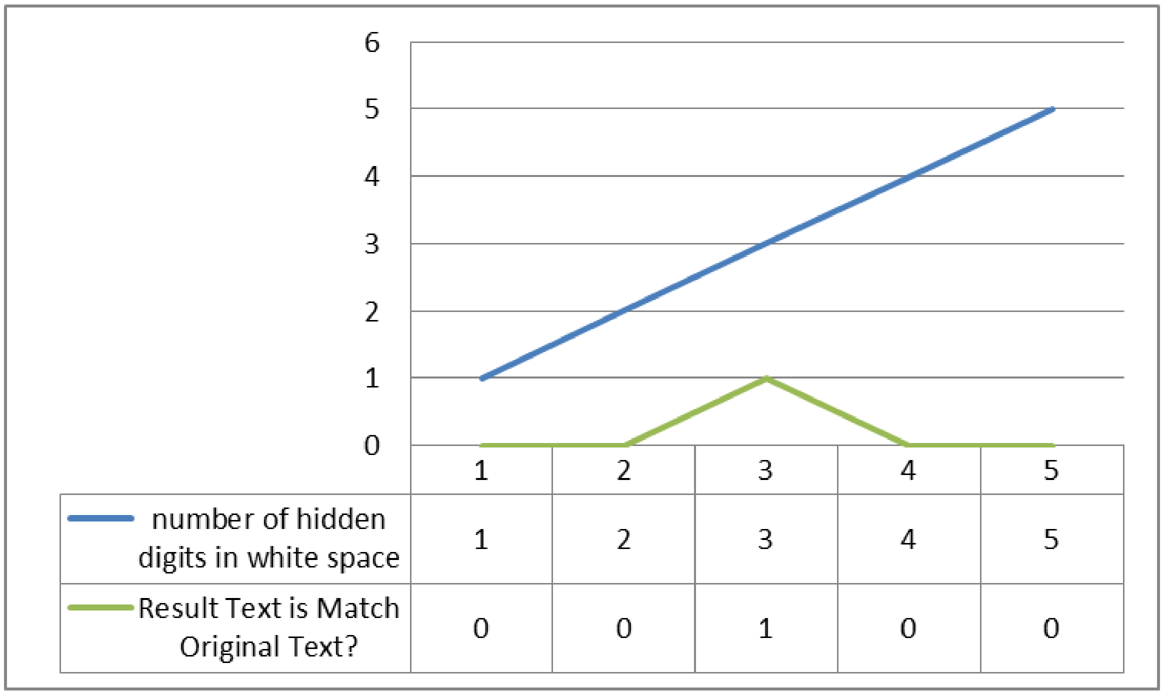
Table 4-11: Result text matching with the original cover text for hiding secret text using cover text size 14pt

number of hidden digits in white space	Result Text is Match Original Text?
1	FALSE
2	FALSE
3	TRUE
4	FALSE
5	FALSE

(a) The proposed implementation

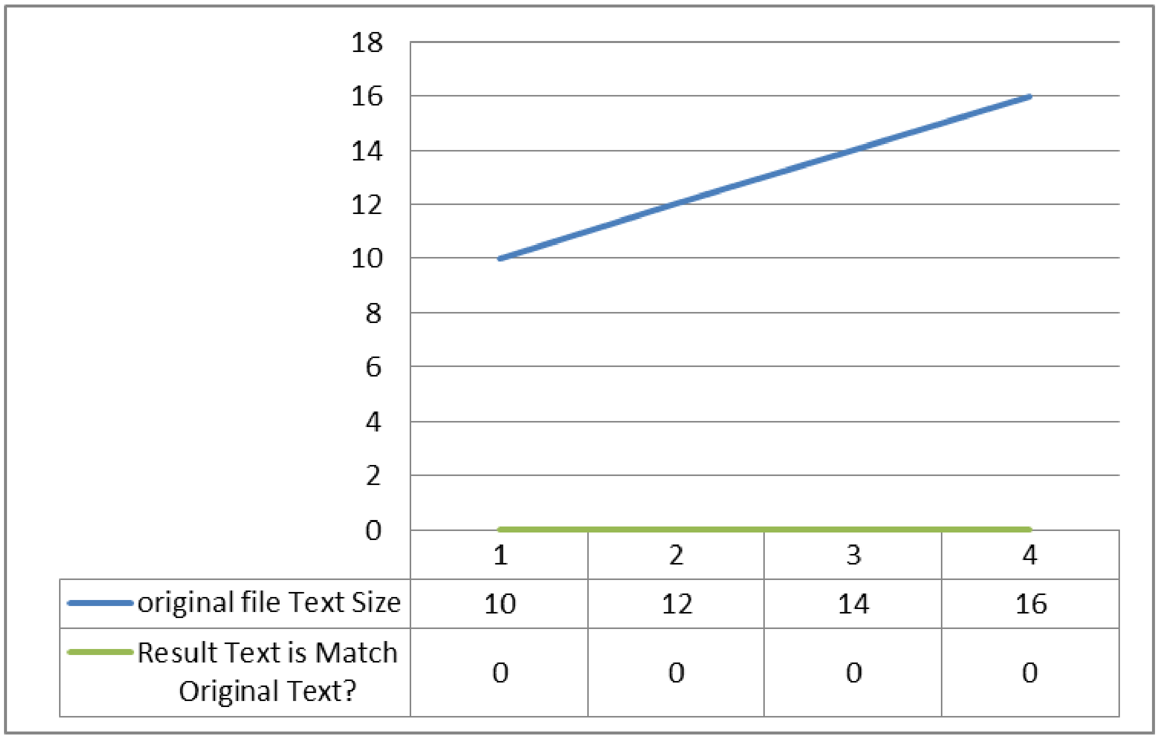
Original file Text Size	Result Text is Match Original Text?
10	FALSE
12	FALSE
14	**FALSE**
16	FALSE

(b) Jebran implementation



	1	2	3	4	5
number of hidden digits in white space	1	2	3	4	5
Result Text is Match Original Text?	0	0	1	0	0

(a) The proposed implementation



	1	2	3	4
original file Text Size	10	12	14	16
Result Text is Match Original Text?	0	0	0	0

(b) Jebran implementation

Figure 4-11: Result text matching with the original cover text for hiding secret text using cover text size 14pt

As it’s shown in above Tables (4-11) and Figures (4-11), the result text matching with the original cover text generated by applying the proposed implementation is different depending on the number of digits are used to hide them within the white space of the cover text “described on 3.2.2.1 Detect the number of digits”, when using three digits in cover text size 14pt the result text matches of the original cover text, and when using one, two, four and five digits in cover text size 14pt the result text does not match the original cover text. By applying the other implementation; the result text does not match the original cover text on variant number of digits used to hide them within the cover text white space in cover text size 14pt.

Hiding data by using size 16pt

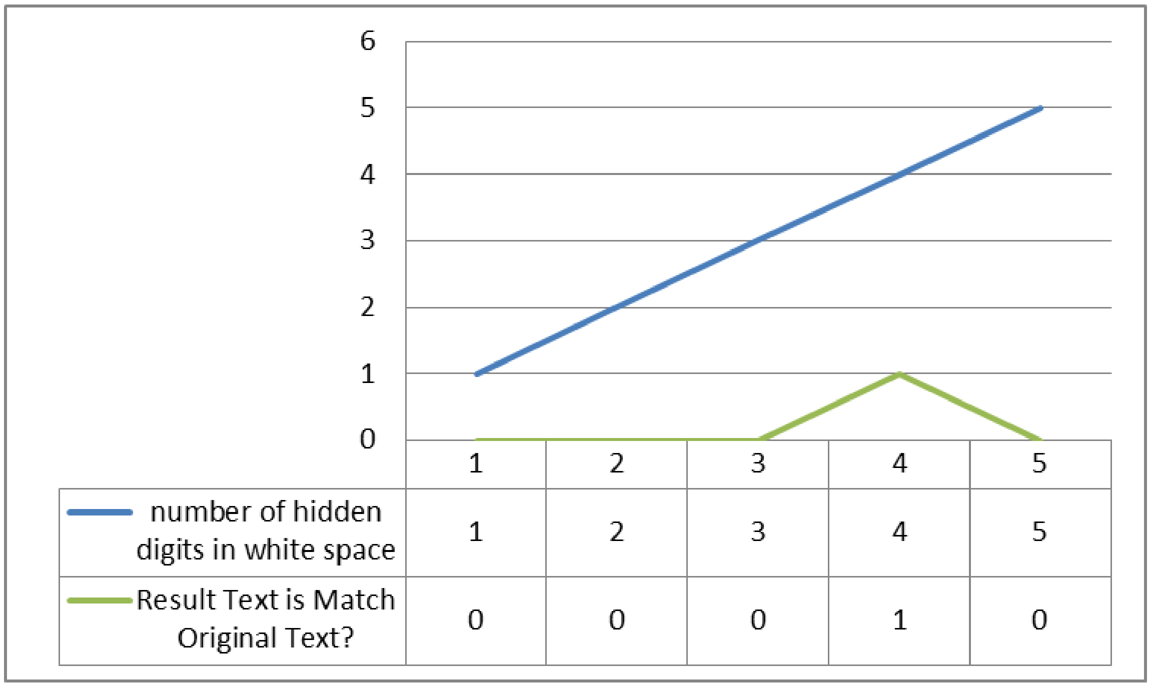
Table 4-12: Result text matching with the original cover text for hiding secret text using cover text size 16pt

number of hidden digits in white space	Result Text is Match Original Text?
1	FALSE
2	FALSE
3	FALSE
4	TRUE
5	FALSE

(a) The proposed implementation

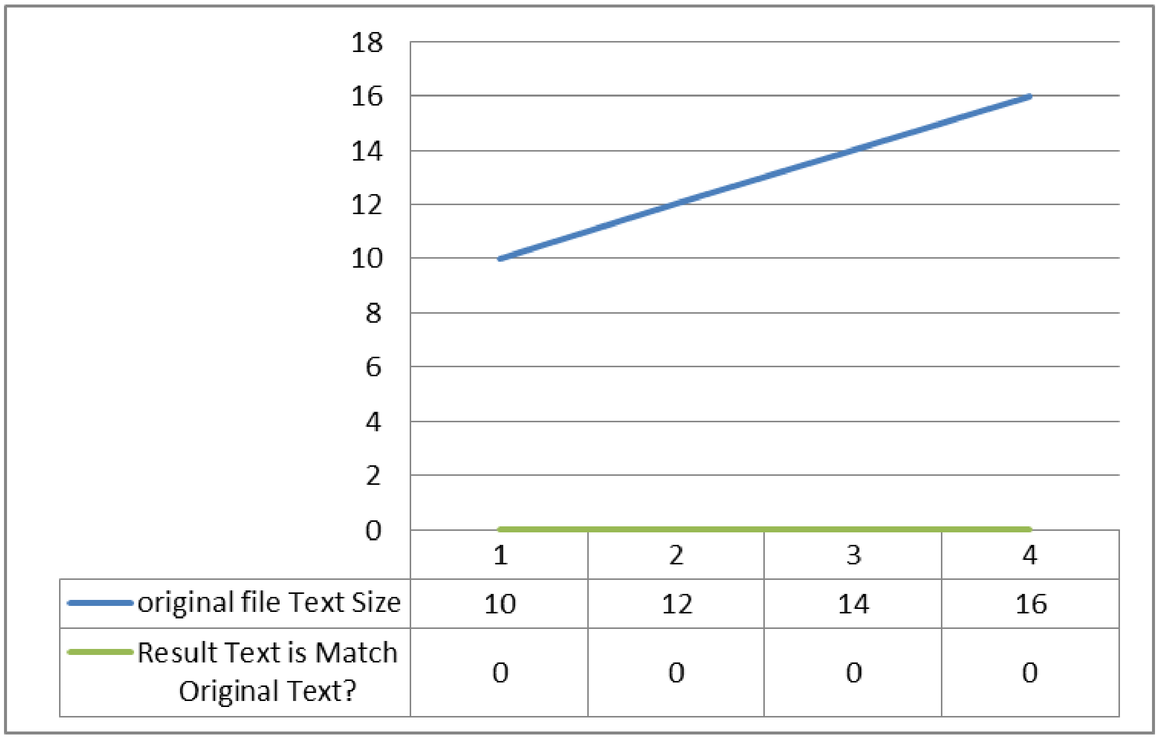
Original file Text Size	Result Text is Match Original Text?
10	FALSE
12	FALSE
14	FALSE
16	**FALSE**

(b) Jebran implementation



	1	2	3	4	5
number of hidden digits in white space	1	2	3	4	5
Result Text is Match Original Text?	0	0	0	1	0

(a) The proposed implementation



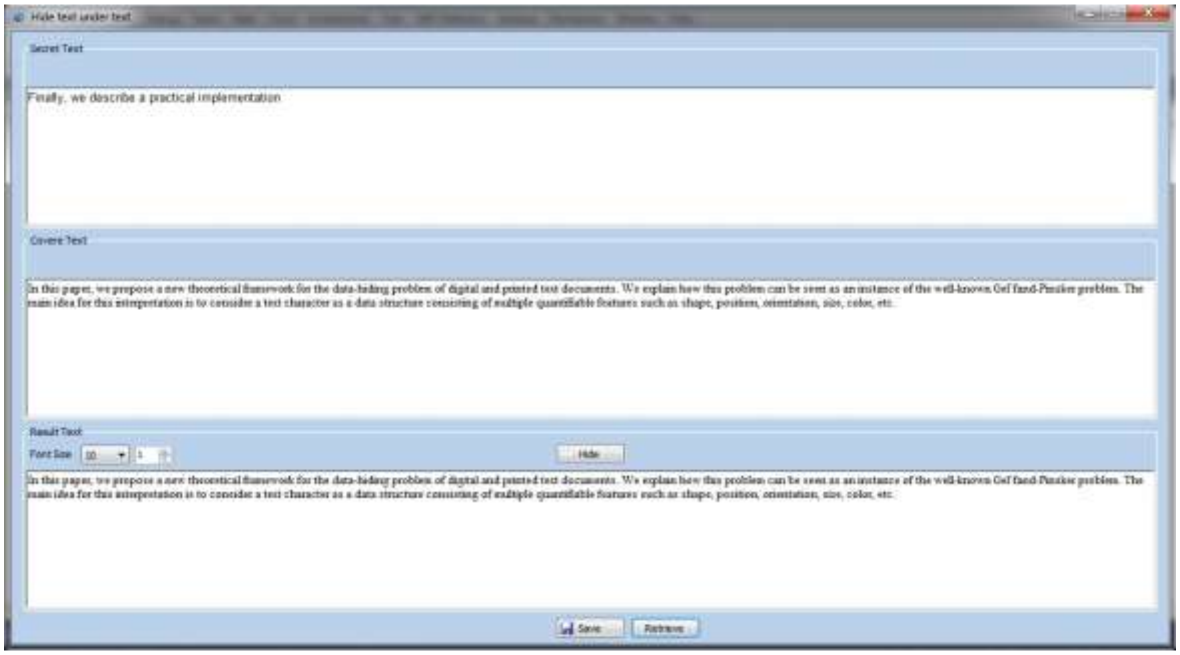
	1	2	3	4
original file Text Size	10	12	14	16
Result Text is Match Original Text?	0	0	0	0

(b) Jebran implementation

Figure 4-12: Result text matching with the original cover text for hiding secret text using cover text size 16pt

As it’s shown in above Tables (4-12) and Figures (4-12), the result text matching with the original cover text generated by applying the proposed implementation is different depending on the number of digits are used to hide them within the white space of the cover text “described on 3.2.2.1 Detect the number of digits”, when using four digits in cover text size 16pt the result text matches the original cover text, and when using one, two, three and five digits in cover text size 16pt the result text does not match the original cover text. By applying the other implementation; the result text does not match the original cover text on variant number of digits used to hide them within the cover text white space in cover text size 16pt.

4.5 The results of the proposed implementation in variant cases



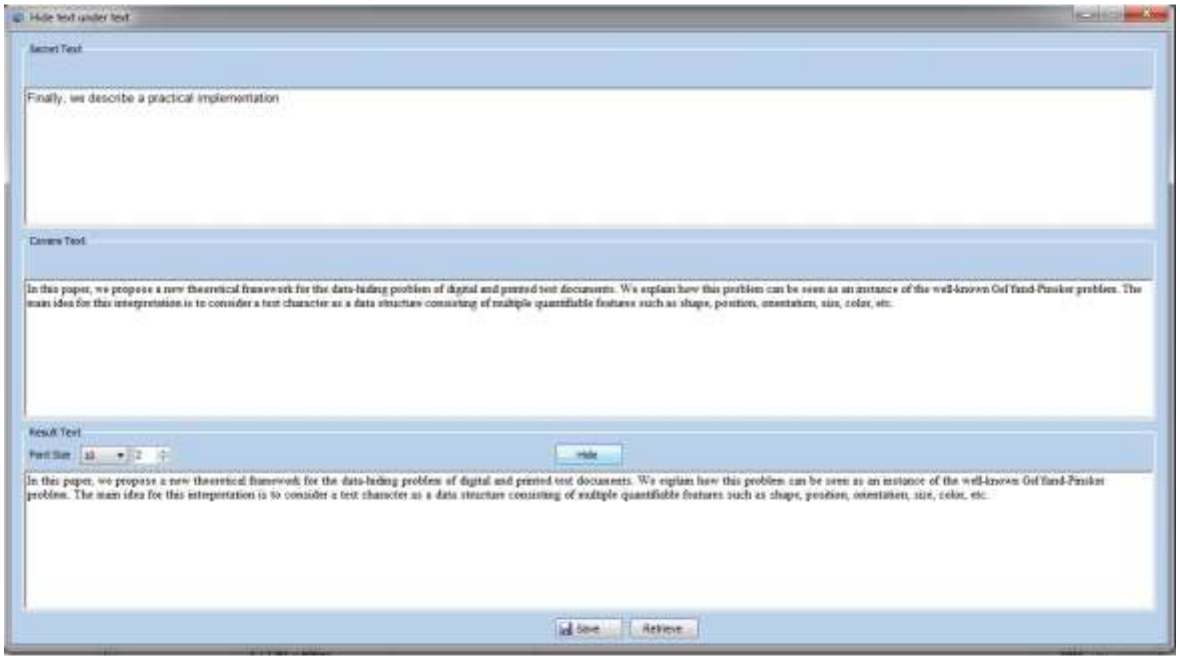
Finally, we describe a practical implementation

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Font Size 10 1

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-13: Result of hiding secret text using cover text size 10pt for one digit



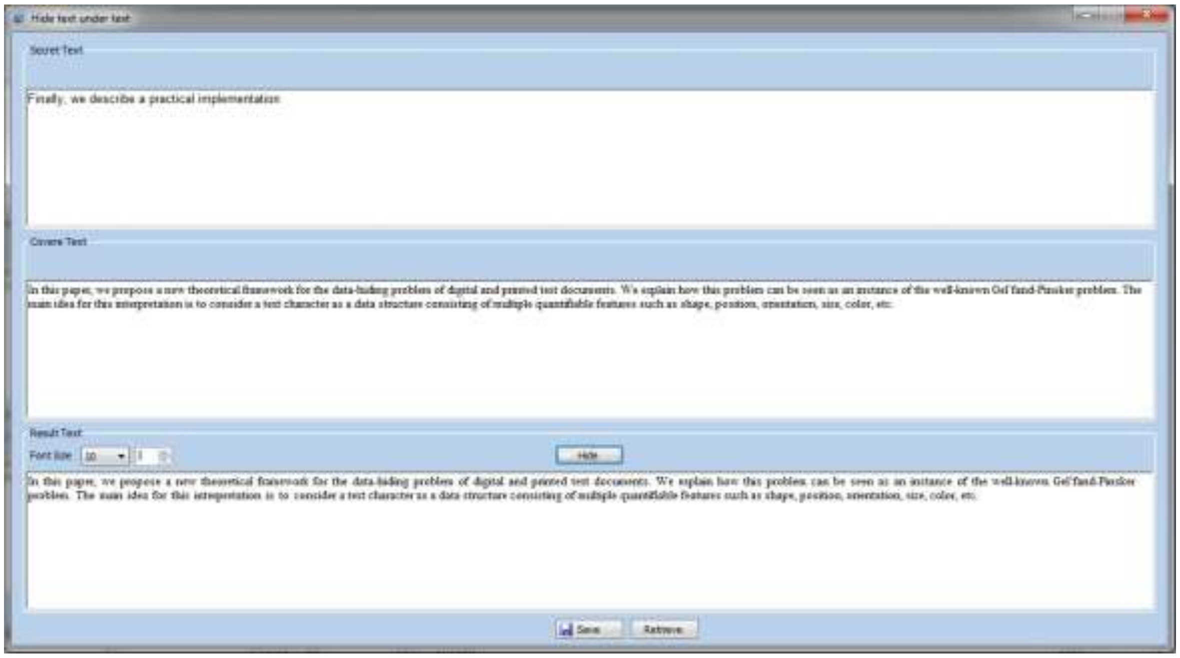
Finally, we describe a practical implementation

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Font Size 10 2

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-14: Result of hiding secret text using cover text size 10pt for two digits



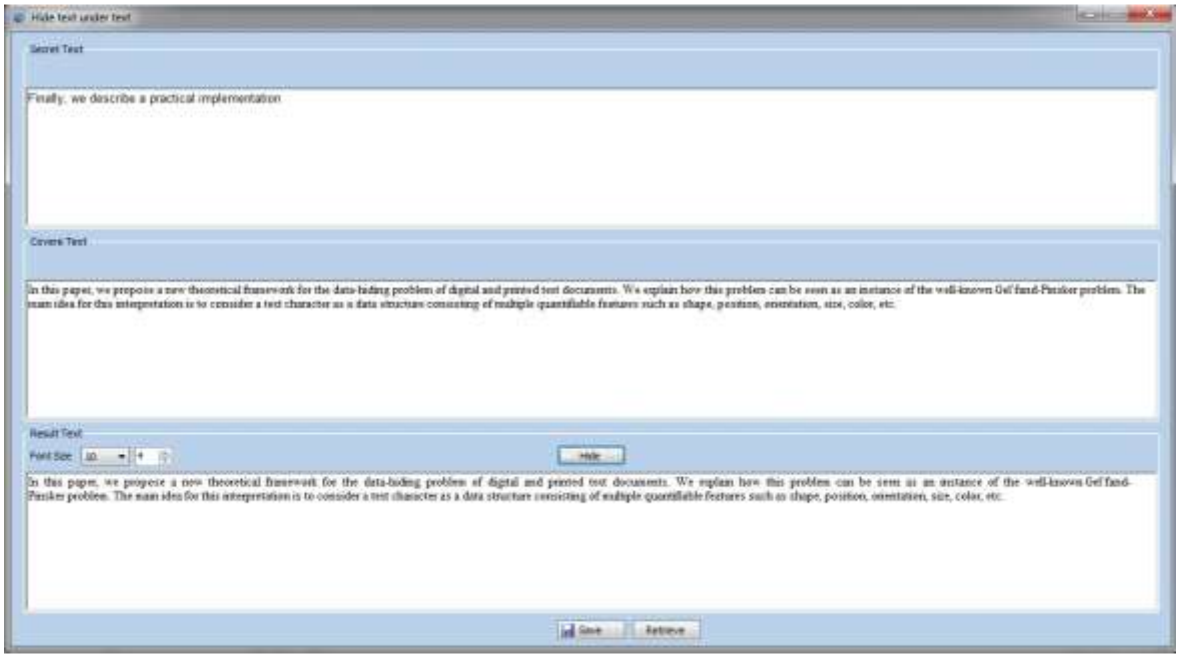
Finally, we describe a practical implementation

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Font Size: 10 3

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-15: Result of hiding secret text using cover text size 10pt for three digits



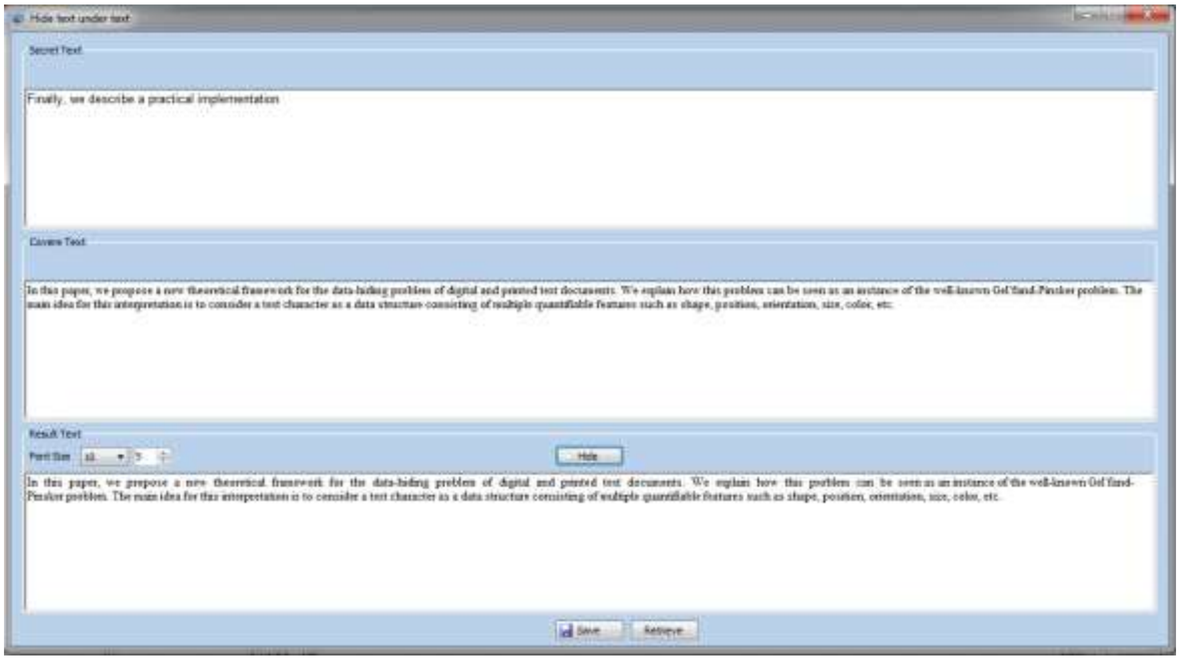
Finally, we describe a practical implementation

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Font Size: 10 4

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-16: Result of hiding secret text using cover text size 10pt for four digits



Secret Text

Finally, we describe a practical implementation

Cover Text

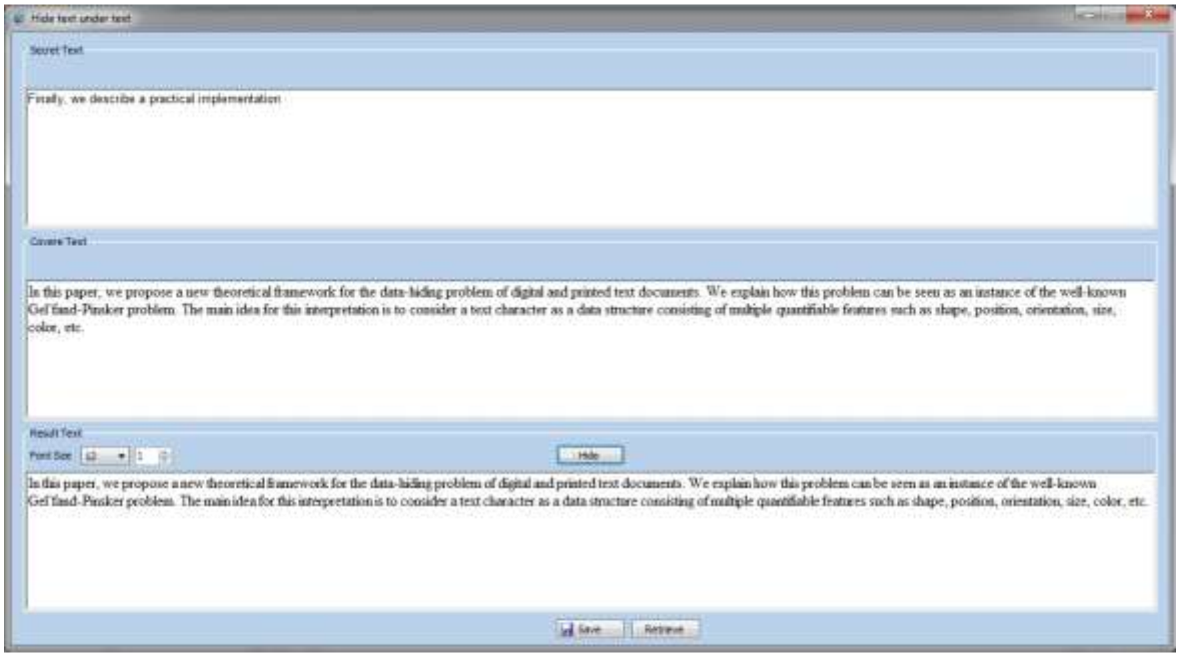
In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Result Text

Font Size 10 5

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-17: Result of hiding secret text using cover text size 10pt for five digits



Secret Text

Finally, we describe a practical implementation

Cover Text

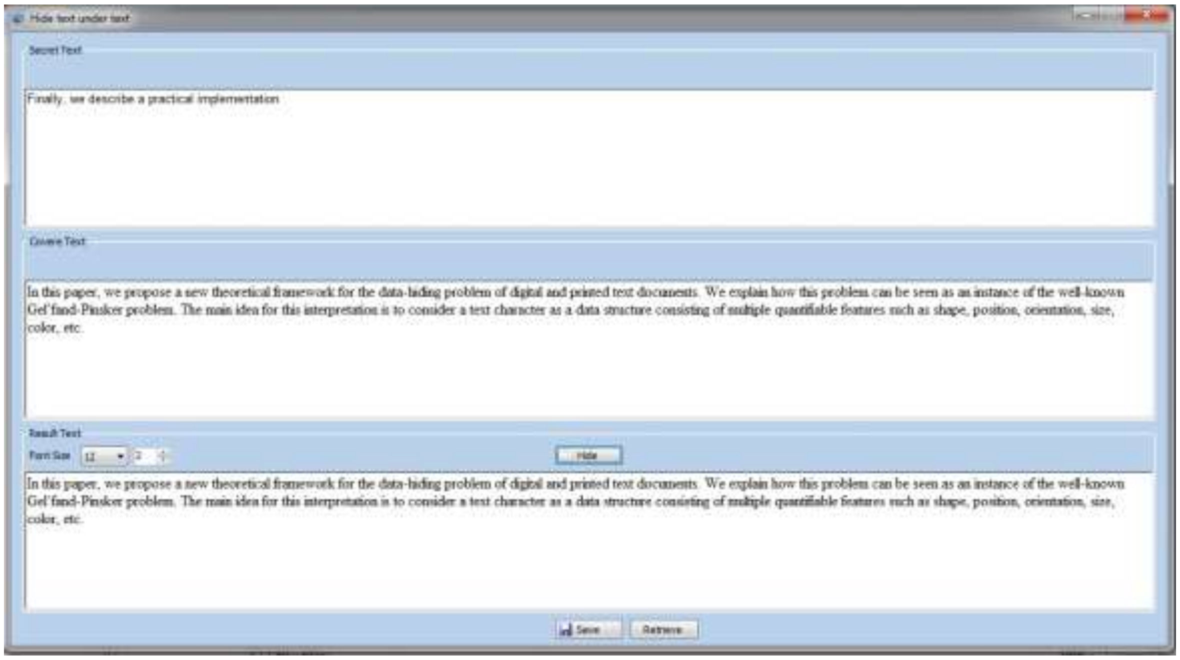
In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Result Text

Font Size 12 1

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-18: Result of hiding secret text using cover text size 12pt for one digit



Hide text under text

Secret Text

Finally, we describe a practical implementation

Cover Text

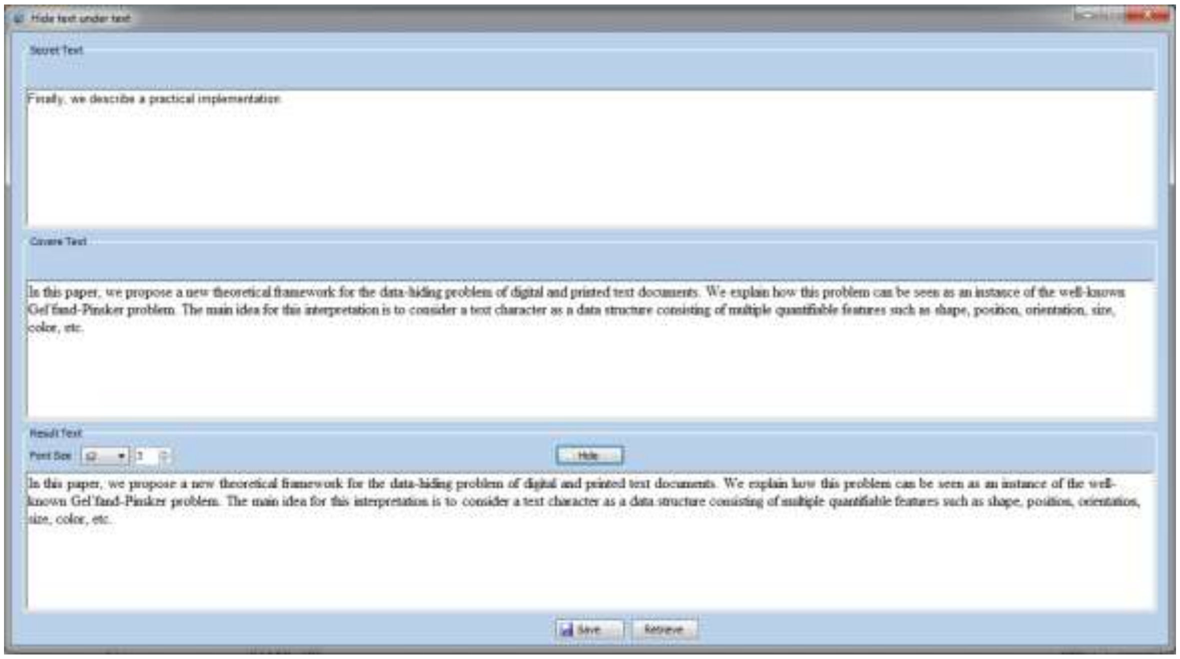
In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Result Text

Font Size 12 2

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-19: Result of hiding secret text using cover text size 12pt for two digits



Hide text under text

Secret Text

Finally, we describe a practical implementation

Cover Text

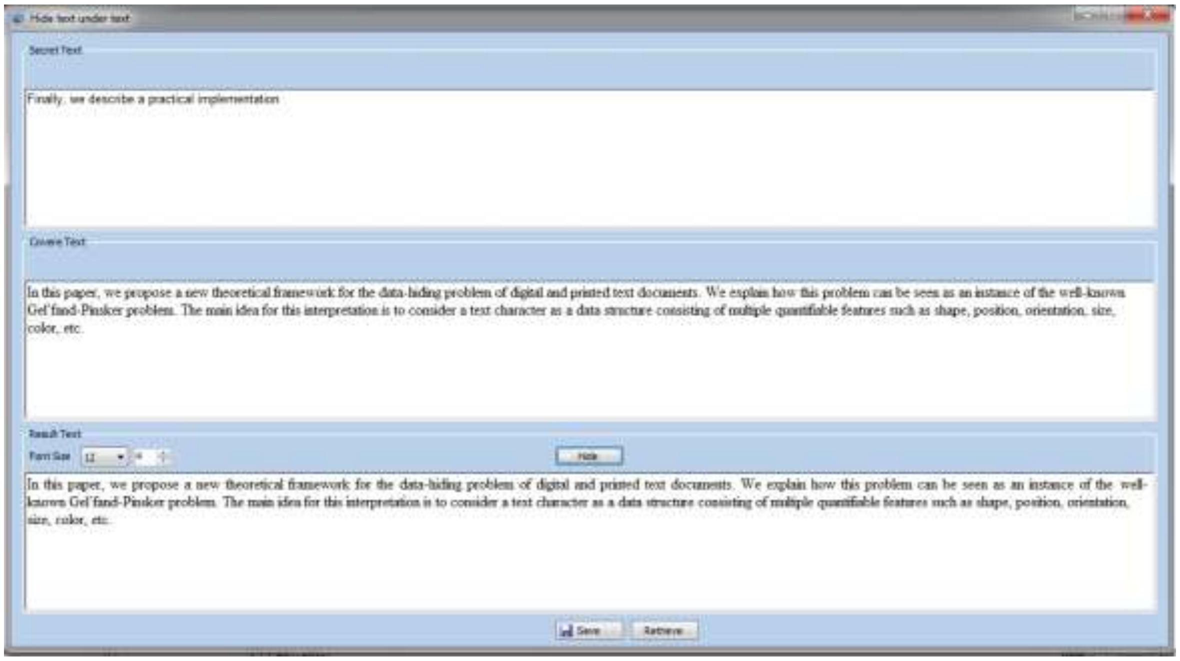
In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Result Text

Font Size 12 3

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-20: Result of hiding secret text using cover text size 12pt for three digits



Secret Text

Finally, we describe a practical implementation

Covere Text

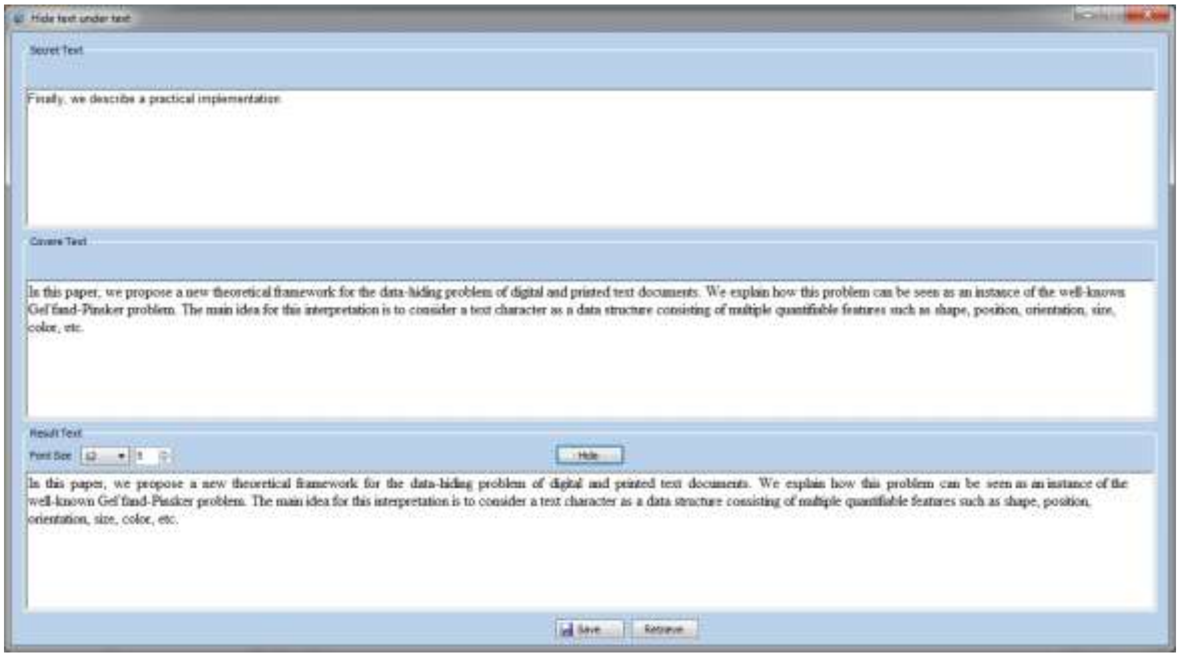
In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Result Text

Font Size 12 4

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-21: Result of hiding secret text using cover text size 12pt for four digits



Secret Text

Finally, we describe a practical implementation

Covere Text

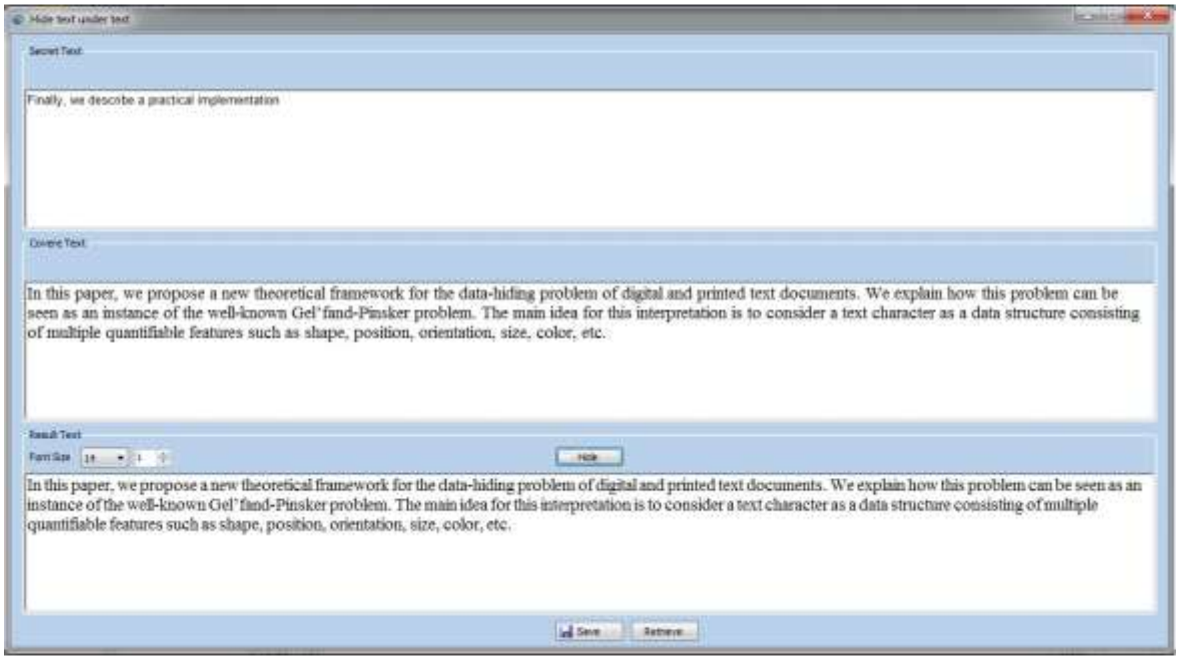
In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Result Text

Font Size 12 5

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-22: Result of hiding secret text using cover text size 12pt for five digits



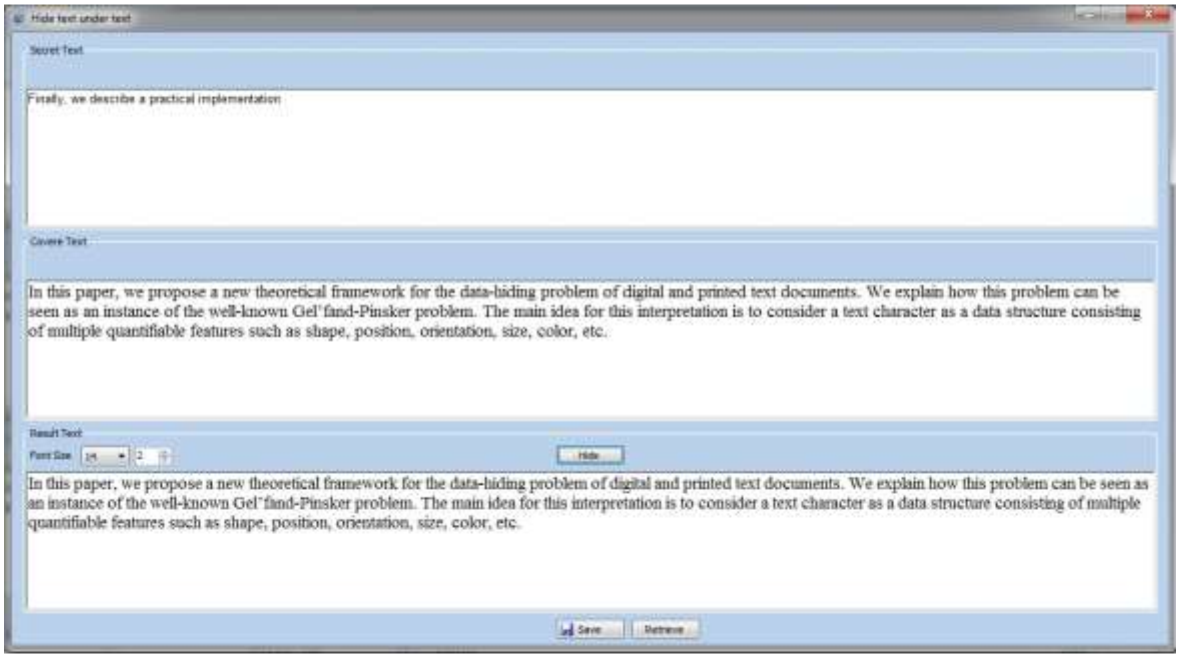
Finally, we describe a practical implementation

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Font Size 14 1

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-23: Result of hiding secret text using cover text size 14pt for one digit



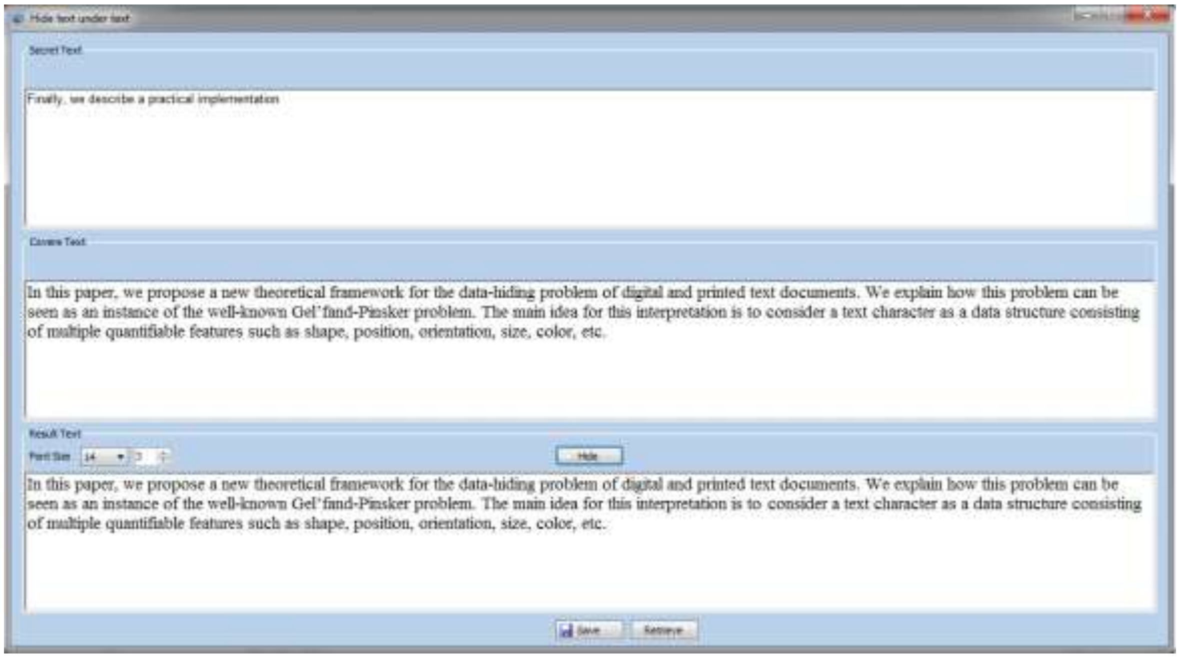
Finally, we describe a practical implementation

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Font Size 14 2

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-24: Result of hiding secret text using cover text size 14pt for two digits



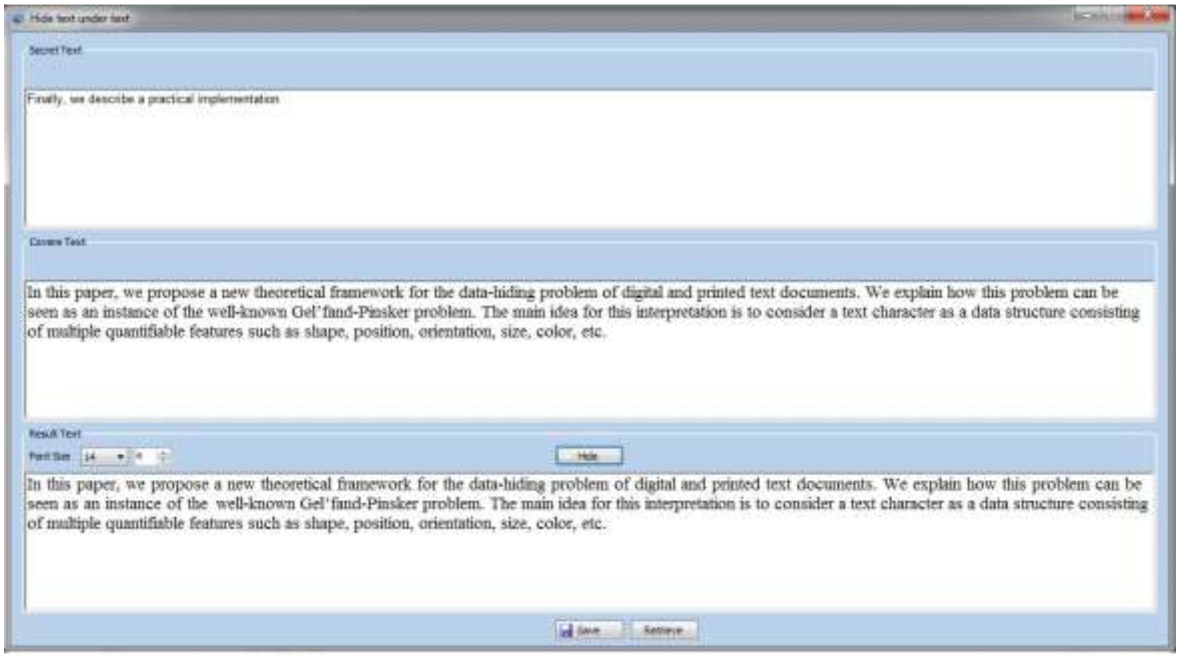
Finally, we describe a practical implementation

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Font Size 14 3

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-25: Result of hiding secret text using cover text size 14pt for three digits



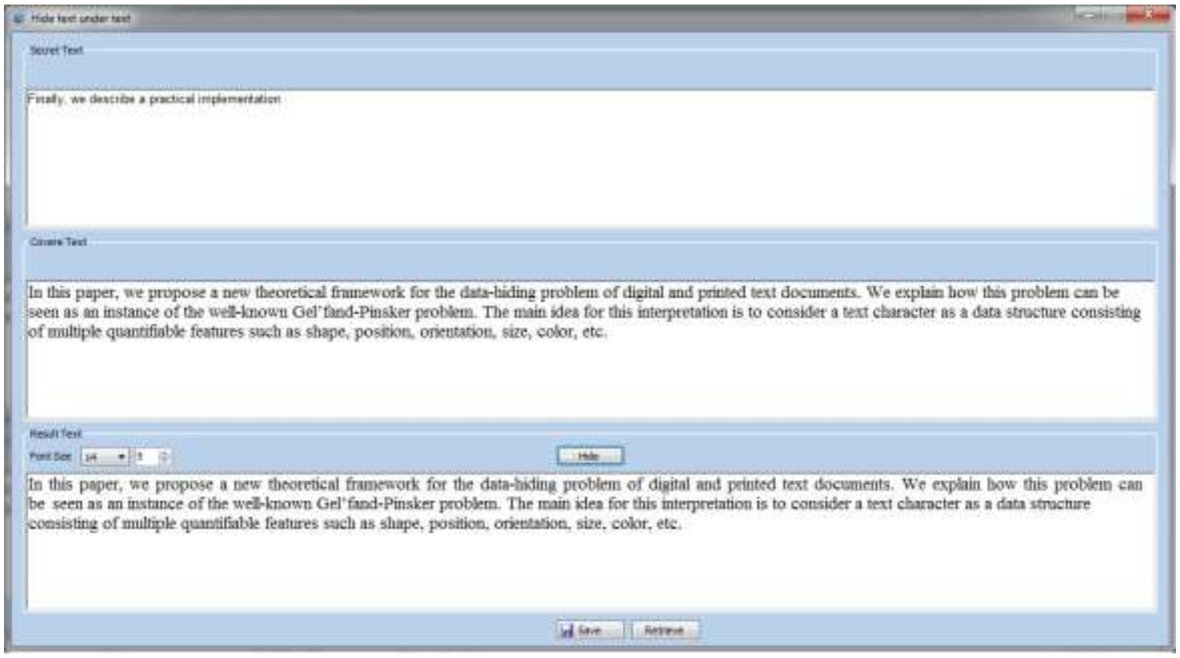
Finally, we describe a practical implementation

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Font Size 14 4

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-26: Result of hiding secret text using cover text size 14pt for four digits



Secret Text

Finally, we describe a practical implementation

Covere Text

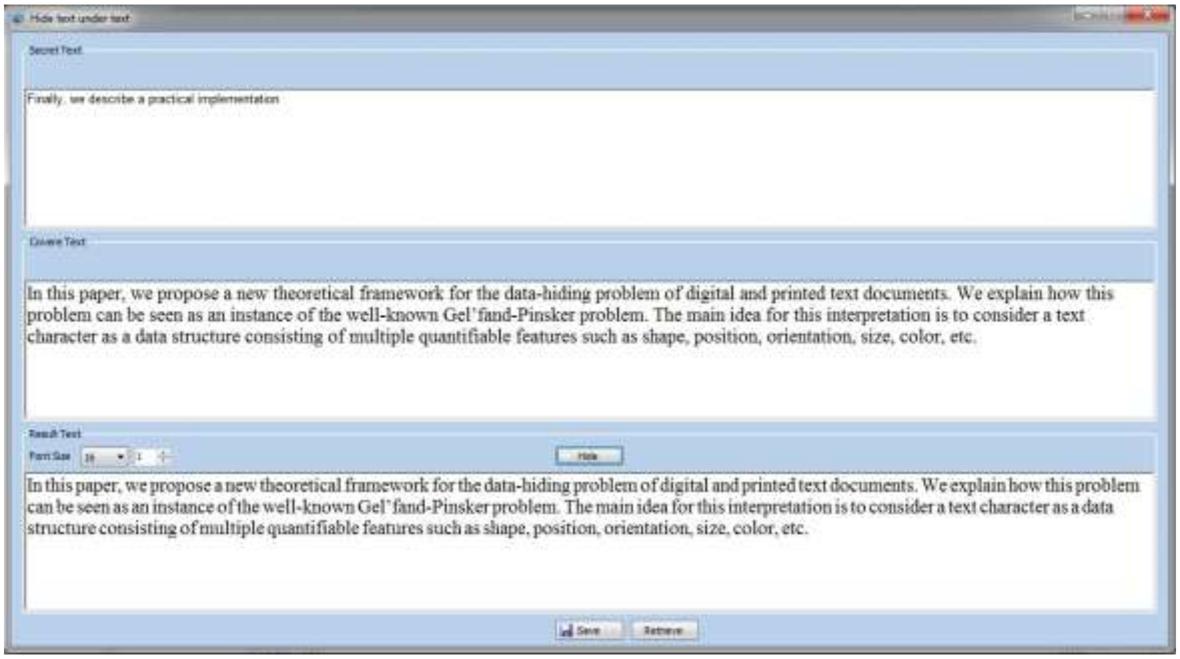
In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Result Text

Font Size 14 5

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-27: Result of hiding secret text using cover text size 14pt for five digits



Secret Text

Finally, we describe a practical implementation

Covere Text

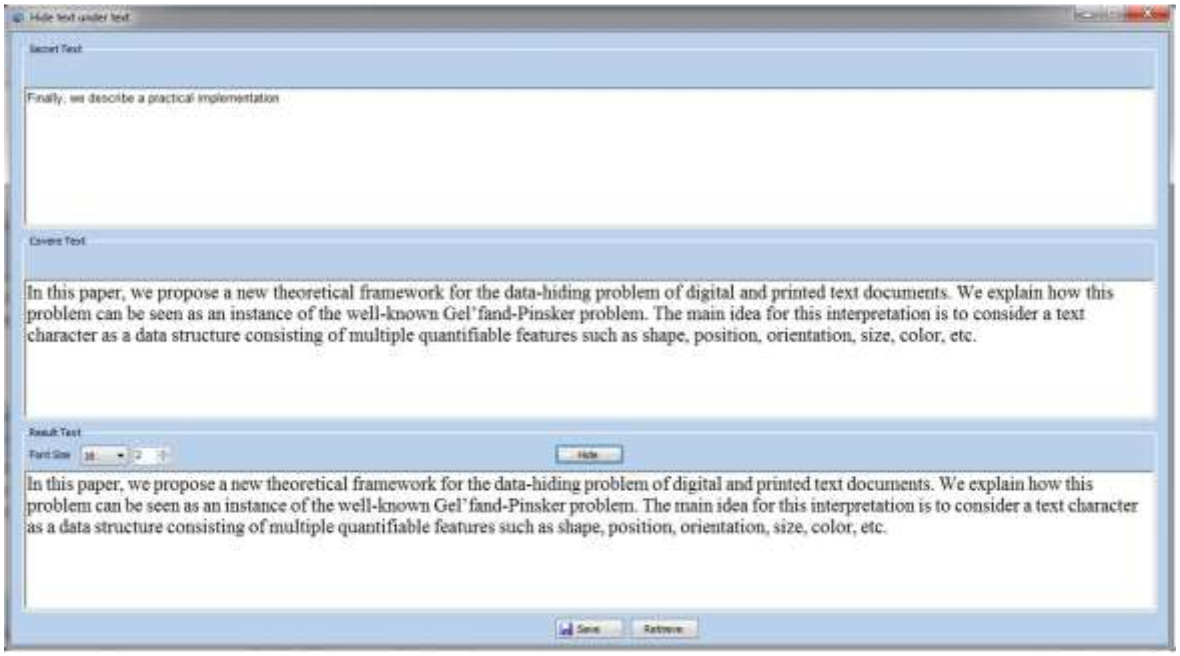
In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Result Text

Font Size 16 1

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-28: Result of hiding secret text using cover text size 16pt for one digit

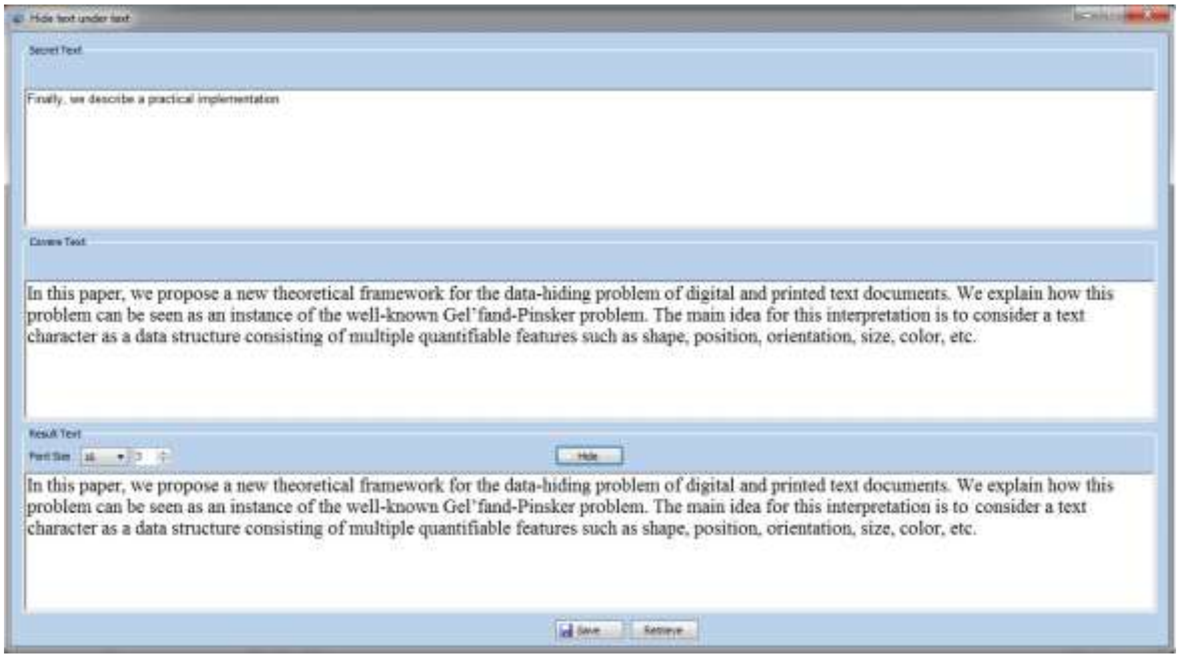


Finally, we describe a practical implementation

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-29: Result of hiding secret text in cover text size 16pt for two digits

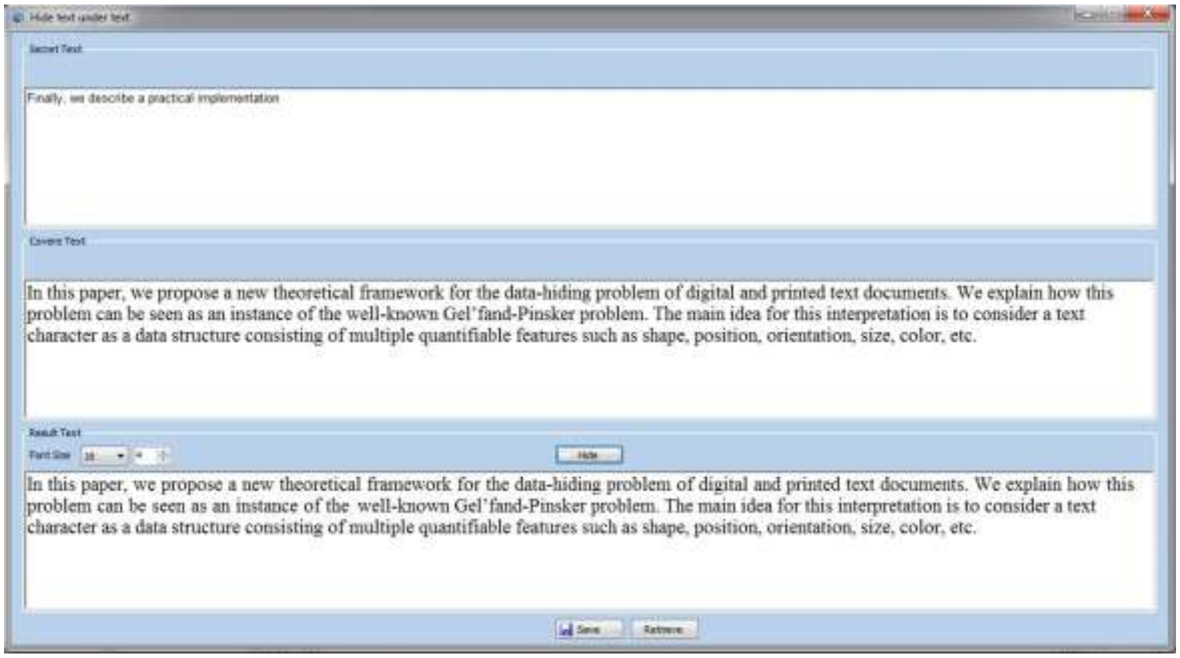


Finally, we describe a practical implementation

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-30: Result of hiding secret text using cover text size 16pt for three digits



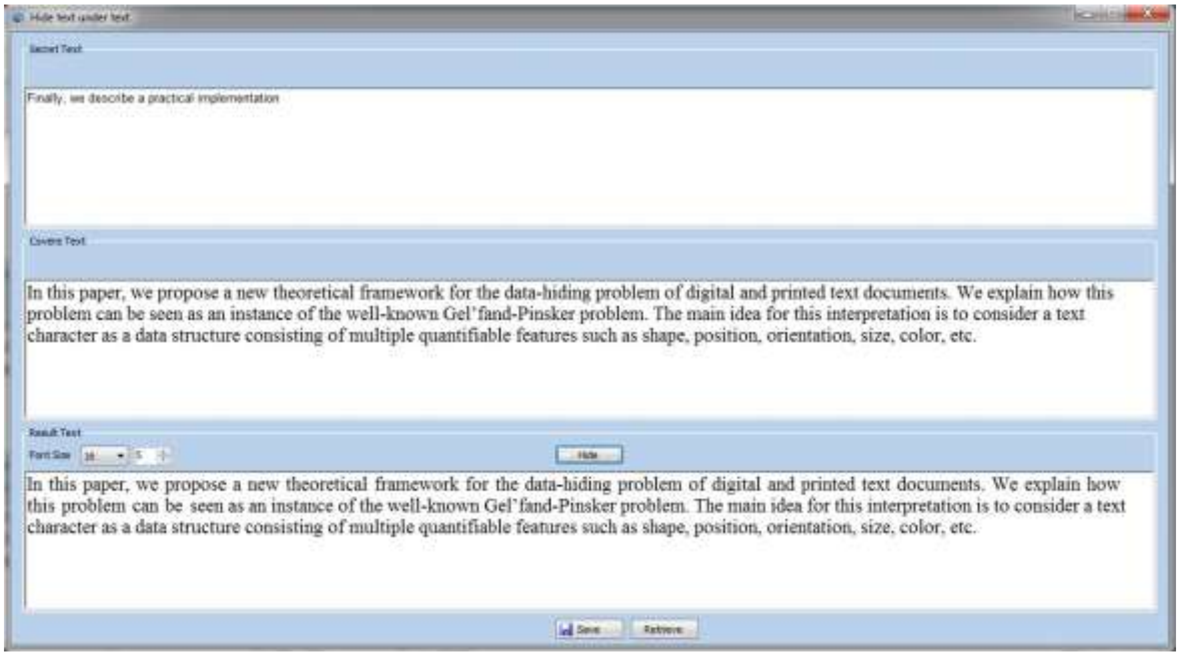
Finally, we describe a practical implementation

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Font Size 16 4

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-31: Result of hiding secret text using cover text size 16pt for four digits



Finally, we describe a practical implementation

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Font Size 16 5

In this paper, we propose a new theoretical framework for the data-hiding problem of digital and printed text documents. We explain how this problem can be seen as an instance of the well-known Gel'fand-Pinsker problem. The main idea for this interpretation is to consider a text character as a data structure consisting of multiple quantifiable features such as shape, position, orientation, size, color, etc.

Figure 4-32: Result of hiding secret text using cover text size 16pt for five digits

Chapter Five
Conclusion and Future Work

Information security has two branches; data encryption and data hiding. Image, audio, and text are used for data hiding. Data hiding in text is to embed text within another text to be invisible. Digital media has become more prevalent and expanding.

There are three major methods for data hiding text under text; open space methods that encode through manipulation of white space (unused space on the printed page), syntactic methods that utilize punctuation, and semantic methods that encode using manipulation of the words themselves.

There are two reasons why the manipulation of white space in particular yields useful results in open space method. First, changing the number of trailing spaces has little chance of changing the meaning of a phrase or sentence. Second, a casual reader is unlikely to take notice of slight modifications to white space.

There are three methods of using white space to encode data. The methods exploit inter-sentence spacing by placing either one or two spaces after each terminating character, end-of-line spaces by insert spaces at the end of lines, the data are encoded allowing for a predetermined number of spaces at the end of each line, and inter-word spacing in justified text by controlling where the extra spaces are placed, one space between words is interpreted as a “0” two spaces are interpreted as a “1”.

Character indexes used in the proposed solution to get the index of secret characters from the cover text to be able to retrieve the secret text in the stage of the text show. The characters' indexes are not enough to indicate the intended character from the cover text, so; ASCII code characters give a unique code for each character in the secret text "with sensitive case".

The octal numeral system is the base-8 number system, and uses the digits 0 to 7. Octal numerals can be made from binary numerals by grouping consecutive binary digits into groups of three (starting from the right).

The proposed solution takes advantage of the unused white space from the text "Cover Text" to hide the data "Secret Data" on the cover text. Changing the format of the secret text by setting the text size to 1px, setting the font color to white as the back color of cover text, then extract the index of the secret text characters from cover text, the remaining secret text characters that do not exist in the cover text will generate a unique code for each none existing characters from ASCII code characters, then convert the result of indexes and ASCII codes from decimal numeral system into octal numeral system by separating the characters with the number "9" and identifying the ASCII code "not index" with the number "8", then merging the secret text with the cover text using white space method to generate the result text which is hiding the secret message within it.

Future works may include converting the extract numbers of indexes and ASCII codes into equations allowing the regeneration of the extract numbers and these equations can be concealed and transported via cover text.

References

Abdul Qadir, M., & Ahmad, I. (2006). Digital Text Watermarking: Secure Content Delivery and Data Hiding in Digital Documents. *Security Technology, 2005. CCST '05. 39th Annual 2005 International Carnahan Conference*, 101 - 104.

Abdullah, Y. F., & Nasereddin, H. H. (2013). Proposed Data Hiding Technique – Text under Text. *American Academic & Scholarly Research Journal (AASRJ)*, 243-248.

Al-Hamami, A. H., & Al-Hamami, M. A. (2008). *Information Hiding – Steganography and Watermark.* Al-Sharjeh: ethraa.

Ali, A. (2007). Qualitative Spatial Image Data Hiding for Secure Data Transmission. *International Journal on Graphics, Vision and Image Processing*, 35-43.

AMIN, M. M., IBRAHIM, S., SALLEH, M., & KATMIN, M. R. (2003). *INFORMATION HIDING USING STEGANOGRAPHY.* Malaysia: University Teknologi.

Asif, A. A., Shaikh, A., Manza, R. R., & Ramteke, R. J. (2010). Conversion of Bitmap Text Images for Data Hiding. *Computational Intelligence and Computing Research (ICCIC), 2010 IEEE International Conference*, 1 – 4.

Bender, W., Gruhl, D., Morimoto, N., & Lu, A. (1996). Techniques for data hiding. *IBM SYSTEMS JOURNAL*, 313-336.

Borges, P. V., Izquierdo, E., & Mayer, J. (2008). Efficient Text Color Modulation for Printed Side Communications and Data Hiding. *Institute of Electrical and Electronics Engineers (IEEE)*, 79 – 86.

Brassil, J. T., Low, S., Maxemchuk, N. F., & O'Gorman, L. (1995). Electronic Marking and Identification Techniques to Discourage Document Copying. *Selected Areas in Communications, IEEE Journal*, 1495 – 1504.

Bulan, O., Sharma, G., & Monga, V. (2008). Adaptive Decoding For Halftone Orientation-Based Data Hiding. *Institute of Electrical and Electronics Engineers (IEEE)*, 1280 - 1283.

Chen, Y.-Y., Pan, H.-K., & Tseng, Y.-C. (2000). A Secure Data Hiding Scheme for Two-Color Images. *Institute of Electrical and Electronics Engineers (IEEE)*, 750 – 755.

Deguillaume, F., Rytsar, Y., Voloshynovskiy, S., & Pun, T. (2005). Protocols for data-hiding based text document security and automatic processing. *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference*.

Dutta, P., Bhattacharyya, D., & Kim, T.-h. (2009). Data Hiding in Audio Signal: A Review. *Kim International Journal of Database Theory and Application*, 1-8.

Harshavardhan, K., & Sugata, S. (2012). A Survey On Various Data Hiding Techniques And Their Comparative Analysis. *Acta Technica Corviniensis - Bulletin of Engineering*, 35 - 40.

Ibrahim, A., & Zabian, A. (2009). Algorithm for Text Hiding in Digital Image for Information Security. *International Journal of Computer Science and Network Security*, 262-268.

Jebran, A. (2007, June 19). *Text 2Text Steganography - Part 2*. Retrieved January 6, 2013, from Code Project: http://www.codeproject.com/Articles/19260/Text-2Text-Steganography-Part-2

Judge, J. C. (2001). Steganography: Past, Present, Future. *Security Essentials Certification: GSEC*, 1-29.

KAHN, D. (1996). *The Codebreakers: The Story of Secret Writing.* New York: Scribner.

Kim, H., & Mayer, J. (2007). Data Hiding for Binary Documents Robust to Print-Scan, Photocopy and Geometric Distortions. *Computer Graphics and Image Processing, 2007. SIBGRAPI 2007. XX Brazilian Symposium*, 105 - 112.

Kuo, W.-C., Jiang, D.-J., & Huang, Y.-C. (2008). A Reversible Data Hiding Scheme Based on Block Division. *Institute of Electrical and Electronics Engineers (IEEE)*, 365 – 369.

Kuo, W.-C., Kuo, S.-H., & Wuu, L.-C. (2010). High Embedding Reversible Data Hiding Scheme for JPEG. *Institute of Electrical and Electronics Engineers (IEEE)*, 74-77.

Kurup, S., Sridhar, G., & Sridhar, V. (2005). Entropy Based Data Hiding for Document Images. *World Academy of Science, Engineering and Technology*, 150-153.

Low, S. H., Maxemchuk, N. F., & Lapone, A. M. (1998). Document Identification for Copyright Protection Using Centroid Detection. *Institute of Electrical and Electronics Engineers (IEEE)*, 372 – 383.

Low, S. H., Maxemchuk, N. F., Brassil, J. T., & O'Gorman, L. (1995). Document Marking and Identification using Both Line and Word Shifting. *Institute of Electrical and Electronics Engineers (IEEE)*, 853 - 860.

Lu, H., Kot, A. C., & Susanto, R. (2002). Binary Image watermarking through Biased Binarization. *Nanyang Technological University, Institute for Infocomm Research*.

Michaud, E. (2003). Current Steganography Tools and Methods. *Current Steganography Tools and Methods*, 1-11.

Mikkilineni, A. K. (2012). INFORMATION HIDING IN PRINTED DOCUMENTS. *For the degree of Doctor of Philosophy Purdue University*, 1-10.

MOULIN, P., & KOETTER, R. (2005). Data-Hiding Codes. *Institute of Electrical and Electronics Engineers (IEEE)*, 2083 – 2126.

Nasereddin, H. H., & Al Farzaeai, M. S. (2010). PROPOSED DATA HIDING TECHNIQUE TEXT IMAGE INSIDE IMAGE (TIII). *International Journal of Research and Reviews in Applied Sciences*, 183- 193.

Net 2000 Ltd. (2010, January 1). *DataMasker.* Retrieved January 1, 2012, from DataMasker: http://www.DataMasker.com

Ni, Z., Shi, Y. Q., Ansari, N., Su, W., Sun, Q., & Lin, X. (2008). Robust Lossless Image Data Hiding Designed for Semi-Fragile Image Authentication. *Institute of Electrical and Electronics Engineers (IEEE)*, 1051-8215.

Por, L. Y., Wong, K., & Chee, K. O. (2012). A text-based data hiding method using Unicode space characters. *The Journal of Systems and Software*, 1075 – 1082.

Rahma, A. S., AbdulWahab, H. B., & Al-Noori, A. Y. (2011). Proposed Steganographic Method for Data Hiding in Microsoft Word Documents Structure. *Al-Mansour Journal*, 1 - 29.

Samphaiboon, N. (2011). Steganography via running short text messages. *Springer Science, Business Media, LLC 2009 Multimed Tools*, 569–596.

Schneier, B. (2004). *Secret & Lies: Digital Security in a Networked World; with new information about post-9/11 security.* Indianapolis, Indiana: Wiley Publishing.

Stallings, W. (1999). *Cryptography and network security.* New Jersey: Prentice Hall.

Stanev, S. (2005). Steganographic Systems. *CSC/MAT*.

Vill´an, R., Voloshynovskiy, S., Koval, O., Vila, J., Topak, E., Deguillaume, F., et al. (2006). Text Data-Hiding for Digital and Printed Documents: Theoretical and Practical Considerations. *Stochastic Information Processing (SIP)*, 15-26.

wikipedia. (2013, April 21). *Octal.* Retrieved Aprile 26, 2013, from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Octal

wikipedia. (2013, March 08). *Point (typography).* Retrieved May 02, 2013, from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Point_(typography)

wikipedia. (2013, April 30). *Sentence spacing.* Retrieved May 02, 2013, from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Sentence_spacing

Wong, P. H., Au, O. C., & Wong, J. W. (2000). Data Hiding and Watermarking in JPEG Compressed Domain by DC Coefficient Modification. *Proc. SPIE Security and Watermarking of Multimedia Contents*, 5-2.

Xuan, G., Shi, Y. Q., Chai, P., Tong, X., Teng, J., & Li, J. (2008). Reversible Binary Image Data Hiding By Run-Length Histogram Modification. *Institute of Electrical and Electronics Engineers (IEEE)*, 1-4.

Yang, H., & Kot, A. C. (2005). DATA HIDING FOR TEXT DOCUMENT IMAGE AUTHENTICATION BY CONNECTIVITY-PRESERVING. *Institute of Electrical and Electronics Engineers (IEEE)*, 505 – 508.

YILMAZ, A. (2003). ROBUST VIDEO TRANSMISSION USING DATA HIDING. *In partial fulfillment of the requirements for the degree of Master of Science, the graduate school of natural and applied sciences of the Middle East technical university*, 20-30.

Zhang, X.-P., Li, K., & Wang, X. (2008). A Novel Look-Up Table Design Method for Data Hiding With Reduced Distortion. *Institute of Electrical and Electronics Engineers (IEEE)*, 769 - 776.

Zou, D., & Shi, Y. Q. (2005). Formatted Text Document Data Hiding Robust to Printing, Copying and Scanning. *Institute of Electrical and Electronics Engineers (IEEE)*, 4971 – 4974.

Appendices

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace DataHidingTextUnderText
{
    public partial class frmMain : Form
    {

        private int iNoOfDigits = 1;
        private float fontSize = 12;
        private string[] iDec = null;
        private string sWholeIndexes = string.Empty;

        public frmMain()
        {
            InitializeComponent();
            rtxtResultText.Font = new Font(rtxtResultText.Font.FontFamily,
fontSize);
            iNoOfDigits = (int)nudNoOfDigits.Value;
        }

        private bool validation()
        {
            bool result = true;
            if (rtxtSecretText.Text.Trim().Equals(string.Empty) ||
rtxtCoveredText.Text.Trim().Equals(string.Empty))
            {
                MessageBox.Show("Please enter Covered and Secret text");
                result = false;
            }
            return result;
        }

        private void exportIndexes()
        {
            iDec = new string[rtxtSecretText.Text.Length];
            sWholeIndexes = string.Empty;
            char strChars;
            for (int i = 0; i < rtxtSecretText.Text.Length; i++)
            {
                strChars = Char.Parse(rtxtSecretText.Text.Substring(i, 1));
                if (rtxtCoveredText.Text.IndexOf(rtxtSecretText.Text.Substring(i,
1)) >= 0)
                {
                    iDec[i] = 
Convert.ToString(rtxtCoveredText.Text.IndexOf(rtxtSecretText.Text.Substring(i, 1)), 
8);
                }
                else
                {
                    iDec[i] = "8" + Convert.ToString(Convert.ToInt32(strChars), 8);
                }
            }

            for (int i = 0; i < iDec.Length; i++)
            {
                if (i + 1 < iDec.Length && !iDec[i + 1].Substring(0, 1).Equals("8"))
                {
                    sWholeIndexes = sWholeIndexes + iDec[i] + "9";
                }
                else
                {
                    sWholeIndexes = sWholeIndexes + iDec[i];
                }
            }
        }

        private void margeData()
        {
            rtxtResultText.Text = rtxtCoveredText.Text;
            int iIndex = 0, leng = iNoOfDigits;
            for (int i = 0; i < rtxtResultText.Text.Length; i++)
                if (rtxtResultText.Text.Substring(i, 1).Equals(" "))
                {
                    if (sWholeIndexes.Substring(iIndex, sWholeIndexes.Length - 
iIndex).Length < iNoOfDigits)
                        leng = sWholeIndexes.Substring(iIndex, sWholeIndexes.Length 
- iIndex).Length;

                    rtxtResultText.Text = rtxtResultText.Text.Remove(i, 1);
                    rtxtResultText.Text = rtxtResultText.Text.Insert(i, " " + 
sWholeIndexes.Substring(iIndex, leng) + " ");
                    iIndex += iNoOfDigits;
                    i += iNoOfDigits + 1;

                    if (iIndex >= sWholeIndexes.Length)
                        break;
                }
        }

        private void hideData()
        {
            int remainingDigitsCount = sWholeIndexes.Length;
            for (int i = 0; i < rtxtResultText.Text.Length && remainingDigitsCount > 
0; i++)
                if (rtxtResultText.Text.Substring(i, 1).Equals(" "))
                {
                    if (remainingDigitsCount < iNoOfDigits)
                    {
                        rtxtResultText.Select(i, remainingDigitsCount + 1);
                        rtxtResultText.SelectionFont = new
Font(rtxtResultText.Font.FontFamily, 1);
                        rtxtResultText.SelectionColor = rtxtResultText.BackColor;
                        rtxtResultText.DeselectAll();
                        break;
                    }
                    else if (rtxtResultText.Text.Length >= i + iNoOfDigits + 1 &&
rtxtResultText.Text.Substring(i + iNoOfDigits + 1, 1).Equals(" "))
                    {
                        rtxtResultText.Select(i, iNoOfDigits + 2);
                        rtxtResultText.SelectionFont = new
Font(rtxtResultText.Font.FontFamily, 1);
                        rtxtResultText.SelectionColor = rtxtResultText.BackColor;
                        rtxtResultText.DeselectAll();
                        i += iNoOfDigits + 1;
                        remainingDigitsCount -= iNoOfDigits;
                    }
                }
        }

        private void frmMain_Load(object sender, EventArgs e)
        {
            cmbFontSize.SelectedIndex = 1;
        }

        private void btnHide_Click(object sender, EventArgs e)
        {
            DateTime time = new DateTime();
            time = DateTime.Now;
            rtxtResultText.Text = string.Empty;
            if (validation())
            {
                exportIndexes();
                margeData();
                hideData();
            }
            lblTime.Text = (DateTime.Now - time).ToString();
            txtTime.Text = lblTime.Text;
        }

        private void btnSave_Click(object sender, EventArgs e)
        {
            SaveFileDialog sfd = new SaveFileDialog();
            sfd.Filter = "Rich File|.rtf";
            if (sfd.ShowDialog() != System.Windows.Forms.DialogResult.Cancel)
                rtxtResultText.SaveFile(sfd.FileName,
RichTextBoxStreamType.RichText);
        }


        private void cmbFontSize_SelectedIndexChanged(object sender, EventArgs e)
        {
            fontSize = float.Parse(cmbFontSize.Text.ToString());
            rtxtResultText.Font = new Font(rtxtResultText.Font.FontFamily,
fontSize);
```

```
            rtxtCoveredText.Font = new Font(rtxtCoveredText.Font.FontFamily,
fontSize);
            hideData();
        }

        private void nudNoOfDigits_ValueChanged(object sender, EventArgs e)
        {
            iNoOfDigits = (int)nudNoOfDigits.Value;
        }

        private void rtxtSecretText_TextChanged(object sender, EventArgs e)
        {
            if (rtxtSecretText.Text != string.Empty && rtxtSecretText.Text.Length >
5)
                nudNoOfDigits.Maximum = 5;
            else
                nudNoOfDigits.Maximum = rtxtSecretText.Text.Length;

            if (nudNoOfDigits.Maximum > 3)
                nudNoOfDigits.Value = 3;
            else
                nudNoOfDigits.Value = nudNoOfDigits.Maximum;
        }

        private void retrieveIndexes()
        {
            string secretText = string.Empty;
            List<int> indexesToRemove = new List<int>();
            sWholeIndexes = string.Empty;
            for (int i = 0; i < rtxtResultText.Text.Length; i++)
            {
                rtxtResultText.Select(i, 1);
                if (!rtxtResultText.SelectedText.Equals(" ") &&
rtxtResultText.SelectionFont.Size.Equals(1) &&
rtxtResultText.SelectionColor.Equals(Color.White))
                {
                    sWholeIndexes = sWholeIndexes + rtxtResultText.SelectedText;
                    indexesToRemove.Add(i);
                    rtxtResultText.DeselectAll();
                }
            }
            for (int i = indexesToRemove.Count - 1; i >= 0; i--)
            {
                rtxtResultText.Text = rtxtResultText.Text.Remove(indexesToRemove[i],
1);
            }
            rtxtResultText.Text = rtxtResultText.Text.Replace("  ", " ");
            iDec = new string[sWholeIndexes.Length];
            if (sWholeIndexes != string.Empty)
                sWholeIndexes = (!sWholeIndexes.Substring(0, 1).Equals("8") &&
!sWholeIndexes.Substring(0, 1).Equals("9") ? "9" : "") + sWholeIndexes;
            int index = -1;
            for (int i = 0; i < sWholeIndexes.Length; i++)
            {
                if (i == 0 || sWholeIndexes.Substring(i, 1).Equals("8") ||
sWholeIndexes.Substring(i, 1).Equals("9"))
```

```
                {
                    index += 1;
                    iDec[index] = sWholeIndexes.Substring(i, 1);
                }
                else
                {
                    iDec[index] = iDec[index] + sWholeIndexes.Substring(i, 1);
                }
            }
            for (int i = 0; i < iDec.Length; i++)
            {
                if (iDec[i] == null)
                {
                    break;
                }
                else if (!iDec[i].Equals("") && !iDec[i].Equals("8") &&
!iDec[i].Equals("9"))
                {
                    if (iDec[i].Contains("8"))
                    {
                        //remove additional number
                        iDec[i] = iDec[i].Replace("8", "");
                        //convert octal to decimal
                        iDec[i] =
Convert.ToInt64(Convert.ToString(Convert.ToInt64(iDec[i], 8), 10)).ToString();
                        //get char value from decimal code
                        secretText = secretText +
Convert.ToChar(long.Parse(iDec[i])).ToString();
                    }
                    else if (iDec[i].Contains("9"))
                    {
                        //remove additional number
                        iDec[i] = iDec[i].Replace("9", "");
                        //convert octal to decimal
                        iDec[i] =
Convert.ToInt64(Convert.ToString(Convert.ToInt64(iDec[i], 8), 10)).ToString();
                        //get char value from decimal code
                        secretText = secretText +
rtxtResultText.Text.Substring(int.Parse(iDec[i]), 1);
                    }
                }
            }

            MessageBox.Show(secretText);
        }

        private void btnRetrieve_Click(object sender, EventArgs e)
        {
            retrieveIndexes();
        }

        private void btnBrowseST_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            if (ofd.ShowDialog() != System.Windows.Forms.DialogResult.Cancel)
            {
                StreamReader reader = new StreamReader(ofd.FileName);
                rtxtSecretText.Text = reader.ReadToEnd();
                reader.Close();
            }
        }

        private void btnBrowseCT_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            if (ofd.ShowDialog() != System.Windows.Forms.DialogResult.Cancel)
            {
                rtxtCoveredText.LoadFile(ofd.FileName);
                rtxtCoveredText.Modified = false;
            }
        }

        private void btnBrowseRT_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            if (ofd.ShowDialog() != System.Windows.Forms.DialogResult.Cancel)
            {
                rtxtResultText.LoadFile(ofd.FileName);
                rtxtResultText.Modified = false;
            }
        }

    }
}
```