

# **Studies on Information Hiding Techniques in Word Processor Documents**

*By*

**BALA KRISHNAN R**

(Enrolment No: ENGG 02 2012 04 010)

**Indira Gandhi Centre for Atomic Research, Kalpakkam**

*A Thesis Submitted to the  
Board of Studies in Engineering Sciences  
In partial fulfillment of requirements  
For the Degree of*

**DOCTOR OF PHILOSOPHY**

*of*

**HOMI BHABHA NATIONAL INSTITUTE**



**August, 2017**

# Homi Bhabha National Institute

## Recommendations of the Viva Voce Committee

As members of the Viva Voce Committee, we certify that we have read the dissertation prepared by **R. Bala Krishnan** entitled **Studies on Information Hiding Techniques in Word Processor Documents** and recommend that it may be accepted as fulfilling the thesis requirement for the award of Degree of Doctor of Philosophy.

Chairman – Dr. U. Kamachi Mudali (Moved out ) <i>f. Sarbajit</i> [G. SASIKALA, Dean-Academic]	Date: 20.09.2018
Guide / Convener – Dr. M. Sai Baba <i>m. Sai baba</i>	Date: 20-9-18
Examiner – Dr. Subhamoy Maitra, ISI, Kolkata <i>Subhamoy Maitra</i>	Date: 20.9.2018
Member 1 – Dr. B. K. Panigrahi <i>B.K. Panigrahi</i>	Date: 20/9/18
Member 2 – Dr. Sharat Chandra <i>Sharat Chandra</i>	Date: 20/9/2018
Technology Adviser – Dr. S. A. V. Satya Murty <i>Satya Murty</i>	Date: 20/9/2018

Final approval and acceptance of this thesis is contingent upon the candidate's submission of the final copies of the thesis to HBNI.

I/We hereby certify that I/we have read this thesis prepared under my/our direction and recommend that it may be accepted as fulfilling the thesis requirement.

Date: 20-9-18

Place: IGCAR, Kalpakkam

*m. Sai baba*

Dr. M. Sai Baba  
Guide

## STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at Homi Bhabha National Institute (HBNI) and is deposited in the Library to be made available to borrowers under rules of the HBNI.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the Competent Authority of HBNI when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

  
[R. Bala Krishnan]

## DECLARATION

I, hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and has not been submitted earlier as a whole or in part for a degree / diploma at this or any other Institution / University.

  
[R. Bala Krishnan]



## LIST OF PUBLICATIONS

### Journal

#### a. Published

1. Text steganography: A novel character-level embedding algorithm using font attribute

**R. Bala Krishnan**, Prasanth Kumar Thandra, S. A. V. Satya Murty, *Security and Communication Networks*, John Wiley & Sons, 9 (18), pp. 6066 — 6079, Feb 2017.

2. Font Attributes based Text Steganographic algorithm (FATS) for communicating images: A nuclear power plant perspective

**R. Bala Krishnan**, Prasanth Kumar Thandra, S. A. V. Satya Murty, P. Thiagarajan, *Kerntechnik*, Carl Hanser Verlag, 82 (1), pp. 98 — 111, Mar 2017.

#### b. In preparation

1. Exploring the font attributes of word processor documents: A steganographic perspective

**R. Bala Krishnan**, Prasanth Kumar Thandra.

2. Text steganography: A novel mixed-type method with high embedding capacity

**R. Bala Krishnan**, Prasanth Kumar Thandra.

3. Text steganography: A case study on embedding images in vector formats

**R. Bala Krishnan**, Prasanth Kumar Thandra, M. Sai Baba.

## Conferences

### a. Published

1. An overview of text steganography

**R. Bala Krishnan**, Prasanth Kumar Thandra, M. Sai Baba, Fourth International Conference on Signal Processing, Communication and Networking, Madras Institute of Technology, Chennai, Tamil Nadu, India, Mar 2017.

2. Exploring the font attributes of Microsoft Word: A steganographic perspective

**R. Bala Krishnan**, Prasanth Kumar Thandra, M. Sai Baba, International Conference on Frontiers in Engineering, Applied Sciences and Technology, National Institute of Technology Tiruchirappalli, Trichy, Tamil Nadu, India, Mar 2017.

### b. In preparation

1. Text steganography: A character-level embedding algorithm for case-sensitive messages

**R. Bala Krishnan**, Prasanth Kumar Thandra, M. Sai Baba.

  
[R. Bala Krishnan]

**DEDICATED**

**To**

**MY FATHER**

## **ACKNOWLEDGEMENTS**

I would like to thank people who have played their part in shaping this thesis and help me to evolve as a researcher.

First, I would like to express my gratitude to Mr. Prasanth Kumar Thandra, who played a major role throughout the course of my PhD work. His support helped me in all the time of research and writing of publications as well as this thesis. I am profoundly indebted to him and I owe my deepest gratefulness and respect to him.

Next, I would like to thank my thesis advisor Dr. M. Sai Baba, for his advice and constant support during the writing of this thesis as well as publications. His motivation and guidance helped me to improve my research skills. I owe my deepest respect to him for his support. I would like to thank the members of my doctoral committee, Dr. U. Kamachi Mudali, Dr. B. K. Panigrahi, Dr. Sharat Chandra and Dr. S. A. V. Satya Murty for their valuable suggestions, tough questions and encouragement which helped me to gain deeper insights and intensify my quest to explore the research in different perspective.

I would like to thank Homi Bhabha National Institute (HBNI) and Department of Atomic Energy (DAE) for funding my research work. I would like to thank the former Director of Indira Gandhi Centre for Atomic Research (IGCAR) Shri. S. C. Chetal, for offering me the opportunity to carry out my research work at Electronics & Instrumentation Group (EIG), IGCAR. I am deeply grateful to the subsequent Directors, Dr. P. R. Vasudeva Rao and Dr. S. A. V. Satya Murty and also to the current Director, Dr. A. K. Bhaduri for allowing me to carry forward my research work in this esteemed institute (IGCAR).

I express my gratitude to Mr. J. Rajan for his constant motivation and assistance during tough situations.

I am also grateful to Dr. P. Thiyagarajan, Dr. D. Karthickeyan, Mr. M. Naveen Raj and Mr. D. Sanjay Kumar for providing valuable suggestions to improve my publications as well as thesis.

I would like to further thank all my colleagues from EIG group. Special thanks to my lab mates Mrs. Tripura Sundari, Mr. Sai, Mr. Raghava Reddy, Mr. Mohit Kumar Yadav, Mr. Prasanth Sharma, Mrs. Vijaya Lakshmi, Mr. Nambirajan, Mr. Siva Shankar, Mrs. Shanmugapriya, Ms. S. Prema, Ms. Karthika, Mr. M. Harish, Mr. J. Harish and Mr. Vikas Kumar for affording comfortable and joyful laboratory environment.

I thank all my batch mates Mr. J. Abuthahir, Mr. K. Srinivasan, Mr. K. G. Raghavendra, Mr. K. Yadagiri, Mr. Sivadasan, Mr. Chandan Kumar Bhagat, Mr. Santhosh, Mr. Gopi, Mrs. Sumathi, Ms. Sravanthi Srikantam, Mrs. Preethi and JRF enclavians for providing a pleasant and conducive environment during my stay in JRF enclave.

I have to express my thankfulness as well to my friends Mr. Bala Sundaram, Mr. Uday, Mr. Devidas, Mr. Vairavel, Mr. Zaibudeen, Mr. Lakshmanan, Mr. Radhikesh, Mr. Irshad, Mr. Raghavendran, Mr. Barath, Mr. Sai Kumaran, Mr. Thangam, Mr. Shiva Kumar, Mr. Santhosh, and Mr. Thirukumaran who had accompanied me during these years of good and stressed moments.

I owe my gratitude to my mother S. Seetha Lakshmi whose innumerable sacrifice, unconditional love and confidence has driven me this far. I thank my brothers, sisters and friends for their affection and wishes.

  
[R. Bala Krishnan]



# CONTENTS

	<b>Page No.</b>
<b>SYNOPSIS</b>	xiii
<b>LIST OF FIGURES</b>	xxiii
<b>LIST OF TABLES</b>	xxviii
<b>LIST OF ABBREVIATIONS AND SYMBOLS</b>	xxxi
<b>LIST OF TERMINOLOGIES</b>	xxxiii
 <b>CHAPTER 1 INTRODUCTION</b>	 1
1.1 Introduction	1
1.2 Watermarking	3
1.3 Cryptography	4
1.4 Steganography	4
1.5 Need for steganography in the current scenario	5
1.6 Recent trends in steganography	6
1.7 Desired characteristics of digital steganography	7
1.8 Discussion	9
1.9 Motivation for the thesis	10
1.10 Objectives of the thesis	11
1.11 Contributions of the thesis	12
1.12 Organization of the thesis	12
 <b>CHAPTER 2 LITERATURE SURVEY</b>	 15
2.1 Introduction	15
2.2 Classification of text steganography	15
2.2.1 <i>Character-level embedding technique (CLET)</i>	17
2.2.2 <i>Bit-level embedding technique</i>	22
2.2.3 <i>Mixed-type embedding technique</i>	34
2.3 Comparison of the existing methods	35
2.4 Summary	37

<b>CHAPTER 3</b>	<b>EXPLORING THE FONT ATTRIBUTES OF WORD PROCESSOR DOCUMENTS</b>	<b>39</b>
3.1	Introduction	39
3.2	Employing the font attributes for steganography	43
3.3	Classification of the font attributes	61
3.4	Comparison of the font attributes	64
3.5	Discussion	68
3.6	Summary	69
<b>CHAPTER 4</b>	<b>EMBEDDING TEXT</b>	<b>71</b>
4.1	Introduction	71
4.2	Handling the non-uniform occurrence frequencies of characters	72
4.3	Theoretical background of the development of Method-A	74
4.4	Generation of Frequency Normalization Set (FNS)	76
4.5	Character & String Mapping (CSM)	77
4.6	Embedding algorithm	78
4.7	Extraction algorithm	80
4.8	Evaluation parameters	82
	4.8.1 <i>Secrecy</i>	82
	4.8.2 <i>Embedding capacity</i>	83
	4.8.3 <i>Uniformity in embedding probability</i>	84
	4.8.4 <i>Comparison with existing methods</i>	86
4.9	Security aspect	87
	4.9.1 <i>Uniformity in embedding probability</i>	87
	4.9.2 <i>Distribution in stego characters</i>	87
	4.9.3 <i>Frequency distribution of stego characters</i>	88
	4.9.4 <i>Cryptographic aspect</i>	91
4.10	Application to case-sensitive letters	92

4.11	Summary	94
<b>CHAPTER 5</b>	<b>EMBEDDING BINARY DATA</b>	<b>97</b>
5.1	Introduction	97
5.2	Binary to Character Converter (BCC)	98
5.3	Embedding algorithm	102
5.4	Extraction algorithm	104
5.5	Evaluation parameters	105
5.5.1	<i>Embedding capacity and Bits per distortion</i>	105
5.5.2	<i>Uniformity in embedding probability</i>	106
5.5.3	<i>Comparison with other methods</i>	106
5.6	Security aspect	107
5.7	Case study on nuclear power plants	109
5.8	Summary	113
<b>CHAPTER 6</b>	<b>EMBEDDING IMAGE</b>	<b>115</b>
6.1	Introduction	115
6.2	Vector format	117
6.3	Custom format to represent an image	120
6.3.1	<i>Elements of an image</i>	123
6.3.2	<i>Code representation of elements</i>	125
6.3.3	<i>Image to code conversion procedure</i>	127
6.3.4	<i>Code to image conversion procedure</i>	129
6.4	Methodology adopted to develop Method-C	130
6.4.1	<i>Image to code conversion algorithm</i>	132
6.4.2	<i>Embedding algorithm</i>	132
6.4.3	<i>Extraction algorithm</i>	135
6.4.4	<i>Image drawing algorithm</i>	140
6.5	Evaluation and Security aspect	141
6.5.1	<i>Size comparison</i>	141
6.5.2	<i>Embedding capacity</i>	145
6.5.3	<i>Bits per distortion</i>	145

6.5.4	<i>Secrecy</i>	146
6.5.5	<i>Comparison of Method-B and Method-C</i>	147
6.5.6	<i>Transmission error</i>	148
6.6	Summary	151
<b>CHAPTER 7</b>	<b>SUMMARY AND SCOPE FOR FUTURE INVESTIGATIONS</b>	153
7.1	Summary and conclusions	153
7.2	Scope for the future work	158
<b>REFERENCES</b>		159
<b>APPENDIX — A</b>		175
	Procedure to generate Frequency Normalization Set (FNS)	175
<b>APPENDIX — B</b>		181
	Illustration of the imperceptibility level of Method-C	181

## SYNOPSIS

Digital communication plays an important role in connecting geographically distributed individuals as well as organizations. To assist the communication, the information is generated (or converted) and shared in digital form. This, also, allows a malicious intender to destroy, corrupt or steal the data at ease.

A malicious intender having the personal data, such as credit or debit card details, can make fraudulent transactions. Similarly, an attacker possessing the data of an organization, like the customer details or medical records, can threaten the organization for ransom or sell them in darknet. Such attacks can result in the loss of reputation and customers of the liable organization. Thereby, this kind of information is *sensitive*. Hence, securing such *sensitive* information, both during storage and in-transit, is of serious concern to individuals as well as organizations.

In general, organizations take efforts to safeguard the *sensitive* information by deploying access controls, security checks and periodic backups. However, these mechanisms can safeguard only against specific internal threats, such as stealing and/or destruction of information, by their own employees. But, securing the information from cyber criminals is a challenging task.

To protect the *sensitive* information, security methodologies like watermarking, cryptography and steganography are used. Watermarking prevents from illicit claim of ownership of the information, by secretly hiding the ownership details in it. However, it does not prevent the reproduction of the content.

On the other hand, cryptography prevents such reproduction by changing the appearance of the content through encryption. Besides confidentiality, cryptography also provides integrity, authentication and non-repudiation. But, it fails to provide secrecy. Hence, during the exchange of information, any third party who is sniffing



the communication channel can identify its presence. Once identified, attackers might be able to use suitable state-of-the-art techniques to extract the encoded information or save them for later use. This can be possible due to the various vulnerabilities such as export grade encryption, man-in-the-middle attacks, default or weak passwords, insecure configurations and advances in cryptanalysis.

Steganography averts this kind of detection by performing the communication in a stealthy manner. It hides the secret information inside an innocent looking cover medium, by making unnoticeable modifications or distortions, preventing it from raising any suspicion. Due to this characteristic, lately security experts advice the use of steganography and cryptography in combination. The idea, here, is to encrypt the information first and, then, send it through steganographic means.

Digital steganography (hereafter referred to as steganography) can be classified based on the type of cover medium used. The cover medium used can be a text (plain text as well as word processor documents), image, audio, video, network packet, etc. Of these various types, text steganography is least preferred as the: (i) amount of redundant information present in a text document is less; (ii) structural and visual appearance of a text document are directly related.

However, text steganography cannot be avoided completely. According to a 2009 report, nearly 80% of organizations use text documents (Microsoft Word documents) for collaboration purpose. Hence in such environment, depending on other media types, that are scarcely used, to perform the covert communication is not preferable. This is because, the: (i) transmission of other media types can make the communication suspicious and can lead to further analysis; (ii) low voluminous traffic of other media types can make it practical for an adversary to investigate them thoroughly. In addition, text documents require low bandwidth

during communication, due to the involvement of smaller file sizes. Considering these facts, this dissertation aims to analyze the existing text steganographic methods and address their drawbacks through innovative solutions.

A detailed literature survey of the existing techniques concluded that the embedding capacity and the number of bits embedded per distortion are inversely proportional, except for the techniques that generate the stego-work directly. Character-level embedding technique (CLET), a variant of text steganography, directly marks the identical character in the cover medium, sequentially, to embed the secret characters. It marks the character by altering the font, font style, position, misspelling, etc., and thereby, embeds 8-bits per distortion. However, it can embed a secret character only if the corresponding character is present in the cover medium. As the occurrence frequency of English alphabets is not uniform in a typical document, the overall embedding capacity of this technique is low.

On the other hand, the existing bit-level embedding technique, UniSpaCh, with highest embedding capacity, embeds the secrets by injecting Unicode space characters in the white spaces. On an average, it embeds 1.046-bits/cover-character with approximately 2-bits/distortion. Hence a large number of distortions are required to embed the secret. Apart from the methods that generate the stego-work directly, from a given secret, no existing method was found to achieve both high number of bits per distortion as well as a high embedding capacity.

In this study, an attempt has been made to address this shortcoming by presenting three novel techniques. These techniques embeds text, image or binary data inside the font attributes of word processor document. The present thesis is organized into seven chapters and the contents of each chapter are summarized below.

## **Chapter 1 — Introduction**

Chapter 1 presents a brief introduction on the importance of data security. It discusses the existing security methodologies in detail, and the need for steganography in the current digital scenario. It, also, describes the desirable characteristics of a digital steganographic algorithm and our motivation for the choice of text steganography along with the challenges involved in it.

## **Chapter 2 — Literature Survey**

Chapter 2 describes the existing text steganographic techniques, in detail, and classifies them into three broad categories, namely character-level, bit-level and mixed-type embedding, based on their nature of embedding. Character-level embedding technique is further classified into two categories as Cover\_Document\_Required (CDR) and Cover\_Document\_Not\_Required (CDNR). CDR embeds a secret character by directly marking the respective character in the cover medium. CDNR directly generates the stego work, based on the secret characters, without using any cover medium. Bit-level embedding technique considers the secret information as a binary string and the embedding is carried out accordingly. Mixed-type embedding technique is a mixture of the above two techniques. It considers the secret information as a binary string and splits them into groups of 2 or 4-bits each. Each group is, then, mapped to one or more alphabet(s) and consequently the bits are embedded in the mapped alphabet(s) of the cover medium. This chapter discusses the advantages and disadvantages of the above methods in detail. Also, it compares them based on their embedding capacity and the number of bits embedded per distortion.

### Chapter 3 — Exploring the Font Attributes of Word Processor Documents

Chapter 3 discusses the font attributes of various word processor documents viz. Microsoft Word, LibreOffice Writer, OpenOffice Writer and WordPerfect. It analyses the font attributes from a steganographic perspective, and categorizes them based on their usability and imperceptibility. In addition, it presents the various ways of embedding the secret using these attributes, viz. restricting their values within a particular range or masking the effect of an attribute by the effect of another attribute. A comparison of the selected attributes, based on their embedding capacity, usability and complexity of the extraction process, revealed that the word processor Microsoft Word suits best for steganographic purpose. Additionally, the comparison illustrated that the attributes *color*, *spacing* and  *Kerning* stands best among the rest.

***Color:*** This attribute specifies the color of a character and it is represented by a 24-bit value using the format (R, G, B). The least significant 1 or 2-bits of each R, G and B can be modified without creating any visual difference.

***Spacing:*** This attribute alters the spacing (expand or condense) between two characters. Since the spacing of characters are not uniform in a justified text, modifying the document using this attribute will go unnoticed.

***Kerning:*** This attribute alters the spacing between the overlapping character pairs such as AV, WA, etc. It takes the size of the font as input, known as the kerning font size, which can be between 1 and 1638 points. The effect is produced only when the specified kerning font size is lesser than or equal to the font size of the character. Else, applying *kerning* produces no effect and hence goes unnoticed.

## Chapter 4 — Embedding Text

Chapter 4 proposes a novel character-level embedding technique (Method-A) which embeds secret text, that contains English alphabets, dot and space characters (ADS), with higher number of bits per distortion as well as high embedding capacity. To achieve this, the method generates a set of 28 strings, using the occurrence frequency of ADS characters, known as Frequency Normalization Set (FNS). These strings have the following properties: (i) Each string contains only ADS characters; (ii) The length of a string,  $L$ , is seven with seven positions  $\{0, 1, 2, 3, 4, 5, 6\}$ ; (iii) A character occurs only once in a string; (iv) A character occurs only once in a given position in the whole FNS. That is, no column-wise repetitions; (v) The cumulative frequency of characters of each string is  $\approx 25$ .

The generated FNS is then injective mapped to the 28 ADS characters. This mapping is called as Character & String Mapping (CSM).

The secret characters are embedded, serially, at the first occurrence of a character in the corresponding mapped string. This allows a low occurring character to get embedded in several cover characters, and thereby boosts its embedding probability in addition to achieving uniformity. A font attribute called *spacing* is used to mark the seven respective positions of characters in a string. The distorted character and the value of *spacing* represent the hidden character. An investigation on the embedding capacity revealed that an average of 2.22-bits/cover-character with 8-bits/distortion was attained by the proposed method.

## Chapter 5 — Embedding Binary Data

Chapter 5 extends the method explained in Chapter 4, Method-A, into a mixed-type embedding technique (Method-B), to embed binary information. To achieve this, the



method first converts the secret information into a binary stream and segments it into quadruples. It then maps the sixteen possible combinations of a quadruple to the 28 strings of FNS, using which it converts the binary stream into a character stream. These characters are then embedded as explained in Chapter 4.

Though the conversion procedure affected the embedding capacity and bits/distortion of the former method, it still attained an average embedding capacity of approximately 1.71-bits/cover-character with 4-bits/distortion which is better than the existing methods.

However, a case study on the nuclear power related images, like engineering drawings, roadmaps, graphs, etc., revealed that, for certain images, the required number of pages in the cover document are still larger than the average size of an academic book. It is noted that this huge size requirement is due to embedding the unnecessary information of images, like color, line thickness, etc., which is present in the pixel representation.

## **Chapter 6 — Embedding Image**

Chapter 6 explores the possibilities to reduce the size of above-mentioned images without losing the necessary information. The various representations of images namely Encapsulated PostScript (EPS), Scalable Vector Graphics (SVG) and Compressed SVG (SVGZ) achieve the same goal. It was observed that, of the available formats, SVGZ attained the least size of a given image. Hence, when the same is embedded using the method proposed in Chapter 5 (Method-B), it considerably reduced the number of pages required to embed them. However, the method failed to extract the image in the case of any data corruption.

Hence, Chapter 6 proposes a novel bit-level embedding method (Method-C) that embeds specific category of images, like engineering drawings, roadmaps, graphs, etc., in a unique way reflecting the structure of the hidden image. The method, first, converts the various elements of a given image into custom defined *codes*. This is done with respect to the grid lines that are drawn over the image. The intersection of two grid lines is called as a control point. *Codes* represent the layout of the image, based on the way the layout traverses through the control points.

The conversion procedure begins by choosing a specific control point called *origin*, to trace the layout. The angle of the tracing line drawn from the *origin* to the next control point defines the *code*. The procedure repeats the process and traces the entire image, to convert it into *codes*. Whenever the conversion procedure encounters a text message, near a control point, it places an appropriate marker in the corresponding *code*, and writes the encountered message into a separate file. Hence, the size of the final *codes* depends only on the number of grid lines drawn but not on the resolution of the image. Thereby, the procedure considerably reduces the number of bits that are required to represent an image.

The proposed method (Method-C) uses the font attributes *color*, *spacing* and *kerning* to embed the *codes*. The embedding procedure starts by choosing a cover character, corresponding to the *origin*, and embeds the *code*. It selects the next character, based on the last embedded *code*, in a manner similar to that of traversing the image. This process achieves the original structure of image to be embedded inside the text document. The details related to *origin* and resolutions of the grid are communicated to the recipient. To extract the *codes*, the extraction algorithm starts from the character, corresponding to *origin*, and traces the embedded characters similarly. The extracted *codes* are then used to reproduce the original image.

In the case of any transmission errors, the extraction algorithm searches the potential characters to continue the traversal and thus, supports error handling mechanisms. The text messages are embedded using UniSpaCh and are extracted accordingly. From the experiments conducted, a high embedding capacity was observed making it suitable for low bandwidth environments.

## **Chapter 7 — Summary and Scope for Future Investigations**

The highlights of this dissertation and the scope for future work are summarized in Chapter 7. The highlights are: 1) The steganographic usage of word processor documents was investigated thoroughly; 2) The best suitable font attributes to embed the secrets were identified; 3) The possibility to normalize the embedding probability of secret characters in CLET algorithms, irrespective of their occurrence, was reported for the first time; 4) A CLET with high embedding capacity and bits/distortion was successfully designed, developed and tested; 5) The requirement of larger cover work to embed the secret information, like image, was reported with a case study; 6) The custom defined format to represent an image with lesser number of bits was illustrated; 7) For the first time, the procedure of embedding an image along with its structure was demonstrated; and 8) The procedure of embedding an image with error handling mechanisms was demonstrated. The future work can focus to explore the ways to embed secrets inside other document formats like Portable Document Format (PDF).

**LIST OF FIGURES**

<b>Figure No.</b>	<b>Figure Caption</b>	<b>Page No.</b>
1.1	Types of information security measures	3
1.2	Procedure of watermarking	4
1.3	Procedure of cryptography ensuring confidentiality	4
1.4	Procedure of steganography	5
1.5	Types of steganography	7
2.1	Classification of the existing text steganographic techniques	16
2.2	Sample input and output of missing letter puzzle technique	21
2.3	Sample input and output of hiding data in wordlist technique	22
2.4	Example for word shifting	26
2.5	Illustration of the white spacing's in text document [66]	26
2.6	Example for end-of-line spacing [76]: (a) Ordinary text; (b) White space encoded text	28
2.7	Example for UniSpaCh technique: (a) Unicode space characters (color-coded for understanding purpose) [66]; (b) Size of Unicode and normal space characters [66]	29
2.8	Vertical displacement of Dot in the Persian character Noon [105]	30
2.9	Exploiting the structure of characters: (a) Basic components of Chinese, Japanese and Korean characters [84]; (b) Representation of characters using the basic components [41]	31

2.10	Exploitation of the Arabic or Urdu characters: (a) Diacritics of Arabic language [107]; (b) Representation of Araabs; (c) Usage of the regular and reverse Fatha [94]	32
2.11	Sample output of the “Change Tracking” technique [83]	33
3.1	WordPerfect revealing the formatting information	40
3.2	Exploitation of the Animation attribute	44
3.3	Exploitation of the EmphasisMark attribute	45
3.4	Exploitation of the Color attribute	47
3.5	Exploitation of the Emboss and Engrave attributes	48
3.6	Exploitation of the Shading attribute in WordPerfect. Middle character of each word has a default shading value of 75%, whereas the left and right characters have varying values which are mentioned above each character (in %)	55
3.7	Exploitation of the StylisticSet attribute	59
3.8	Classification of the font attributes of word processors. * – represents the case that the attribute is applied on space character alone; † – represents the case that care has been taken to separate an identical modified and unmodified character; MS – Microsoft Word; LO – LibreOffice; OF – OpenOffice; WP – WordPerfect	62
4.1	Sample output of Method-A	82
4.2	English secret in English cover document. CW1 – cover document; Sec1 – secret message; Stego – stego document	90



4.3	Random secret in English cover document. CW1 – cover document; Sec2 – secret message; Stego – stego document	90
4.4	Random secret in random cover document. CW3 – cover document; Sec2 – secret message; Stego – stego document	91
4.5	Method-A combined with Format Preserving Encryption system	92
5.1	Schematic diagram of Method-B. The dotted lines represent the modules that are introduced in Method-B and bold line represents the modified module of Method-A	98
5.2	Sample Character & Bit Mapping using the Character & String Mapping provided in Table 4.4. “□” – represents “Space” and “.” – represents “Dot”; ADS – English alphabets, Dot and Space	99
5.3	Sample tested images: (a) Road map from kalpakkam to anupuram [135]; (b) Graph [136]; (c) Electronic circuit diagram [137]; (d) Civil drawing of stairs [138]; (e) Boiling water reactor [139]; (f) Schematic diagram of thermal power plant [140]; (g) Nuclear power plant steam generation [141]; (h) Reactor flow sheet [142]; (i) Reactor core [143]	111
6.1	Vector image formats. * – represents the compressed version of SVG format	118
6.2	Sample SVGZ file and corresponding image: (a) SVGZ file; (b) Generated image	121
6.3	Sample SVGZ File (corrupted) and corresponding image: (a) Corrupted SVGZ file; (b) Generated image	122
6.4	Sample image: (a) Without grid; (b) With grid	123
6.5	Picture depicting the various elements of an image	124

6.6	Picture depicting the three possible branches of an image	124
6.7	Sample image with additional grid line: (a) Without grid; (b) With grid	128
6.8	Sample image along with the corresponding code: (a) Without grid; (b) With grid	128
6.9	Modules of Method-C. $N_{HGL}$ – Number of Horizontal Grid Line; $N_{VGL}$ – Number of Vertical Grid Line	131
6.10 (A)	Extraction algorithm: flowchart 1	137
6.10 (B)	Extraction algorithm: flowchart 2	138
6.10 (C)	Extraction algorithm: flowchart 3	139
6.11 (A)	Generated images of SVGZ and custom formats — part 1: (a) Roadmap from kalpakkam to anupuram; (b) Graph; (c) Electronic circuit diagram	142
6.11 (B)	Generated images of SVGZ and custom formats — part 2: (d) Civil drawing of stairs; (e) Boiling water reactor; (f) Schematic diagram of thermal power plant	143
6.11 (C)	Generated images of SVGZ and custom formats — part 3: (g) Nuclear power plant steam generation; (h) Reactor flow sheet; (i) Reactor core	144
6.12	Sample output of Method-C. *Stego characters are highlighted for understanding purpose	147
6.13 (A)	Illustration of handling transmission error: (a) Original image; (b) Transmitted image	149
6.13 (B)	Illustration of handling transmission error: (c) Stego characters at sender side (highlighted for understanding purpose);	150

(d) Identified stego characters after the occurrence of error (15% of stego characters have been distorted); (e) Applied color patterns for the stego characters in Fig. 6.13 (B) (d) during extraction; (f) Extracted image from Fig. 6.13 (B) (d)

A.1	Flowchart to generate Frequency Normalization Set (FNS)	176
B.1	Boiling water reactor [139]	181

## LIST OF TABLES

Table No.	Table Caption	Page No.
1.1	Various cover types and their embedding strategies	9
2.1	Occurrence frequencies of alphabets in English text [38]	19
2.2	Sample words and the respective synonyms	23
2.3	Sample United States and United Kingdom spellings [97]	24
2.4	Number of bit and space character combinations	28
2.5	Reflection symmetry properties of English alphabets [86]	34
2.6	Comparison of the existing techniques	36
3.1	Details of word processors	40
3.2 (A)	List of attributes in word processors: part 1	41
3.2 (B)	List of attributes in word processors: part 2	42
3.3	Exploitation of the Spacing attribute	57
3.4 (A)	Comparison of the <i>average</i> and <i>high imperceptible</i> attributes: part 1	65
3.4 (B)	Comparison of the <i>average</i> and <i>high imperceptible</i> attributes: part 2	66
4.1	Occurrence frequencies of ADS characters in English text	73
4.2	Respective NCC and P values	75
4.3	Sample Frequency Normalization Set	77

4.4	Sample Character & String Mapping	78
4.5	Results of embedding the secrets in cover document	84
4.6	Uniformity in embedding probability	85
4.7	Comparison of Method-A with existing techniques	86
4.8	Spacing values of the identified stego characters	89
4.9	Sample Character & String Mapping for case-sensitive messages	93
5.1	Mapping of ADS characters and the nibbles (Character & Bit Mapping)	100
5.2	Results of embedding the secrets in cover document	106
5.3	Uniformity in embedding probability at nibble-level	107
5.4	Uniformity in embedding probability at character-level	108
5.5	Distribution of nibbles among the possible tuples	110
5.6	Details of images given in Fig. 5.3	112
5.7	Number of cover characters required by UniSpaCh and Method-B	112
6.1	Software that generate vector file format	118
6.2	Sizes of generated vector images	119
6.3	Results of embedding raster and vector images by Method-B	119
6.4	Used notations and their descriptions	124
6.5	Elements of an image and their respective codes	126

6.6	Possible angles and their respective $H_4$	126
6.7	Kerning value	132
6.8	Flags and their respective spacing values	133
6.9	Selecting one character from eight neighbors based on $H_4$	133
6.10	Sizes of images in custom and SVGZ formats	141
6.11	Results of embedding the custom format	146
6.12	Comparison of Method-B and Method-C	148
6.13	Used color patterns to color the currently selected character	149

## **LIST OF ABBREVIATIONS AND SYMBOLS**

PAN	Permanent Account Number
U.S. or USA	United States of America
JPG	Joint Photographic Experts Group
PNG	Portable Network Graphics
TCP/IP	Transmission Control Protocol/Internet Protocol
CLET	Character-level embedding technique
CDR	Cover_Document_Required
CDNR	Cover_Document_Not_Required
EoL	End-of-Line Spacing
UniSpaCh	Unicode Space Characters
CJK	Chinese, Japanese and Korean
L-R	Left-Right
U-D	Up-Down
ASCII	American Standard Code for Information Interchange
MS	Microsoft
LO	LibreOffice
OF	OpenOffice
WP	WordPerfect
R, G, B or RGB	Red, Green, Blue (color)
ADS	English alphabets, Dot and Space
NCC	Number of characters cumulated
FNS	Frequency Normalization Set
CSM	Character & String Mapping
EoS	End-of-Secret
No.	Number

FPE	Format Preserving Encryption
BCC	Binary to Character Converter
CBM	Character & Bit Mapping
KB	Kilebyte
GIF	Graphics Interchange Format
AI	Adobe Illustrator
EPS	Encapsulated PostScript
PDF	Portable Document Format
SVG	Scalable Vector Graphics
SVGZ	Scalable Vector Graphics Compressed
DP	Directional point
DCP	Direction changing point
TP	Temporary point
SP	Split point
FPoB	First point of a branch
LPoB	Last point of a branch
OPB	The only point of a branch
$N_{HGL}$	Number of horizontal grid line
$N_{VGL}$	Number of vertical grid line
NoB	Number of branch
$S_p$	Single page
$H_4$	Hexadecimal value (Nibble)



## **List of Terminologies**

Cover work/document	Cover medium that is used to carry the secret
Cover character	Character in a cover document
Embedding	Process of hiding a secret
Embedding capacity	Maximum amount of information that can be embedded inside a chosen cover document
Embedding space	Place in which the secret can be embedded
Distortion	Modification performed to embed a secret
Bits/distortion or bits per distortion	Number of bits that can be embedded in a distortion
Stego character	Cover character that has undergone modification
Non-stego character	Cover character that has not undergone modification
Stego work/document	Cover medium (resultant file) after embedding the secret
Extraction	Process of extracting the secret

## INTRODUCTION

---

*This chapter furnishes a detailed introduction on the importance of data security both during storage and in-transit. A detailed discussion on existing security methodologies and the need of steganography in the current digital scenario has been provided. The desirable characteristics of digital steganographic algorithms and motivation for the choice of text steganography along with the challenges involved in it has, also, been described.*

### 1.1 Introduction

In the modern era, predominant amount of information is generated, stored and shared in the digital form [1]. This is mainly due to the ease with which it can be handled and the flexibility it offers in terms of efficient storage, distribution [2], retrieval [3], etc. Information in the digital form is also environment-friendly [3].

Information can be personal as well as organizational. Personal data can be name, date of birth, marital status, medical history, passport details, permanent account number (PAN), email ids, contact numbers, debit or credit card numbers, income details, assets, finger prints, etc. [4-6]. Organizational data can be employee details, medical records of customers, minutes of meetings, tender quotations, architectural blueprints, engineering drawings, maps, graphs, financial records, etc. [7]. Also, several organizations have technical notes and documents that need to be shared among the colleagues associated with the activity.

These types of information are *sensitive* and the access of such information by unauthorized persons can be harmful to individuals as well as organizations [8].

Malicious third parties can misuse them by:

- selling the phone numbers or contact details to advertising agencies [9]
- filing false income returns, selling properties, applying loans, etc. [10,11]
- making fraudulent transactions using credit or debit card details [12]
- selling the medical records of patients to hospitals, health insurance and pharmaceutical companies [13]
- selling the chemical composition of explosives or locations of army camps to anti-social elements [14,15]

Hence, securing sensitive data is of utmost priority to organizations as well as individuals [16]. Typically, organizations take extensive efforts to protect them from both internal and external threats. Internal threats include destroying, stealing or misusing of organization's data by their own employees [17]. Organizations deploy strict access controls [18], security checks and periodic backups to minimize the internal threats.

However, most of the organizations are geographically distributed and are forced to be inter-connected using public networks like the internet [19]. Hence, securing information from the external threats (active or passive) of cyber criminals is a complex task. Through regular patching and proper training, active threats like firewall penetration, phishing [20], botnets [21], etc., can be mitigated to an extent. But preventing the networks from passive threat, like sniffing [22], is a difficult task even to organizations that use leased line or virtual private networks.

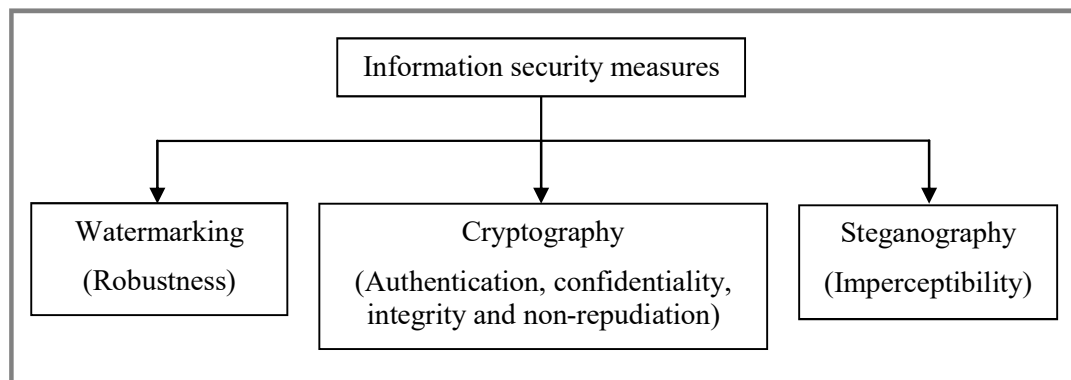
Regardless of the communication line used, cyber criminals sniff the network and collect the sensitive information. This information can later be exploited due to

vulnerabilities such as export grade encryptions [23], man-in-the-middle attacks [24], default or weak passwords [25,26], bad configurations and advances in cryptanalysis [27,28]. A number of such incidents were reported in the past and are growing in number [29-34]. Therefore, securing the digital information, in-transit, is extremely important.

Security of data, in-transit, must ensure the prevention from:

- (i) illicit copying and claim of ownership
- (ii) illicit extraction and reproduction
- (iii) detection of communication and the communicating parties

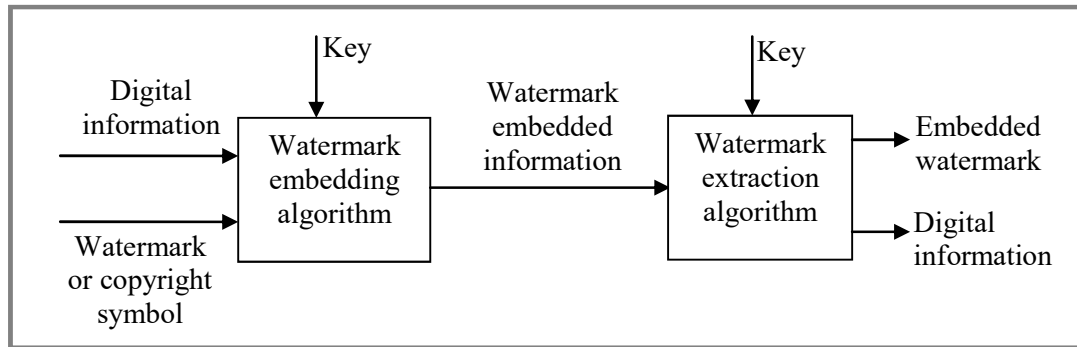
These goals are achieved through information security methodologies like watermarking, cryptography and steganography respectively [35] (refer Fig. 1.1) which are as follows.



**Figure 1.1** Types of information security measures

## 1.2 Watermarking

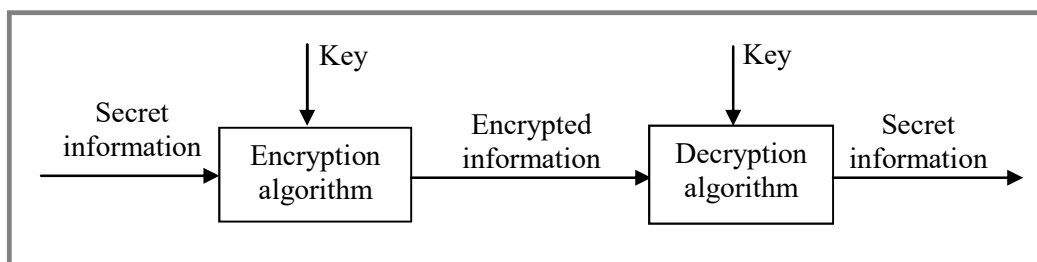
Watermarking is typically used to identify the owner of the information and protect copyright [36]. It injects or hides the trademark or copyright symbol in the information without compromising its quality (refer Fig. 1.2). Such injected trademark must be robust against tampering and should not be able to be removed without destroying a substantial quality of information [37].



**Figure 1.2** Procedure of watermarking

### 1.3 Cryptography

Cryptography means “secret writing” [38]. It hides the meaning of the information by encoding it using secret key. Hence a party possessing the key, can only extract the information (refer Fig. 1.3). This protects the information from the illicit extraction and reproduction (confidentiality). Besides this, cryptography also provides authentication (identify the sender), integrity (identify data modification) and non-repudiation (denial of sending) [39].

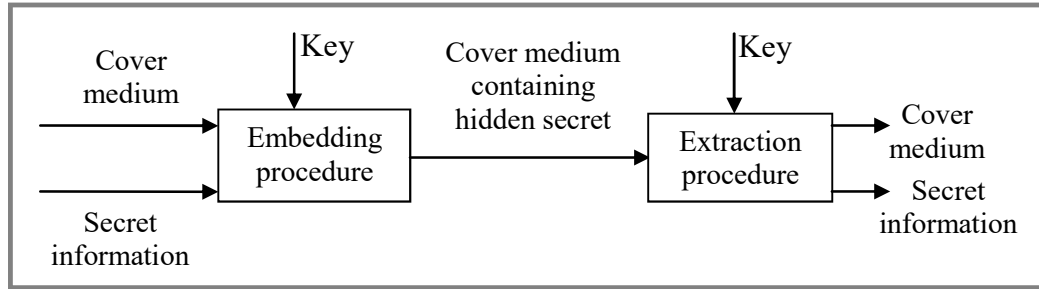


**Figure 1.3** Procedure of cryptography ensuring confidentiality

### 1.4 Steganography

Steganography means “covered writing” [40]. Steganography hides secret information covertly inside an innocent looking cover medium by making unnoticeable modifications in the cover medium [41] (refer Fig. 1.4). Therefore, communicating using steganographic techniques hides the information and in certain

cases the transmission of information and communicating parties altogether (explained later in Section 1.6).



**Figure 1.4** Procedure of steganography

### 1.5 Need for Steganography in the Current Scenario

Watermarking protects the information from illicit claim of ownership [42], but does not prevent anyone from reproducing or using the content [43]. Only, cryptographic and steganographic techniques achieve this by changing the appearance (encoding) or by hiding the existence (hidden communication) of information respectively.

Cryptographic techniques succeed in providing *confidentiality*, *integrity*, *non-repudiation* and *authentication* of the information [38]. But, it sends the encrypted information in plain sight making it available to preying eyes [41,44]. Latest reports emphasize the sniffing of public networks by cyber criminals as well as government agencies [45,46], for such encrypted information. In addition, in the past, various law-enforcing authorities forced the product developers to reduce the strength of cryptographic techniques (by incorporating export-grade encryption techniques [23], backdoors [47], weak primes [46], elliptical curves, etc.), so that they can decrypt the sniffed information at ease [48]. However, in some cases, the captured information is just stored, till an efficient cryptanalysis method on the used algorithm or advanced computation mechanisms like quantum computing [49,50] becomes

available. Hence, relying on encryption techniques alone to send sensitive information is, clearly, not sufficient anymore. This makes the secrecy of information during communication a necessity one.

Considering this, security experts suggest the incorporation of secrecy along with security [45,51]. The idea here is to protect the information, first, by using cryptographic techniques and communicate them by means of steganographic techniques [52]. This brings steganography into the digital communication field, for enhancing the security of information during transit.

## **1.6 Recent Trends in Steganography**

Though cryptography is used by the mainstream, due to the Internet and World Wide Web, steganography is still practiced by people who wish to circumvent the spying activities [53]. After weeks of interviews with U.S. officials and experts, Jack Kelley of USA Today wrote an article headlined ‘Terrorist instructions hidden online’ [54]. Following this, the 9/11 attacks created awareness across nations that terrorists may use steganography for secretly communicating their targets [55]. It is believed that the target locations along with the relevant information are embedded in images and posted in websites like eBay, Reddit, etc. [56]. Thereby, to a casual observer, this post looks like a genuine user who is about to sell a product but only the intended recipient(s) gets the original message. This method of communication not only hides the message but also the intended recipient(s).

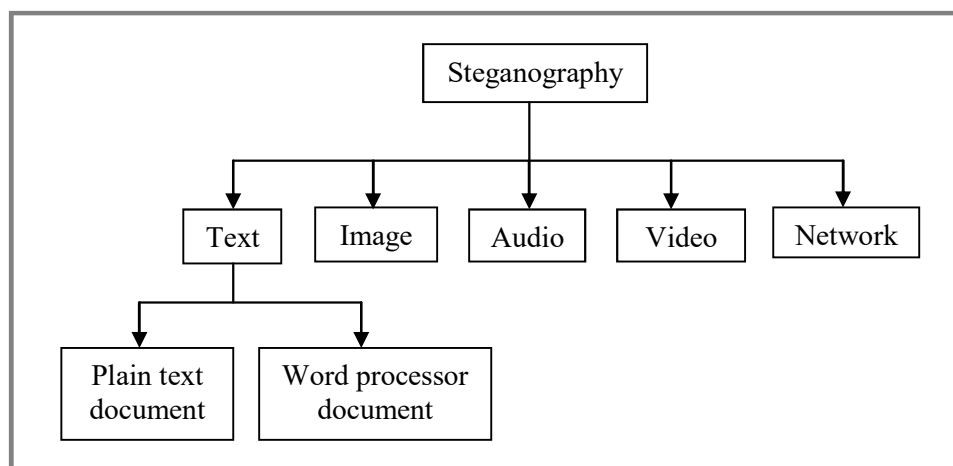
More recently, malicious code writers started to use steganography either to collect sensitive information from a malware infected machine [57] or to deploy malwares. Zeus or Zbot malware (first identified in 2007) appends its encrypted configuration file to a JPG (Joint Photographic Experts Group) file [58].

Duqu malware (discovered in 2011) collects sensitive information from an infected machine, encrypts and sends it by embedding it in a JPG file [59]. Stegoloader malware (emerged in 2015) completely hides its malicious code within a PNG (Portable Network Graphics) file [60].

On the positive side, steganography is used to prevent users from phishing attacks in online banking scenario [61]. For this purpose, a browser plug-in that can hide secret messages inside images (logos) and validate the same has been developed. When the resultant stego image (image with hidden secret) is placed in the corresponding bank's website, the user's browser can validate the website using the developed plug-in [61]. These clearly show the growing use of steganography in the digital communication, with a new paradigm.

### 1.7 Desired Characteristics of Digital Steganography

Digital steganography (hereafter referred as steganography) can be classified based on the type of cover medium used [62] (refer Fig. 1.5). The cover medium used can be a text message or document, image, audio, video, network packet, etc. [36].



**Figure 1.5** Types of steganography



Choosing an appropriate cover medium in a given environment is the first precautionary measure that the designer of a steganographic algorithm must look into [63]. This is because an attacker can sniff the traffic flowing in the communication channel and identify the communicating parties, first. Later, he/she can qualify the association between them based on the contents of the message. Any questionable or non-justifiable exchange can lead to the thorough inspection of transmitted data. This is known as traffic analysis attack [62]. Hence, to counter the attack, a steganographic algorithm must use a cover medium that is being exchanged in abundance by the communicating parties. This makes the communication look legitimate and also make the life of an adversary tougher as thorough inspection of such exchange is computationally intensive.

Second, as the secrets are embedded by making modifications or distortions in the cover medium, an algorithm must focus to make the modifications in an unsuspecting manner [63,64]. To achieve this, modern techniques either rely on the redundant information in cover medium or the properties of cover medium that human perceptual system fails to identify. Details of some of the existing methods are given in Table 1.1.

Third, a good steganographic algorithm must aim to minimize the number of distortions that are required to embed a secret [63]. That is, the number of secret bits embedded per distortion must be maximized.

Fourth, an algorithm must utilize the available embedding space efficiently and try to reduce the size of cover medium required to embed the secret [65]. This reduces the load on the network and also avoids suspicion.

Fifth, the embedding method should not be cover medium dependent. That is, designing a steganographic method which is specific to a cover document restricts the

method to that particular environment. For example, designing a text steganographic method that can be applied only to Chinese characters, make it to be non-usable in other languages.

**Table 1.1** Various cover types and their embedding strategies

Cover type	Embedding strategy	Exploited parameter	Reference
Text	Embeds the secrets in white spaces	Human visual system	[66]
	Embeds the secrets by exploiting the structure of characters		[41]
Image	Embeds the secrets in the least significant bit of color	Redundant information	[67]
Audio	Embeds the secrets as noise or in frequencies beyond human audible range	Human auditory system	[68]
Video	Embeds the secrets using both image and audio steganographic techniques	Both human auditory and visual systems	[69,70]
Network	Embeds the secrets in the unused or insignificant bits of TCP/IP (Transmission Control Protocol/Internet Protocol) packet	Redundant information	[71]

Sixth, the method must generate a meaningful stego document that is suitable to any given scenario [72,73]. For example, designing a steganographic method which generates a puzzle or unconnected contents (words or sentences) as a stego document makes it unsuitable for organizations.

## 1.8 Discussion

Amongst the various steganographic types, text steganography is hard to manage [74,75]. This is because, the:

- (i) amount of redundant information present is relatively less [73,76,77]. For example, when a character “r” is represented as text, the whole character has only 24-bits to represent its color. Whereas in a 24-bit image, each pixel is represented with 24-bits leaving a lot of redundant information

- (ii) structural and visual appearance of text document is directly related [78]. For example, altering the least significant bits of two neighbor pixels that has similar values won't draw any attraction. Whereas, altering the font size of one of the two consecutive similar letters in a word, say **balloon**, will create suspicion
- (iii) number of pages of a normal academic text document is  $\approx 250$  (average number of words per page is 300 [79] and average number of words in an academic book is 70,000 [80]). Hence requiring a cover document that is larger than the available size is not easy to accommodate and the usage of which can, also, raise suspicion. This is not the case for the cover types like image, audio and video

Due to the above-mentioned factors, text steganography is not a preferred method for steganographers [77].

## 1.9 Motivation for the Thesis

Though employing text steganography involves complications, one cannot completely avoid it. This is because many organizations are expected to exchange documents more often than other media types such as image, audio, video, etc. Hence in such environment depending on other media types, that are scarcely used, to perform covert communication will not be beneficial. Besides, the involvement of smaller file size makes text document to require low bandwidth during communication [41,81].

Further exploration demonstrated that the existing techniques consider secret message either as characters [78] or bits [66]. Techniques that consider the secret message as characters have the advantage of embedding them directly and thereby

achieve 8-bits/distortion (each character in a file is represented by single byte [82]). But these techniques suffer from either wasting the available embedding space or generating meaningless stego documents. Whereas, techniques that consider the secret message as bits manage to utilize the embedding space efficiently, but suffers from low bits/distortion.

This shows that no existing method, that generates a meaningful stego document, utilizes the available embedding space efficiently and achieves high bits/distortion. As a result, embedding secrets in smaller size text document is a challenging task.

Despite this limitation, the advantage of using text document for steganography and the advantage associated in transmitting it motivated us to take up the problem of developing newer text steganographic methods. This forms the basis of work to be carried out in the thesis.

### **1.10 Objectives of the Thesis**

Text steganography is the focus of the present study and objective of the thesis is:

- Identify the possibilities to achieve maximum number of bits per distortion
- Utilize the available embedding space in efficient manner
- Design and develop a method that achieves both high embedding capacity and bits/distortion, while maintaining meaningful stego document
- Design and develop a method that embeds larger message, like multimedia data, in smaller size documents

### **1.11 Contributions of the Thesis**

A summary of the contributions made, based on the work carried out, in the thesis is given below:

- (i) A brief discussion on the font attributes of various word processors and the possibilities to employ them for steganography are explained and demonstrated
- (ii) Novel techniques that embed text content and binary data, with high embedding capacity and bits/distortion, are designed and developed
- (iii) The requirement of larger cover document to embed multimedia data, like image, audio, video, etc., is highlighted with a case study. The possibility to reduce the size requirement through vector formats is demonstrated
- (iv) A custom defined format to represent an image with lesser number of bits is described and developed
- (v) A novel method that embeds images along with their structure, to support error correction, is designed and developed

### **1.12 Organization of the Thesis**

Remaining part of the thesis is organized into six chapters. The details of the content in each chapter are:

- Chapter 2 presents a survey on existing text steganographic techniques
- Chapter 3 presents a brief description on the font attributes of various word processor documents and analyses them from a steganographic perspective
- Chapter 4 presents a novel character-level embedding technique that was developed to embed text content inside Microsoft Word documents

- Chapter 5 presents the extension of the developed character-level technique that embeds binary data like image, audio, video, etc.
- Chapter 6 presents a novel method that was developed to represent images in reduced size and embed them with in-built error handling capabilities
- Chapter 7 provides a brief summary of the investigations and conclusion made towards the thesis

### LITERATURE SURVEY

---

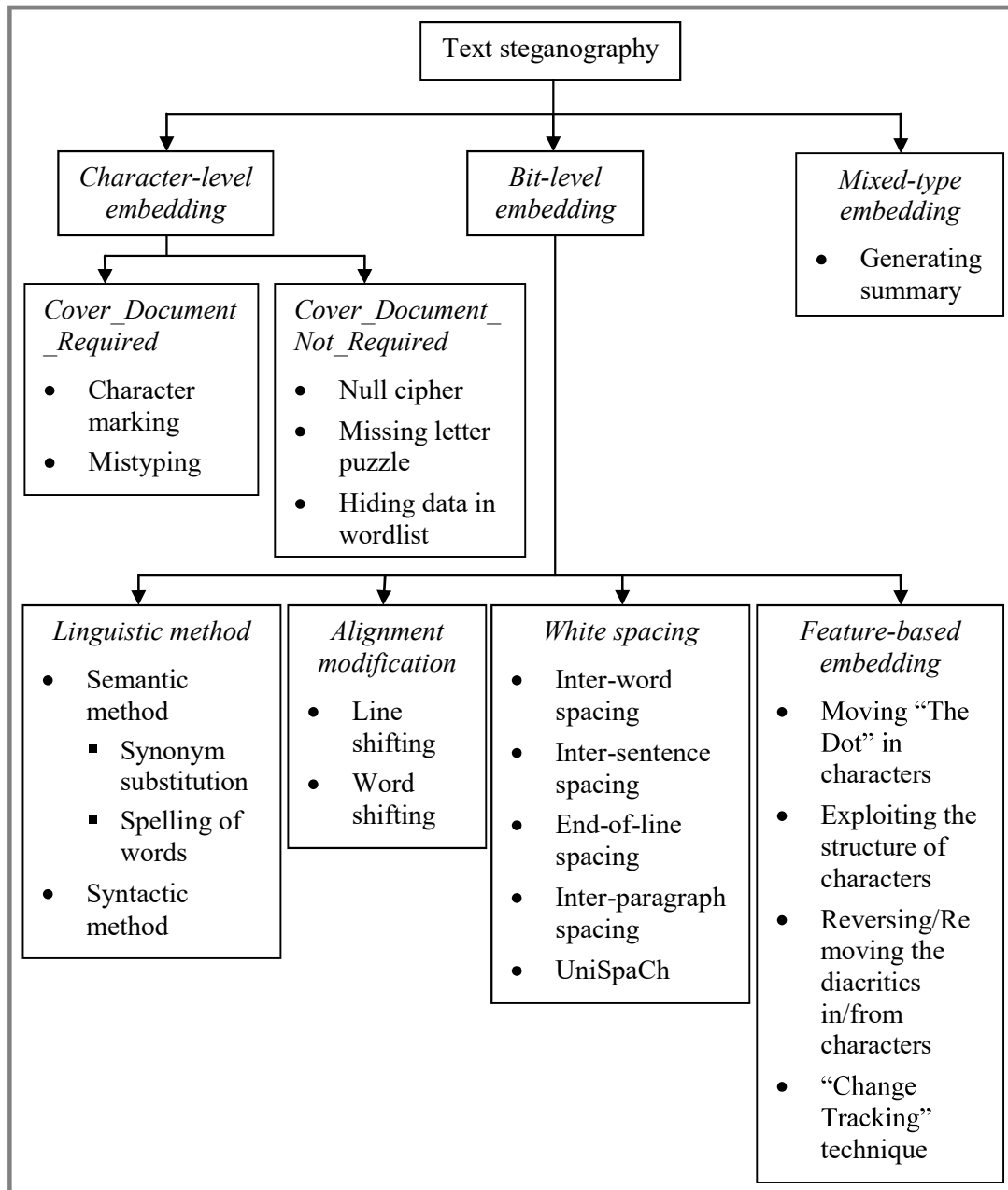
*In this chapter, the text steganographic techniques that are available in the literature are summarized. The methods are categorized based on the nature of embedding. It also discusses the merits and demerits of each of the methods in some detail. A comparison on the embedding capacity and number of bits embedded per distortion, for each of the methods, is provided.*

#### 2.1 Introduction

As discussed in the introduction (Chapter 1), embedding secret information inside text documents is considered to be harder, when compared with other cover types [74,75]. Existing techniques mostly rely on the properties of cover document [83], properties of characters [41,84], properties of languages [85], etc., to embed the secrets. As a result, the embedding methodologies of these techniques are not applicable to all the characters, words or languages. Hence, the overall embedding capacity of such methods is relatively less and these aspects are discussed below in some detail.

#### 2.2 Classification of Text Steganography

Embedding strategy of the existing techniques can be broadly classified into three categories as character-level, bit-level and mixed-type embedding techniques (details are shown in Fig. 2.1). First category, character-level embedding technique (CLET), considers the secret message as a *string of characters* and uses them accordingly [78].



**Figure 2.1** Classification of the existing text steganographic techniques

Second category, bit-level embedding technique, considers the secret message as a *string of binary bits* and consecutively embeds them inside the cover document [66]. Third category, mixed-type embedding technique, is a mixture of the character-level and bit-level embedding techniques. It considers the secret message as a *string of binary bits* and converts them into a *string of characters* through mapping. Subsequently, the method embeds the mapped characters inside the cover



document [86]. All these categories and their respective sub-categories are described below in some detail.

### **2.2.1 Character-Level Embedding Technique (CLET)**

Character-level embedding technique (CLET) directly uses the *string of characters* and the embedding strategy of this technique can further be classified into two categories as:

- A. Cover\_Document\_Required (CDR)
- B. Cover\_Document\_Not\_Required (CDNR)

These techniques are discussed below.

#### **A. Cover\_Document\_Required (CDR)**

As the name implies, Cover\_Document\_Required (CDR) techniques need a cover document. It embeds the string of secret characters, serially, by creating distortions in the cover document. Character marking and mistyping are used for this purpose.

#### ***Character Marking* [62,87-89]**

This technique searches for the occurrence of secret characters in the cover document, serially, and marks the identical characters to embed them. It exploits the properties of fonts, like bold, italic, underline, size, style, etc., to mark the characters in the cover document. The receiver extracts the secret message by identifying and grouping the marked characters together (see Example 2.1).

**Example 2.1:**

*Cover work:* Techniques like cryptography do not ensure the secrecy.

*Generated stego work:* Techniques like cryptography do not ensure the secrecy.

*Embedded secret:* secret

***Mistyping***

This technique embeds the secret characters in the cover document, serially, by intentionally creating spelling mistakes [62,90,91] or by creating changes in the position of characters [87]. That is, placing the characters slightly over or under the baseline (the imaginary line over which all the characters of a line are placed is called baseline [92]). To extract the hidden message, the receiver has to identify and group the original characters of the misspelled word or the misplaced characters (see Example 2.2).

**Example 2.2:**

*Cover work:* He jumped off the boat.

*Generated stego work:* He dumped off the boat.

*Embedded secret character:* j

***Discussion on CDR Techniques***

As the embedding takes place at character-level, these methods embed 8-bits/distortion. This helps to reduce the number of distortions that are required to embed the secret which is the advantage of these methods.

On the downside, these techniques are not case-sensitive. Also, they will succeed, only, when all the characters in the secret message are present in the cover document specifically in that order. But, the occurrence frequencies of characters in English text are not uniform (refer Table 2.1). This makes the embedding probabilities of characters non-uniform, and results in the wastage of cover characters (embedding space) while embedding low occurring characters.

**Table 2.1** Occurrence frequencies of alphabets in English text [38]

<b>Letter</b> *	E	T	A	O	I	N	S	H	R	D	L	C	U
<b>Frequency</b>	12.7	9.1	8.2	7.5	7.0	6.7	6.3	6.1	6.0	4.3	4.0	2.8	2.8
<b>Letter</b> *	M	W	F	G	Y	P	B	V	K	J	Q / X / Z		
<b>Frequency</b>	2.4	2.3	2.2	2.0	2.0	1.9	1.5	1.0	0.08	0.02	0.01		

\* – not case-sensitive

In addition, making perceptible alterations for marking the characters or creating several spelling mistakes can draw attention. Hence, these methods can be used, stealthily, only when the amount of secret information to be embedded is very small as compared with the size of cover document. This non-uniformity/perceptible-alterations make/force the embedding capacity of these methods to be low.

## **B. Cover\_Document\_Not\_Required (CDNR)**

Cover\_Document\_Not\_Required (CDNR) techniques do not use a cover document. They generate stego documents, directly, based on the secret. Null cipher, missing letter puzzle and hiding data in wordlist are some of the CDNR techniques which are described below.

### ***Null Cipher*** [66,93]

This technique generates words or sentences, directly, based on the secret characters. It generates them in such a way that the particular position of a letter from each word (say second letter from each word) or sentence or paragraph represents the secret character (see Example 2.3). This method is complicated because the generated stego work should be meaningful and, also, inter-connect the sentences. Hence, it involves manual intervention and requires an experienced person to perform the task.

#### **Example 2.3:**

*Generated stego work (from [66]):* Apparently neutral's protest is thoroughly discounted and ignored. Islam hard hit. Blockade issue affects pretext for embargo on by-products, ejecting suets and vegetable oils.

*Embedded secret (considering the second character from each word):*  
Pershing sails from NY June 1.

### ***Missing Letter Puzzle*** [78]

This technique generates a list of words, of length between six and fifteen (not including space), as a stego work. It uses the three-digit decimal value associated with each secret character to generate a word. It uses the middle digit of the decimal value to find the length of the word. Then, based on the last digit of the decimal value, it replaces one or more character(s) of the generated word with question mark and provides a hint. This makes the generated stego work to disguise like a puzzle (refer Fig. 2.2).

The length of each word along with the presence of hint, position and number of question marks together represent the embedded character. A drawback of this method is that the communication of a list of words with special character, like

question mark, can attract attention. In addition, this method cannot be used in all scenarios, like organizations, as it tries to disguise like a puzzle.



```

Secret message: Secretdata
Generated stego work:
La?ender
Cheiran?hus
Sunflowe?
?ntirrhinum
Gillyflo?er
Schiz?nthus
Xeranthem?m
Helleb?re
?oneysuckle
Digita?is

```

**Figure 2.2** Sample input and output of missing letter puzzle technique

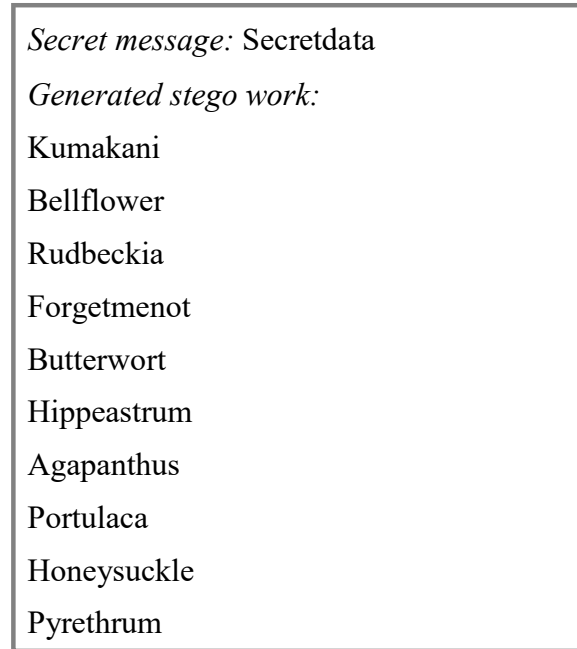
### ***Hiding Data in Wordlist [78]***

Similar to missing letter puzzle, this technique also generates a list of words of length between six and fifteen (refer Fig. 2.3). This method uses the three-digit decimal value associated with each secret character to generate a word. It uses the sum of digits in the decimal value to decide the first character and the middle digit of the decimal value to find the length of each word. Though this method avoids the usage of question mark and hint, it still generates a list of unrelated words which can draw attention.

### ***Discussion on CDNR Techniques***

Similar to CDR techniques, these techniques also embed 8-bits/distortion. But one advantage of these methods over CDR is that they achieve a higher embedding

capacity by generating the stego document directly. On the downside, the generation of each word or sentence depends on each secret character. Hence, there is no guarantee that the generated stego work would be related unless performed manually.



*Secret message:* Secretdata  
*Generated stego work:*  
Kumakani  
Bellflower  
Rudbeckia  
Forgetmenot  
Butterwort  
Hippeastrum  
Agapanthus  
Portulaca  
Honeysuckle  
Pyrethrum

**Figure 2.3** Sample input and output of hiding data in wordlist technique

### 2.2.2 Bit-Level Embedding Technique

This method considers secret message as a *string of binary bits* and embeds them accordingly. Based on the type of embedding, it is further classified into four categories as:

- A. Linguistic method
- B. Alignment modification
- C. White spacing
- D. Feature-based embedding

These techniques are discussed below.

## A. Linguistic Method

This method embeds secrets by exploiting the flexibility of languages. Semantic and syntactic are the methods used in linguistic method.

### *Semantic Method*

Semantic method exploits the flexibility in the choice of words to embed one bit at a time. That is, 1-bit/distortion. It embeds, a bit, by replacing one word with another without altering the original meaning.

- *Synonym Substitution* [90,94,95]

This method embeds a bit by substituting a word with its synonym (refer Table 2.2). Due to this, both the sender and receiver must have the complete list of words and their respective synonyms for embedding and extraction process respectively. It substitutes the first synonym to embed the bit “0” and the other to embed the bit “1”. This method has two major drawbacks such as:

- (i) the substituted synonym may not suit the sentence [90,96]
- (ii) the generated stego work may not match the author’s narration style

**Table 2.2** Sample words and the respective synonyms

Words	Synonyms	
	Bit 0	Bit 1
Leave	Depart	Go away
Subsequent	Successive	Later
Port	Harbour	Dock
Consequence	Result	Effect

- *Spelling of Words* [85]

Some words have different spellings in American and British English (refer Table 2.3). This method exploits this variation to embed the bits secretly. It represents a word with one spelling to embed the bit “0” and the other to embed the bit “1”. Since this method can be applied only to a particular set of words, which have different spellings, the embedding capacity of this method is low. In addition, it leaves clues to a third party, as the generated stego work will contain a mixture of spelling styles [96].

**Table 2.3** Sample United States and United Kingdom spellings [97]

United Kingdom spelling	United States spelling
Ageing	Aging
Colour	Color
Colonise	Colonize
Computerise	Computerize

### ***Syntactic Method***

This method exploits the syntax of sentences to embed the bits secretly. In English, the occurrence of the punctuation mark, like comma, becomes optional in some cases [94]. For example, the phrases, “Milk, bread, and butter” and “Milk, bread and butter” both convey the same [98]. This method explores this flexibility to embed the secret bits. The presence of a comma embeds the bit ‘0’ and the vice-versa embeds the bit ‘1’. However, this method requires utmost care as the improper use of such punctuations can draw attention.



## B. Alignment Modification

As the name implies, this method alters the alignment of text to embed one bit at a time. That is, 1-bit/distortion. Line and word shifting are the techniques used in alignment modification method.

### *Line Shifting* [87,99]

This technique shifts a line up or down to embed the bit 0 or 1 respectively. It considers three consecutive lines together as a group and marks a line only if all the lines in the considered group are sufficiently long. In each group, it shifts the middle line alone and leaves the other two neighbor lines undisturbed (see Example 2.4). During the decoding process, it uses these neighbor lines to check whether the middle line has been shifted or not. Hence, this method requires minimum three lines to embed one bit of information.

### Example 2.4:

*Cover work (taken from [66]):*

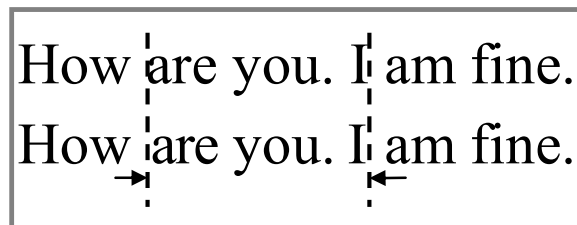
↑ Apparently neutral's protest is thoroughly discounted and ignored.  
 ↓ Islam hard hit. Blockade issue affects pretext for embargo on  
 ↓ by-products. Apparently neutral's protest is thoroughly discounted.

*Stego work (after embedding the bit "1"):*

↑ Apparently neutral's protest is thoroughly discounted and ignored.  
 ↓ Islam hard hit. Blockade issue affects pretext for embargo on  
 ↓ by-products. Apparently neutral's protest is thoroughly discounted.

**Word Shifting** [87,99]

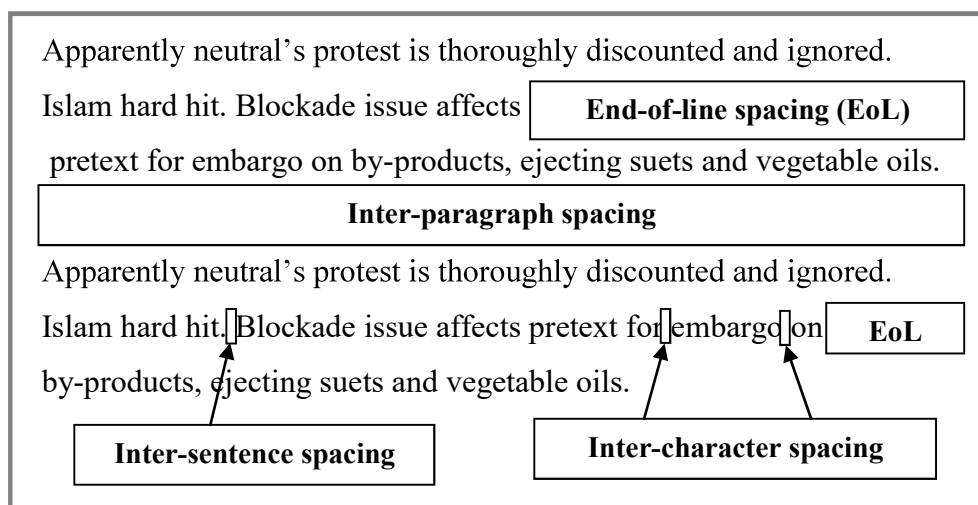
This method is similar to the line shifting technique mentioned above. The only difference between these two techniques is that, instead of considering lines, this method uses words to embed the bits. It partitions the words in each line into groups, each consisting of three words. Keeping the first and last word in each group constant, it shifts the middle word left to embed the bit “0” or right to embed the bit “1” (refer Fig. 2.4). It should be noted that, the embedding capacity of this method is better than the line shifting method as it can embed 1-bit/3-words.



**Figure 2.4** Example for word shifting

**C. White Spacing**

In a typical text document, white spaces are present in between words, sentences, paragraphs and at the end of lines (refer Fig. 2.5). This method exploits



**Figure 2.5** Illustration of the white spacing's in text document [66]

these spaces to embed the bits secretly. Inter-word spacing, inter-sentence spacing, end-of-line spacing, inter-paragraph spacing and Unispach are the techniques used in white spacing method.

### ***Inter-Word Spacing*** [76,100,101]

This method uses the white space between words to embed a secret bit. It injects an extra space character to embed the bit “1” or leaves undisturbed to represent the bit “0” (see Example 2.5). The embedding capacity of this method is higher than the word shifting method as the former can embed 1-bit/2-words.

#### **Example 2.5:**

<i>Cover work</i> <sup>*</sup> :	Hai·how·are·you.
<i>Stego work</i> <sup>*</sup> :	Hai·how·are·you.
<i>Embedded secret</i> :	010

<sup>\*</sup>*For understanding purpose, the white spaces in the text are highlighted using the character “.”.*

### ***Inter-Sentence Spacing*** [76]

This method exploits the white space between two sentences to embed a secret bit. It injects an extra space character to embed the bit “1” and leaves it undisturbed to represent the bit “0”. Since the number of sentences in a typical paragraph is less (average number of words in a sentence is between 15 and 20 [102] and average number of words in a paragraph is 150 [103]), this method suffers from low embedding capacity. However, this method performs better than the line shifting method, as the latter requires minimum three lines to embed one bit of information.

**End-of-Line Spacing** [76]

In a non-justified text, the presence of white space at the end of a line is very common. This method exploits this white space and injects space characters in it to embed the bits secretly (refer Fig. 2.6). It injects characters in the power of two to match the number of bit possibilities. That is, two characters to embed one bit (either “0” or “1”), four characters to embed two bits (00, 01, 10, 11), and so on (refer Table 2.4).

C	R	Y	P	T	O	G	R	A	P	H	I	C		T	E	C	H	N	I	Q	U	E	S	
D	O		N	O	T		E	N	S	U	R	E		S	E	C	R	E	C	Y	.			
(a)																								
C	R	Y	P	T	O	G	R	A	P	H	I	C		T	E	C	H	N	I	Q	U	E	S	
D	O		N	O	T		E	N	S	U	R	E		S	E	C	R	E	C	Y	.			
(b)																								

**Figure 2.6** Example for end-of-line spacing [76]: (a) Ordinary text; (b) White space encoded text

**Table 2.4** Number of bit and space character combinations

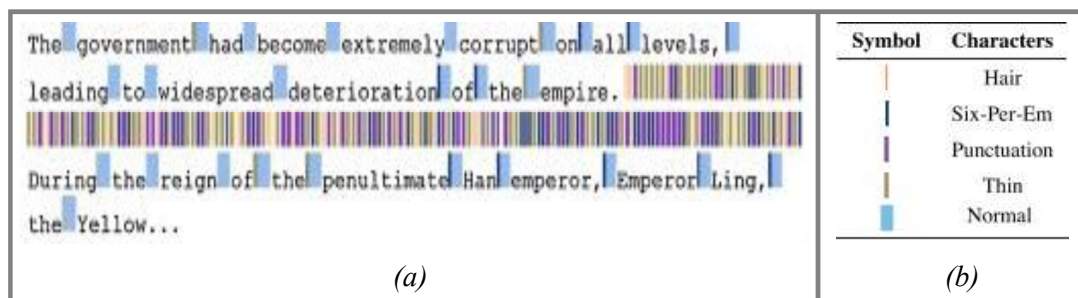
Number of bits	Number of possibilities	Possible bits	Number of space characters to be injected
1	2	0	1
		1	2
2	4	00	1
		01	2
		10	3
		11	4

**Inter-Paragraph Spacing** [101,104]

This method exploits the white space between two paragraphs to embed the bits. It injects space and tab characters in this white space to embed the bits “0” and “1” respectively.

**UniSpaCh** [66]

This technique is an improved version of the white spacing techniques that are mentioned above. It injects Unicode space characters like Punctuation, Thin, Six-per-Em, and Hair in inter-sentence, inter-word, inter-paragraph and end-of-line spacings' to embed 2-bits at a time (refer Fig. 2.7 (a)). The advantage of these characters over the ordinary space character is that the width of these characters is too small (refer Fig. 2.7 (b)). Hence more characters can be injected which increases its embedding capacity.



**Figure 2.7** Example for UniSpaCh technique: (a) Unicode space characters (color-coded for understanding purpose) [66]; (b) Size of Unicode and normal space characters [66]

**Discussion on White Spacing Methods**

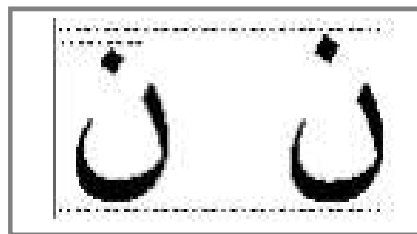
Though the methods are successful in injecting characters in white spaces, unnoticeably, a simple matching analysis on the number of space and tab characters with the number of words and paragraphs reveals the presence of secret message. In addition, except UniSpaCh, all methods are vulnerable when “Show paragraph marks and other hidden formatting symbols” button is selected [66]. When this is done, the ordinary space character is visible as “.” and tab character is visible as “→”, which helps to identify the intentionally injected characters.

## D. Feature-Based Embedding

This method explores the features of characters or document formats to embed the secret bits. A drawback of these methods is that they are restricted either to a particular language or document format. Moving “The Dot” in characters, exploiting the structure of characters, reversing/removing the diacritics in/from characters, “Change Tracking” technique are the methods used in feature-based embedding.

### *Moving “The Dot” in Characters [105]*

Similar to the lower-case letters of the English alphabets “i” and “j”, Arabic and Persian alphabets also have dots. In Persian language, out of the 32 alphabets eighteen have dots (three letters have two points each, five letters have three points each and ten letters have one point each) and in Arabic language, out of the 28 alphabets fifteen have dots [105]. This method exploits these dots to embed the bits secretly. It moves the dot upward to embed the bit “1” or leaves it undisturbed to represent the bit “0” (refer Fig. 2.8).



**Figure 2.8** Vertical displacement of Dot in the Persian character Noon [105]

### *Exploiting the Structure of Characters [41,84]*

In CJK (Chinese, Japanese and Korean) characters, there are totally 20,902 characters [84]. Of these characters, nearly 14,571 characters have left-right (L-R) structure [84] and nearly 4700 characters have up-down (U-D) structure [41]. It is

possible to generate most of these L-R and U-D characters by combining certain characters from 580 basic components [41] (refer Figures 2.9 (a) and (b)).



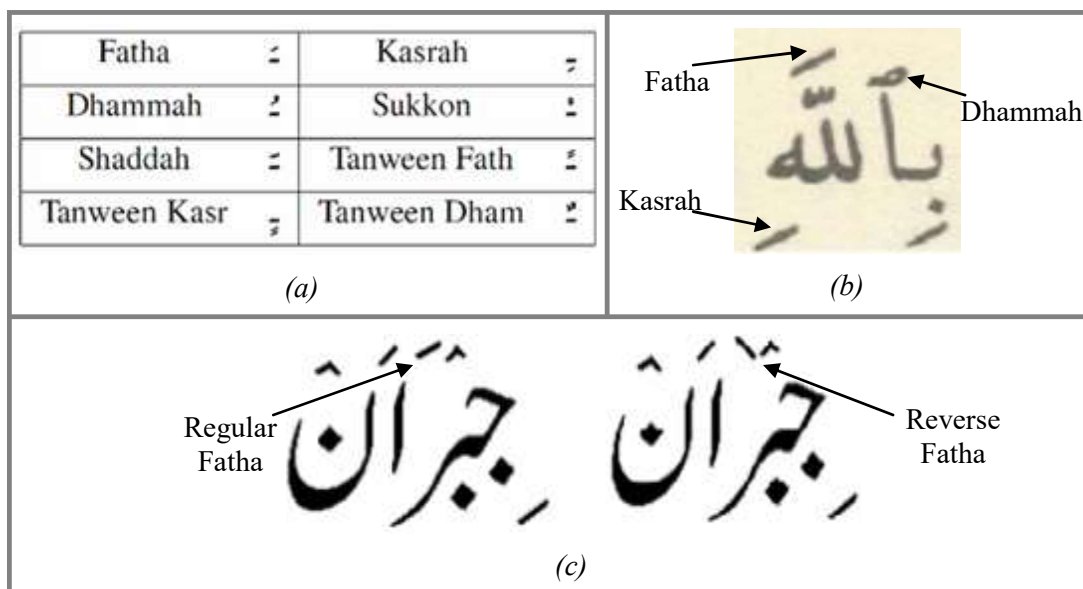
**Figure 2.9** Exploiting the structure of characters: (a) Basic components of Chinese, Japanese and Korean characters [84]; (b) Representation of characters using the basic components [41]

This method exploits this flexibility and partitions the available characters into three-character sets namely L-R structured character set, U-D structured character set and neither L-R nor U-D character set. To embed the secret bits, the method first scans the document serially until it identifies any of the L-R or U-D structured characters. If the bit to be embedded is “0”, then it leaves the encountered character undisturbed. Else, it replaces the character with the two basic components and alters the spacing between them. This will make it to disguise like a single character and avoids attention (refer Fig. 2.9 (b)).

### ***Reversing/Removing the Diacritics in/from Characters*** [94,106]

Arabic or Urdu language have many diacritics of which eight are most common (refer Fig. 2.10 (a)). It uses these diacritics to alter the pronunciation of

words [106] and the usage of these diacritics in written text is optional (refer Fig. 2.10 (b)). There exist two methods which exploits this flexibility to embed the secret bits.



**Figure 2.10** Exploitation of the Arabic or Urdu characters: (a) Diacritics of Arabic language [107]; (b) Representation of Araabs; (c) Usage of the regular and reverse Fatha [94]

One method [106] uses the diacritic Fatha to represent the bit “1” and uses the other seven diacritics to represent the bit “0”. Hence, to embed the bit “1”, it scans the cover document serially and removes all other diacritics till it encounters the diacritic Fatha. Similarly to embed the bit “0”, it removes all the diacritic Fatha till it encounters any one of the other seven diacritics.

Another method [94] reverses the diacritic Fatha to embed the bit “1” and leaves it undisturbed to represent the bit “0” (refer Fig. 2.10 (c)). In both the methods, to extract the embedded secret, the receiver scans the document serially and identifies the equivalent bits for the used diacritics.



**“Change Tracking” Technique [83]**

Some word processors like Microsoft Word facilitate the user to keep track of the modifications that are performed in the document. This technique explores this feature to embed the bits secretly. It first scans the chosen cover document and identifies the possible degenerations, like misspelling, typos, synonym replacement, etc., that can be performed in it. It then recognizes the typical occurrence probabilities of each of the possible degeneration and creates a Huffman tree [108] using such probabilities.

Next the sender purposefully degenerates the document, based on the secret bit, by inserting the corresponding mistake from the Huffman tree. Then with the help of the available commenting tools, the sender corrects the mistakes by himself and sends it to the receiver. Thus, the generated stego work contains both the degenerated and corrected texts.

Therefore, to a casual observer, the stego work will look like an experienced person correcting the mistakes of a novice (refer Fig. 2.11). Using the above-mentioned facility, the receiver extracts the hidden bits by recognizing the deliberately created mistakes and the choice made from the Huffman tree. This method embeds an average of 0.33-bits/word and the number of bits embedded per distortion depends upon the generated Huffman tree of each of the degeneration.

In Table 2, the PSNR values of the images recovered with right keys are all -1, which~~that~~ mean that the  $MSE_R$  values are all zero. That is, the recovered images and the original images are exactly the same. And the PSNR values of the images recovered with ~~incorrect~~wrong keys are smaller ~~than~~than 20dB, which ~~display~~show that the recovery results are still very difficult from the original ones ~~due~~to the noise surviving~~interference~~ ~~living~~in the watermark areas of the ~~healed~~recovered images.

**Figure 2.11** Sample output of the “Change Tracking” technique [83]

### 2.2.3 Mixed-Type Embedding Technique

This method is a mixture of the character-level and bit-level embedding techniques. Like bit-level embedding, it considers the secret message as a *string of binary bits*. It then converts these bits to characters using mapping. Finally, like character-level embedding, it embeds these mapped characters inside the cover document. Generating summary is the method used in mixed-type embedding technique.

#### *Generating Summary* [86]

This technique embeds 2-bits at a time. It chooses sentences from the cover work, based on the secret bits, to generate the stego work. To do so, it first partitions the 26 English alphabets into four groups using the reflection symmetry property (refer Table 2.5). It then maps these four groups to the four possible bit pairs.

**Table 2.5** Reflection symmetry properties of English alphabets [86]

Group no.	Reflection property	Alphabets	Secret bits
1	Reflection property followed along neither axis	C, F, G, J, L, N, P, Q, R, Z	00
2	Reflection property followed along horizontal axis	B, D, E, K, S	01
3	Reflection property followed along vertical axis	A, M, T, U, V, W, Y	10
4	Reflection property followed along both axis	H, I, O, X	11

Depending on each bit pair, it selects sentences from the cover work whose first alphabet (not an article) matches with any one of the alphabets of the respective group. This makes the generated stego work to look like a summary of the used cover work. To extract the embedded bits, the receiver has to identify the reflection

symmetry property of the first alphabet of each sentence. The major drawback of this method is that it requires one complete sentence to embed 2-bits of information.

### **2.3 Comparison of the Existing Methods**

The comparison of the existing methods is provided in Table 2.6. The methods are compared based on their embedding capacity and the number of bits embedded per distortion. For calculation purpose, it has been considered that the average length of a word is 4.50 (not including space) [109], average number of characters in a line is 60 (including space) [110], average number of words in a sentence is 15 and average number of sentences per paragraph (150 words per paragraph [103]) is 10 for English language.

From Table 2.6 it can be seen that, CLET techniques (both CDR and CDNR techniques) achieve the highest bits/distortion, followed by end-of-line spacing, UniSpaCh and generating summary. Also, CDNR techniques achieve the highest embedding capacity, followed by UniSpaCh. This shows that, only, CDNR techniques achieve both high embedding capacity and bits/distortion. However, they cannot guarantee a meaningful stego document. This drawback makes UniSpaCh the best alternative available to general users. Hence the same has been considered as a benchmark for comparing the methods that are developed and presented in upcoming chapters.

**Table 2.6** Comparison of the existing techniques

Technique	Type of embedding	Bits/distortion (approximate)	Embedding capacity (approximate)	Reference
Character marking	Character-level	8	Low (due to the non-uniform occurrence of characters)	[62,87-89]
Mistyping	Character-level	8	Low (due to the non-uniform occurrence of characters)	[62,87,90,91]
Null cipher	Character-level	8	8-bits/5.5-cc	[66,93]
Missing letter puzzle	Character-level	8	8-bits/11.5-cc	[78]
Hiding data in wordlist	Character-level	8	8-bits/11.5-cc	[78]
Synonym substitution	Bit-level	1	1-bit/5.5-cc*	[90,94,95]
Spelling of words	Bit-level	1	1-bit/5.5-cc*	[85]
Line shifting	Bit-level	1	1-bit/180-cc	[87,99]
Word shifting	Bit-level	1	1-bit/16.5-cc	[87,99]
Inter-word spacing	Bit-level	1	1-bit/5.5-cc	[76,100,101]
Inter-sentence spacing	Bit-level	1	1-bit/83.5-cc	[76]
End-of-line spacing	Bit-level	2	2-bits/56-cc	[76]
UniSpaCh	Bit-level	2	1.046-bits/cc	[66]
Moving “The Dot” in characters	Bit-level	1	1-bit/cc*	[105]
Exploiting the structure of characters	Bit-level	1	0.5-bits/cc	[41,84]
Reversing/Removing the diacritics in/from characters	Bit-level	1	1-bit/cc*	[94,106]
“Change Tracking” technique	Bit-level	Variable due to Huffman tree	0.33-bits/5.5-cc	[83]
Generating summary	Mixed-type	2	2-bits/83.5-cc	[86]

\* – represents the assumption that any character or word can be exploited by the method; cc – cover-character

## **2.4 Summary**

Existing text steganographic methods are described briefly, and the merits and demerits of each of the methods are discussed in some detail. The methods are compared based on their embedding capacity and bits/distortion. It was noticed that, apart from the methods that generate the stego document directly, no existing method was found to achieve both high bits/distortion as well as high embedding capacity.

Due to this, embedding secret information inside text document is difficult as the size of a typical cover document is limited. Situation gets worse, while embedding multimedia information like image, audio, video, etc., as it is of typically in the order of megabytes. Hence, there is a need for formulating a method with high embedding capacity and bits/distortion.

The work carried out as part of this thesis aims to achieve the same. After detailed analysis, it was understood that this can be achieved by designing a method that:

- (i) embeds maximum number of data bits in a distortion
- (ii) utilizes the available embedding space in an efficient manner

Bearing this in mind, in the present work, various word processors were studied to identify the suitable features that can carry a large number of bits per distortion. Three novel methods that utilize the identified features and achieve high embedding capacity have been developed and applied.

# EXPLORING THE FONT ATTRIBUTES OF WORD PROCESSOR DOCUMENTS

---

*This chapter discusses the font attributes of various word processor documents and analyses them from a steganographic perspective. It, also, presents the various ways of employing these attributes for steganography, and categorization of them based on their usability and imperceptibility. A comparison of the selected attributes is performed to identify the best word processor, suitable for steganographic purpose.*

### 3.1 Introduction

As mentioned in Chapter 1 (Fig. 1.5), text steganography considers both plain text and word processor documents as text document. However, differences exist between them. Plain text documents can contain only ASCII (American Standard Code for Information Interchange) characters [82] (not graphics) and supports fewer formatting features or attributes like font name, font style and font size. These make it advantageous to be independent of computer architectures. As a result of the same, a plain text document once created can be opened in any operating system seamlessly [82]. However, less formatting features make the possibility to embed secrets inside plain text document a complex task.

Word processors, on the other hand, can contain texts, images, tables, etc., and provide advanced formatting attributes like underline, font color, etc. [111,112]. These attributes are often stored as metadata and are applied over the underlying plain text content, to create visual effects. Hence these attributes expect the corresponding

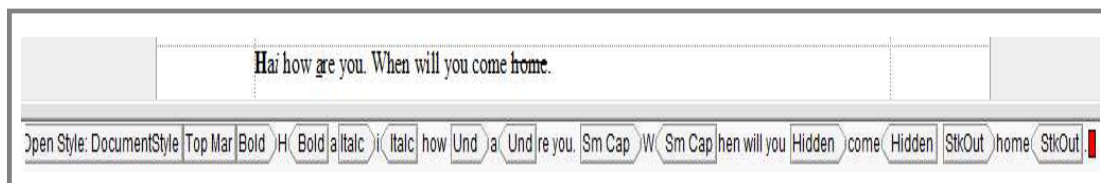
document to be interpreted correctly [113]. This makes the document to be compatible only with the corresponding program or operating system [114] (refer Table 3.1).

**Table 3.1** Details of word processors

Word processor	Owned by	Copyright status	Operating system	Initial release	Reference
Microsoft Word	Microsoft	Proprietary	Windows	1983	[115]
LibreOffice	The Document Foundation	Open source	Linux, Windows, Mac	2011	[116]
OpenOffice	Apache	Open source	Linux, Windows, Mac	2012	[117]
WordPerfect	Corel	Proprietary	Windows	1996	[118,119]

As mentioned, word processors have a rich set of font attributes, each performing a particular task. A list of major attributes, that is present in word processors like Microsoft (MS) Word (2007 and 2010), LibreOffice (LO), OpenOffice (OF) and WordPerfect (WP) are presented in Tables 3.2 (A) and (B). The availability of these attributes facilitates to embed secrets inside word processor documents in an efficient manner.

In the case, however, of WP, any formatting related modifications performed in a WP document can be viewed readily by selecting *View > Reveal Codes* (refer Fig. 3.1). This single feature makes WP not suitable for steganography. But, as features can be added or removed at later point of time, the present study considers WP for further discussion with an expectation that this particular feature will be removed in future versions.



**Figure 3.1** WordPerfect revealing the formatting information

**Table 3.2 (A)** List of attributes in word processors: part 1

Font attributes	MS Word 2007 & 2010	WP	LO & OF	Font attributes	MS Word 2007 & 2010	WP	LO & OF
AllCaps/Capitals	✓	NA	✓	Engrave	✓	NA	✓
Animation	✓	NA	NA	Hidden	✓	✓	✓
Blinking	NA	NA	✓	Highlight	☒	✓	✓
Bold/BoldBi	✓	✓	✓	HighlightColor	☒	✓	✓
Bold StrikeThrough	NA	NA	☒	Italic/ItalicBi	✓	✓	✓
Border	✓	✓	✓	Kerning	✓	✓	☒
Border	(Character)	(Paragraph)	(Character)	Ligatures	✓	NA	NA
BorderColor	✓	✓	✓	LowerCase	×	×	✓
ColorIndex/ColorIndexBi/Fo ntColor	✓	✓	✓	Name/NameAscii/NameBi/NameFa rEast/NameOther	✓	×	✓
DoubleStrikeThrough	☒	NA	☒	NumberForms	✓ (2010)	NA	NA
Duplicate/FormatPainter/Qui ckFormat/Formatpaintbrush	×	×	×	NumberSpacing	✓ (2010)	NA	NA
Emboss	✓	NA	✓	Outline	✓	✓	✓
EmphasisMark	✓	NA	NA				

MS – Microsoft; WP – WordPerfect; LO – LibreOffice; OF – OpenOffice; ✓ – represents the presence of attribute and suitability for steganography; ☒ – represents the presence of attribute but not suitable for steganography (performs perceptible modifications); × – represents the presence of attribute but cannot be exploited; NA – represents the absence of attribute



**Table 3.2 (B)** List of attributes in word processors: part 2

Font Attributes	MS Word 2007 & 2010	WP	LO & OF
Overlining	NA	NA	✓
OverlineColor	NA	NA	✓
Position	✓	☒	✓
RedLine	NA	✓	NA
Rotation	NA	NA	☒
Scaling	✓	NA	✓
Shading	✓	✓	✓
ShadingColor/BackgroundColor	✓	×	✓
Shadow	✓	✓	✓
Size/SizeBi	✓	✓	✓
SmallCaps/SmallCapitals	✓	✓	✓
Spacing	✓	✓	✓
StrikeThrough/SingleStrikeThrough/StrikeOut	☒	☒	☒
StrikeThrough with X	NA	NA	✓
StrikeThrough with /	NA	NA	✓
StylisticSet	✓	NA	NA
	(2010)		
Subscript	✓	☒	✓
Superscript	✓	☒	✓
Title/Capitalize each Word/Initial Capitals	×	×	✓
Underline	✓	✓	✓
UnderlineColor	✓	✓	✓
UpperCase	×	×	×

*MS – Microsoft; WP – WordPerfect; LO – LibreOffice; OF – OpenOffice;*  
*✓ – represents the presence of attribute and suitability for steganography;*  
*☒ – represents the presence of attribute but not suitable for steganography*  
*(performs perceptible modifications); × – represents the presence of attribute but*  
*cannot be exploited; NA – represents the absence of attribute*

From Tables 3.2 (A) and (B), it can be observed that most of the attributes are common to all the processors. The details of these attributes and the various ways to employ them for steganography are given in Section 3.2. For the convenience of readers, hereafter, attributes having multiple names are represented using single name.

For example, attributes Duplicate/FormatPainter/QuickFormat/Formatpaintbrush are represented using Duplicate.

### **3.2 Employing the Font Attributes for Steganography**

Font attributes have a major role to play while displaying the characters. Each attribute is responsible to produce certain visual effect. However, applying some attributes on particular character(s) is ineffective. For example, applying the Bold attribute on space character produces no visual effect. In addition, when the values of certain attributes are varied between a certain range, it produces no noticeable effect. For example, altering the least significant bits of the 24-bit Color attribute is indistinguishable. In some cases, it is possible to replicate the effect of certain attributes by carefully mixing the effects of other attributes. For example, making a character superscript and lowering its position appropriately, will make it to appear like a subscript character.

These characteristics are discussed below in detail.

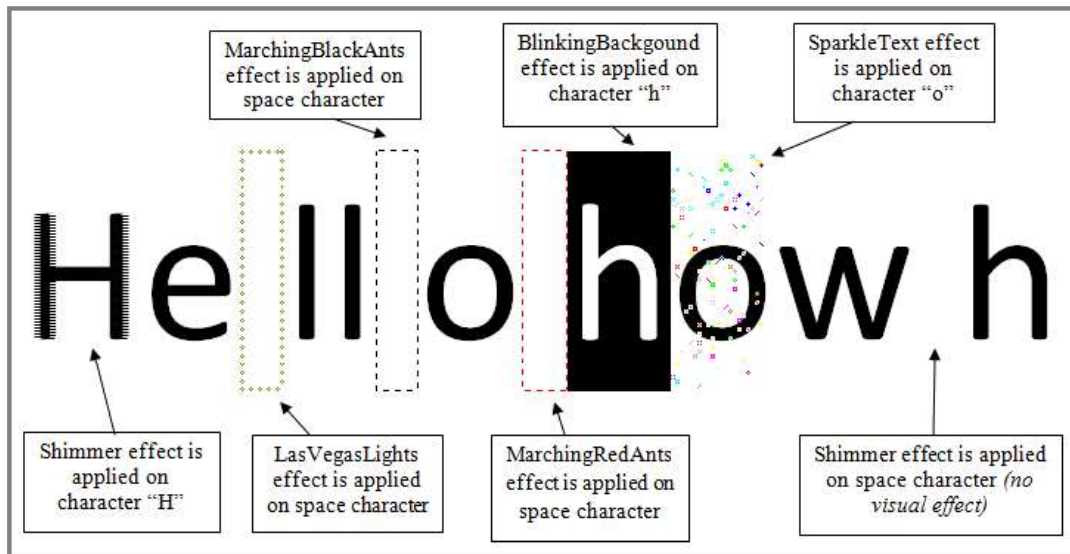
#### ***AllCaps***

AllCaps attribute is used to change the selected alphabets to uppercase. However, applying this attribute on an already capitalized character or a non-alphabet produces no effect. For example, HEEL99 – AllCaps attribute is set for the characters first “E” and first “9”.

#### ***Animation***

Animation attribute is used to produce visible animation effects on the selected text. The various animation effects that could be produced are BlinkingBackground, MarchingBlackAnts, MarchingRedAnts, LasVegasLights, Shimmer and SparkleText

(refer Fig. 3.2). However, when the Shimmer effect is applied on space character it produces no visual effect.



**Figure 3.2** Exploitation of the Animation attribute

### **Blinking**

Blinking attribute is used to create blinking effect on the selected text. However, applying this attribute on space character produces no visual effect.

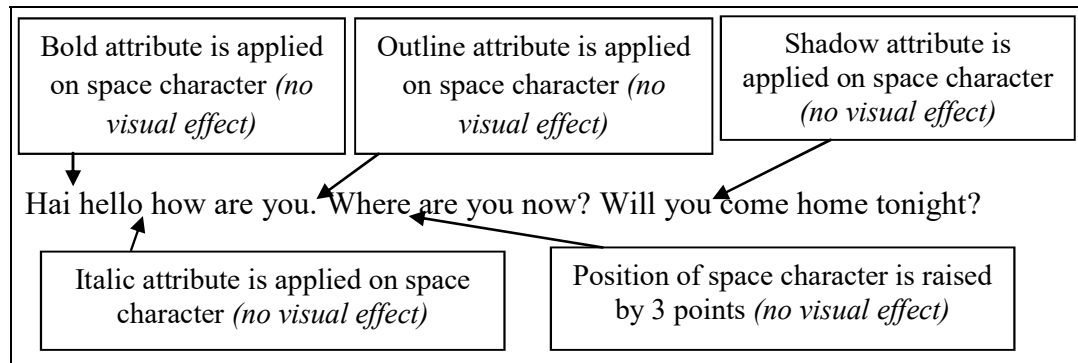
### **Bold, EmphasisMark, Italic, Outline, Position and Shadow**

Bold attribute is used to make the selected text bold. For example, **b**. EmphasisMark is used to represent stress on a particular character by using symbols like Over Comma, Over Solid Circle, Over White Circle and Under Solid Circle (refer Fig. 3.3). Italic attribute is used to make the selected text italic. For example, *i*. Outline attribute is used to provide an outline to the selected text. For example, **b**.

Position attribute is used to lower or raise the selected text at point level (in typography, a point is equivalent to 1/72 of an inch [120]). For example, **b** **b** **b** – Represents the normal, raised and lowered characters. Shadow attribute is used to

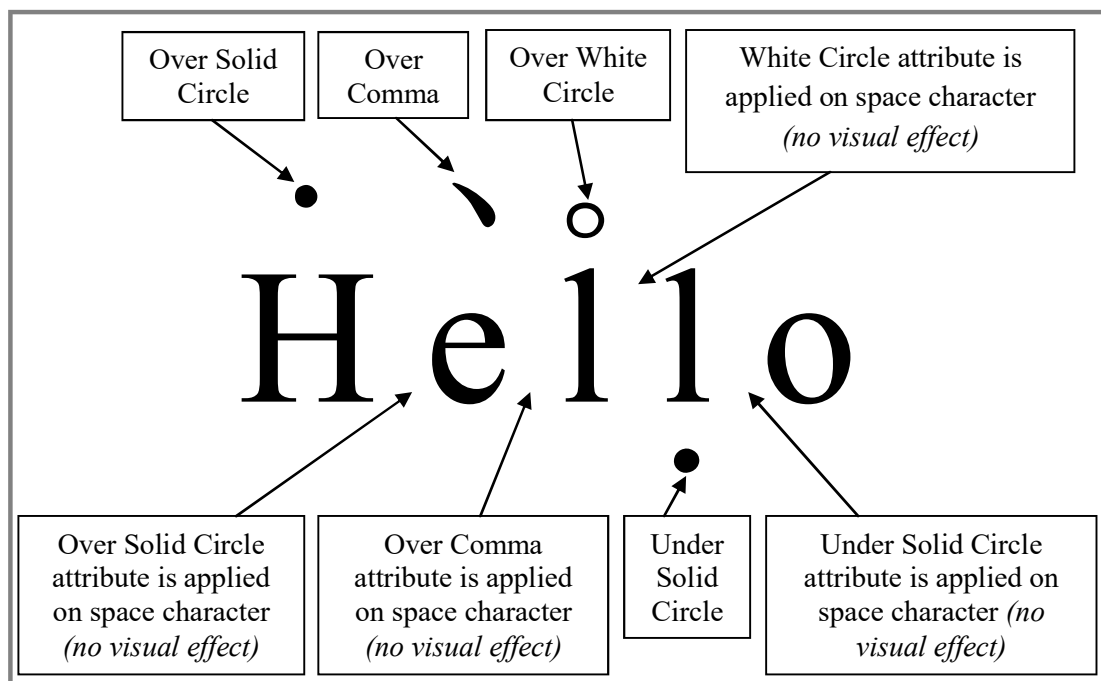
create a shadow underneath the selected text. For example, g. Although, these attributes create perceptible alterations on other characters, applying them on space character produces no visible effect (see Example 3.1).

### Example 3.1:



### ***Bold StrikeThrough, DoubleStrikeThrough and SingleStrikeThrough***

Bold StrikeThrough strikes the selected text, boldly, once. DoubleStrikeThrough and SingleStrikeThrough attributes strikes the selected text twice and once respectively. For example, Hai ~~how~~ are you. – DoubleStrikeThrough attribute is applied to the word “how” and StrikeThrough attribute is applied to the word “you”.



**Figure 3.3** Exploitation of the *EmphasisMark* attribute

### ***Border and BorderColor***

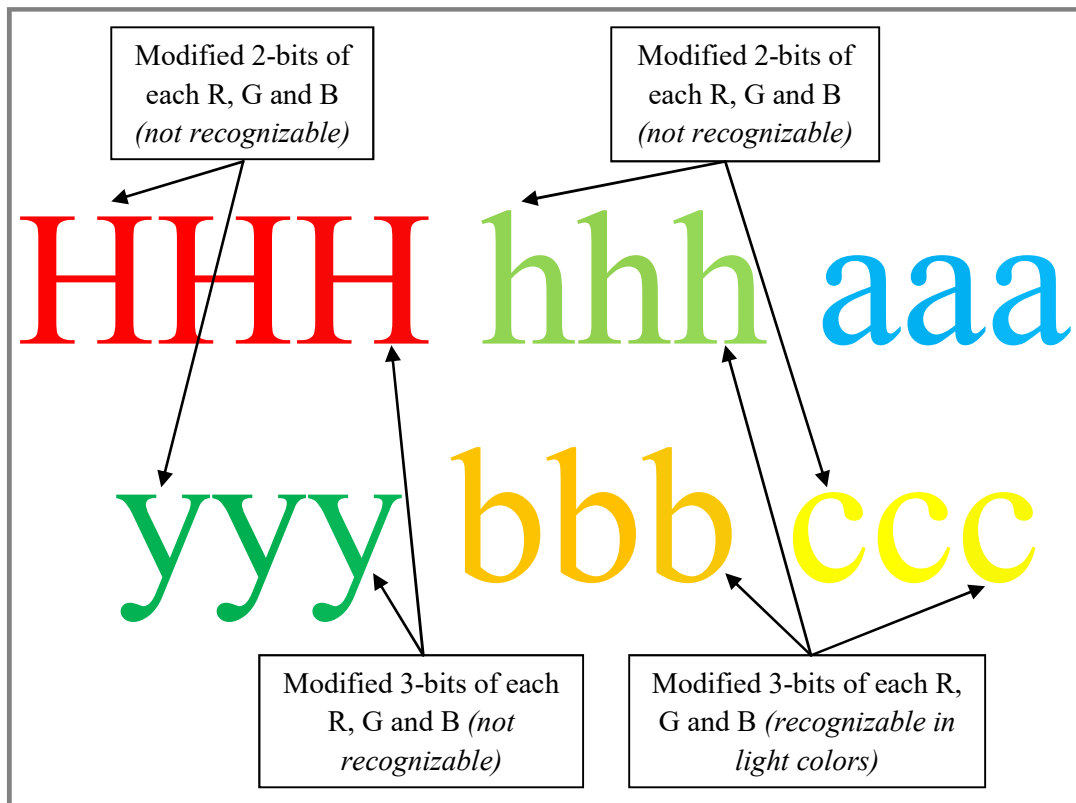
Border attribute is used to highlight the selected text by setting a box around it. Due to this, the spacing between the selected text and its neighbors gets altered. For example, Hai – Border attribute is applied on the character “a”. BorderColor attribute is used to specify the color in which the Border should be displayed. Setting the default background color of the document to BorderColor, makes the Border invisible. When this is done, altering the least significant (R, G, B) value of BorderColor attribute, also, goes unnoticed on screen (but noticeable in hard copy). For example, Hai Hai – Border attribute is applied on the second occurrence of “a” and the (R, G, B) value of BorderColor attribute is set as (254, 254, 254).

### ***Color***

Color attribute is used to specify the color in which a particular character should appear. It is represented by a 24-bit value using the format (R, G, B). The least significant 1 or 2-bits of each R, G and B can be modified without creating any visual difference (refer Fig. 3.4). Also, applying this attribute on space character produces no visual effect.

### ***Duplicate***

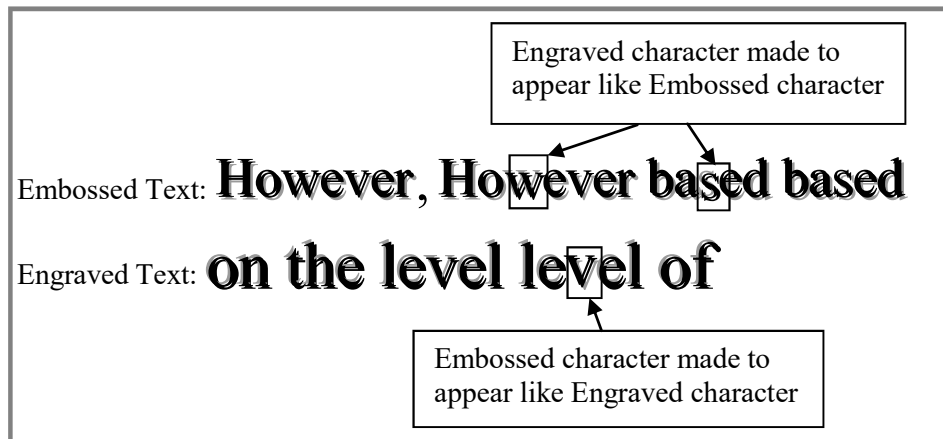
Duplicate is used to copy the formatting of one text and apply it to another, directly.



**Figure 3.4** Exploitation of the Color attribute

### ***Emboss and Engrave***

Emboss attribute is used to place the selected text slightly above the baseline and apply shadow to the edges inward. For example, **H**. On the other hand, Engrave attribute is used to place the selected text below the baseline and apply shadow to the edges outwards. For example, **H**. Though one cannot change the position of the shadow (whether to fall inward or outward), it is possible to change the position of the character up or down and make an embossed character to look like an engraved character and vice versa. Finding such modified characters from a document containing a large amount of such embossed or engraved characters is a difficult task (refer Fig. 3.5). Also, applying these attributes on space character produces no effect.

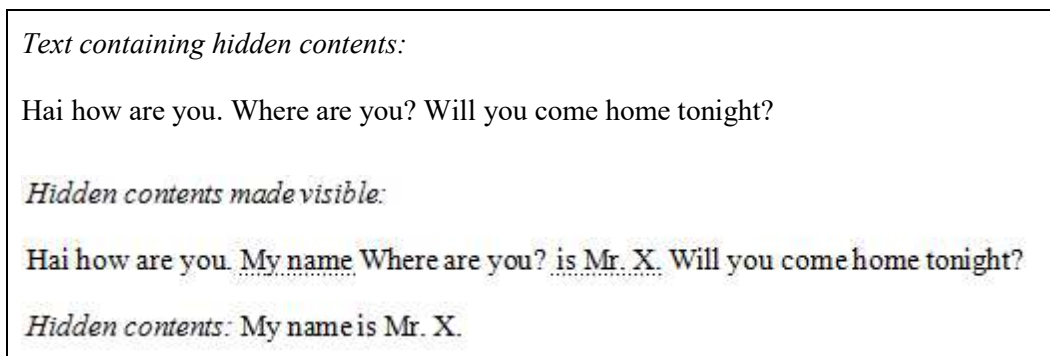


**Figure 3.5** Exploitation of the Emboss and Engrave attributes

### **Hidden**

Hidden attribute is used to make the selected text invisible in a text document. The hidden text can be made visible or printed by changing the settings. In MS Word, hidden contents will be made visible by selecting *Hidden text* checkbox in *Word Options > Display > Always show these formatting marks on the screen* (see Example 3.2). In OF, it will be made visible by selecting *View > Nonprinting Characters*. In WP, it will be made visible by selecting *View > Hidden Text*.

### **Example 3.2:**

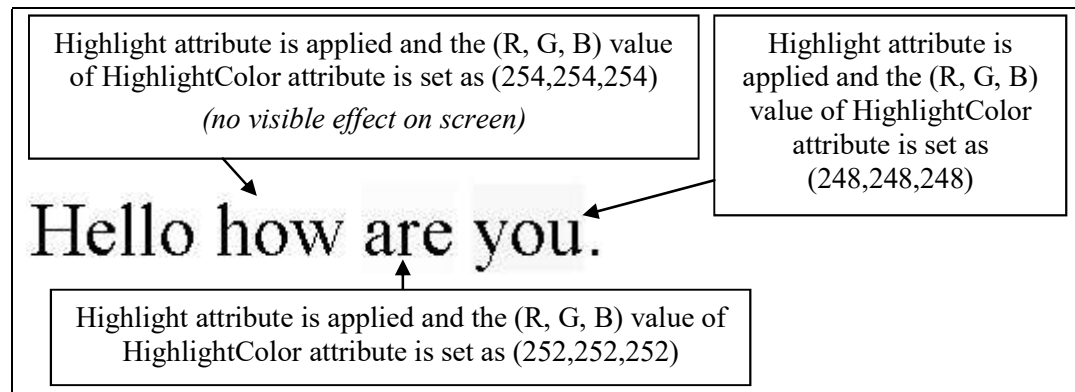


### **Highlight and HighlightColor**

Highlight attribute is used to highlight (mark) the selected text by changing the background color. HighlightColor attribute is used to specify the color that should be

used for highlighting. Setting the default background color of the document to HighlightColor, makes the highlighting invisible. When this is done, altering the least significant (R, G, B) value of HighlightColor attribute, also, goes unnoticed on screen (but noticeable in hard copy). The same is shown in Example 3.3.

**Example 3.3:**



***Kerning***

Kerning attribute is used to alter the spacing between overlapping character pairs like AV, WA, etc. In WP, this attribute allows to manually control the spacing between characters by taking the kerning value as input. Hence, changing the value by a 0.1 or 0.2 point does not produce noticeable effect.

Whereas in MS Word, this attribute automatically adjusts the space between characters. But it allows to control whether the kerning effect should be produced or not, even though the attribute is set. It does so, by taking the size of font as kerning value, which can vary between 1 and 1638 points. The effect is produced only when the specified value is lesser than or equal to the font size of the character. Hence, setting a value greater than the character's size will produce no effect. For example, AV AV AV AV – The Kerning attribute is applied to the second and fourth pairs, and the effect is made to produce for the second pair alone.



### ***Ligatures***

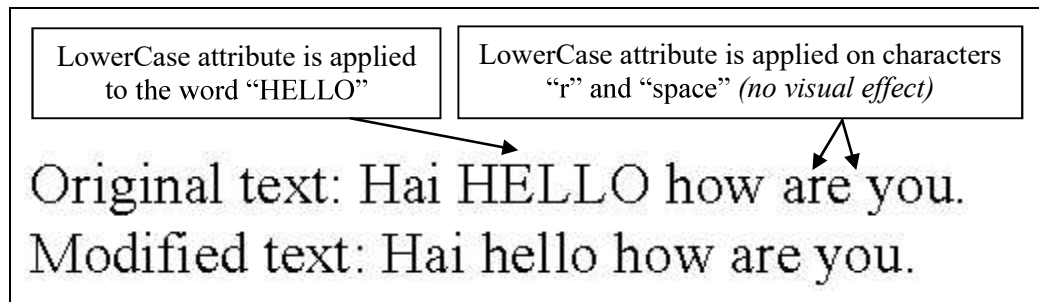
Ligatures are letters that are merged together as one character and are mainly used for calligraphic purpose [121,122]. There are four possible ligatures styles namely Standard Only, Standard and Contextual, Historical and Discretionary, and All. These styles produce visual effects only on a particular set of characters like fi, fl, ff, ffi, etc. [122,123] (see Example 3.4). Applying this attribute on other character pairs produces no effect. Also, applying this attribute on non-alphabets or non-neighbor characters does not cause any visual difference.

#### **Example 3.4:**

Default	: Different, Difficult, Diffie-Hellman.
Standard Only	: Different, Difficult, Diffie-Hellman.
Standard and Contextual	: Different, Difficult, Diffie-Hellman.
Historical and Discretionary	: Different, Difficult, Diffie-Hellman.
All	: Different, Difficult, Diffie-Hellman.

### ***LowerCase***

LowerCase attribute is used to change all the selected alphabets to lowercase. But, applying this attribute on an already lowercase character or a non-alphabet produces no effect (see Example 3.5).

**Example 3.5:*****Name***

Name attribute is used to specify font style, like Times New Roman, Calibri, etc., that should be used to display the character. If the specified font style is not present in the system, a default font style will be used to display that particular character (in MS Word, the used default font style can be checked at *Word options > Advanced > Show document content > Font Substitutions*). This attribute can take any character as a font style. Hence, setting the secret characters itself as a font style will go unnoticed.

***NumberForms*** [124,125]

NumberForms attribute is used to alter the way the numbers are displayed in text document. There are two possible styles namely Lining and Old-style. Lining style numbers appear over the baseline with the tops and bottoms of each number line up exactly. Old-style numbers look a little more uneven. That is, some letters fall below the baseline, and some even change their shape (see Example 3.6). However, applying this attribute on non-numerical characters produces no visual effect.

**Example 3.6:**

Lining	:	00112233445566778899
Old-style	:	00112233445566778899

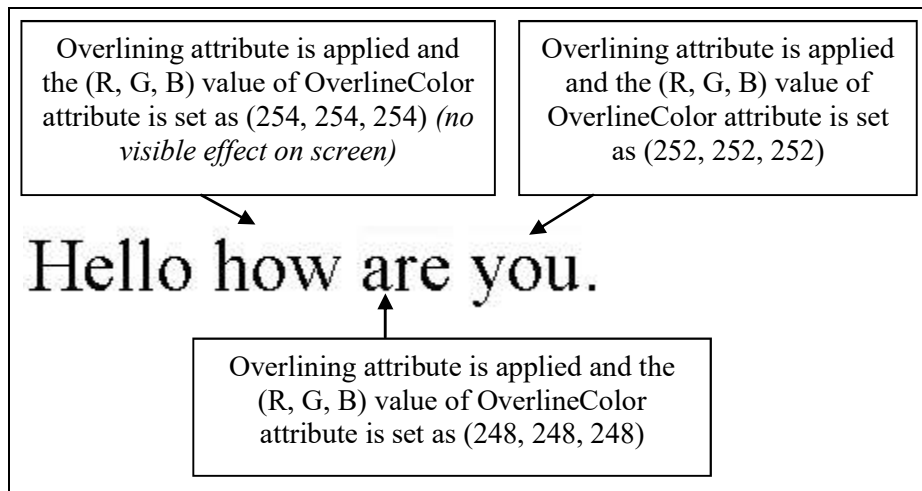
***NumberSpacing*** [124,126]

NumberSpacing attribute is used to alter the way the numbers are displayed inside a chart or table. There are two possible styles namely “Tabular” and “Proportional”. Tabular numbers have exact width as one another and hence line up perfectly in a vertical column of the table. Proportional numbers are more visually pleasing, and work well for dates and phone numbers. However, applying this attribute on non-numerical characters produces no visual effect.

***Overlining and OverlineColor***

Overlining attribute is used to insert a line (bar) over the selected text. There are totally sixteen different styles of overline available. OverlineColor attribute is used to specify the color that should be used to overline. Setting the default background color of the document to OverlineColor, makes the overline invisible. When this is done, altering the least significant (R, G, B) value of OverlineColor attribute, also, goes unnoticed on screen (but noticeable in hard copy). The same is shown in Example 3.7.

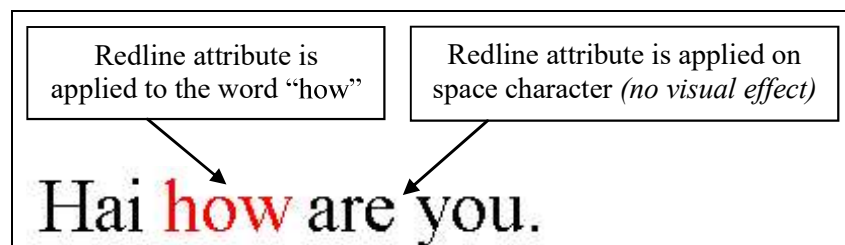
**Example 3.7:**



**Redline**

Redline attribute is used to change the font color of the selected text to red. However, applying this attribute on space character produces no visual effect (see Example 3.8).

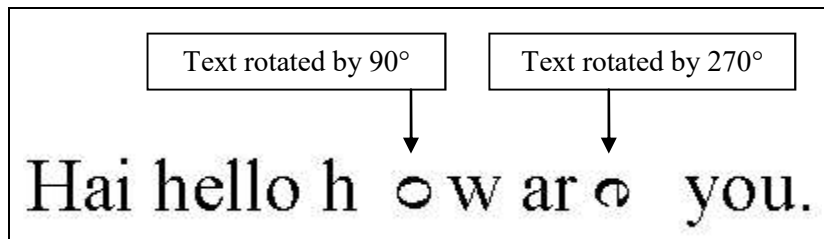
**Example 3.8:**



**Rotation**

Rotation attribute is used to rotate the selected text by 90° or 270° (see Example 3.9).

**Example 3.9:**



***Scaling***

Scaling attribute is used to change the width of a character. The characters “O” and “0” looks alike except their width. By using this attribute, it is possible to replace “O” with “0” and vice versa. For example, 000000000 – The Scaling attribute is applied to the fifth and sixth characters (from left).

***Shading (WP)***

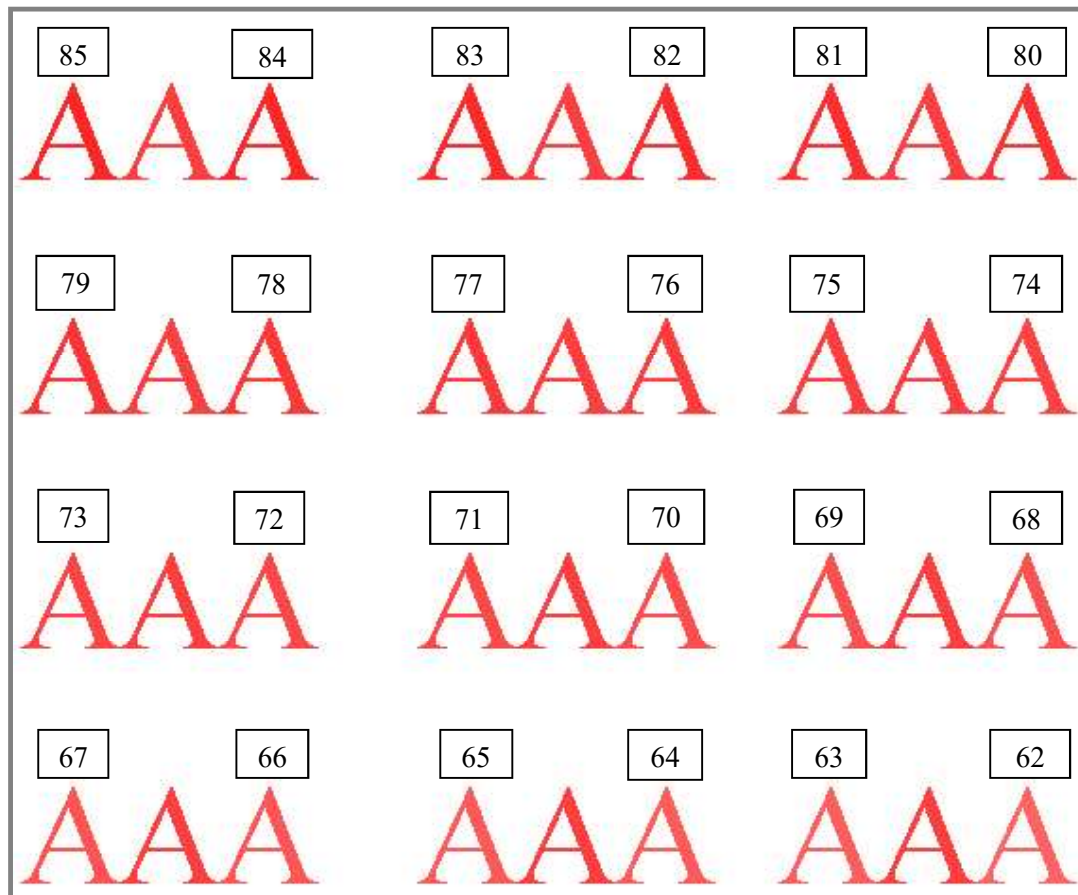
Shading attribute in WP is used to alter the darkness of font color of the selected text. The value of this attribute can be varied, at percentage level, between 1 and 100 (inclusive). Varying this value in a range of  $\pm 4$  produces no observable effects (refer Fig. 3.6).

***Shading and ShadingColor (MS Word, LO & OF)***

Shading attribute is used to change the background of each cell in a table. ShadingColor attribute is used to specify the color that should be used for shading (see Example 3.10).

**Example 3.10:**

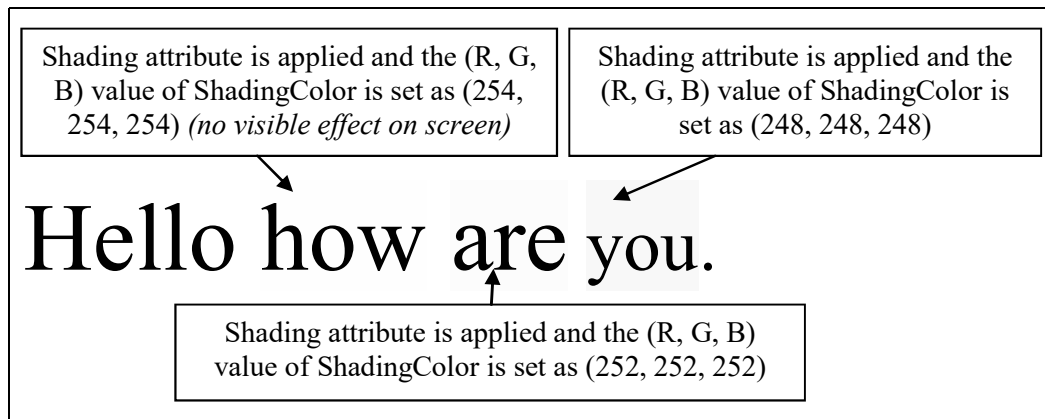
Shading attribute is applied to this cell	Highlight attribute is applied on this text
Highlight attribute is applied on this text	Shading attribute is applied to this cell



**Figure 3.6** Exploitation of the Shading attribute in WordPerfect. Middle character of each word has a default shading value of 75%, whereas the left and right characters have varying values which are mentioned above each character (in %)

However, when the Shading attribute is applied to texts, which are not in table, it behaves like the attribute Highlight. For example, **Hai** – Shading attribute is applied on the character “H” and Highlight attribute is applied on the character “a”. Setting the default background color of the document to ShadingColor, makes the shading invisible. When this is done, altering the least significant (R, G, B) value of ShadingColor attribute, also, goes unnoticed on screen (but noticeable in hard copy). The same is shown in Example 3.11.

**Example 3.11:**



***Size***

Size attribute is used to specify the font size of a character. The character looks alike even when the font size is altered by some points. For example, hello hello – The character “h” in the second word is increased by 0.5 point. It should be mentioned that, in MS Word the value can be altered at 0.5 point levels whereas in LO, OF and WP, it can be varied at 0.1 point levels. This shows that LO, OF and WP facilitates more to exploit this attribute, when compared with MS Word.

***SmallCaps***

SmallCaps attribute is used to change the selected alphabets to uppercase. Hence, applying this attribute on non-alphabets produces no visual effect. Also, the only difference between this attribute and AllCaps is that the font size of the former is smaller than the latter. For example, H H – The first character is SmallCaps and the second character is AllCaps. Hence, setting the SmallCaps attribute and increasing the font size of a character will make it to appear like the effect of AllCaps. For example, Yellow Yellow – The SmallCaps attribute of the second “Y” is set and the font size is increased by 2 points.

## Spacing

Similar to Kerning attribute, Spacing attribute is also used to alter the spacing between two characters either by expanding or condensing (the difference between them is that, in MS Word, Kerning attribute produces effect only on specific character pairs). Since the spacing of characters are not uniform in a justified text, modifying the document using this attribute will go unnoticed (refer Table 3.3).

**Table 3.3** Exploitation of the Spacing attribute

Unmodified text string	Modified text string	Performed modification (every third character in the modified text string is altered)
PhD Guide	PhD Guide	Expanded by 0.05 point
Doctoral	Doctoral	Expanded by 0.1 point
Committee	Committee	Expanded by 0.15 point
Doctorate	Doctorate	Expanded by 0.2 point
Conference	Conference	Expanded by 0.25 point
Publication	Publication	Expanded by 0.3 point
Presentation	Presentation	Expanded by 0.35 point
Meeting	Meeting	Expanded by 0.4 point
Recommend	Recommend	Expanded by 0.45 point
PhD Guide	PhD Guide	Condensed by 0.05 point
Doctoral	Doctoral	Condensed by 0.1 point
Committee	Committee	Condensed by 0.15 point
Doctorate	Doctorate	Condensed by 0.2 point
Conference	Conference	Condensed by 0.25 point
Publication	Publication	Condensed by 0.3 point
Presentation	Presentation	Condensed by 0.35 point
Meeting	Meeting	Condensed by 0.4 point

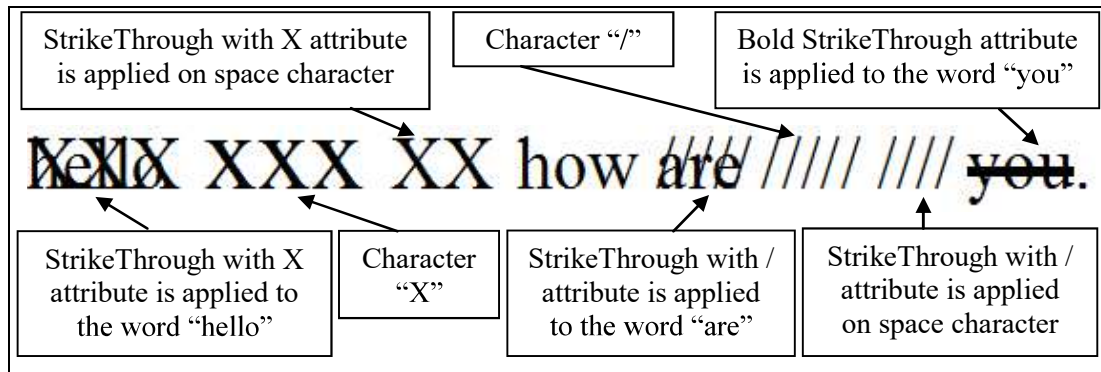
## StrikeThrough with X and StrikeThrough with /

StrikeThrough with X or / is used to strike the selected text with the character “X” or “/” respectively. These attributes can be used to disguise as characters



“X” or “/”. To do so, empty white spaces are inserted and stroked out with the attribute “StrikeThrough with X” or “StrikeThrough with /” (see Example 3.12).

**Example 3.12:**



***StylisticSet***

StylisticSet attribute is used to create swashes on alphabets [124] (a swash is a typographical flourish on a glyph [127]). Applying this attribute on other characters produces no visual effect. MS Word has twenty different styles (refer Fig. 3.7). Of these, only the first seven styles produce visual effect and applying the styles between eight and twenty (inclusive) is ineffective.

***Subscript and Superscript***

Subscript and Superscript attributes are used to place a character below and above the baseline respectively, in addition to decreasing its font size. For example,  $g^g$   $g_g$  – Represents a normal, superscripted and subscripted “g” characters respectively. Hence, lowering the position of a superscripted character to certain points will make it to appear like a subscripted character and vice versa. However, in MS Word, altering this attribute affects the inter-space between the adjacent lines. Hence, they can be used only when the inter-space between the adjacent lines is sufficiently large. This

limitation is not present in LO and OF. In such case, altering the position and font size appropriately can make it to appear like a normal character.

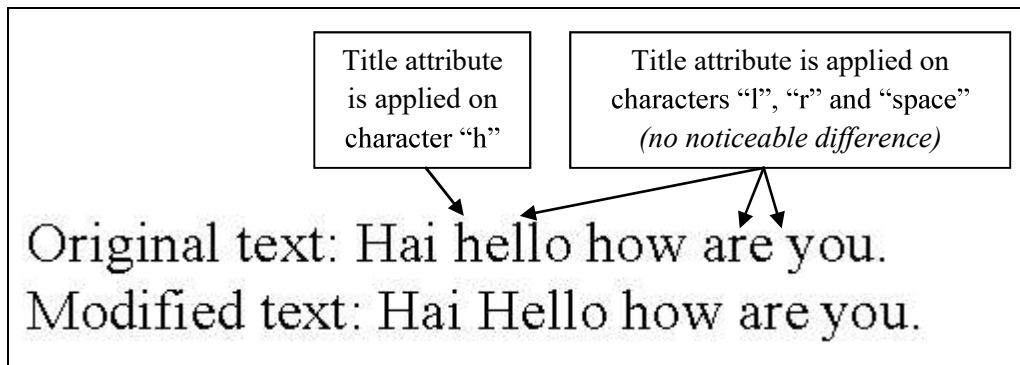
Default: Hai how are you. I am fine.	Default: Hai how are you. I am fine.
Style 1: Hai how are you. I am fine.	Style 11: Hai how are you. I am fine.
Style 2: Hai how are you. I am fine.	Style 12: Hai how are you. I am fine.
Style 3: Hai how are you. I am fine.	Style 13: Hai how are you. I am fine.
Style 4: Hai how are you. I am fine.	Style 14: Hai how are you. I am fine.
Style 5: Hai how are you. I am fine.	Style 15: Hai how are you. I am fine.
Style 6: Hai how are you. I am fine.	Style 16: Hai how are you. I am fine.
Style 7: Hai how are you. I am fine.	Style 17: Hai how are you. I am fine.
Style 8: Hai how are you. I am fine.	Style 18: Hai how are you. I am fine.
Style 9: Hai how are you. I am fine.	Style 19: Hai how are you. I am fine.
Style 10: Hai how are you. I am fine.	Style 20: Hai how are you. I am fine.

**Figure 3.7** Exploitation of the *StylisticSet* attribute

### ***Title***

Title is used to change the first character of every word of the selected alphabets to uppercase. Hence, applying this attribute on a non-alphabet or an already uppercase character produces no effect. Also, applying this attribute on any character, other than the first, of each word produces no noticeable effect (see Example 3.13).

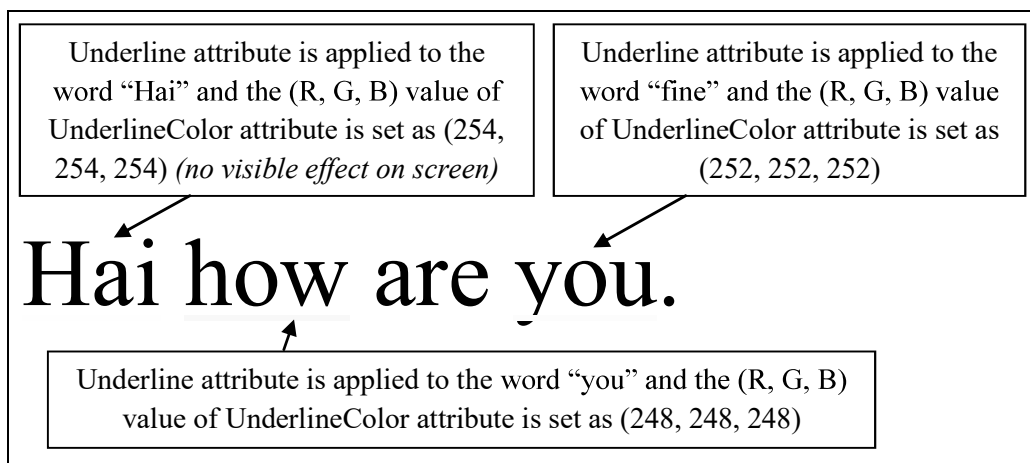
**Example 3.13:**



***Underline and UnderlineColor***

Underline attribute is used to underline the selected text. There are totally sixteen different styles of underline available. UnderlineColor attribute is used to set the color of line that should be used to underline. Setting the default background color of the document to UnderlineColor makes the underline invisible, except for those characters that have a descender (descender is the portion of a character that extends below the baseline [128]. For example, g, j, p, q and y). When this is done, altering the least significant (R, G, B) value of UnderlineColor attribute, also, goes unnoticed on screen (but noticeable in hard copy). The same is shown in Example 3.14.

**Example 3.14:**



### ***UpperCase***

Similar to AllCaps attribute, UpperCase attribute is used to change the selected alphabets to uppercase.

### **3.3 Classification of the Font Attributes**

From steganographic point of view, font attributes can be divided into two broad categories as *usable* and *unusable* (see Fig. 3.8). Attribute Duplicate is used to copy and apply the formatting of one text to another. This attribute cannot be altered. Similarly, attribute UpperCase can be applied to texts but cannot be exploited. Hence, they cannot be used in steganography and are considered as *unusable* attributes.

Based on the level of imperceptibility, the *usable* attributes can further be divided into three categories as *low*, *average* and *high imperceptible* attributes.

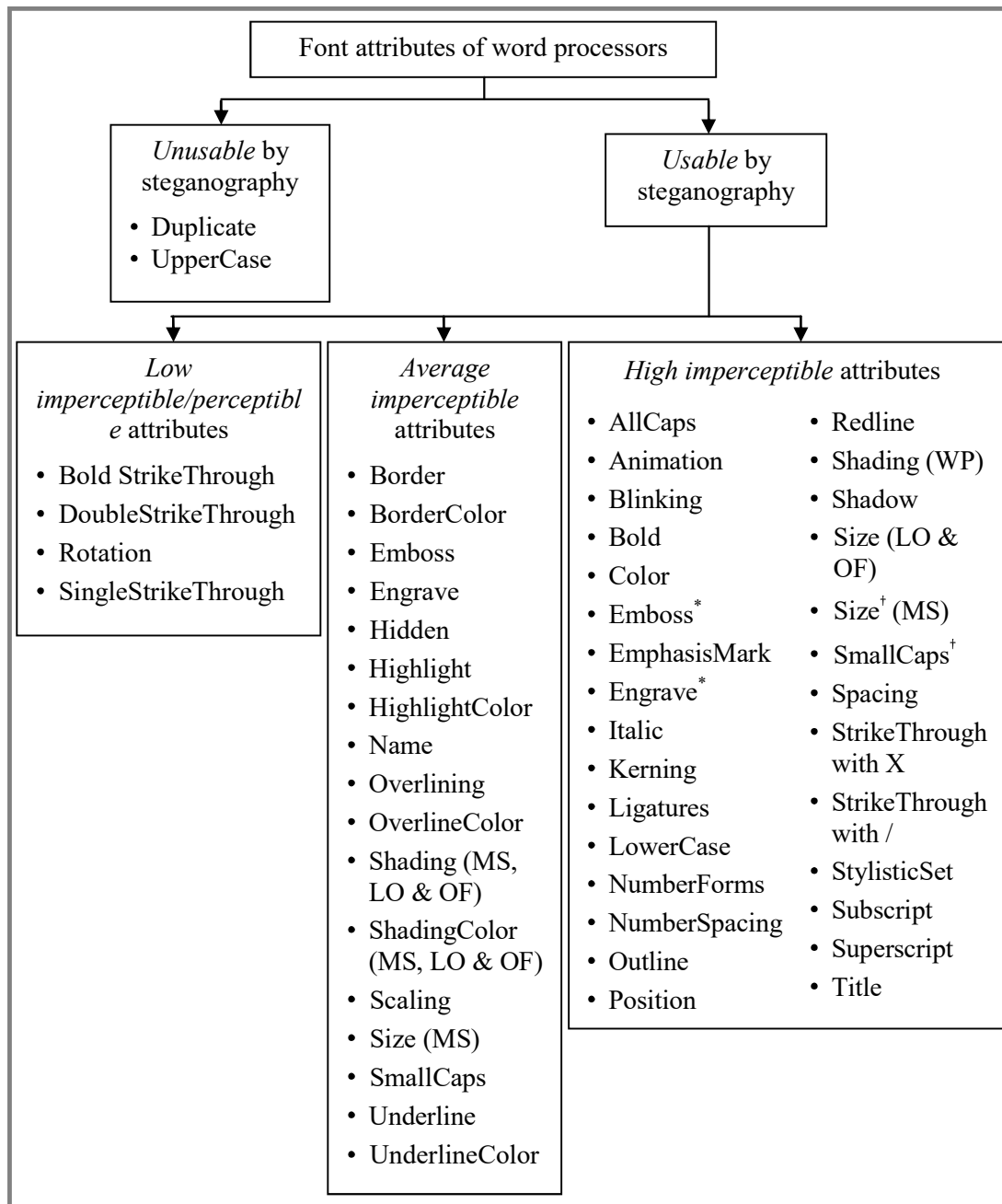
#### ***Low Imperceptible Attributes***

Attributes Bold StrikeThrough, DoubleStrikeThrough, Rotation and SingleStrikeThrough make perceptible changes. The characters that are modified using these attributes can easily be identified by a casual observer. Therefore, they are considered as *low imperceptible* attributes.

#### ***Average Imperceptible Attributes***

When the effect of the attribute Emboss is combined with the effect of the attributes Spacing and Position, it appears identical to that of the attribute Engrave (with minimal variation) and vice versa. Similarly, the attribute SmallCaps when combined with the attribute Size (MS Word), they produce the effect of AllCaps

attribute (with minimal variation). Without careful inspection, an observer cannot distinguish the effects produced by the corresponding attribute and by the combination.



**Figure 3.8** Classification of the font attributes of word processors. \* – represents the case that the attribute is applied on space character alone; † – represents the case that care has been taken to separate an identical modified and unmodified character; MS – Microsoft Word; LO – LibreOffice; OF – OpenOffice; WP – WordPerfect

Attribute Size (MS Word) create imperceptible alterations on characters. However, when an identical modified and unmodified character appears nearby, they can be distinguished. Attribute Scaling produces visible alterations on all characters. But, when used in a string of O's or 0's, the alteration cannot be distinguished without careful examination. Attribute Hidden completely hides the selected text without producing any visual effect. Similarly, the effect produced by the attribute Name, due to the non-existence of specified font style, becomes unnoticeable when the default font style is used to write the whole text content. But, exploitation of these two attributes can easily be identified using the available graphical user interface tools.

Also, though the attributes BorderColor, HighlightColor, OverlineColor, ShadingColor (MS Word, LO & OF) and UnderlineColor succeed in masking the effects of Border, Highlight, Overlining, Shading (MS Word, LO & OF) and Underline respectively, on screen, the same is perceptible in hard copy. Therefore, these attributes are considered as *average imperceptible* attributes.

### ***High Imperceptible Attributes***

Attributes AllCaps, Animation, Blinking, Bold, EmphasisMark, Italic, Outline, Position, Redline, Shadow and Title create perceptible alterations on characters. However, applying them on space character produces no visible effect. Similarly, the attributes LowerCase, NumberForms and NumberSpacing create visible modifications on uppercase characters and numbers. But, applying them on others does not produce any change.

Attribute StrikeThrough (with X or /) make noticeable changes on characters. But, when applied on space character, it produces effects indistinguishable from characters "X" and "/".

Although attribute Ligatures produce visible alterations to specific character pairs, applying this attribute on others produces no visual effect. Attributes Color, Kerning, Shading (WP), Size (LO & OF), Spacing and StylisticSet when varied within a specific range produces no visual change. For example, modification of the least significant 1 or 2-bits of the Color attribute. Also, when the attributes Subscript and Superscript are combined with the effect of the attribute Position, the changes produced are indistinguishable by the human visual system. Hence, even after careful inspection, the text that is modified using these attributes cannot be distinguished. Therefore, these kinds of attributes are considered as *high imperceptible* attributes.

Similar imperceptibility level can be obtained by the attributes Size (MS Word) and SmallCaps, if care has been taken to separate an identical modified and unmodified characters. That is, the farther the characters are, the more imperceptible they will be. Also, as mentioned earlier in Section 3.2, applying the attributes Emboss and Engrave on space character produces no visible effect. Hence, in such cases, these attributes can also be considered as *high imperceptible* attributes.

### 3.4 Comparison of the Font Attributes

The comparison of *average* and *high imperceptible* attributes (others are not considered due to their low imperceptibility) is provided in Tables 3.4 (A) and (B). The attributes are compared based on their embedding capacity. Embedding capacity is the amount of information that can be hidden in the chosen cover medium [36]. It, directly, depends upon two factors: (i) the number of bits that can be embedded in a given attribute; (ii) the usability of that attribute in a typical text document. The usability of an attribute further depends upon two factors: (i) the number of characters

in which the attribute can be applied; (ii) the occurrence frequencies of corresponding characters in a typical document.

**Table 3.4 (A)** Comparison of the *average* and *high imperceptible* attributes: part 1

Attribute	Usability in a typical text document	Embedding capacity		
		MS Word (2007 & 2010)	LO & OF	WP
AllCaps	Moderate	1-bit/UC & 1-bit/NaT	1-bit/UC & 1-bit/NaT	NA
Animation	Moderate	1-bit/SC	NA	NA
Blinking	Moderate	NA	1-bit/SC	NA
Bold, Italic, Outline and Shadow	Moderate	1-bit/SC	1-bit/SC	1-bit/SC
Border and BorderColor	High	3-bits/AC	3-bits/AC	3-bits/paragraph
Color	High	6-bits/AC & 24-bits/SC	6-bits/AC & 24-bits/SC	6-bits/AC & 24-bit/SC
Emboss	Moderate	1-bit/EnC & 1-bit/SC	1-bit/EnC & 1-bit/SC	NA
EmphasisMark	Moderate	2-bits/SC	NA	NA
Engrave	Moderate	1-bit/EmC & 1-bit/SC	1-bit/EmC & 1-bit/SC	NA
Hidden	High	High <sup>w</sup>	High <sup>w</sup>	High <sup>w</sup>
Highlight and HighlightColor	High	×	3-bits/ANBC	3-bits/AC
Kerning	High	10-bits/AC	×	3-bits/AC
Ligatures	High	2-bits/NNGC & 2-bits/NaT	NA	NA
LowerCase	High	×	1-bit/LC & 1-bit/NaT	×
Name	High	31-characters/AC	65288-characters/AC	×

*MS* – Microsoft; *LO* – LibreOffice; *OF* – OpenOffice; *WP* – WordPerfect; *NA* – represents the absence of attribute; *×* – represents the presence of attribute but cannot be exploited; *AC* – any character; *ANBC* – any character that does not appear on top and left borders of the page; *EmC* – embossed character; *EnC* – engraved character; *LC* – lowercase character; *NaT* – non-alphabet; *NNGC* – non-neighbor character; *SC* – space character; *UC* – uppercase character; *W* – depends on the number of visible characters in cover document



**Table 3.4 (B)** Comparison of the *average* and *high imperceptible* attributes: part 2

Attribute	Usability in a typical text document	Embedding capacity		
		MS Word (2007 & 2010)	LO & OF	WP
NumberForms	High	1-bit/NNC	NA	NA
NumberSpacing	High	1-bit/NNC	NA	NA
Overlining and OverlineColor	High	NA	7-bits/AC <sup>Z</sup>	NA
Position	Moderate	11-bits/SC	4-bits/SC	×
Redline	Moderate	NA	NA	1-bit/SC
Scaling	Low	1-bit/ZC	1-bit/ZC	NA
Shading and ShadingColor	High	3-bits/AC	3-bits/ANBC	3-bits/AC
Size	High	1-bit/AC	3-bits/AC <sup>X1</sup>	3-bits/AC <sup>X1</sup>
SmallCaps	Moderate	1-bit/UC & 1-bit/NaT	1-bit/UC & 1-bit/NaT	1-bit/UC & 1-bit/NaT
Spacing	High	3-bits/AC <sup>X1</sup>	3-bits/AC <sup>X1</sup>	3-bits/AC <sup>X1</sup>
StrikeThrough with X	Low	NA	1-bit/XC	NA
StrikeThrough with /	Low	NA	1-bit/SLC	NA
StylisticSet	High	3-bits/AC <sup>X1</sup>	NA	NA
Subscript and Superscript	Low (MS Word) & High (OF & LO)	1-bit/SPC & 1-bit/SBC	1-bit/AC & 4-bits/SC	×
Title	High	×	1-bit/UC & 1-bit/NFCW & 1-bit/NaT	×
Underline and UnderlineColor	High	7-bits/AND <sup>Y</sup>	7-bits/AND <sup>Y</sup>	7-bits/AND <sup>Y</sup>

MS – Microsoft; LO – LibreOffice; OF –OpenOffice; WP – WordPerfect; NA – represents the absence of attribute; × – represents the presence of attribute but cannot be exploited; AC – any character; AND- any character that does not have descender; ANBC – any character that does not appear on top and left borders of the page; NaT – non-alphabet; NFCW – non-first character in a word; NNC – non-numeric character; SC – space character; SBC- subscripted character; SLC – character “/”; SPC – superscripted character; UC – uppercase character; XC – character “X”; ZC – character “O” or “0”; X1 – eight variations are considered; Y – 4-bits are embedded by Underline style and 3-bits by UnderlineColor; Z - 4-bits are embedded by Overlining and 3-bits by OverlineColor

From Tables 3.4 (A) and (B), it can be seen that the attributes Scaling, StrikeThrough (with X or /) can only be exploited by applying them as a substitute for the characters O, 0, X and / respectively. Attributes Superscript and Subscript, of MS Word, can be used interchangeably to produce the effect of each other. It is clear that these attributes can be applied only on special characters whose occurrence frequencies are low (refer Table 2.1) in a typical text document. Hence, the usability of these attributes is limited, which in turn affects the overall embedding capacity.

Similarly, attributes like AllCaps, Animation, Blinking, Bold, Italic, Emboss, EmphasisMark, Engrave, Outline, Position, Redline, SmallCaps and Shadow can be applied on space character without creating any visual attention. Though these attributes can be applied only on single character, space, it is the highest occurring character in any document. Hence the usability of these attributes is considered as moderate. Also, among these attributes, Position attribute of MS Word, stands best in terms of embedding capacity as it can embed 11-bits/space-character.

Attributes like Border & BorderColor (MS Word, LO & OF), Color, Hidden, Highlight & HighlightColor, Kerning, Name, Overlining & OverlineColor, Shading & ShadingColor, Size, Spacing, StylisticSet, Subscript (LO & OF), Superscript (LO & OF) and Title can be applied on every character without creating any attention on screen. Attribute Ligatures can be applied on every non-neighbor character (including space) and attribute LowerCase can be applied on every lowercase character (including space) without causing any visual change. Also, attributes NumberForms and NumberSpacing can be applied on every non-numerical character. Since these attributes can be applied on majority of characters, their usability is considered as high.

Attributes Underline and UnderlineColor can be applied on every character except the lowercase characters g, j, p, q and y. But, the occurrence probabilities of

these five characters in English document are low. Hence, the usability of these attributes is also considered as high.

Among these highly usable attributes, Border & BorderColor, Color, Hidden, Kerning, Name, Shading & ShadingColor, Spacing, StylisticSet and Underline & UnderlineColor attributes of MS Word embeds minimum 3-bits/character. Similarly, Border & BorderColor, Color, Hidden, Highlight & HighlightColor, Name, Overlining & OverlineColor, Shading & ShadingColor, Size, Spacing, and Underline & UnderlineColor attributes of LO & OF embeds minimum 3-bits/character. Also, Color, Hidden, Highlight & HighlightColor, Kerning, Shading, Size, Spacing and Underline & UnderlineColor attributes of WP embeds minimum 3-bits/character. Hence the overall embedding capacity achievable by these attributes will also be high.

### **3.5 Discussion**

From Section 3.4, it is observed that a large number of attributes of various word processors achieve a high embedding capacity. But, attribute StylisticSet is available only in MS Word 2010 and higher versions. Attributes Border & BorderColor, Hidden, Highlight & HighlightColor, Name, Overlining & OverlineColor, Shading & ShadingColor (MS Word, LO & OF) and Underline & UnderlineColor have average imperceptibility level. Due to the above reasons, these attributes are not considered for further discussion.

The following attributes:

- (i) Color, Kerning and Spacing of MS Word
- (ii) Color, Size and Spacing of LO & OF
- (iii) Color, Kerning, Shading, Size and Spacing of WP

are found to be more suitable for steganographic purpose. This further shows that MS Word stands best as it can embed a maximum of 19-bits ( $6 + 10 + 3$ ), using only three attributes, in any given character. Also, these attributes are available in all versions. This makes the steganographic methods developed using these attributes compatible across various organizations as a majority of them utilize MS Word [66].

### **3.6 Summary**

A brief study on the font attributes of word processors, like MS Word, LO, OF and WP, was presented in this chapter. Attributes were classified based on their usability, for steganography, and imperceptibility level. Embedding capacity of these attributes and the usability of the corresponding word processor, in organizations, were considered to identify the best word processor along with the respective attributes. MS Word was found to be having the best attributes namely, Color, Kerning and Spacing.

In view of the above findings, Word document has been chosen as a cover medium for further study. However, the methodologies developed in this work can be adapted to other processors as well. Based on the work carried out in the present study, three novel methods have been developed to secretly embed information like text, image or binary data in any Word document. The first and second methods use the attribute Spacing, and the third method uses the attributes Color, Kerning and Spacing. These methods are described in detail in the subsequent chapters.

### EMBEDDING TEXT

---

*This chapter describes the method devised to embed secret text (containing English alphabets, dot and space characters) inside Microsoft (MS) word documents. A novel character-level embedding technique, referred to as Method-A, that marks the cover characters using the font attribute Spacing has been evolved and the same is described. The method so developed addresses the non-uniform embedding probabilities of secret characters and the overall low embedding capacity of these techniques. Method-A is assessed for its embedding capacity and uniformity in embedding probability. Various security features of Method-A have been discussed and a comparison with other existing methods is, also, provided.*

#### 4.1 Introduction

A secret text can be embedded inside a text document in three different ways namely character-level, bit-level and mixed-type embedding techniques (described in Chapter 2). Bit-level and mixed-type embedding techniques consider secret message as binary bits and subsequently embed them. Hence, even the best existing technique, UniSpaCh, requires four distortions ( $\approx 2$ -bits/distortion) to embed a character, which is of 8-bits. Character-level embedding techniques (CLET), on the other hand, consider secret message as characters and embed them accordingly. This enables CLET to embed a secret character, as a whole, in single distortion. This nature of embedding attracts CLET techniques to embed text.

Among the two CLET techniques, Cover\_Document\_Required (CDR) and Cover\_Document\_Not\_Required (CDNR), the latter methods are hard to manage.

This is because developing a generalized method which can generate a meaningful stego document that suits all scenarios is not easy. Cover\_Document\_Required (CDR) techniques can meet this purpose as they generate stego document from an existing cover document that is meaningful by its own. However, proper handling of low frequency characters and efficient utilization of available embedding space, to achieve high embedding capacity, are still challenging tasks in CDR techniques.

Though one cannot control the occurrence frequencies of characters in a given language (here English), it is still possible to change the way the low occurring characters are handled during embedding. For example, embedding a low occurring character in several other characters boosts its embedding probability and avoids wastage of embedding space during the process. Besides, an optimal embedding capacity is possible, only, by choosing an appropriate font attribute that can be applied on every cover character. Such attributes enable each cover character potential to carry a secret character.

Taking the above findings into consideration, a novel CDR method (referred to as Method-A, in the rest of the thesis) is developed which can embed secrets with high embedding capacity using the attribute Spacing. This attribute can be replaced by any attribute that has been found most suitable, for steganography, in Chapter 3. The development of Method-A is described below.

## **4.2 Handling the Non-Uniform Occurrence Frequencies of Characters**

The standard occurrence frequencies of characters in English text, that are available in the literature, does not consider the special characters “Dot” and “Space” (refer Table 2.1). However, these characters are mandatory for making the meaning out of a given text. Hence, a normal English text is considered and the occurrence

frequencies of English alphabets, dot and space (ADS) characters are identified and presented in Table 4.1.

**Table 4.1** Occurrence frequencies of ADS characters in English text

Occurrence probability	Character	Frequency	Occurrence probability	Character	Frequency
High	Space	20.30%	Low	G	1.69%
	E	9.63%		Y	1.55%
	T	7.56%		F	1.50%
	A	6.84%		Dot	1.39%
	O	6.31%		B	1.36%
	I	5.45%		C	1.21%
	S	5.06%		K	1.14%
Average*	H	4.97%		P	0.86%
	N	4.84%		V	0.78%
	D	3.93%		J	0.30%
	R	3.57%		Z	0.08%
	L	3.01%		X	0.06%
	U	2.27%		Q	0.05%
	W	2.21%			
	M	2.08%			

*ADS – English alphabets, Dot and Space; \* – Ideal occurrence probability =  $100 / 28 = 3.57$ . For experimental purpose, any values between 2 and 5 are considered as average occurrence probability*

From the table, it can be inferred that a typical CDR technique will require 2000 ( $100 / 0.05 = 2000$ ) cover characters to embed the character “Q”, and will require only five cover characters to embed the “Space” character. The difference in their embedding probabilities is due to the difference in their occurrence frequencies. As a result of this, embedding space (1999 cover characters in the case of “Q”) gets wasted while embedding low occurring characters.

To overcome the limitation and for attaining an optimal embedding capacity, the following methodology is adapted:

- (i) Increase the embedding probability of low occurring secret characters by embedding them across multiple cover characters

For example, embedding “Q”, in several different cover characters, say “S”, “G”, “K”, “P”, “H”, “E” and “M” will boost its cumulative embedding probability from 0.05% to 25.43%. This facilitates “Q” to require only four ( $100 / 25.43 \approx 4$ ) cover characters and get embedded even in its absence in the cover document.

- (ii) Make the embedding probability of all the secret characters uniform

This can be met by increasing the embedding probabilities of all the secret characters equivalent to or greater than that of the “Space” character (20.3%). Besides, a careful choice of characters into which a given secret character should be embedded, must be identified.

To achieve the above, first, the number of characters that must be cumulated (NCC) needs to be identified, and choices of characters must be recognized. The number is derived mathematically, and choices of characters are defined by introducing a novel idea called Frequency Normalization Set (FNS) and Character & String Mapping (CSM). FNS defines the choices of characters that must be cumulated and CSM describes the procedure of mapping these cumulated character strings to ADS characters.

### 4.3 Theoretical Background of the Development of Method-A

Let “P” be the ideal cumulative probability of embedding a secret character inside a cover character. Then, “P” can be defined as:

$$P = \left[ \frac{100}{\text{Number of ADS characters}} \right] \times \text{NCC} \quad (4.1)$$



The respective “P” values of various NCC values are listed in Table 4.2. From the table, it can be observed that a uniform embedding probability cannot be achieved when  $NCC < 6$ . Because, the achievable cumulative embedding probability still falls short of 20.3% (which is the embedding probability of “Space” character alone).

**Table 4.2** Respective NCC and P values

Number of characters cumulated (NCC Value)	Ideal cumulative probability of embedding a secret character inside a cover character (P Value)
1	3.57
2	7.14
3	10.71
4	14.29
5	17.86
6	21.43
7	25.00
8	28.57
9	32.14
10	35.71

At  $NCC = 6$ , the value of “P” is 21.43%, which is higher than 20.3% with a marginal increase of 1.13% ( $21.43 - 20.3\%$ ). This makes the choices of characters difficult, whenever the cumulated character string involves a “Space” character. Because the “Space” character itself contributes 20.3% to the “P” value, and leaves the mere 1.13% for the rest of the five characters.

These limitations are not present when  $NCC > 6$ . From the available values, in the present work, a value of seven is considered and substituting the value  $NCC = 7$  in equation 4.1 we get:

$$P = \frac{100}{28} \times 7 = 25 \quad (4.2)$$

which means that, on an average, out of four consecutive characters encountered in a cover document, a secret character would be embedded. That is, an average of 2-bits/cover-character will be embedded.

#### 4.4 Generation of Frequency Normalization Set (FNS)

The properties of Frequency Normalization Set (FNS), which decides the choices of characters that must be cumulated to achieve the uniform embedding probability, are defined and are as follows:

- (i) FNS contains 28 strings (equivalent to the number of ADS characters)
- (ii) Each string contains ADS characters
- (iii) Each string is of length seven, with seven positions or columns  
 $\{0, 1, 2, 3, \dots, 6\}$
- (iv) A character occurs only once in a string
- (v) A character occurs only once in a given position in the whole FNS. That is, no column-wise repetitions
- (vi) When the individual frequency of characters present in any string is summed up, it converges and falls close to the value 25

An algorithm has been designed to generate such a FNS (a flowchart of the developed algorithm and the necessary pseudo codes are provided in Appendix – A). The algorithm takes the occurrence frequencies of ADS characters, minimum and maximum allowed error (deviation from the value 25), etc., as inputs and generates a FNS as output. A sample FNS along with the respective cumulative frequency values are provided in Table 4.3.

**Table 4.3** Sample Frequency Normalization Set

Frequency Normalization Set (FNS)	Cumulative frequency	Frequency Normalization Set (FNS)	Cumulative frequency
WRZFOIN	23.96	MNSLAB.	24.58
F□CWQXP	26.19	TVIZM.A	24.18
JMFX Y□Z	25.87	SGKPHEM	25.43
XA.MNTU	25.04	HEUVIYJ	24.95
YCHIJOS	24.85	IYLNIAK	25.10
KUX□ZJC	25.36	OFJGCDE	24.57
□.BCPQX	25.23	VDEHKLY	25.01
BLOSWMI	25.48	LONQSUR	25.11
PJGALHO	23.98	QP□KXRV	26.76
NSVOBGH	25.01	EZDYGPT	25.30
UWY.TSL	23.05	.KQUVZ□	26.01
DIREFCQ	25.34	AXMTRVD	24.82
GHTR.WB	22.75	RTWBDFN	24.97
ZBADEFG	25.03	CQPJ□KW	26.07

“□” – represents “Space”; “.” – represents “Dot”

#### 4.5 Character & String Mapping (CSM)

CSM maps the generated 28 strings of FNS to the 28 possible ADS characters, which limit the secret message to have only ADS characters. The characters are mapped in such a way that:

- (i) The mapped character does not exist in the selected string
- (ii) Map a high frequency character to a string that contains at least 2-low, 1-average and 1-high frequency characters
- (iii) Map an average or a low frequency character to a string that contains at least 1-low, 1-average and 1-high frequency characters
- (iv) The mapping should not make any cover character to carry more secret characters. For example, mapping several high frequency characters to

strings containing “Space” will make the “Space” character to carry more secret characters which must be avoided

Mapping the ADS characters in this manner will distribute the secret character across all the seven possible characters of the mapped string. This avoids the possibility of a particular cover character, say the highest frequency character, carrying more secret characters. A sample mapping is provided in Table 4.4.

**Table 4.4** Sample Character & String Mapping

Mapped character	Frequency Normalization Set (FNS)	Cumulative frequency	Mapped character	Frequency Normalization Set (FNS)	Cumulative frequency
A	WRZFOIN	23.96	O	MNSLAB.	24.58
B	F□CWQXP	26.19	P	TVIZM.A	24.18
C	JMFXY□Z	25.87	Q	SGKPHEM	25.43
D	XA.MNTU	25.04	R	HEUVIYJ	24.95
E	YCHIJOS	24.85	S	IYLNIAK	25.10
F	KUX□ZJC	25.36	T	OFJGCDE	24.57
G	□.BCPQX	25.23	U	VDEHKLY	25.01
H	BLOSWMI	25.48	V	LONQSUR	25.11
I	PJGALHO	23.98	W	QP□KXRV	26.76
J	NSVOBGH	25.01	X	EZDYGPT	25.30
K	UWY.TSL	23.05	Y	.KQUVZ□	26.01
L	DIREFCQ	25.34	Z	AXMTRVD	24.82
M	GHTR.WB	22.75	□	RTWBDNF	24.97
N	ZBADEFG	25.03	.	CQPJ□KW	26.07

“□” – represents “Space”; “.” – represents “Dot”

#### 4.6 Embedding Algorithm

Embedding algorithm of Method-A takes a secret message, cover document and CSM as input values. The cover document is checked for its size first, as the method requires an average of four cover characters to embed a secret character.

Embedding begins by selecting the first character from the secret message. The string that is mapped to the selected character, in the CSM, is identified. Embedding algorithm searches, the cover document serially, for the first occurrence of any of the characters of the mapped string. When a match is found, the position of the encountered cover character in the mapped string is identified. Based on the position, the cover character is marked by altering the value of its attribute Spacing.

As each string of CSM has seven positions, the attribute requires seven different spacing values to mark a character. From a detailed analysis, it was observed that the value of the attribute can be made to expand till 0.4 points (+0.4) or condense till 0.3 points (-0.3), at an interval of 0.05 points, without creating any visual attention (as discussed in Chapter 3 and shown in Table 3.3). For experimental purpose, the spacing values of  $\{-0.1, -0.2, -0.3, +0.1, +0.2, +0.3, +0.4\}$  points are considered, respectively, and a sample embedding is provided in Example 4.1.

**Example 4.1:**

Secret character	Mapped string in CSM	Cover document	Encountered character	Position of the encountered character in the mapped string	Modification to be performed
Q	SGKPHE M	Cool morning all.	Cool <span style="border: 1px solid black;">m</span> orning all.	6	Expand the spacing value of “m” by 0.4 points (+0.4)

After embedding a secret character, all the strings of CSM are circular left shifted by one. This avoids the possibility of embedding the same secret character inside an identical cover character with the same spacing value, frequently. The advantage of doing this is explained later in Section 4.9.2.

The procedure is continued till all the secret characters are embedded. After this, the method embeds *End-of-Secret* (EoS) characters. EoS can be anything that

does not appear in the secret message. For experimental purpose, it has been considered as “. . .” → “Dot Space Dot Space Dot”. During extraction process, these characters indicate the receiver that the end of the embedded secret is reached.

The pseudo code of the embedding procedure is as follows:

#### **Pseudo code of embedding procedure**

---

**Input:** Secret\_Message, Cover\_Work, CSM, EoS characters, character spacing and their respective positions

**Output:** Modified\_Cover\_Work

---

```
Int count ← 1
Secret_Message ← Secret_Message + EoS Characters
For each character “Xi” in Secret_Message do
    String Sk ← String that is mapped to Xi in CSM
    L: Yj ← Read the character @ position “count” in Cover_Work
    count++
    If Yj ∈ Sk
        Int pos ← Position of Yj in Sk
        Change the character spacing of Yj based on pos
    End if
    Else Goto L
    CSM ← Perform a Circular Left Shift on all the Strings in CSM
    //makes the CSM dynamic
End for
Return Modified_Cover_Work
```

---

The modified cover document is the required stego document which has to be communicated to the receiver. In addition, the used CSM and EoS characters should be communicated.

#### **4.7 Extraction Algorithm**

The extraction process is the reverse of the embedding process. The method takes the stego document, EoS and the CSM as input values. Extraction begins by

checking the spacing value of characters in the stego document. When the algorithm encounters a character, whose spacing value is altered, it recognizes the position equivalent to the spacing value. Then, the algorithm searches for the string, in CSM, that contains the encountered character at the recognized position. The ADS character that is mapped to the identified string is the embedded secret character (see Example 4.2).

**Example 4.2:**

Stego document	Encountered character	Identified spacing value	Position equivalent to the spacing value	Mapped string in CSM	Embedded secret character
Cool morning all.	Cool <span style="border: 1px solid black;">m</span> orning all.	+0.4 (Expanded by 0.4 points)	6	SGKPHE M	Q

After extracting a secret character, the CSM is circular left shifted by one, and the procedure is continued till the algorithm extracts the EoS characters.

The pseudo code of the extraction procedure is given below:

**Pseudo code of extraction procedure**


---

**Input:** Stego\_Work, CSM, EoS characters, character spacing and their respective positions

**Output:** Secret\_Message

---

Int count  $\leftarrow$  1

String Secret\_Message  $\leftarrow$  ""

Repeat

Char  $X_i \leftarrow$  Read the character @ position "count" in Stego\_Work

count++

If character spacing of  $X_i \neq 0.0$

Int pos  $\leftarrow$  Respective position, based on the character spacing of  $X_i$

String  $S_k \leftarrow$  String in CSM that has  $X_i$  @ position pos

Secret\_Message  $\leftarrow$  Secret\_Message + Character that is mapped to  $S_k$

in CSM

```
        CSM ← Perform a Circular Left Shift on all the Strings in CSM
    End if
Until ((count > Total no. of characters in Stego_Work) || (EoS is read))
Return Secret_Message
```

---

It should be mentioned that reaching the end of the stego document without encountering the EoS characters indicates the receiver, that a corrupted stego document has been received.

## 4.8 Evaluation Parameters

The method so developed, Method-A, is evaluated using the following three parameters: secrecy, embedding capacity and uniformity in embedding probability. Each of the parameter is discussed below in some detail. In addition to these, a comparison with the existing methods is provided.

### 4.8.1 Secrecy

Secrecy represents the imperceptibility level of the embedded secret [63,65,129,130]. To test the imperceptibility level, secret messages of various lengths are experimented using Method-A. A sample output is provided in Fig. 4.1.

*Cover document:*

INTRODUCTION: Internet which is extensively used to share any kind of information does not imply any strict rules for the security of data on its own.

*Secret message:* Come to my home tomorrow.

*Stego document:*

INTRODUCTION: Internet which is extensively used to share any kind of information does not imply any strict rules for the security of data on its own.

**Figure 4.1** Sample output of Method-A



By observing both the cover and stego documents of Fig. 4.1, it is evident that the stego document does not create any attraction in its visual appearance.

#### 4.8.2 Embedding Capacity

As explained earlier, embedding capacity is the measure of the maximum size of secret that a chosen cover document can hide [36,65]. Hence, in general, embedding capacity can be defined as in equation 4.3.

$$\text{Embedding capacity per cover – character} = \frac{\text{No. of bits in secret message}}{\text{No. of characters required in cover document}} \quad (4.3)$$

As Method-A considers the secret message as characters (instead of bits), equation 4.3 can be rewritten as:

$$\text{Embedding capacity per cover – character} = \frac{\text{No. of characters in secret message (excluding EoS)}}{\text{No. of characters required in cover document}} \times 8 \quad (4.4)$$

To evaluate the embedding capacity per cover-character exercise was carried out by embedding secret messages, of various lengths, inside a given cover document. Table 4.5 furnishes the number of characters present in the secret message and the number of characters used in the cover document to embed the same.

From Table 4.5, it can be observed that Method-A achieves an average embedding capacity of  $2.22 \pm 0.05$  bits/cover-character which is slightly higher than the theoretically expected value of 2-bits/cover-character.

**Table 4.5** Results of embedding the secrets in cover document

No. of characters in secret message (excluding EoS)	No. of characters encountered while embedding secret + EoS characters					Embedding capacity per cover-character *
	Alphabet	Dot	Space	Other	Total	
500	1484	13	282	8	1787	2.24
1000	2842	27	556	12	3437	2.33
1500	4535	48	864	26	5473	2.19
2000	5999	63	1139	32	7233	2.21
2500	7612	85	1461	45	9203	2.17
3000	9238	104	1769	59	11170	2.15
3500	10459	116	2032	83	12690	2.21
4000	11736	130	2287	100	14253	2.25
4500	13403	146	2608	114	16271	2.21
5000	14848	164	2879	128	18019	2.22

\* – using equation 4.4; EoS – End-of-Secret

### 4.8.3 Uniformity in Embedding Probability

A good CDR technique must handle the secret characters uniformly and should maintain uniform embedding probability. Achieving such uniformity will avoid the wastage of embedding space and will reduce the size of required cover document.

Uniformity of a method can be tested by embedding a secret message that contains all the possible characters, in large numbers, inside an English cover document. Since the occurrence frequencies of characters are not uniform, creating such a secret message and embedding it, is a tedious task. Hence, for experimental purpose, a random string of ADS characters (of length 5000) that satisfies the above requirement is generated, and considered as a secret message. It is, then, embedded inside an English cover document.

The total number of occurrences of each character in the secret message, and the average number of cover characters used to embed them are provided in Table 4.6.

**Table 4.6** Uniformity in embedding probability

Characters	Secret message (5000 + End-of-Secret characters)		Characters	Secret message (5000 + End-of-Secret characters)	
	No. of times occurred	Average no. of cover characters used to embed one secret character		No. of times occurred	Average no. of cover characters used to embed one secret character
A	184	4.01	O	180	4.00
B	173	3.57	P	202	4.33
C	176	3.70	Q	172	3.19
D	159	3.87	R	174	3.18
E	155	3.14	S	173	4.02
F	205	3.62	T	192	3.01
G	184	3.55	U	192	3.94
H	196	3.67	V	173	3.31
I	164	4.24	W	179	3.16
J	172	3.62	X	171	3.42
K	166	4.27	Y	191	3.82
L	174	3.05	Z	160	3.82
M	194	4.36	Space	189	3.39
N	180	3.58	Dot	175	3.58

From the table, it can be observed that the average number of cover characters required to embed any secret character uniformly falls within the range  $3.66 \pm 0.4$ . This shows that the developed method utilizes the available embedding space efficiently.

#### 4.8.4 Comparison with Existing Methods

A comparison of Method-A with the existing methods is provided in Table 4.7 (calculations are done similar to Section 2.3). The comparison is carried out in terms of number of distortions and number of cover characters that are required to embed a secret character.

**Table 4.7** Comparison of Method-A with existing techniques

Technique	Requirement to embed a secret character	
	No. of distortions	No. of cover characters (approximate)
Character marking, Misspelling	1	Variable
Null cipher	1	5.5
Missing letter puzzle	1	11.5
Hiding data in wordlist	1	11.5
Synonym substitution, Spelling of words	8	44
Line shifting	8	1020
Word shifting	8	100
Inter-sentence spacing	8	668
Inter-word spacing	8	44
End-of-line spacing	4	224
UniSpach	4	7.65
Moving “The Dot” in characters	8	8
Exploiting the structure of characters	8	16
Reversing/Removing the diacritics in/from characters	8	8
“Change Tracking” technique	Variable	133
Generating summary	4	334
Method-A	1	3.6

From Table 4.7, it can be observed that Method-A records the least number of cover characters required to embed a secret character. In addition, it stands best by

making only one distortion to embed a secret character. It is worth mentioning that UniSpaCh, one of the best available methods requires four distortions.

## **4.9 Security Aspect**

Method-A embeds an English secret text inside an English cover text. Due to the non-uniform occurrence of characters, an attacker can try to break the system and identify the characters by using the variation in their occurrence frequencies. This is known as frequency analysis attacks [38,131]. Bearing this in mind, Method-A has been developed with some in-built security features to resist such attacks. It, also, facilitates the ways to enhance the security by combining it with other existing methods which are discussed below.

### **4.9.1 Uniformity in Embedding Probability**

As mentioned, the non-uniform occurrence of characters cannot be controlled. Hence, a secret character having high embedding probability will get embedded in fewer chances, whereas the other may require more cover characters to get embedded. An attacker can break the method, using the frequency analysis attacks, which follows such variation. Method-A avoids such attacks by maintaining the embedding probabilities of all the characters uniform (refer Section 4.8.3).

### **4.9.2 Distribution in Stego Characters**

Method-A embeds the secrets by altering the attribute Spacing. This makes the secret characters to get distributed among the seven possible levels  $\{-0.1, -0.2, -0.3, +0.1, +0.2, +0.3, +0.4\}$  of a cover character. An attacker can try to analyze the distribution of characters among these levels. That is, the presence of high variations

among the levels of a particular stego character can represent the possibility of carrying high frequency secret characters.

Method-A avoids such analysis by performing a circular left shift on the used CSM (the embedding procedure is described in Section 4.6). Doing so distributes the secret characters across the different levels of a cover character and thereby avoids the occurrence of high values at any level of a stego character. This prevents the method from producing such high variations.

The same has been verified by embedding an English secret message, of length 5000 (excluding EoS), inside an English cover document. The distribution of secret characters among the seven levels of each stego character is identified and provided in Table 4.8.

From the table, it is evident that the method distributes the secret characters, almost uniformly, across all the possible levels of a cover character. It should be noted that the variation in distribution of characters in “P”, “Dot”, “K”, “V”, “X”, “Z”, “Q” and “J” is due to their low occurrences in cover document. That is, the low occurrence of these characters made them to carry less secret characters, which resulted in high standard deviation values.

### **4.9.3 Frequency Distribution of Stego Characters**

An attacker can try to gain knowledge, about the hidden secret, by performing the frequency analysis on the stego characters. Suppose, if the frequency profile of stego characters follow the frequency profile of secret characters, then the presence of high peaks at both profiles represent the correlation between them.

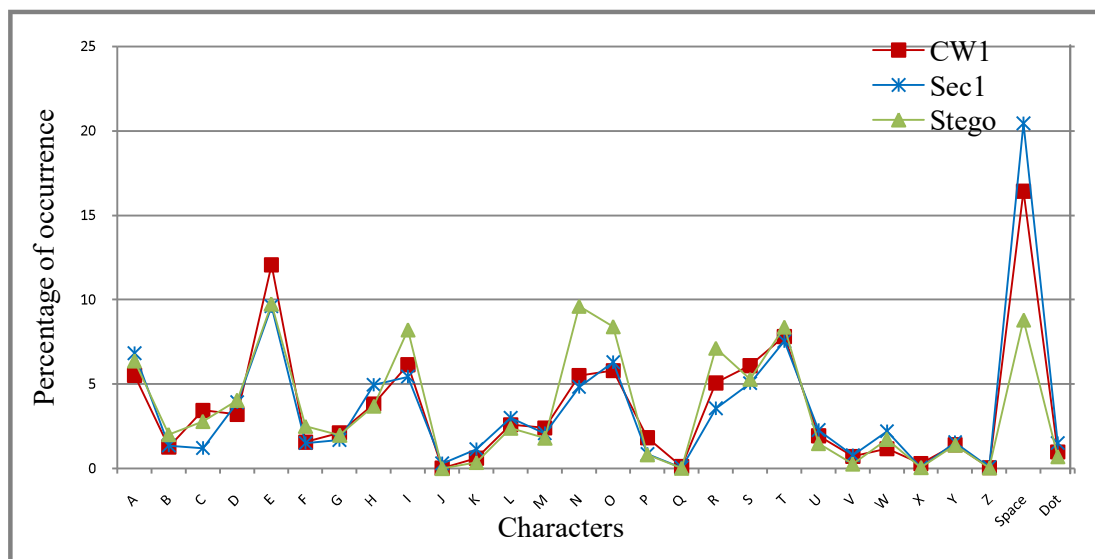
**Table 4.8** Spacing values of the identified stego characters

Possible stego characters	No. of times identified as stego character	Identified spacing values at each level (in %)							Standard deviation
		-0.1	-0.2	-0.3	+0.1	+0.2	+0.3	+0.4	
E	625	15.68	13.76	13.12	13.60	14.56	12.96	16.32	1.29
N	616	14.61	14.45	13.15	18.34	13.80	11.85	13.80	2.01
Space	564	15.96	14.36	15.25	13.30	13.83	14.00	13.30	1.00
O	539	11.88	13.73	13.17	16.88	15.77	15.03	13.54	1.70
T	537	13.59	10.99	14.15	16.76	14.34	14.34	15.83	1.82
I	527	16.70	12.14	18.60	12.90	11.77	14.23	13.66	2.51
R	457	12.25	15.32	16.41	15.75	13.57	13.13	13.57	1.54
A	409	13.94	14.43	11.98	15.89	12.96	16.38	14.42	1.54
S	339	15.93	11.51	14.16	15.04	14.75	13.57	15.04	1.43
D	259	11.20	11.97	15.83	13.90	14.28	15.83	16.99	2.13
H	237	13.93	16.03	16.03	11.82	15.61	13.50	13.08	1.64
C	179	20.11	16.20	11.17	13.97	14.52	13.97	10.06	3.30
F	160	13.75	11.88	18.12	11.25	13.12	17.50	14.38	2.64
L	153	15.03	15.03	13.73	15.69	13.73	11.76	15.03	1.33
B	129	12.40	10.85	17.06	15.50	8.53	20.16	15.50	3.95
G	127	11.02	14.17	7.88	14.17	16.54	18.90	17.32	3.81
M	116	16.38	17.24	11.21	14.66	8.62	17.24	14.65	3.26
W	111	10.81	12.61	12.61	16.22	15.32	17.12	15.31	2.29
U	95	13.69	20.00	12.63	10.53	13.68	12.63	16.84	3.15
Y	89	15.73	13.49	17.98	13.48	13.48	13.48	12.36	1.92
P	52	9.62	9.62	15.38	26.92	9.62	15.38	13.46	6.16
Dot	45	13.34	20.00	13.33	22.22	8.89	2.22	20.00	7.13
K	23	8.69	8.70	4.35	21.74	17.39	21.74	17.39	6.97
V	17	11.77	5.88	29.41	17.65	5.88	17.65	11.76	8.22
X	4	0	0	0	0	25.00	50.00	25.00	19.67
Z	2	0	50.00	50.00	0	0	0	0	24.40
Q	1	0	0	0	100	0	0	0	37.80
J	0	-	-	-	-	-	-	-	-

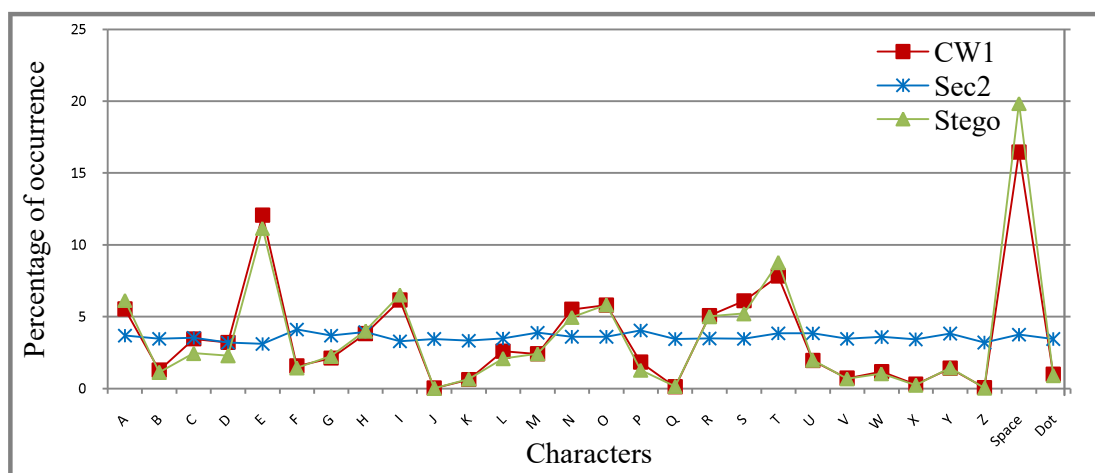
The rules of CSM are designed to prevent Method-A from such attacks. It does so, by distributing the secret characters across all the seven possibilities of the

mapped string. This avoids the occurrence of high peaks in the frequency profile of stego characters (refer Fig. 4.2).

To study the correlation between the frequency profile of secret and stego characters, English and random secret messages (of length 6407 and 5000 respectively) are embedded inside English and random cover documents. Figures 4.2, 4.3 and 4.4 illustrate the occurrence frequencies of characters in the secret message, cover document and stego characters.

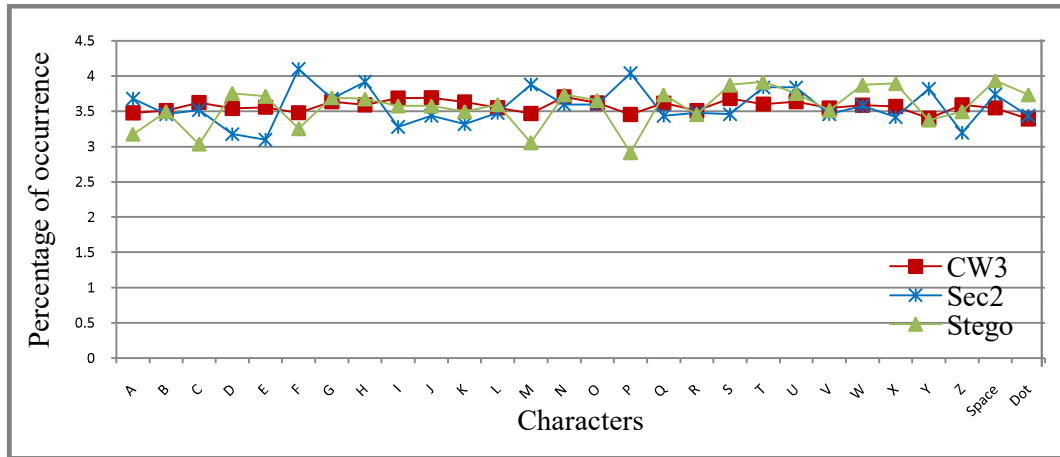


**Figure 4.2** English secret in English cover document. CW1 – cover document; Sec1 – secret message; Stego – stego document



**Figure 4.3** Random secret in English cover document. CW1 – cover document; Sec2 – secret message; Stego – stego document





**Figure 4.4** Random secret in random cover document. CW3 – cover document; Sec2 – secret message; Stego – stego document

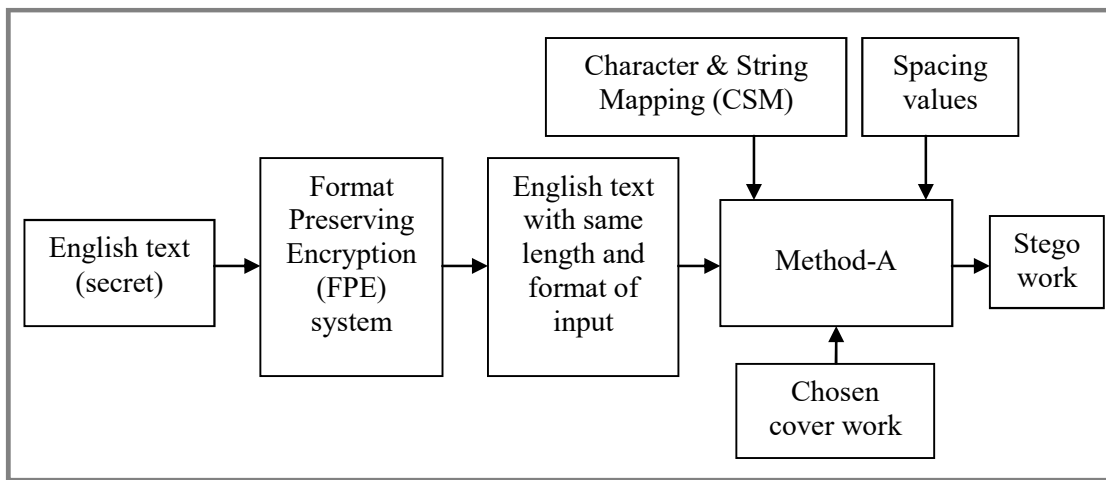
From Figures 4.2, 4.3 and 4.4, it can be observed that the frequency profile of stego character is predominantly due to the frequency profile of cover document but not due to that of secret message. Hence, performing such attacks, on the stego characters of Method-A, will not provide the expected information to an attacker.

#### 4.9.4 Cryptographic Aspect

Though the aim of the present study is to develop best steganographic methods with high embedding capacity and bits/distortion, it is worth mentioning that the formulated method has some in-built security features comparable with the cryptographic techniques. Also, it allows the method to be combined with existing cryptographic techniques to increase the security further. This has been emphasized in this section.

With the help of CSM, Method-A embeds a secret character in several cover characters. Hence, when the used CSM is kept secret, a security level comparable with that of a polyalphabetic substitution ciphers of cryptography [38] can be expected.

Also, combining Method-A with Format Preserving Encryption (FPE) system [132] can further enhance the confidentiality of the embedded secret. This combination is feasible as FPE preserves the length and format of given input, unlike other cryptographic techniques [133]. Thereby, the original secret text containing ADS characters must be encrypted, using FPE, first. The generated cipher text which is again a string of ADS characters of same length, can then be considered as secret and embedded using Method-A (see Fig. 4.5).



**Figure 4.5** Method-A combined with Format Preserving Encryption system

To break this type of dual security, first, an adversary has to identify the presence of hidden message and extract it. After this, he/she has to break the FPE security system in order to get the original secret message. Hence, the combined system can provide a greater challenge to an adversary than when the FPE and Method-A are individually applied.

#### 4.10 Application to Case-Sensitive Letters

The method so developed, Method-A, is not case-sensitive. However, it can be achieved by utilizing another attribute, to differentiate the case, or by following the procedure mentioned below.

The number of characters in each string of CSM must be extended to fourteen by representing them in both upper- and lowercase letters (refer Table 4.9). It is well-known that the characters “Space” and “Dot” are not case-sensitive. However, to make this method feasible, they are treated as case-sensitive.

**Table 4.9** Sample Character & String Mapping for case-sensitive messages

Mapped character	Frequency Normalization Set (FNS) <sup>†</sup>	Mapped character	Frequency Normalization Set (FNS) <sup>†</sup>
A/a	WwRrZzFfOoLiNn	O/o	MmNnSsLlAaBb●.
B/b	Ff■□CcWwQqXxPp	P/p	TtVvIiZzMm●.Aa
C/c	JjMmFfXxYy■□Zz	Q/q	SsGgKkPpHhEeMm
D/d	XxAa●.MmNnTtUu	R/r	HhEeUuVvIiYyJj
E/e	YyCcHhIiJjOoSs	S/s	IiYyLlNnUuAaKk
F/f	KkUuXx■□ZzJjCc	T/t	OoFfJjGgCcDdEe
G/g	■□●.BbCcPpQqXx	U/u	VvDdEeHhKkLlYy
H/h	BbLlOoSsWwMmIi	V/v	LlOoNnQqSsUuRr
I/i	PpJjGgAaLlHhOo	W/w	QqPp■□KkXxRrVv
J/j	NnSsVvOoBbGgHh	X/x	EeZzDdYyGgPpTt
K/k	UuWwYy●.TtSsLl	Y/y	●.KkQqUuVvZz■□
L/l	DdIiRrEeFfCcQq	Z/z	AaXxMmTtRrVvDd
M/m	GgHhTtRr●.WwBb	■/□	RrTtWwBbDdNnFf
N/n	ZzBbAaDdEeFfGg	●./	CcQqPpJj■□KkWw

<sup>†</sup> – uppercase “Space” and “Dot” are represented by “■” and “●”, and lowercase “Space” and “Dot” are represented by “□” and “.” respectively

The resulting fourteen different positions in CSM must be represented using fourteen different spacing values  $\{-0.05, -0.1, -0.15, -0.2, -0.25, -0.3, +0.05, +0.1, +0.15, +0.2, +0.25, +0.3, +0.35, +0.4\}$ .

Now, to embed a secret character, the cover document is searched for the occurrence of any of the characters in the corresponding CSM string. When a match is found, the case of the secret character along with the encountered cover character is used to identify the corresponding spacing value (by default, the characters “Space”

and “Dot” are considered as lowercase). That is, if the secret character is in uppercase then the spacing value corresponding to the uppercase of the encountered cover character is marked and vice-versa. A sample embedding is shown below in Example 4.3.

**Example 4.3:**

Secret character	Mapped string in CSM	Cover document	Encountered character	Case of secret character	Position of the encountered character in the mapped string*	Modification to be performed
Q	SsGgKk PpHhEe Mm	Cool morning all.	Cool <span style="border: 1px solid black;">m</span> orning all.	Upper case	12 (“M”)	Expand the spacing value of “m” by 0.35 points (+0.35)
q	SsGgKk PpHhEe Mm	Cool morning all.	Cool <span style="border: 1px solid black;">m</span> orning all.	Lower case	13 (“m”)	Expand the spacing value of “m” by 0.4 points (+0.4)

\* – based on the case of secret character

The extraction procedure is straight-forward and is the reverse of the embedding process. The spacing value of the identified stego character defines the secret character along with its case.

#### 4.11 Summary

CDR techniques embed secret information directly inside cover documents by marking them. This makes these methods an optimum choice for embedding text as it embeds 8-bits/distortion. However, due to the non-uniform occurrence probabilities of characters in cover document, the available embedding space gets wasted whenever a low occurring character needs to be marked. This affects the overall embedding capacity of these techniques.

With an aim to address this limitation, the necessary measures that need to be taken are identified and a Frequency Normalization Set (FNS) in combination with Character & String Mapping (CSM) are introduced. The combination efficiently handled the low occurring characters, by embedding them in multiple cover characters, and made the embedding probabilities of all the characters uniform. This allowed the method (Method-A) to achieve an average embedding capacity of 2.22-bits/cover-character, with 8-bits/distortion, which is slightly higher than the theoretically expected value of 2-bits/cover-character. Hence, the size of the required cover document and the number of modifications that are performed in the document gets reduced.

Method-A alters the attribute Spacing to mark the cover characters. The imperceptible changes made in the cover document ensured high secrecy and hence created no attraction in their visual appearance. As this attribute can be applied even on non-English characters, Method-A is not restricted to any particular language.

A security level comparable with that of a polyalphabetic substitution cipher of cryptography is expected when the used CSM is considered as a secret key. The use of the Format Preserving Encryption system to further enhance the security has also been described. In addition to these, various in-built security features that prevent Method-A from well-known frequency analysis attacks have also been discussed.

Though Method-A has the above-mentioned advantages, it restricts the secret message to contain only ADS characters. Due to this limitation, it cannot be used to embed:

- (i) messages that contain numbers and special characters viz. mobile, credit card, debit card, etc.
- (ii) binary and multimedia data like image, audio, video, etc.

However, these limitations can be overcome by extending the method into a mixed-type embedding technique which is discussed in the next chapter.

### EMBEDDING BINARY DATA

---

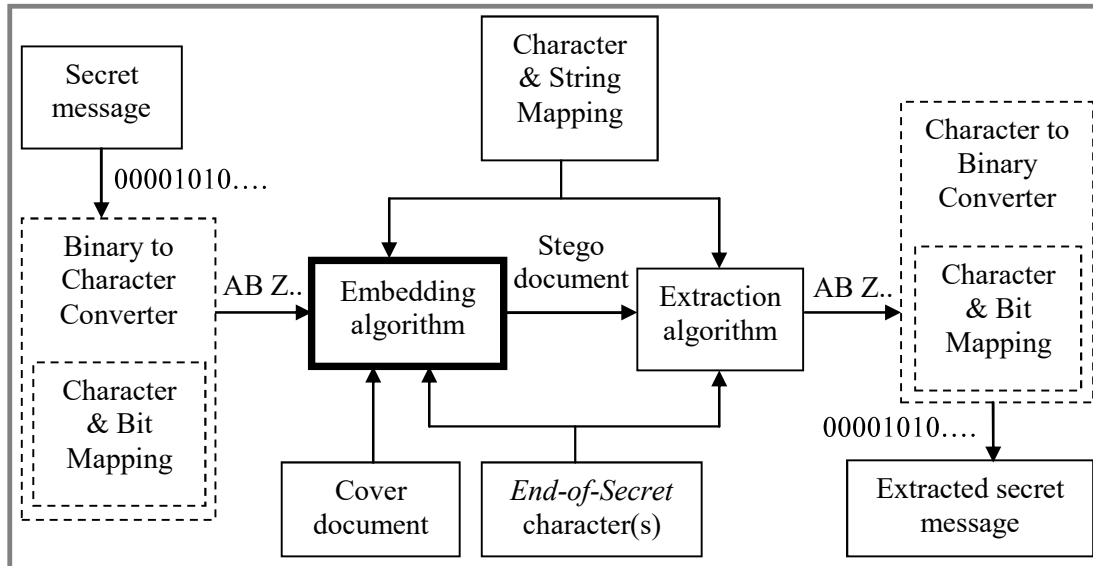
*This chapter describes the method developed to embed binary data inside Microsoft (MS) Word documents. It extends the method described in Chapter 4, Method-A, into a mixed-type embedding technique, referred to as Method-B, that embeds binary data (secret) using the attribute Spacing. Method-B is assessed for its embedding capacity and uniformity in embedding probability. A comparison with one of the best existing methods available in the literature is provided. Various security aspects of Method-B have been discussed and a case study using images related to nuclear power plants has, also, been conducted.*

#### 5.1 Introduction

The method (Method-A) developed in the present study is successful in embedding the secret message and communicating the same efficiently using smaller size cover documents. But, it restricts the messages to contain only English alphabets, Dot and Space (ADS) characters. This limits the method from communicating other kinds of message such as text with special characters and numbers. In addition, it is not possible to embed messages such as multimedia data, compressed and encrypted data, etc.

As these data types are also commonly used and important, there is a need for communicating the same in a secure manner. Method-A is modified to accommodate the above-said data types. This was achieved by extending the method to a mixed-type embedding technique (referred to as Method-B, in the rest of the thesis) which embeds binary data.

For this purpose, a new module named “Binary to Character Converter” (BCC) has been introduced. This module takes the secret, binary data, as input and generates an equivalent character stream which can, then, be embedded by Method-A (refer Fig. 5.1). The method is described below.

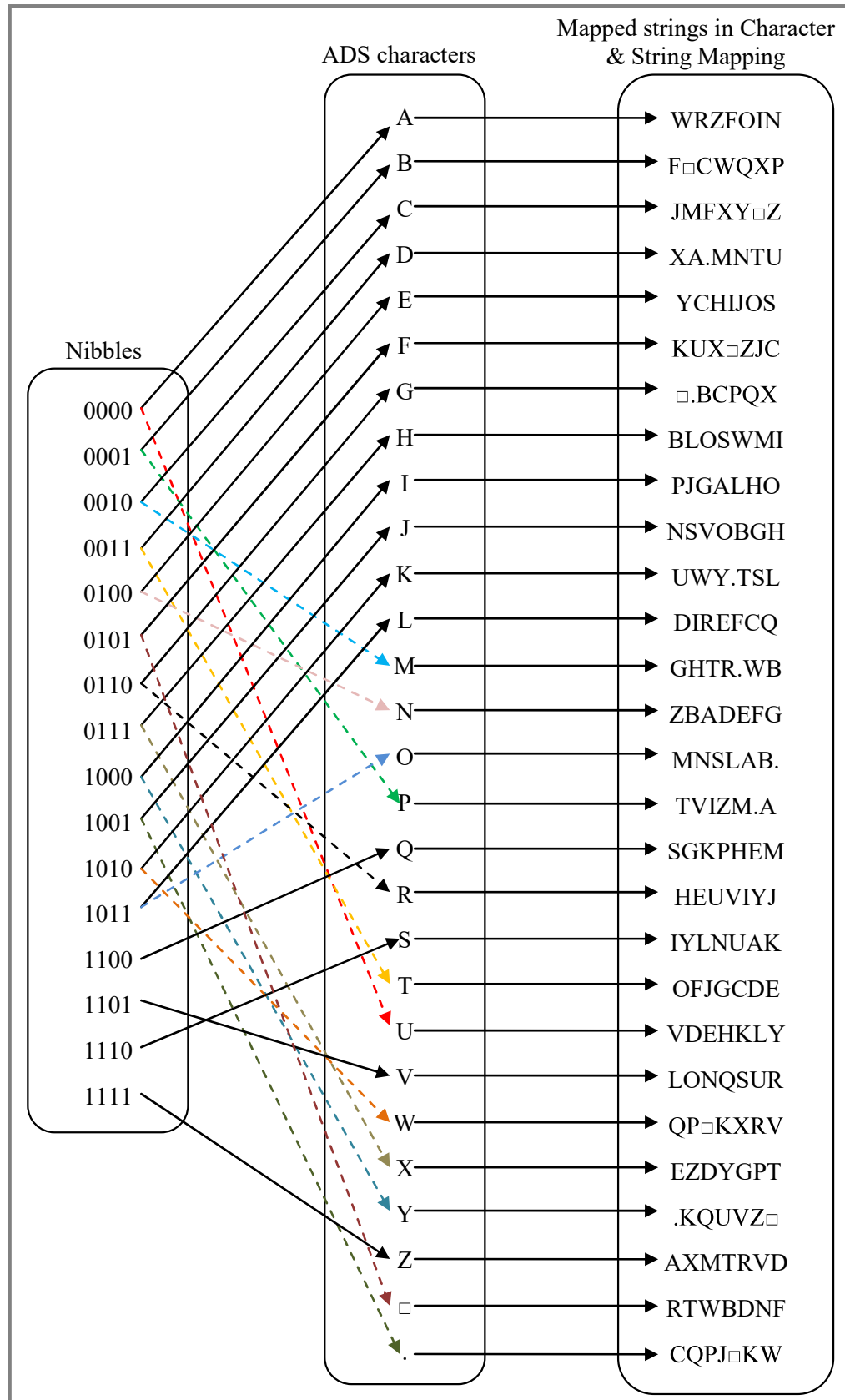


**Figure 5.1** Schematic diagram of Method-B. The dotted lines represent the modules that are introduced in Method-B and bold line represents the modified module of Method-A

## 5.2 Binary to Character Converter (BCC)

As mentioned, the BCC module converts a binary secret into character stream by using a mapping called Character & Bit Mapping (CBM). This process is similar to Base 64 encoding (it partitions the binary bits into groups of 6-bits and maps the 64 possibilities to 64 characters [134]). But, unlike Base 64, CBM considers a nibble as a group and the sixteen possible elements are mapped to the 28 ADS characters. The difference in numbers (16 and 28) leads to a one-to-many mapping with  $\frac{3}{4}$  elements mapped to two ADS characters, a 2-tuple. A sample mapping is provided in Fig. 5.2.





**Figure 5.2** Sample Character & Bit Mapping using the Character & String Mapping provided in Table 4.4. “□” – represents “Space” and “.” – represents “Dot”; ADS – English alphabets, Dot and Space

It should be noted that these ADS characters are in turn mapped to the CSM (Character & String Mapping) strings of length seven. This leads CBM to map an element to distinct characters ranging between seven and fourteen ( $7 + 7$ ) inclusive.

As discussed earlier (Section 4.3), the embedding capacity of Method-A is directly influenced by the number of possible characters in which a secret character can get embedded. Hence, avoiding the common characters between the two CSM strings of a 2-tuple will facilitate Method-B to achieve an optimal embedding capacity.

Table 5.1 provides a sample mapping, using the CSM provided in Table 4.4, along with the respective cumulative probability values.

**Table 5.1** Mapping of ADS characters and the nibbles (Character & Bit Mapping)

Nibble	Mapped character(s) <sup>†</sup>	No. of common characters between the two strings	Cumulative probability to embed the corresponding nibble
0000	A/U	0	48.97
0001	B/P	0	50.37
0010	C/M	0	48.62
0011	D/T	0	49.61
0100	E/N	0	49.88
0101	F/□	0	50.33
0110	G/R	0	50.18
0111	H/X	0	50.78
1000	I/Y	0	49.99
1001	J/.	0	51.08
1010	K/W	0	49.81
1011	L/O	0	49.92
1100	Q	NA	25.43
1101	V	NA	25.11
1110	S	NA	25.10
1111	Z	NA	24.82

ADS – English alphabets, Dot and Space; “□” – represents “Space”; “.” – represents “Dot”; † – elements of a 2-tuple are separated using the symbol “/”; NA – Not applicable

Using the mapping provided in Fig. 5.2, BCC module can convert a binary secret into an equivalent character stream by following the procedure given below.

### Procedure to Convert Binary to Character Stream

---

**Input:** Audio/video/compressed/encrypted binary\_string, CBM

**Output:** Character\_Stream

---

String binary\_string  $\leftarrow$  binary string

String Character\_Stream  $\leftarrow \epsilon$

For each nibble in binary\_string

    Character\_Stream  $\leftarrow$  Identify the character(s) equivalent to the nibble, using CBM, and append it to the Character\_Stream

    CBM  $\leftarrow$  Circular Left Shift the elements in CBM by 1 //makes the CBM dynamic

End for

Return Character\_Stream

---

It should be noted that, after converting each nibble to its equivalent ADS character(s), 1-tuple or 2-tuple, the elements in CBM are circular left shifted by one by the BCC module. This allows a nibble to be mapped to different ADS character(s) at different times and thus makes the CBM dynamic (see Example 5.1).

### Example 5.1:

Secret message	— 123?
Equivalent binary stream (ASCII)	— 00110001 00110010 00110011 00111111
Binary stream (after grouping)	— 0011 0001 0011 0010 0011 0011 0011 1111
Equivalent character stream	— D/T A/U B/P Z Z S V I/Y

From Example 5.1, it is evident that the character stream generated by BCC looks similar to the secret message used in Chapter 4. The only difference is the presence of a 2-tuple, which represents the choice of characters that has to be

embedded at a particular instance. That is, D/T represents the choice to embed either “D” or “T” inside a cover document. Thus, the generated character string is the required secret message which needs to be communicated, secretly, using a cover document.

### 5.3 Embedding Algorithm

Embedding algorithm of Method-B is similar to the procedure explained in Chapter 4 (Section 4.6). It takes a secret message, cover document and CSM as input values. It uses the same attribute Spacing with the seven levels,  $\{-0.1, -0.2, -0.3, +0.1, +0.2, +0.3, +0.4\}$ , to embed the secret character.

Embedding begins by identifying whether the first secret character is a 1-tuple or 2-tuple. If it is a 1-tuple, then the embedding algorithm identifies the CSM string that is mapped to the character. The algorithm searches, the cover document serially, for the first occurrence of any of the characters of the mapped string. When a match is found, the position of the encountered cover character in the mapped string is identified. Based on the position, the cover character is marked by altering the value of its attribute Spacing. A sample embedding is presented in Example 5.2.

#### Example 5.2:

Secret character	Mapped string in CSM	Cover document	Encountered character	Position of the encountered character in mapped string	Modification to be performed
Q	SGKPH EM	Cool morning.	Cool <span style="border: 1px solid black;">m</span> orning.	6	Expand the spacing value of “m” by 0.4 points (+0.4)

Suppose, if the character is a 2-tuple, then the embedding algorithm identifies the strings corresponding to the two ADS characters. It, then, searches the cover document serially, for the first occurrence of any of the characters of the two

identified strings. When a match is found, the cover character is marked accordingly (see Example 5.3).

**Example 5.3:**

Secret character	Mapped strings in CSM	Cover document	Encountered character	Position of the encountered character in the mapped string	Modification to be performed
K/W	K $\leftarrow$ UWY.TSL W $\leftarrow$ QP□KXRV	Cool morning all.	Cool□ morning all.	6	Expand the spacing value of “l” by 0.4 points (+0.4)

After embedding a secret character, all the strings of CSM are circular left shifted by one. This avoids the possibility of embedding the same secret character inside an identical cover character with the same spacing value, frequently. The advantage of doing this was explained earlier in Section 4.9.2.

The procedure is repeated till all the secret characters are embedded. After achieving this, the algorithm embeds *End-of-Secret* (EoS) characters. For experimental purpose, it has been considered as “. . .”  $\rightarrow$  “Dot Space Dot Space Dot”.

The pseudo code of the embedding procedure is as follows:

**Procedure of Embedding Algorithm**

---

**Input:** Secret\_Message, Cover\_Work, CSM, EoS characters, character spacing and their respective positions

**Output:** Modified\_Cover\_Work

---

Int count  $\leftarrow$  1

String  $S_{k1} \leftarrow \varepsilon$

String  $S_{k2} \leftarrow \varepsilon$

Secret\_Message  $\leftarrow$  Secret\_Message + EoS Characters

For each character “ $X_i$ ” in Secret\_Message do

    String  $S_{k1} \leftarrow$  String that is mapped to  $X_i$  in CSM

    If  $X_{i+1}$  is not empty and  $X_{i+1}$  equals “/” then

```
String Sk2 ← String that is mapped to Xi+2 in CSM
i ← i + 2
End if
L: Yj ← Read the character @ position “count” in Cover_Work
count++
If Yj ∈ Sk1 or Sk2
    Int pos ← Position of Yj in Sk1 or Sk2
    Change the character spacing of Yj based on pos
End if
Else Goto L
CSM ← Perform a Circular Left Shift on all the Strings in CSM
//makes the CSM dynamic
End for
Return Modified_Cover_Work
```

---

The modified cover document is the required stego document which has to be communicated to the receiver along with the CBM, CSM and EoS characters.

#### 5.4 Extraction Algorithm

The extraction algorithm is identical to that of the extraction procedure employed and described in Chapter 4 (Section 4.7). It reads the characters in stego document one by one, in a serial manner, and checks the Spacing of the read character. If the value is altered, it identifies the embedded secret by using the read character and the position equivalent to the spacing value. The procedure is repeated until it reads the EoS. Once extracted, the receiver uses the Character to Binary Converter, which is the reverse process of BCC, and converts the extracted character stream into an equivalent binary stream.

## 5.5 Evaluation Parameters

As Method-B uses the same attribute Spacing and the same seven levels to embed the secret, it provides the same level of secrecy illustrated in Section 4.8.1. But, the difference exists in embedding capacity, bits/distortion and uniformity in embedding probability which are discussed below in some detail.

### 5.5.1 Embedding Capacity and Bits per Distortion

It can be seen from Table 5.1, that Method-B embeds a nibble at a time viz. 4-bits/distortion. Also, out of the sixteen possible elements,  $\frac{3}{4}$  of the elements use two CSM strings and the remaining  $\frac{1}{4}$  use one string to get embedded. From the discussion provided in Section 4.3, this means that  $\frac{3}{4}$  of the elements have 50% probability to get embedded inside an encountered cover character, and the remaining has 25% probability. Hence, the overall probability of embedding a nibble inside an encountered cover character is:

$$\text{Embedding probability} = \left(\frac{3}{4} \times 50\right) + \left(\frac{1}{4} \times 25\right) = 43.75\% \quad (4.1)$$

Now, the possible number of cover characters that are required to embed a nibble is:

$$\text{Number of cover characters required to embed a nibble} = \frac{100}{43.75} = 2.29 \quad (4.2)$$

This shows that the extended method can embed an average of 1.75-bits/cover-character.

To verify the same, audio files (in mp3 format) of various sizes were considered as secret message and embedded inside a chosen text document. The obtained results are listed in Table 5.2.

**Table 5.2** Results of embedding the secrets in cover document

File no.	Size in KB	Size in bits	No. of characters encountered in cover document while embedding the secret + End-of-Secret characters					Embedding capacity per cover-character*
			Alphabet	Dot	Space	Other	Total	
1	19.3	158128	74980	1060	16063	1149	93252	1.70
2	10.7	87856	41246	618	9172	342	51378	1.71
3	13.3	109744	51203	744	11526	463	63966	1.72
4	16.6	136240	63954	926	13954	793	79627	1.71
5	10.3	84400	39802	590	8795	342	49529	1.70

\* – calculated using equation 4.3; KB – kilobyte

From Table 5.2, it can be seen that Method-B has achieved an average embedding capacity of  $1.71 \pm 0.01$  bits/cover-character.

### 5.5.2 Uniformity in Embedding Probability

As discussed earlier, a good steganographic method must maintain uniformity while embedding the secret inside a cover document. This avoids the wastage of embedding space and facilitates to embed secrets in smaller size documents.

To evaluate the uniformity of Method-B, an audio file (in mp3 format) of size 13.3 KB is embedded inside a cover document and the result is provided in Table 5.3.

From the table, it can be seen that the average number of cover characters required to embed a nibble uniformly falls within the range  $2.33 \pm 0.07$ . This proves that Method-B embeds the nibbles uniformly and thereby utilizes the available embedding space efficiently.

### 5.5.3 Comparison with other Methods

As mentioned in Section 4.3, Method-A embeds an average of 2-bits/cover-character (theoretical) with 8-bits/distortion. Whereas, Method-B embeds an average of 1.75-bits/cover-character (theoretical) with 4-bits/distortion. Hence, even though



the procedure of converting the binary secret to character stream has reduced the bits/distortion by 50%, the attained embedding capacity falls short only by 12.5%. However, Method-B outperforms UniSpaCh which can embed binary data with  $\approx 1.046$ -bits/cover-character and  $\approx 2$ -bits/distortion.

**Table 5.3** Uniformity in embedding probability at nibble-level

Nibbles	Secret message (109744 bits)		
	No. of times occurred	No. of times occurred (in %)	Average no. of cover characters used to embed one nibble
0000	3588	13.08	2.30
0001	1045	3.81	2.46
0010	1329	4.84	2.37
0011	1400	5.10	2.40
0100	1586	5.78	2.19
0101	2700	9.84	2.42
0110	1192	4.34	2.29
0111	1259	4.59	2.39
1000	1035	3.77	2.31
1001	1239	4.52	2.31
1010	3021	11.01	2.36
1011	1291	4.71	2.33
1100	1033	3.77	2.30
1101	1396	5.09	2.39
1110	1459	5.32	2.21
1111	2863	10.44	2.29

## 5.6 Security Aspect

Frequency analysis is employed by the attackers to break systems that rely on the occurrence frequencies of characters to embed secrets. Method-B provides the same security features as Method-A, such as uniformity in distributing the stego characters among the seven levels (refer Section 4.9.2) and non-correlation between the frequency profile of secret and stego characters (refer Fig. 4.3). However, the

discrepancy in mapping some nibbles to 1-tuple and others to 2-tuple, in CBM, creates difference in their embedding probabilities (50 and 25%). This causes variations in the intervals of marked cover characters (refer Table 5.4) and thereby results in the leakage of information.

**Table 5.4** Uniformity in embedding probability at character-level

Tuples	Secret message after BCC (27436 characters)				
	No. of times occurred	No. of times occurred (in %)	No. of cover characters used to embed one character		
			Minimum no.	Maximum no.	Average no.
A/U	1802	6.57	1	11	1.92
B/P	1856	6.76	1	8	1.88
C/M	1738	6.33	1	10	2.04
D/T	1724	6.28	1	9	1.89
E/N	2015	7.34	1	7	1.77
F/□	1491	5.43	1	9	1.88
G/R	1646	6.00	1	9	1.79
H/X	1765	6.43	1	7	1.73
I/Y	1706	6.22	1	13	2.07
J/.	1642	5.98	1	8	1.93
K/W	1769	6.45	1	11	1.94
L/O	1715	6.25	1	10	1.85
Q	1586	5.78	1	25	3.77
V	1654	6.03	1	32	3.49
S	1675	6.11	1	28	4.03
Z	1652	6.02	1	22	3.65

“□” – represents “Space”; “.” – represents “Dot”; BCC – Binary to Character Converter

But, Method-B secures such leakage of information by performing a circular left shift operation after converting every nibble to its corresponding tuple (refer Section 5.2). This distributes the nibbles across all the possible tuples and also makes the output character stream uniform. This can be inferred from Tables 5.3 and 5.4 (the occurrences of nibbles in secret message are not uniform, in

Table 5.3, and have standard deviation 3.01%. Whereas, the output character stream generated by BCC is fairly uniform, in Table 5.4, and has standard deviation 0.43%). Hence even though an attacker succeeds in identifying the intervals of marked characters and attribute them to the characters, Q, V, S and Z, mapping them back to the corresponding nibbles is not possible (refer Tables 5.4 and 5.5).

### **5.7 Case Study on Nuclear Power Plants**

To understand the impact of Method-B while communicating images, a case study using the images related to nuclear power plant has been conducted. For the study, various categories of images like engineering drawings (civil drawing, mechanical design, electronic circuit, etc.), roadmaps, graphs, minimal line drawings, etc., were considered. The reasons for considering these images are:

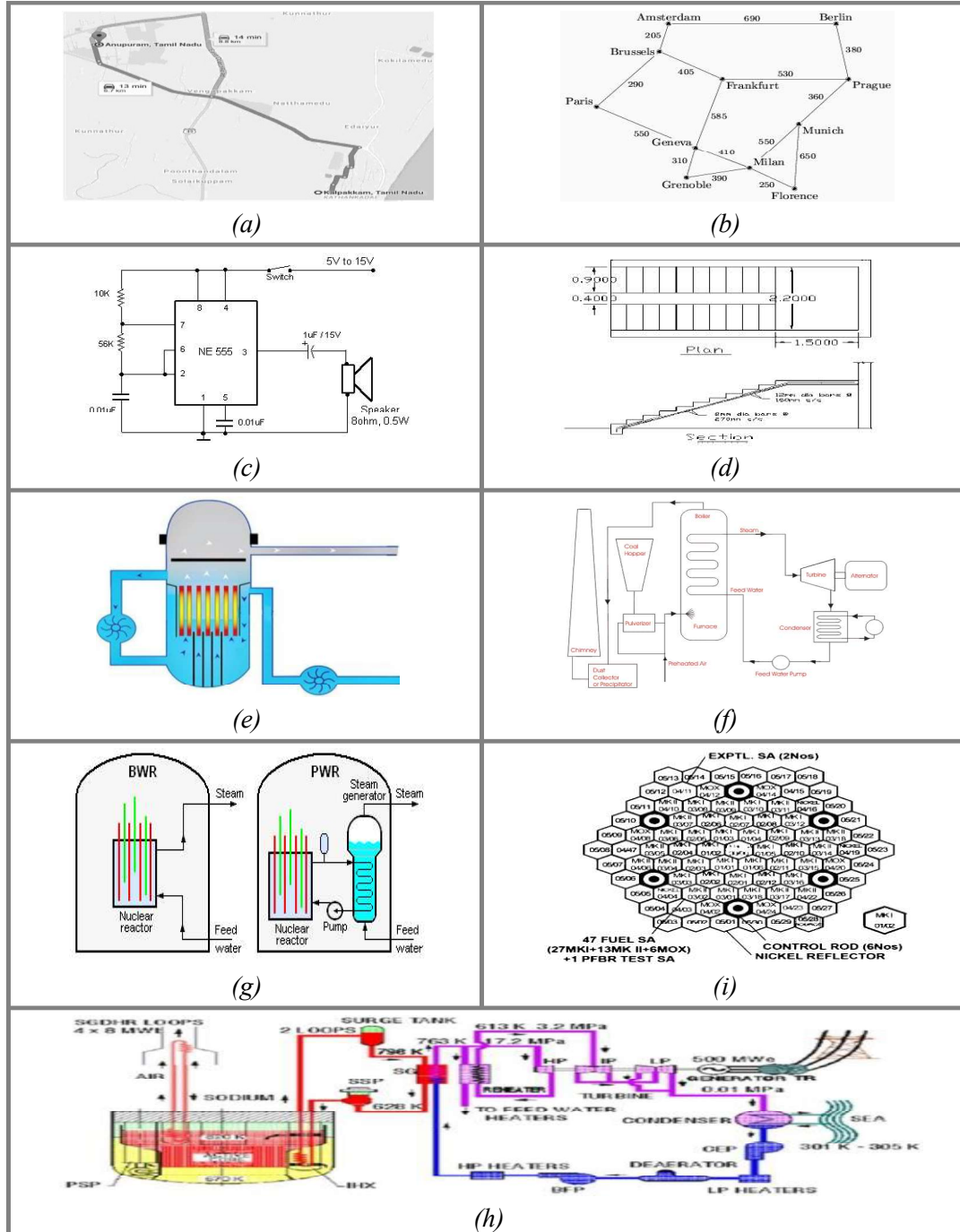
- Graphs can be used to represent the interconnectivity among geographically distributed nuclear power plants
- Roadmaps can be used to represent the existing secret passages between nuclear power plants or escape routes during disasters
- Application of engineering drawings in nuclear power plants is straight forward

Table 5.5 Distribution of nibbles among the possible tuples

Nibbles	No. of times occurred	Tuples															
		A/D	B/P	C/M	D/T	E/N	F/□	G/R	H/X	I/Y	J/.	K/W	L/O	Q	V	S	Z
0000	3588	271	203	204	211	198	190	242	221	196	195	364	203	258	182	199	251
0001	1045	58	103	52	55	117	56	79	54	52	54	56	49	82	67	61	50
0010	1329	102	57	72	65	76	50	91	83	162	87	76	59	53	82	119	95
0011	1400	104	107	58	67	145	82	51	93	69	84	54	136	46	166	73	65
0100	1586	61	168	135	83	215	63	114	91	136	56	119	64	71	81	56	73
0101	2700	155	150	158	156	217	155	133	158	162	177	161	194	196	174	216	138
0110	1192	72	89	64	71	149	55	55	87	71	53	78	70	65	66	78	69
0111	1259	99	60	88	71	96	72	66	75	66	90	82	82	68	83	75	86
1000	1035	64	60	59	70	61	56	59	176	46	42	37	81	50	55	40	79
1001	1239	144	63	133	65	65	65	68	57	73	59	64	115	80	67	66	55
1010	3021	188	224	212	178	174	182	182	186	185	174	188	200	171	204	184	189
1011	1291	76	82	81	82	88	78	88	87	99	75	73	68	90	70	75	79
1100	1033	50	107	41	59	61	80	51	75	75	67	55	66	45	60	67	74
1101	1396	80	69	83	78	83	88	98	76	78	110	78	87	75	79	139	95
1110	1459	72	77	72	181	88	79	118	90	69	134	106	67	77	68	62	99
1111	2863	206	237	226	232	182	140	151	156	167	185	178	174	159	150	165	155

“□” – represents “Space”; “.” – represents “Dot”

Some of the images used for experimentation are depicted in Fig. 5.3. The details of these images are provided in Table 5.6 and the average number of cover characters used to embed them, both by Method-B and UniSpaCh, are listed in Table 5.7.



**Figure 5.3** Sample tested images: (a) Road map from Kalpakkam to Anupuram [135]; (b) Graph [136]; (c) Electronic circuit diagram [137]; (d) Civil drawing of stairs [138]; (e) Boiling water reactor [139]; (f) Schematic diagram of thermal power plant [140]; (g) Nuclear power plant steam generation [141]; (h) Reactor flow sheet [142]; (i) Reactor core [143]

**Table 5.6** Details of images given in Fig. 5.3

Figure no.	Image size in KB	Image resolution	Image format	Bit depth
5.3 (a)	27.10	555 x 593	JPG	8
5.3 (b)	5.84	260 x 194	PNG	8
5.3 (c)	1.32	405 x 255	PNG	1
5.3 (d)	1.74	400 x 444	PNG	1
5.3 (e)	40.90	518 x 405	JPG	24
5.3 (f)	18.30	550 x 381	PNG	8
5.3 (g)	6.18	386 x 188	PNG	24
5.3 (h)	86.20	1091 x 635	JPG	24
5.3 (i)	19.10	744 x 726	PNG	1

*KB – kilobyte; JPG – Joint Photographic Experts Group; PNG – Portable Network Graphics*

**Table 5.7** Number of cover characters required by UniSpaCh and Method-B

Figure no.	Requirement of UniSpaCh		Requirement of Method-B		Efficiency of the Method-B compared with UniSpaCh <sup>†</sup> (in %)
	Average no. of cover characters <sup>†</sup>	Average no. of pages <sup>‡</sup>	No. of cover characters	Average no. of pages <sup>‡</sup>	
5.3 (a)	212240	127	133388	80	37.01
5.3 (b)	45737	28	28117	17	39.29
5.3 (c)	10337	7	6344	4	42.86
5.3 (d)	13627	9	8751	6	33.33
5.3 (e)	320318	192	191459 <sup>±</sup>	115	40.10
5.3 (f)	143321	86	88955	54	37.21
5.3 (g)	48401	29	30139	18	37.93
5.3 (h)	675096	405	403514 <sup>±</sup>	242	40.25
5.3 (i)	149586	90	93173	56	37.78

<sup>†</sup> and <sup>±</sup> – represents the estimated value by considering the embedding capacity per cover-character as 1.046-bits and 1.75-bits respectively; <sup>†</sup> – Relative Efficiency =  $100 - ((\text{Average no. of pages required by Method-B} / \text{Average no. of pages required by UniSpaCh}) * 100)$ ; <sup>‡</sup> - represents the estimated value by considering the average length of a word as 5.5 (including space), average number of words in a sentence as 15 and average number of words per page as 300 — Same page calculation is followed for the rest of the thesis

From Table 5.7, it can be seen that, Method-B stands best by embedding the images in a 38.42% (on an average) smaller size cover document when compared with UniSpaCh. However, the number of pages that are required to embed, even, an image of size 40.9 KB crosses hundred. This shows that the page requirement is not easy to be met when the size of the image is in the order of megabytes. For example, one megabyte image will require a document with  $\approx 2950$  pages.

This situation can be handled by reducing the size of images. One way of achieving this is by using appropriate image formats that minimizes the redundant information or does not store the information, such as color, line thickness, intensity, transparency, etc., at all.

## **5.8 Summary**

In this chapter the limitations of Method-A, such as the non-capability of embedding messages containing numbers, special characters, multimedia data, etc., has been addressed. This was achieved by converting it into a mixed-type embedding technique (Method-B) which is capable of embedding binary data. It is noticed that this conversion procedure has reduced the embedding capacity by 12.5% and bits/distortion by 50%, when compared to what was achieved by employing Method-A. However, it is still better than the best existing method available in the literature namely, UniSpaCh.

As Method-B embeds a single ADS (secret) character in multiple cover characters, it is comparable with that of the poly-alphabetic substitution ciphers of cryptography (similar to Method-A). However, the additional circular left shift operation introduced in BCC module makes it superior when compared with the latter.

A case study on nuclear power plant related images concluded that Method-B has reduced the size of required cover document by 38.42%, when compared with UniSpaCh. But, it still requires more than hundred pages when the image size is  $\approx 40$  KB. Also, the situation gets worse when the file sizes are in the order of megabytes. This limitation is not only applicable for images but, also, to any type of binary data. However, handling such limitation differs from one data type to another. In case of images, one can reduce their size by minimizing the redundant information. This can be achieved by using vector formats which has been explored in the next chapter.



### EMBEDDING IMAGE

---

*In this chapter, the work carried out to embed image files inside smaller size cover documents is described. Initially, the possible ways to reduce the size of images by representing them in vector formats is discussed. Then, the procedure of embedding them using the method described in Chapter 5, Method-B, is explained. As Method-B failed to handle transmission errors, a novel method referred to as Method-C is developed. Method-C converts an image into a custom format and then embeds the same, along with the structure of image, using the attributes Color, Kerning and Spacing. This method of embedding facilitated the extraction algorithm to handle transmission errors and avoided retransmission. A comparison of the custom format and best vector format is given. Method-C is, also, inspected for its embedding capacity and error handling capabilities.*

#### 6.1 Introduction

The extended mixed-type embedding method, Method-B, successfully embeds binary data inside text documents. However, when the size of the secret message exceeds a certain limit, say one megabyte, the number of pages required to embed it, is not easy to be met.

Hence, there is a need to address the issue by developing a method for such types of documents. This chapter focuses to address the shortcoming for the categories of images mentioned in Section 5.7. The reasons for choosing these images are that, they:

- (i) are widely used in organizations
- (ii) do not require the complete information, like color, line thickness, etc., to convey their meaning. Often, their structure or layout is sufficient

This offers the sender a choice to reduce their size by minimizing redundancy through known methods such as compression or vector representation. Of these, the second possibility has been explored in this chapter.

Various vector representations have been studied and experimented with the above-mentioned images. The results indicated that the format SVGZ attains the smallest file size. Hence embedding the resulting images, using Method-B, required a cover document that are, considerably, smaller in size. However, the method failed to recover a substantial portion of the image, even in case of single bit error. This shows that the SVGZ format can be utilized, efficiently, in text steganography only when the used communication line is free from error or the underlying method (both embedding and extraction) can provide the necessary error correction mechanisms.

The latter requirement can be met by introducing error correcting codes, like hamming code [144], as a part of embedding procedure. But, this increases the size of message, that needs to be embedded, which is not preferable.

To address these issues, a novel method (referred to as Method-C, in the rest of the thesis) is developed which includes:

- (i) a custom format that represents images in smaller sizes along with error handling capabilities
- (ii) an embedding algorithm that interprets the custom format and embed the same accordingly

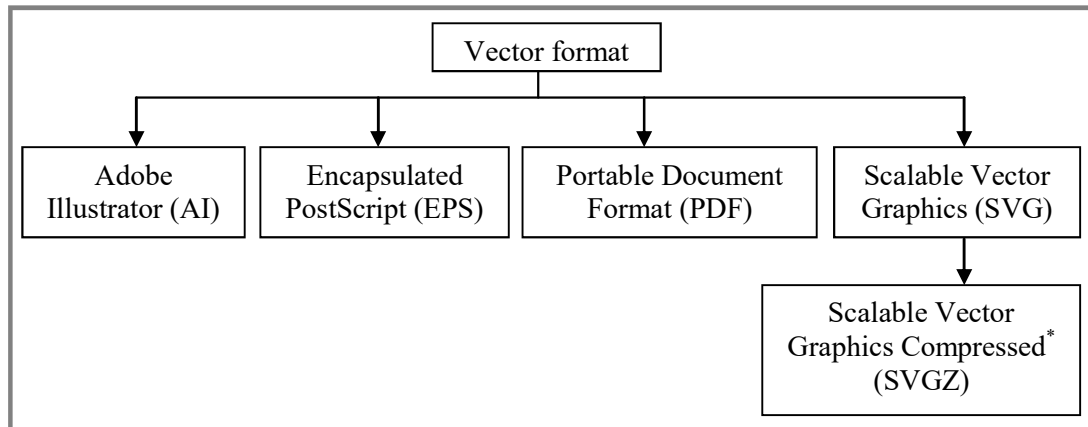
These are explained in the subsequent sections of this chapter. A short introduction to vector formats, highlighting the above-mentioned difficulty, is also provided.

## 6.2 Vector Format

Image formats can be, broadly, classified into two categories namely raster and vector [145]. Raster image formats, like Joint Photographic Experts Group (JPEG), Portable Network Graphics (PNG), Graphics Interchange Format (GIF), etc., considers an image as a grid of pixels each with a depth in the powers of 2 [146]. That is, 1-bit, 2-bit, 4-bit, 8-bit or 24-bit. This makes them to compromise their resolution during resizing [146]. For example, when the image is resized (stretched), the additional pixels are filled using the existing values. This results in the distortion (pixilation) of the image [146]. Also, during storage, these formats do not distinguish the contents (object or element) that are present in the image [146] viz. an image that contains a square or a circle is expressed using the same representation.

This makes these formats inefficient, in terms of memory space, while representing certain types of images, specifically computer generated images, such as line drawings, cartoons, maps, graphical images, etc. This is because these images contain various elements that can be represented by means of simple geometric primitives such as lines, splines, polygons, circles, ellipses, etc. [147]. These primitives can be stored and manipulated more efficiently when they are stored as mathematical expressions rather than as pixels [147]. For example, a triangle object can be represented by defining three points and properties such as fill, color, edge thickness, etc. This approach saves space and also facilitates geometric operations such as scaling, sheering, etc., without losing quality.

Vector image formats (refer Fig. 6.1) achieve the same and some of the well-known software that generates these formats are listed in Table 6.1. Further aspects of these formats are not being discussed as they do not form scope of the present work.



**Figure 6.1** Vector image formats. \* – represents the compressed version of SVG format

**Table 6.1** Software that generate vector file format

Software	Developed by	Copyright status	Operating system	Initial release	Reference
Adobe Illustrator	Adobe Systems	Proprietary	Mac, Microsoft Windows	1987	[148]
Corel Draw	Corel Corporation	Proprietary	Microsoft Windows	1989	[149]
Inkscape	—	Open Source	Linux, Mac, Microsoft Windows	2003	[150]
Sketch	Bohemian Coding	Proprietary	Mac	2010	[151]

For experimental purpose, the images in Fig. 5.3 are converted to vector formats using Adobe Illustrator and Inkscape. Table 6.2 provides a comparison of the obtained results.

From the table, it can be noticed that the SVGZ image format, generated by the Adobe Illustrator software, achieves the smallest size. Hence, the same has been considered as the secret message and the number of pages required to embed them, using Method-B, are estimated in Table 6.3.

**Table 6.2** Sizes of generated vector images

Image no.	Size in KB	Output file size (in KB)								
		Adobe Illustrator					Inkscape			
		AI	Illustrator or EPS	Adobe PDF	SVG	SVG Z	EPS	PDF	SVG	SV GZ
5.3 (a)	27.10	35.0	223	27.6	3.28	905 Bytes	13.90	6.17	6.16	1.40
5.3 (b)	5.84	25.2	215	39.6	4.31	898 Bytes	25.20	9.85	16.10	1.87
5.3 (c)	1.32	26.9	208	32.0	8.71	1.33	22.00	10.20	14.70	2.24
5.3 (d)	1.74	28.8	217	33.4	11.90	1.63	10.80	5.06	15.50	2.65
5.3 (e)	40.90	37.7	226	24.5	16.00	2.38	5.01	1.91	13.10	3.05
5.3 (f)	18.30	28.2	221	32.8	14.10	2.25	17.10	7.27	20.10	3.66
5.3 (g)	6.18	38.2	227	27.6	11.00	1.80	13.50	6.39	19.90	3.39
5.3 (h)	86.20	45.3	254	38.0	50.50	6.10	25.80	10.20	50.30	7.99
5.3 (i)	19.10	47.1	263	44.9	47.90	4.54	23.90	9.53	65.80	7.97

*AI – Adobe Illustrator; EPS – Encapsulated Postscript; PDF – Portable Document Format; SVG – Scalable Vector Graphics; SVGZ – SVG Compressed; KB – kilobyte*

**Table 6.3** Results of embedding raster and vector images by Method-B

Figure no.	Image size in KB (Raster)	Requirement to embed raster images		SVG Z file size in KB	Requirement to embed SVGZ images		Efficiency <sup>†</sup> (in %)
		No. of cover characters	No. of pages (A)		No. of cover characters <sup>†</sup>	No. of pages (B)	
5.3 (a)	27.10	133388	80	905 Bytes	4137	3	96.25
5.3 (b)	5.84	28117	17	898 Bytes	4105	3	82.35
5.3 (c)	1.32	6344	4	1.33	6226	4	0
5.3 (d)	1.74	8751	6	1.63	7630	5	16.67
5.3 (e)	40.90	191459 <sup>†</sup>	115	2.38	11141	7	93.91
5.3 (f)	18.30	88955	54	2.25	10533	7	87.04
5.3 (g)	6.18	30139	18	1.80	8426	5	72.22
5.3 (h)	86.20	403514 <sup>†</sup>	242	6.10	28555	18	92.56
5.3 (i)	19.10	93173	56	4.54	21253	13	76.79

*† – represents the estimated value by considering the embedding capacity per cover-character as 1.75-bits; ‡ – Relative Efficiency =  $100 - ((B / A) * 100)$ ; SVGZ – Scalable Vector Graphics Compressed; KB – kilobyte*

From Table 6.3, it can be noticed that the communication of images using the SVGZ format has, considerably, reduced the size of required cover document. However, one drawback noticed while communicating images in this format is that the embedded image cannot be extracted completely in the case of transmission errors. That is, even for single bit error a substantial portion of the image cannot be retrieved. This can be inferred from Figures 6.2 and 6.3.

This motivated to develop:

- (i) a custom format that encompasses the structure or layout of image in the form of codes
- (ii) a text steganographic algorithm that interprets these codes and embeds them using the attributes Color, Kerning and Spacing

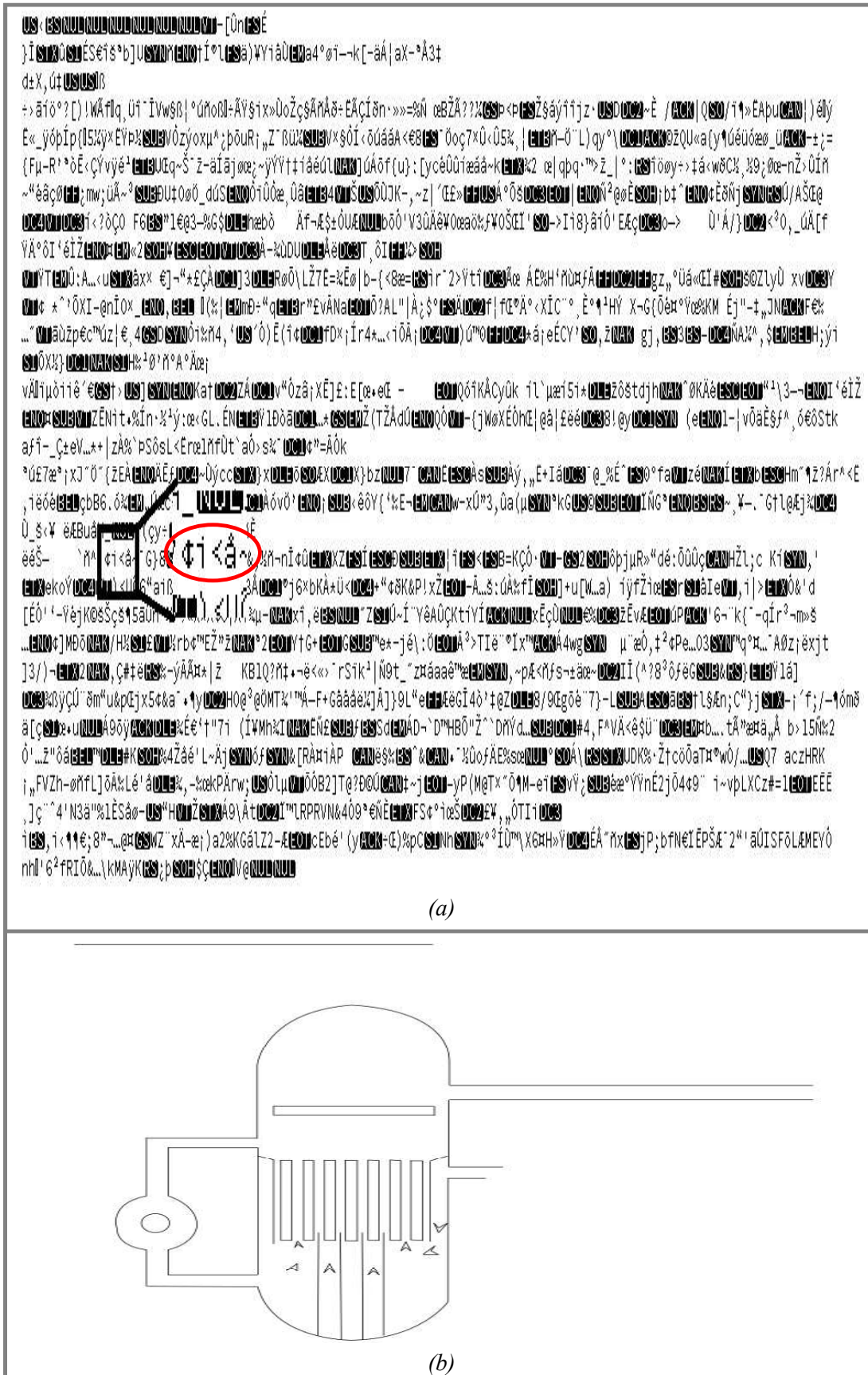
The combination embeds the original structure of an image, as it is, inside a cover document. Hence, in the case of transmission errors, extraction algorithm searches the potential cover characters and continues extracting the codes, accordingly. This provides the required error handling capabilities and, thereby, reduces loss of information in addition to avoiding retransmission. The details are given in the subsequent sections.

### **6.3 Custom Format to Represent an Image**

The format defines various elements of an image and represents them as codes. The procedure to convert an image into codes and vice versa is described below.





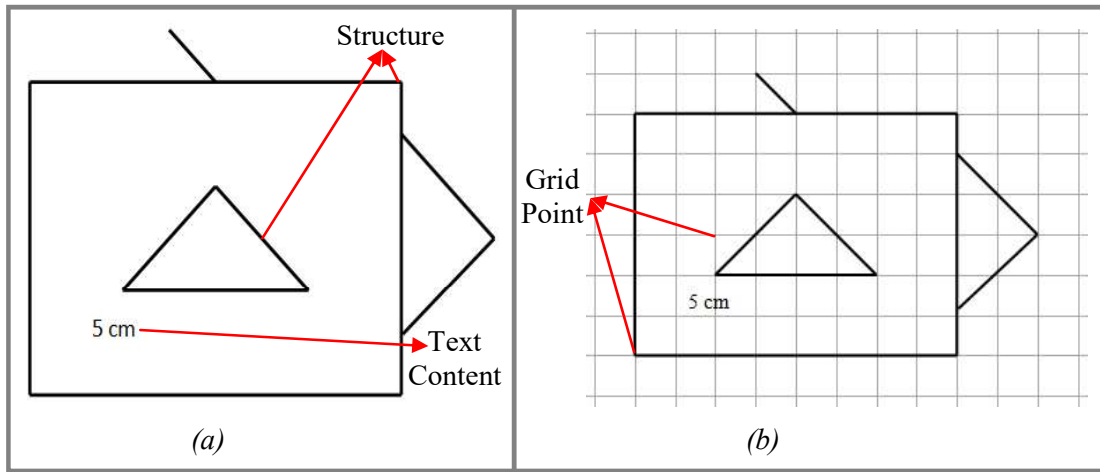


**Figure 6.3** Sample SVGZ File (corrupted) and corresponding image: (a) Corrupted SVGZ file; (b) Generated image



### 6.3.1 Elements of an Image

Elements of an image are defined with respect to a *grid*, of equally spaced horizontal and vertical lines, whose intersection points are called *grid points*. These elements define the way the structure of the image traverses with respect to the grid viz. from one grid point to another, and the text content that appeared in the image. For example, in Fig. 6.4 (a), black lines form the structure of the image and “5 cm” is the text content.



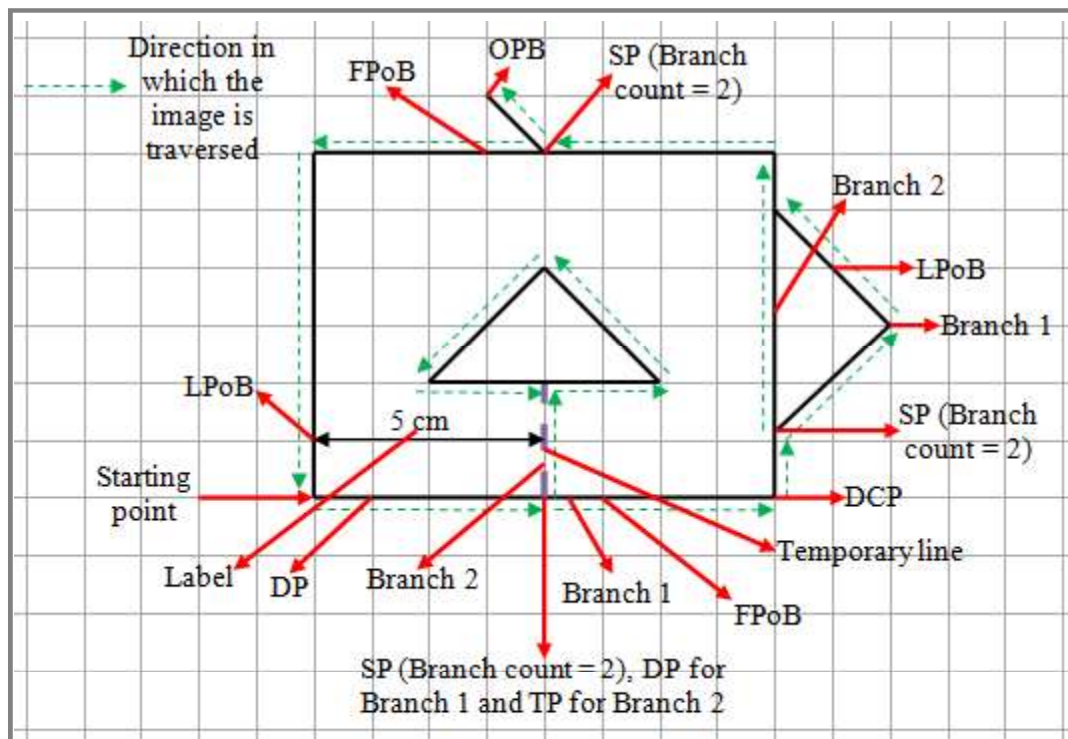
**Figure 6.4** Sample image: (a) Without grid; (b) With grid

The classification and definition of various elements are discussed below, in detail, using Figures 6.5 and 6.6. Various notations mentioned in these figures and the rest of this chapter, are summarized in Table 6.4.

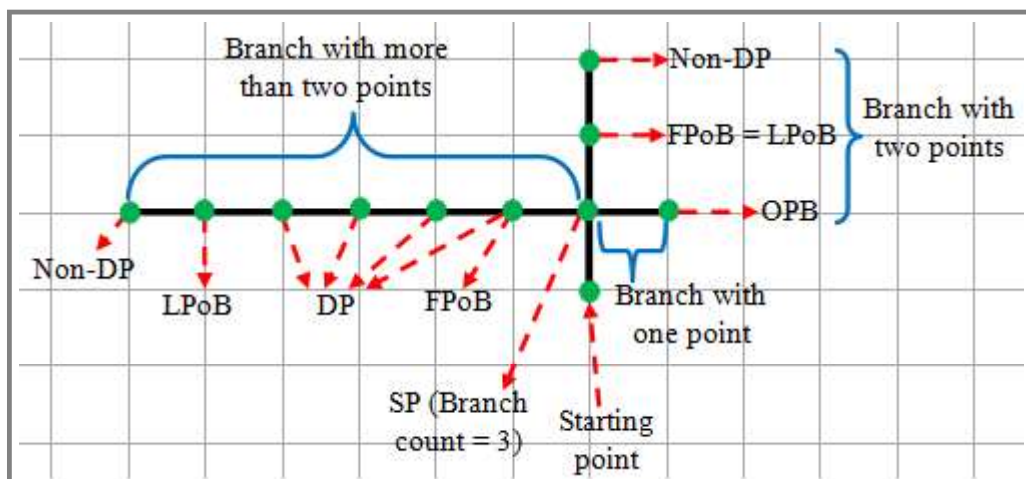
- *Label*: Text content of the image
- *DP*: It is the point of the image that coincides with the grid point. It represents the direction to move to reach the next point where the structure of the image traverses (exceptions are mentioned below)
- *DCP*: It is the point at which the structure changes its direction from one to another

**Table 6.4** Used notations and their descriptions

Notation	Description	Notation	Description
DP	Directional point	OPB	The only point of a branch
DCP	Direction changing point	N <sub>HGL</sub>	Number of horizontal grid line
TP	Temporary point	N <sub>VGL</sub>	Number of vertical grid line
SP	Split point	H <sub>4</sub>	Hexadecimal value (Nibble)
FPoB	First point of a branch	NoB	Number of branch
LPoB	Last point of a branch	S <sub>p</sub>	Single page



**Figure 6.5** Picture depicting the various elements of an image



**Figure 6.6** Picture depicting the three possible branches of an image

- *TP*: The disconnected objects of the image are connected using temporary lines. There is no restriction in the direction or the point at which the temporary line is drawn (but maintain the length of temporary line small). The point at which the temporary line is drawn is the TP
- *SP*: It is a branching point where the structure splits (branches) in more than one direction
- *Branch or Path*: It is nothing but a path after a SP. The branches of a SP are traversed in the anti-clockwise direction starting from degree zero. The number tagged with the branch represents the order in which they will be traversed from the corresponding SP, while converting them to codes
- *FPoB*: The point of a branch, after a SP, which falls on the grid point
- *LPoB*: Except the last point (non-DP), all other points of a branch can be a DP, DCP or SP (exception is OPB). The previous point of such non-DP is the LPoB
- *OPB*: It is the only point of a branch

It should be noted that for a branch with only one or two points, the FPoB and LPoB will be the same. Also, based on the starting point and the direction of the movement chosen, the terminologies marked in Figures 6.5 and 6.6 will vary.

### 6.3.2 Code Representation of Elements

Table 6.5 provides the code representation of various elements of an image. DP and DCP are represented by a nibble,  $H_4$ , representing the direction to reach the next grid point on the structure, from the current grid point. To find the direction, the angle of the line obtained by connecting these grid points is used. All the possible angles and their respective  $H_4$  values are given in Table 6.6.

**Table 6.5** Elements of an image and their respective codes

Element	Respective code	Element	Respective code
Label	\$    #    \$#	FPoB <sup>†</sup>	(H <sub>4</sub> )    (Y, H <sub>4</sub> ) and DP    DCP    TP    SP    OPB    LPoB
DP or DCP	(*H <sub>4</sub> )	OPB	( <sup>†</sup> Y, 0000, X)
TP	(*Y, H <sub>4</sub> )	LPoB	(*H <sub>4</sub> , X)
SP	( <sup>†</sup> X, H <sub>4</sub> )		

“\*” and “+” represents the Label, which can be “\$ || # || \$# || NULL” and “\$ || NULL” respectively; <sup>†</sup> – represents that (Y, H<sub>4</sub>) and OPB cannot occur

**Table 6.6** Possible angles and their respective H<sub>4</sub>

H <sub>4</sub>	Original angle of line	Consolidated angle	H <sub>4</sub>	Original angle of line	Consolidated angle
0000	0    >350 & <360	0	1000	>157.5 & <=180	180.0
0001	>0 & <=22.5	22.5	1001	>180 & <=202.5	202.5
0010	>22.5 & <=45	45.0	1010	>202.5 & <=225	225.0
0011	>45 & <=67.5	67.5	1011	>225 & <=247.5	247.5
0100	>67.5 & <=90	90.0	1100	>247.5 & <=270	270.0
0101	>90 & <=112.5	112.5	1101	>270 & <=292.5	292.5
0110	>112.5 & <=135	135.0	1110	>292.5 & <=315	315.0
0111	>135 & <=157.5	157.5	1111	>315 & <=350	337.5

TP is represented as (Y, H<sub>4</sub>) where “Y” represents the TP and H<sub>4</sub> represents the direction.

SP is represented as (X, H<sub>4</sub>) where “X” represents that the structure moves in more than one direction from this point onward. H<sub>4</sub> (exceptional for SP) represents the NoB existing at this SP (exclude the path through which it reached this SP, see Fig. 6.5). This makes the code to support a maximum of fifteen branches at a SP.

The FPoB is represented by two codes. The *First code* is the direction, from the corresponding SP to the FPoB, which can either be (H<sub>4</sub>) or (Y, H<sub>4</sub>). The *Second code* depends on the type of the second point of that branch which can be any code

other than the FPoB. It should be mentioned that the FPoB cannot have codes  $(Y, H_4)(^+Y, 0000, X)$ , where, “+” represents the Label “\$ || NULL”, as they represent a temporary line.

The OPB is represented as  $(Y, 0000, X)$  where “Y” represents that the value “0000” has no significance (dummy) and “X” represents the end of the branch.

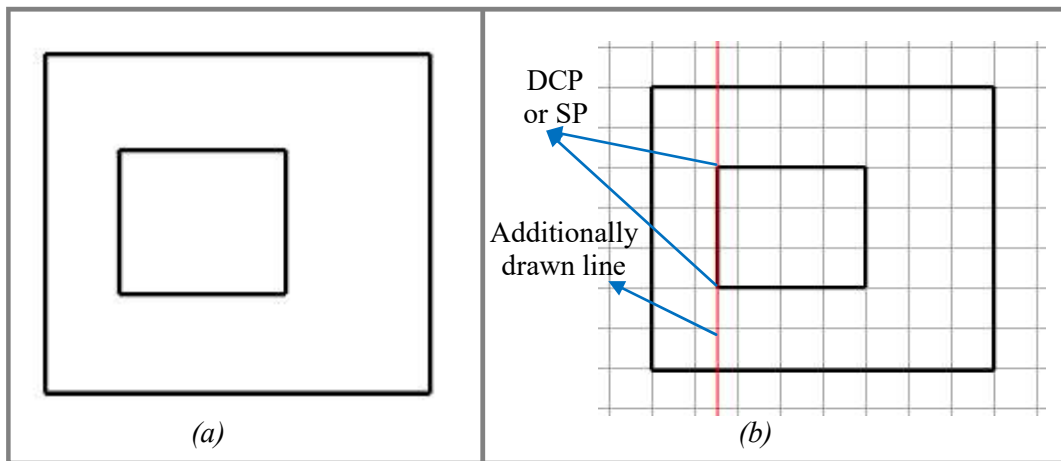
The LPoB is represented as  $(H_4, X)$ .  $H_4$  represents the direction and “X” represents the end of the branch.

It should be noted that the Label is represented as a *Flag* in the code of other elements, through which the structure traverses. It is inserted in the grid point that is closer to the first character of the Label. The *Flag* “\$”, “#” and “S#” represents that a Label is attached to the current, next and both (current and next) grid points respectively. It should be mentioned that, whenever a non-DP has a Label attached to it, the previous grid point that has a DP or DCP or FPoB will carry the *Flag* “#”, as the non-DP does not have a code (exception is OPB).

The Labels are written separately, with a delimiter, in the same order as the *Flags* are inserted.

### 6.3.3 Image to Code Conversion Procedure

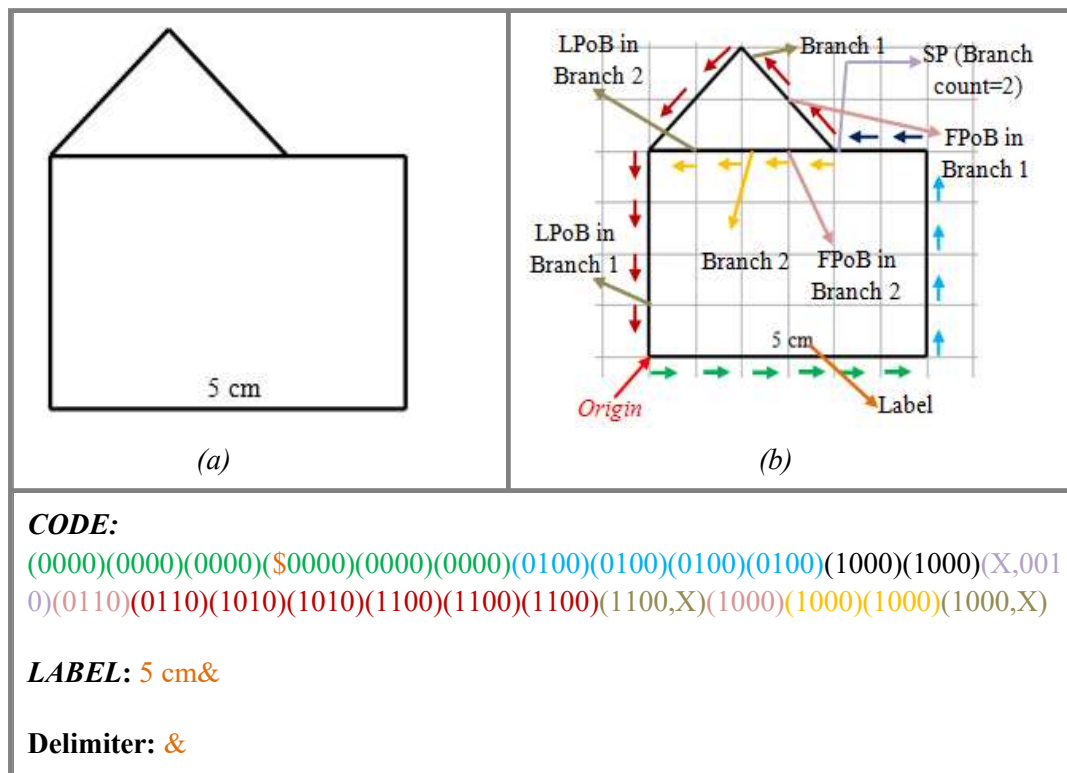
The procedure to convert a given image into custom defined codes is given below. The disconnected objects in the image are connected using temporary lines and the elements of the image like SP and DCP are identified (identification of such elements is possible even in the absence of grid). A grid is chosen in such a way that the maximum number of SP and DCP falls on grid points. Additional lines are drawn to accommodate the points that do not fall on grid points (refer Fig. 6.7 (b)) and the newly drawn lines are considered as normal grid lines.



**Figure 6.7** Sample image with additional grid line: (a) Without grid; (b) With grid

A random point on the grid, which coincides with the structure, is selected as the *origin* (refer Fig. 6.8 (b)).

If the selected point is a DP, DCP, FPoB or TP, then the respective code is written and the procedure is continued by selecting the next point of the structure.



**Figure 6.8** Sample image along with the corresponding code: (a) Without grid; (b) With grid

If the selected point is a SP, then the respective code is written and the location (x, y) of the SP (with respect to the grid) is stored repeatedly in a stack, corresponding to the number of branches (for  $H_4-1$  times) at the SP. After this, each branch is traversed separately in anti-clockwise direction, starting from zero degree, and converted into codes.

Whenever the OPB or LPoB is encountered, the respective code is written and the stack is checked for emptiness. If the stack is not empty, then an element is popped from it and the procedure is continued from the popped (x, y) location viz. traversing the other non-traversed branches. When the stack is empty, it represents that the whole structure has been converted into codes successfully, which ends the procedure.

Whenever a Label is encountered in the image, respective *Flag* is inserted in the corresponding code and the encountered Label is stored separately.

This facilitates the custom format to represent the structure of an image in the form of codes. Also, the number of codes does not depend on the resolution of the image but depends, only, on that of the grid and the number of SP present in it. This property makes the custom format to considerably reduce the number of bits that are required to represent an image (details are provided later in Section 6.5.1).

A sample traversal of an image along with its corresponding codes is depicted in Fig. 6.8. Following the arrows, the complete image is converted to its respective code by employing the details given in Tables 6.5 and 6.6.

#### **6.3.4 Code to Image Conversion Procedure**

The procedure to convert the, custom defined, codes into an image is described below. To draw the image, the procedure uses line drawing application like

Microsoft (MS) Word, that allows drawing lines of predefined length (base length is considered as 0.5 cm and the length of the line, to be drawn, is determined using its angle and trigonometric equations [152]).

First, a stack is declared. The point corresponding to the *origin* is chosen and the codes are read one by one. Based on the read code, a line is drawn in the direction of the consolidated angle as specified in Table 6.6. Whenever a TP is encountered, the cursor is moved in the specified direction without drawing any line. Whenever a SP is encountered, the location of the current point, say (x, y), is stored for (H<sub>4</sub>-1) times in the stack. After this, the codes are read sequentially and the lines are drawn accordingly.

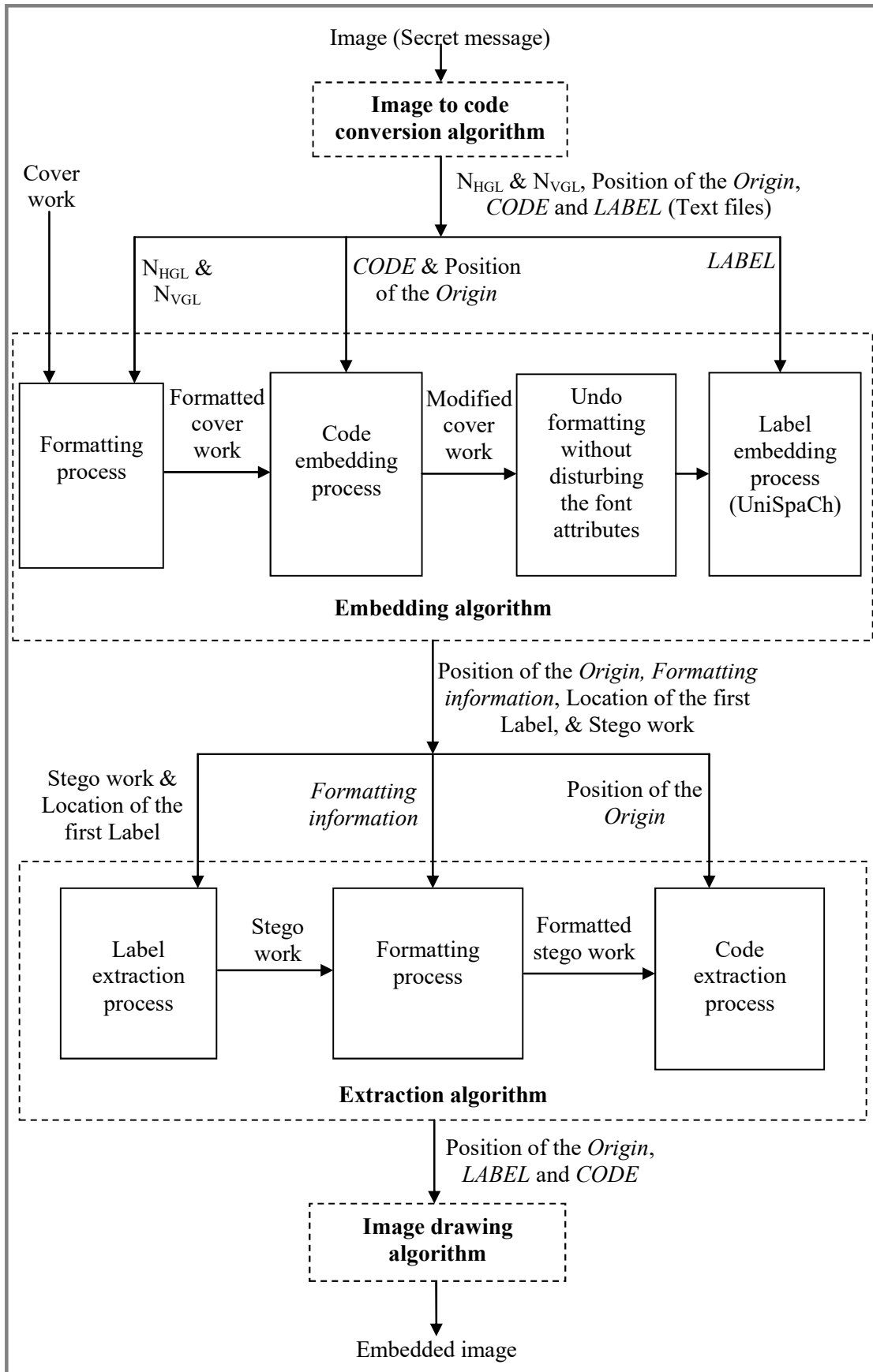
Whenever a LPoB is encountered, a line is drawn and an element is popped from the stack. Now the cursor is moved to the respective location in the application. After this, the codes are read sequentially and the lines are drawn as explained. Whenever an OPoB is encountered, an element is popped from the stack and the same procedure explained for LPoB is followed.

Whenever a *Flag* is encountered, during the process, corresponding Label is retrieved and inserted in the image accordingly. The procedure ends once all the codes are converted into corresponding lines.

#### 6.4 Methodology Adopted to Develop Method-C

Method-C has been developed as four modules (see Fig. 6.9). The first module, *image to code conversion algorithm*, converts an image into codes. The output of this module and the cover document are fed as input to the, second module, *embedding algorithm*.





**Figure 6.9** Modules of Method-C.  $N_{HGL}$  – Number of Horizontal Grid Line;  $N_{VGL}$  – Number of Vertical Grid Line

The *embedding algorithm* formats the cover document and then embeds the codes inside it by modifying the font attributes. In addition, it embeds the Labels using UniSpaCh (the reason for choosing UniSpaCh over Method-B is explained later in Section 6.5.5). The third module, *extraction algorithm*, extracts the embedded codes and Labels, and stores them in separate files. The fourth module, *image drawing algorithm*, draws the embedded image using the extracted codes and inserts the Labels, accordingly. All these algorithms are explained below in detail.

#### 6.4.1 Image to Code Conversion Algorithm

As mentioned in Section 6.3.3, the algorithm converts image into codes. It generates two text files, called *CODE* (contains codes) and *LABEL* (contains Labels), and information such as  $N_{HGL}$ ,  $N_{VGL}$ , position of the *Origin*, as output.

#### 6.4.2 Embedding Algorithm

Embedding algorithm considers *CODE* and *LABEL* as the secret message and embeds them inside the cover document using the attributes Color, Kerning and Spacing. The information related to FPoB and TP are embedded in the least significant bit of Color (RGB), indicated as A, B and C respectively.

FPoB, LPoB, SP and  $H_4$  values are embedded in Kerning, indicated by D to M as shown in Table 6.7. After modifying the corresponding bits, they are converted to their equivalent decimal value, which is then added to a default value (say 100). The resultant is the required kerning value of the character.

**Table 6.7** Kerning value

Bit position	9	8	7	6	5	4	3	2	1	0	Kerning value
Indication	M	L	K	J	I	H	G	F	E	D	100 + Decimal value of D to M

*Flags* are embedded in the Spacing as specified in Table 6.8. Although, the algorithm supports the codes to be embedded in multiple pages (for bigger images), we restrict our discussion to a single page,  $S_p$ .

**Table 6.8** Flags and their respective spacing values

Flag	Spacing
\$	-0.1
#	-0.2
\$#	+0.1
NULL	Default

To begin with, the embedding algorithm formats  $S_p$  into a monospaced font (every letter is the same width [153]), like Courier New, with equal number of characters in each line. The number of characters per line should be  $\geq N_{VGL}$  and the number of lines should be  $\geq N_{HGL}$  (this can be achieved by varying the font size and line spacing). The number of characters per line and the number of lines in  $S_p$  are the *Formatting information*. Except the characters on edges, all others have eight neighbors (Right, Top-Right, Top, Top-Left, Left, Bottom-Left, Bottom and Bottom-Right). These eight neighbors are used to accommodate the sixteen possible directions (refer Table 6.6), as shown in Table 6.9. This limits the algorithm to support a maximum of seven branches at a SP.

**Table 6.9** Selecting one character from eight neighbors based on  $H_4$

$H_4$	Selected neighbor	$H_4$	Selected neighbor
0000	Right	1000	Left
0001, 0010, 0011	Top-Right	1001, 1010, 1011	Bottom-Left
0100	Top	1100	Bottom
0101, 0110, 0111	Top-Left	1101, 1110, 1111	Bottom-Right

Now, a stack is declared and the character corresponding to the *origin* is selected to embed the first code. The codes are embedded based on the type of Form, which is discussed below in detail. After embedding a code, the  $H_4$  value of the embedded code is used to select the next character to embed the next code.

- *Form 1: ( $^*H_4$ )* – The  $H_4$  value is embedded at position “F G H I”. If  $H_4 = 0$ , then “D” is set. By referring Table 6.9, the next character is selected using  $H_4$
- *Form 2: ( $^+X, H_4$ )* – The  $H_4$  value is embedded at position “F G H I” and “X” is embedded at position “D”. The location of the character (line no., char no.), where this code is embedded, is stored in the stack for  $(H_4-1)$  times. By default, Form 2 is followed by Form 3
- *Form 3: The two (First and Second) codes that immediately follow Form 2* – The code of this Form represents the FPoB corresponding to a SP. The  $H_4$  value of the *First code* is used to choose one of the neighboring characters to embed Form 3. The  $H_4$  value of the *First code* is embedded at position “J K L M” and the  $H_4$  value of the *Second code* is embedded at position “F G H I”. The alphabet “Y” of the *First code*, if any, is embedded at position “A”. In addition, if the  $H_4$  value of the *First code* is all 0 then “B” is set. Based on the Form of *Second code*, it is embedded in the appropriate position and corresponding action is taken based on its Form
- *Form 4: ( $^*Y, H_4$ )* – The  $H_4$  value is embedded at position “F G H I” and “Y” is embedded at position “C”. By referring Table 6.9, the next character is selected using  $H_4$
- *Form 5: ( $^*H_4, X$ )* – The  $H_4$  value is embedded at position “F G H I” and “X” is embedded at position “E”. By default, Form 5 is followed by Form 7

- *Form 6: (<sup>+</sup>Y, 0000, X)* – The H<sub>4</sub> value is embedded at position “F G H I” and “X” is embedded at position “B”. “Y” is embedded by setting “C”. By default, Form 6 is followed by Form 7
- *Form 7: The two (First and Second) codes that immediately follow Form 5 or Form 6* – The code of this Form represents the FPoB corresponding to a SP. When Form 7 is encountered, an element is popped from the stack. The character at the popped location is selected and the procedure is continued further. Form 7 is embedded in the same manner as explained in Form 3. After embedding the codes, appropriate action is taken based on the Form of *Second code*

In all the above Forms, “\*” and “+” represent the respective Labels (refer Table 6.5). These are embedded by setting the corresponding spacing values as specified in Table 6.8.

After embedding the codes, all format related modifications are reverted without disturbing the font attributes. Now, a character is selected and the Labels that are present in *LABEL* are embedded using UniSpaCh.

After embedding all the Labels, the embedding procedure ends. The generated stego work is communicated to the receiver. Also, information such as the position of the *origin*, *Formatting information*, Label separator and position of the first Label are communicated.

### 6.4.3 Extraction Algorithm

Extraction procedure is the reverse of the embedding process. To start with, the Labels are extracted from the received document using UniSpaCh and stored in a file called *LABEL*. Now, the formatting process identical to that of the embedding

algorithm is performed. Locations (line no., char no.) of all the stego characters are identified and stored in an ArrayList “SC”. Also, a stack is declared and a file called *CODE* is used to write the extracted codes. Now, the character corresponding to the *origin* is selected and the procedure followed is shown in Figures 6.10 (A), (B) and (C).

The extraction algorithm identifies the *Form* of code embedded in the selected character. When either a *Form 1* or *Form 4* is identified, the code is written in the file and the next character is selected (using  $H_4$ ) to proceed further.

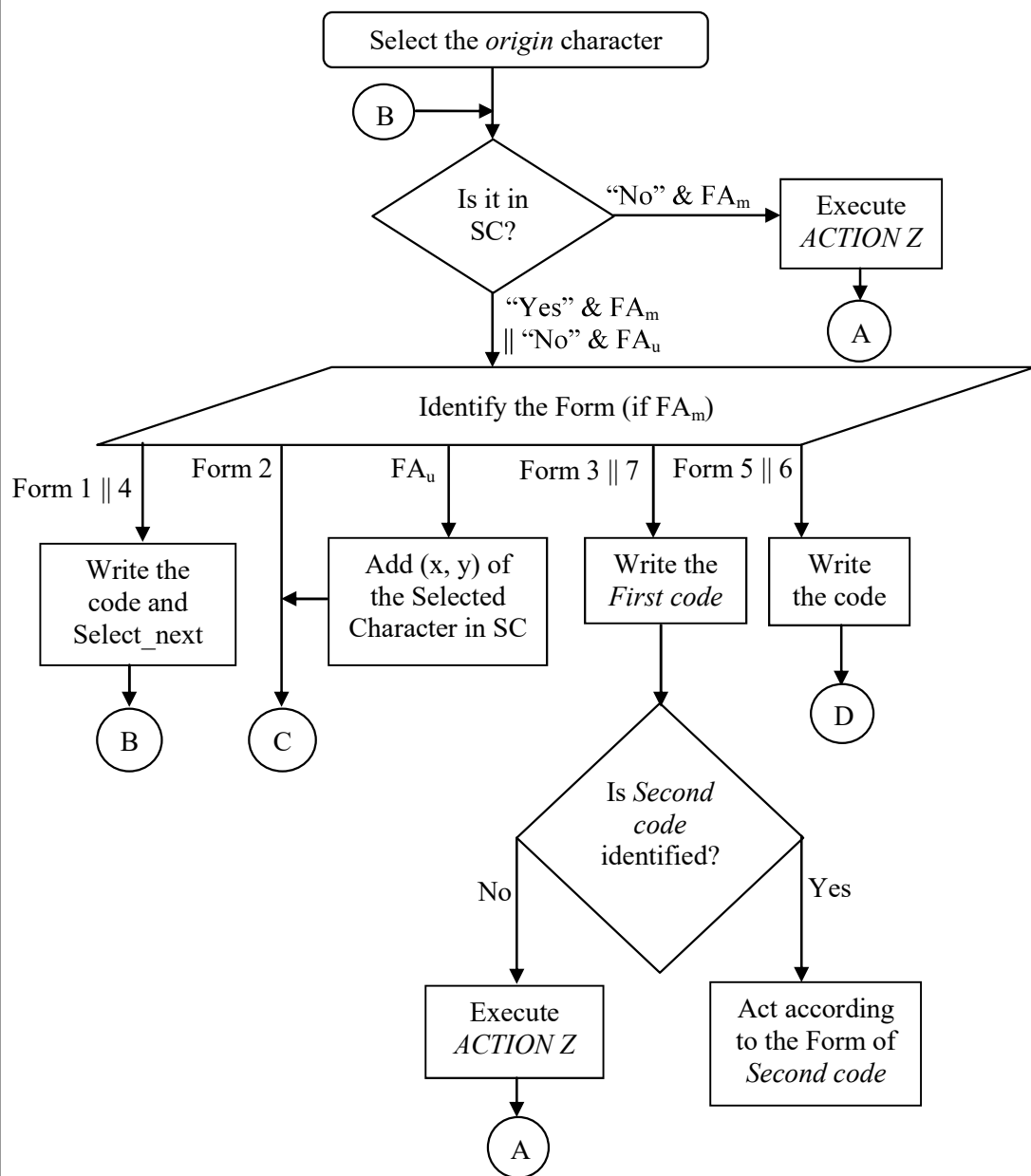
When a *Form 2* is identified, the neighbors that carry the associated *Form 3* are recognized in the clockwise direction starting from the Bottom-Right neighbor. Location(s) of the recognized neighbor(s) are stored in that order in the stack, and the code (X, no. of neighbors recognized) is written in file. Doing so, automatically handles the errors that are caused in *Form 2*, and in the *First code* of *Form 3* and *Form 7*. Now, the topmost element from the stack is popped and the extraction process continues from that location.

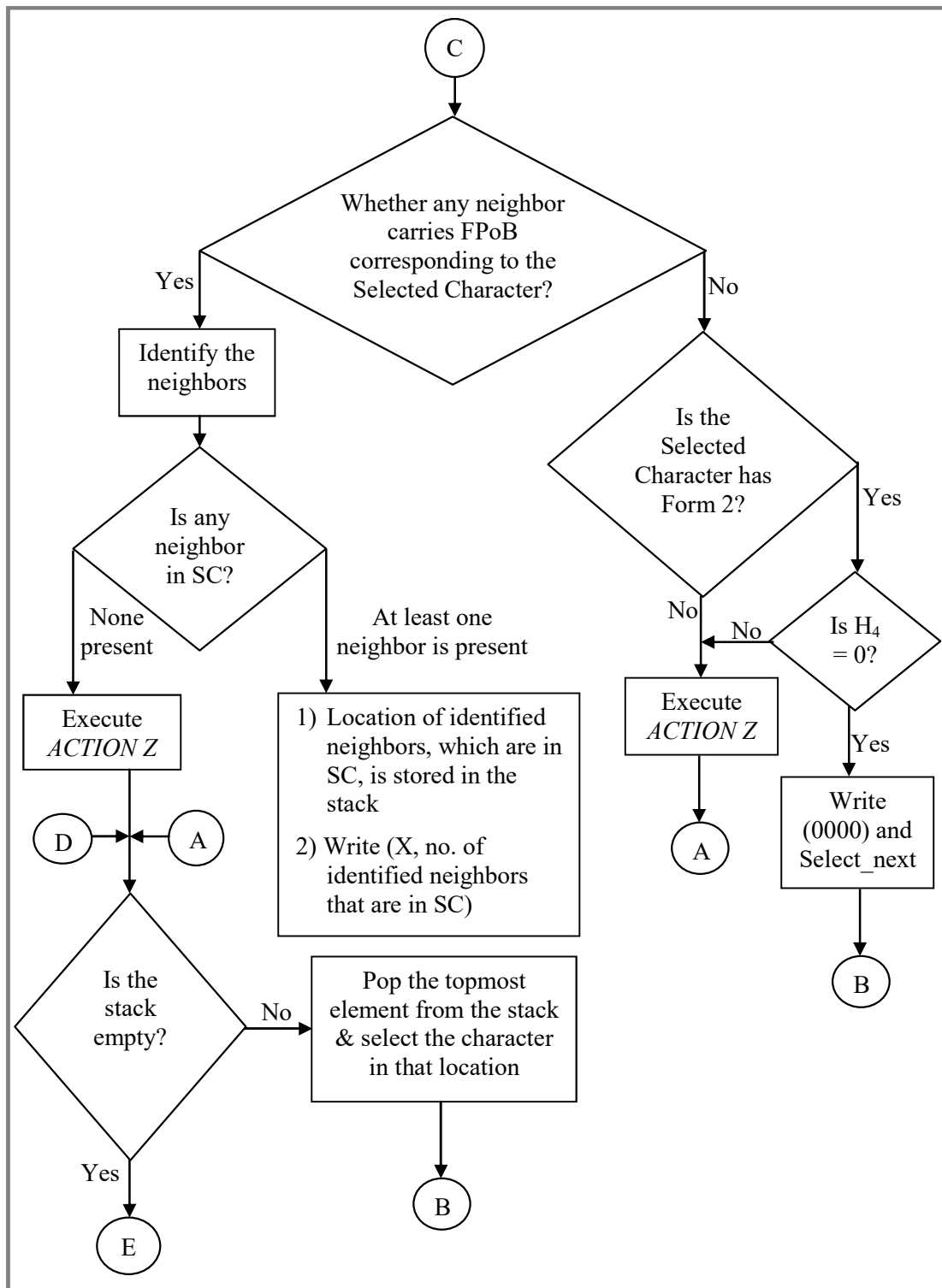
Suppose, when a *Form 2* is identified and no neighbors carry an associated *Form 3*, then the value of  $H_4$  is checked. If it is “0”, then the procedure of *Form 1* is followed. Else, a tag “, X” is appended to the most recently written code and the other branches are traversed by popping an element from the stack.

When either a *Form 3* or *Form 7* is identified, the *First code* is written in the file. If the *Second code* is present, then appropriate action is taken based on the *Form* of *Second code*. Else, the code (Y, 0000, X) is written in the file to handle the error that is caused in the *Second code*. After this, the other branches are traversed, if any, as explained above.

**Note:**

- SC – ArrayList
- Whenever a code is extracted from a character or ACTION Z is executed, remove the currently Selected Character's location from SC
- $FA_u$  – Font attribute is unmodified
- $FA_m$  – Font attribute is modified
- Select\_next – Select next character using Table 6.9
- ACTION Z – If the Selected Character has Form 3 || 7, then write (Y, 0000, X). Else, append the tag “, X” to the most recently written code

**Figure 6.10 (A)** Extraction algorithm: flowchart 1



**Figure 6.10 (B)** Extraction algorithm: flowchart 2



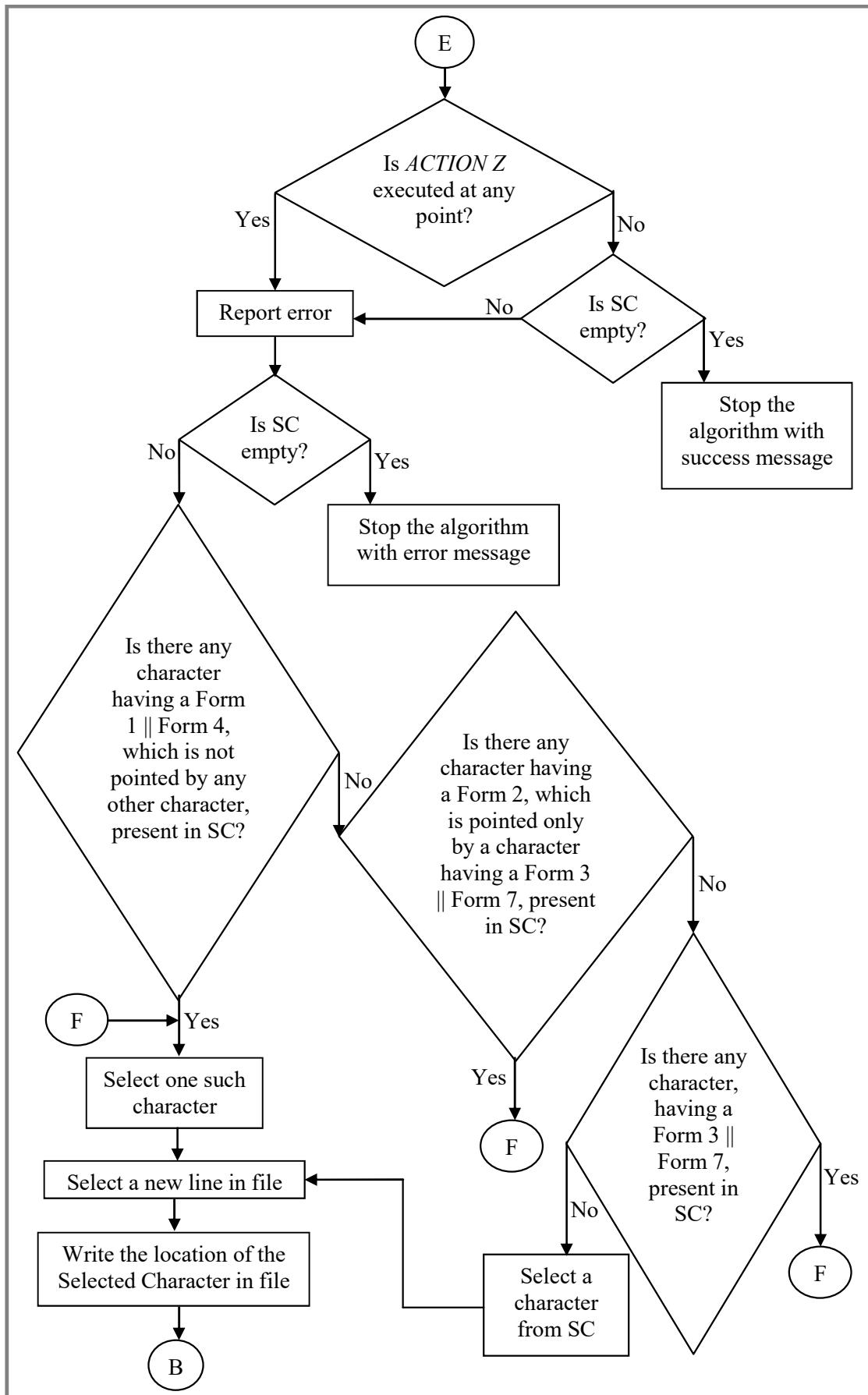


Figure 6.10 (C) Extraction algorithm: flowchart 3

When either a *Form 5* or *Form 6* is identified, the code is written in the file and the other branches are traversed, if any.

### **Error Handling**

It should be noted that, whenever a code is extracted from a character, the location of that character is removed from “SC”. Hence, trying to pop from an empty stack, with some elements left in “SC”, indicates the occurrence of error in *Form 1*, *Form 3* or *Form 7*. These errors are handled by picking an element from “SC” and begin extracting codes from that location (refer Fig. 6.10 (C)).

Also, whenever an unmodified character is encountered, during the extraction process, it first checks whether any of its neighbors carry *Form 3* corresponding to the selected character (refer Figures 6.10 (A) and (B)). If any, then it considers that the selected character has carried *Form 2* and proceeds further with the procedure of *Form 2*. If no such *Form 3* is present, then the tag “, X” is appended to the most recently written code. After this, an element is popped from the stack and the same procedure is followed as explained above. This shows that the errors caused during transmission will affect only that particular code but not the complete image. This is achieved due to the structural embedding nature of the embedding algorithm which facilitated the extraction algorithm to handle the above-mentioned errors.

#### **6.4.4 Image Drawing Algorithm**

The *image drawing algorithm* follows the same procedure explained in Section 6.3.4. During the process, any code with a tag is considered as a normal code and the procedure is followed accordingly. Also, whenever a new line is encountered

in *CODE*, the cursor is moved to the corresponding location in the line drawing application and the procedure is continued as explained.

## 6.5 Evaluation and Security Aspect

This section discusses the various parameters that are used to evaluate Method-C and the security aspects in some detail.

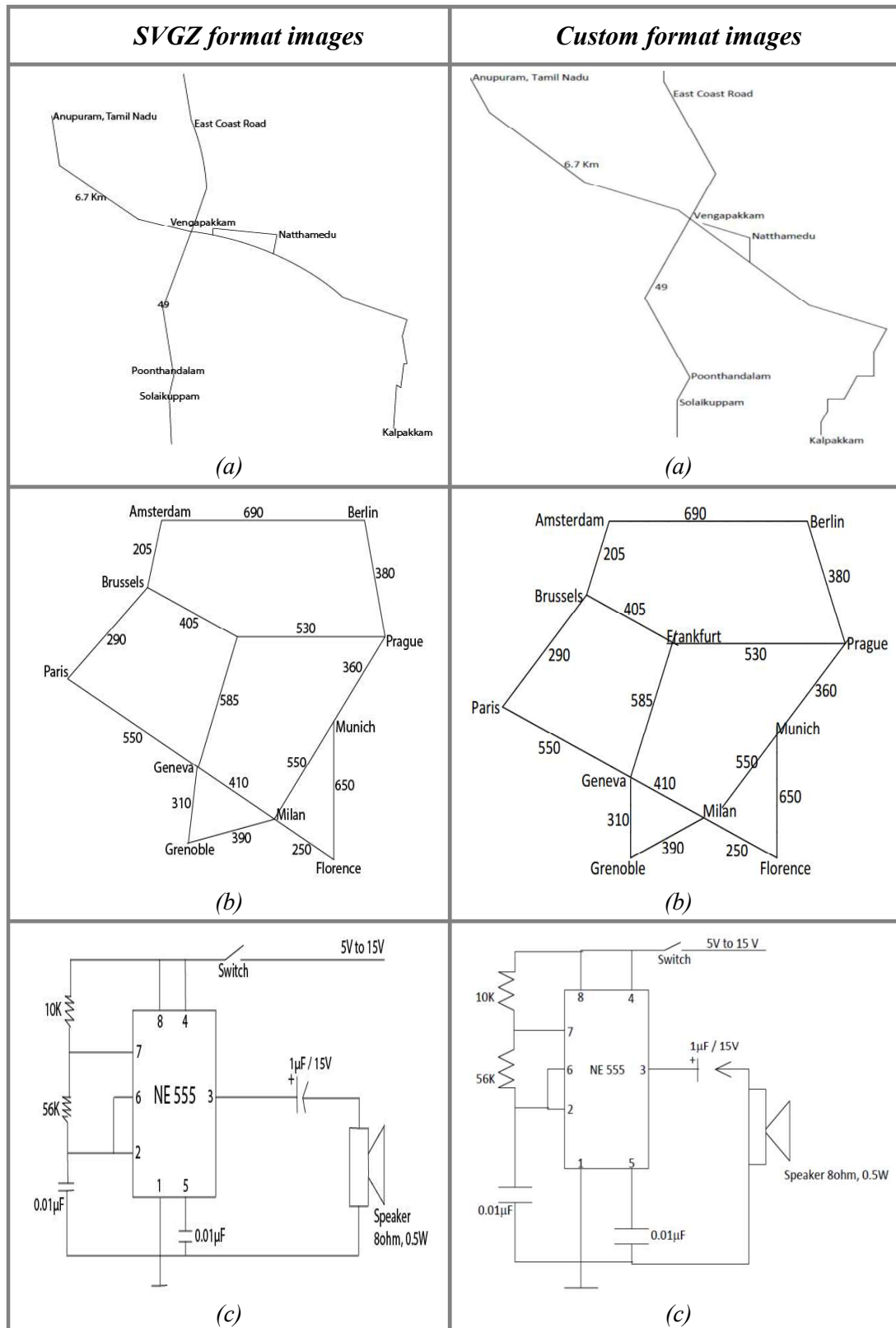
### 6.5.1 Size Comparison

An image is converted into custom defined format and then embedded inside a cover document. Hence to find the efficiency of the custom format, it is compared with the vector format (SVGZ file of Adobe Illustrator). The results are listed in Table 6.10, and the generated images of both formats are shown in Figures 6.11 (A), (B) and (C).

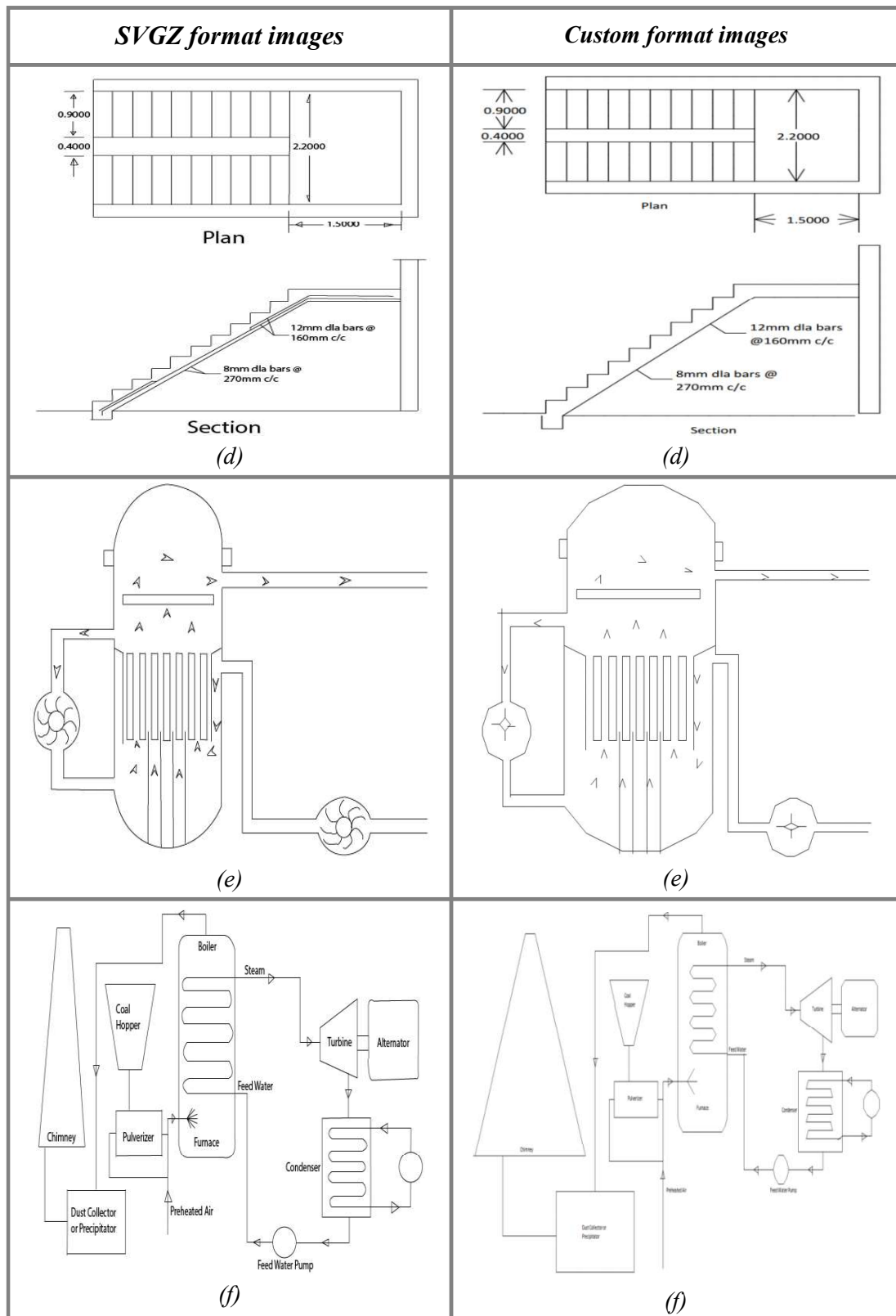
**Table 6.10** Sizes of images in custom and SVGZ formats

Figure no.	SVGZ file in bits (A)	Custom format (in bits)			Efficiency <sup>†</sup> (in %)
		<i>CODE</i> <sup>‡</sup>	<i>LABEL</i> <sup>†</sup>	Total (B)	
5.3 (a)	7240	765	848	1613	77.72
5.3 (b)	7184	765	1176	1941	72.98
5.3 (c)	10895	1611	784	2395	78.02
5.3 (d)	13353	3312	728	4040	69.74
5.3 (e)	19497	5517	NA	5517	71.70
5.3 (f)	18432	5913	1224	7137	61.28
5.3 (g)	14746	5346	760	6106	58.59
5.3 (h)	49971	16101	2208	18309	63.36
5.3 (i)	37192	7272	7216	14488	61.05

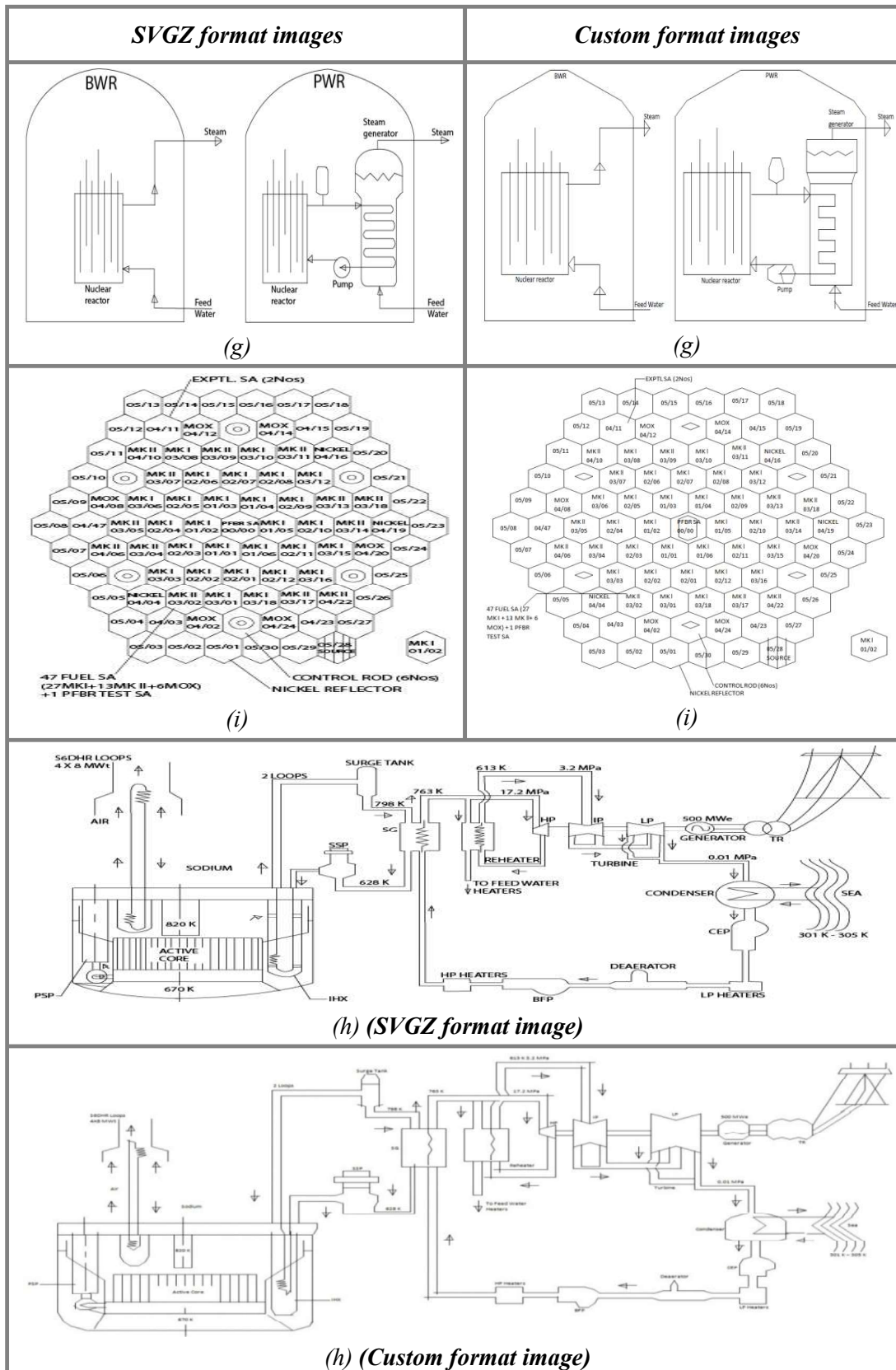
<sup>‡</sup> – each code can be represented as a 9-bit binary string using the structure “\$#XYH<sub>4</sub>X”; <sup>†</sup> – each character in *LABEL* is represented by an 8-bit ASCII value; <sup>†</sup> – Relative Efficiency =  $100 - ((B / A) * 100)$ ; SVGZ – Scalable Vector Graphics Compressed; NA – not applicable



**Figure 6.11 (A)** Generated images of SVGZ and custom formats — part 1: (a) Road map from kalpakkam to anupuram; (b) Graph; (c) Electronic circuit diagram



**Figure 6.11 (B)** Generated images of SVGZ and custom formats — part 2:  
 (d) Civil drawing of stairs; (e) Boiling water reactor;  
 (f) Schematic diagram of thermal power plant



**Figure 6.11 (C)** Generated images of SVGZ and custom formats — part 3:  
 (g) Nuclear power plant steam generation; (h) Reactor flow sheet; (i) Reactor core

From Table 6.10, it is evident that the custom format efficiently represents images in smaller sizes, when compared with the SVGZ file format.

### 6.5.2 Embedding Capacity

Embedding capacity is the maximum amount of information that can be hidden in a given cover medium. Since an image is embedded, in the form of codes, its embedding capacity can be defined in two different ways: one is based on the size of the image (equation 6.1) and the other is based on the total size of its *CODE* and *LABEL* (equation 6.2).

$$\begin{array}{l} \text{Embedding capacity per cover – character} \\ \text{(Image Space)} \end{array} = \frac{\text{Size of image in bits}}{\text{Total no. of cover characters required}} \quad (6.1)$$

$$\begin{array}{l} \text{Embedding Capacity per cover – character} \\ \text{(Code Space)} \end{array} = \frac{\text{Total size of } CODE \text{ and } LABEL \text{ in bits}}{\text{Total no. of cover characters required}} \quad (6.2)$$

To verify the embedding capacity, the custom format of the images (mentioned in Table 6.10) is embedded inside a cover document and the results are listed in Table 6.11.

From Table 6.11, it is noticed that the embedding capacity of Method-C varies from one image to another and is not uniform, unlike the methods Method-A and Method-B. This is due to the structure of the respective images.

### 6.5.3 Bits per Distortion

The custom format first identifies the various elements of an image and represents them as codes. These codes are then embedded inside the cover document. Hence, bits/distortion of the method, also, varies from code to code and is not

uniform. That is, different stego characters carry different number of secret bits. For example, the cover character which embeds a DP or DCP, represented by  $(H_4)$  — without considering Labels, embeds 4-bits/distortion whereas a SP, TP and LPoB, represented by  $(X, H_4)$ ,  $(Y, H_4)$  and  $(H_4, X)$  respectively, embeds 5-bits/distortion.

**Table 6.11** Results of embedding the custom format

Figure no.	No. of cover characters encountered while embedding			Embedding capacity	
	<i>CODE</i> <sup>*</sup>	<i>LABEL</i> <sup>T</sup>	<i>CODE + LABEL</i>	Image Space	Code Space
5.3 (a)	1787	811	2598	2.79	0.62
5.3 (b)	764	1124	1888	3.81	1.03
5.3 (c)	557	750	1307	8.34	1.83
5.3 (d)	821	696	1517	8.80	2.66
5.3 (e)	3667	NA	3667	5.32	1.50
5.3 (f)	6057	1170	7227	2.55	0.99
5.3 (g)	1188	727	1915	7.70	3.19
5.3 (h)	6996	2111	9107	5.49	2.01
5.3 (i)	2695	6899	9594	3.88	1.51

*\* – includes the space characters that was removed during the formatting process; <sup>T</sup> – embedding capacity of UniSpaCh is considered as 1.046-bits/cover-character; NA – not applicable*

Further analysis concluded that the method embeds a minimum number of bits (4-bits/distortion), when a cover character carries a DP or DCP, and a maximum number of bits (12-bits/distortion), when a cover character carries a FPoB with codes  $(Y, H_4)(\$H_4, X)$  or  $(Y, H_4)(\$Y, H_4)$ .

#### 6.5.4 Secrecy

As three attributes Color, Kerning and Spacing are utilized to embed the secrets, the imperceptibility level of the method is tested with different images. For



example, the code given in Fig. 6.8 is considered and its working is illustrated in Fig. 6.12 by taking the *origin* as (7, 1).

Cover:	Stego:	Stego*:
Interne	Interne	Interne
twhichi	twhichi	twhichi
sextens	sextens	sextens
ivelyus	ivelyus	ivelyus
edtosha	edtosha	edtosha
reanyki	reanyki	reanyki
ndofinf	ndofinf	ndofinf

**Figure 6.12** Sample output of Method-C. \*Stego characters are highlighted for understanding purpose

By looking at both the cover and stego work of Fig. 6.12, it can be verified that the embedding procedure has not created any visual difference between them. This validates that the developed method achieves high imperceptibility.

### 6.5.5 Comparison of Method-B and Method-C

In Section 5.7 of Chapter 5, it was shown that Method-B outperforms the best existing method, UniSpaCh. Hence, Method-B has been considered as a benchmark and compared with Method-C based on the number of stego characters (cover characters whose font attributes are modified) and page requirements. The results are provided in Table 6.12.

From Table 6.12, it can be noticed that Method-C uses UniSpaCh to embed the Labels, and still it embeds the images in smaller size cover document with less number of modifications. This shows that the results can further be improved by using Method-B to embed the Labels.

### 6.5.6 Transmission Error

During communication between sender and recipient, transmission errors can occur which can corrupt some of the stego characters. As the developed method embeds the structure of an image, as it is, corruption of an embedded secret in a stego character will affect only that code but not the complete image. This property paves a way for the receiver to handle transmission errors to some extent and extract the embedded image.

**Table 6.12** Comparison of Method-B and Method-C

Figure no.	Method-C				Method-B embedding SVGZ images		Efficiency based on no. of	
	No. of stego characters to embed			No. of pages (B)				
	<i>COD E</i>	<i>LAB EL</i>	<i>CODE + LABEL</i> (A)		No. of stego characters (C)	No. of pages (D)	Stego characters <sup>‡</sup>	Pages <sup>†</sup>
5.3 (a)	80	424	504	2	1810	3	72.15	33.33
5.3 (b)	76	588	664	2	1796	3	63.03	33.33
5.3 (c)	144	392	536	1	2724	4	80.32	75.00
5.3 (d)	280	364	644	1	3338	5	80.71	80.00
5.3 (e)	613	0	613	3	4874	7	87.42	57.14
5.3 (f)	657	612	1269	5	4608	7	72.46	28.57
5.3 (g)	490	380	870	2	3687	5	76.40	60.00
5.3 (h)	1420	1104	2524	6	12493	18	79.80	66.67
5.3 (i)	591	3608	4199	6	9298	13	54.84	53.85

<sup>‡</sup> – Relative Efficiency =  $100 - ((A / C) * 100)$ ; <sup>†</sup> – Relative Efficiency =  $100 - ((B / D) * 100)$ ; SVGZ – Scalable Vector Graphics Compressed

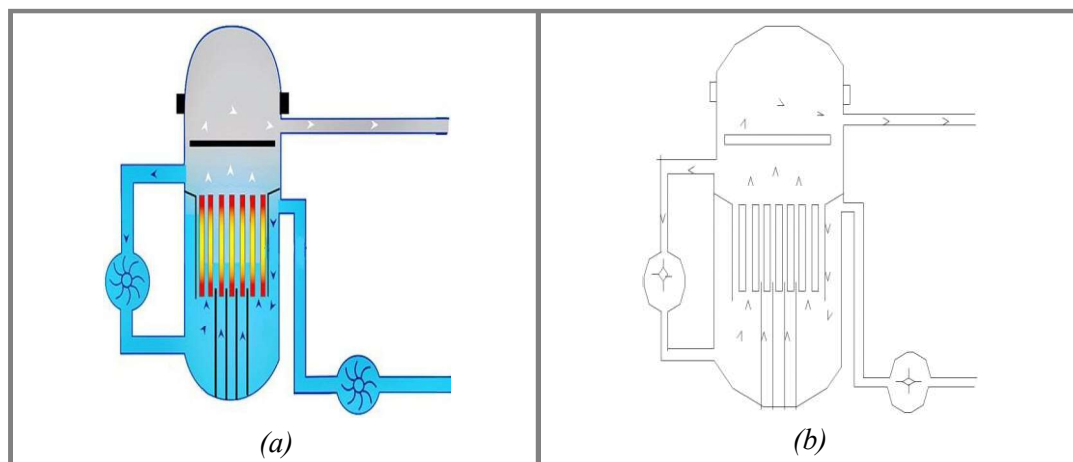
To illustrate this, the color patterns given in Table 6.13 were used while executing the procedure shown in Figures 6.10 (A), (B) and (C). Based on the correctness of extracted code and the execution status of extraction algorithm, an appropriate color is applied to the currently selected character for the primary

identification of transmitted image. As the process of distorting a non-stego character does not affect the method developed in the present work, an illustration is provided in Figures 6.13 (A) and (B), by distorting 15% of the stego characters in a stego work. It should be noted that the extraction algorithm can extract the embedded image, with ease, even when 15% of the embedded secrets were distorted. In addition to applying the color patterns, the *image drawing algorithm* draws the embedded image as explained in Section 6.4.4.

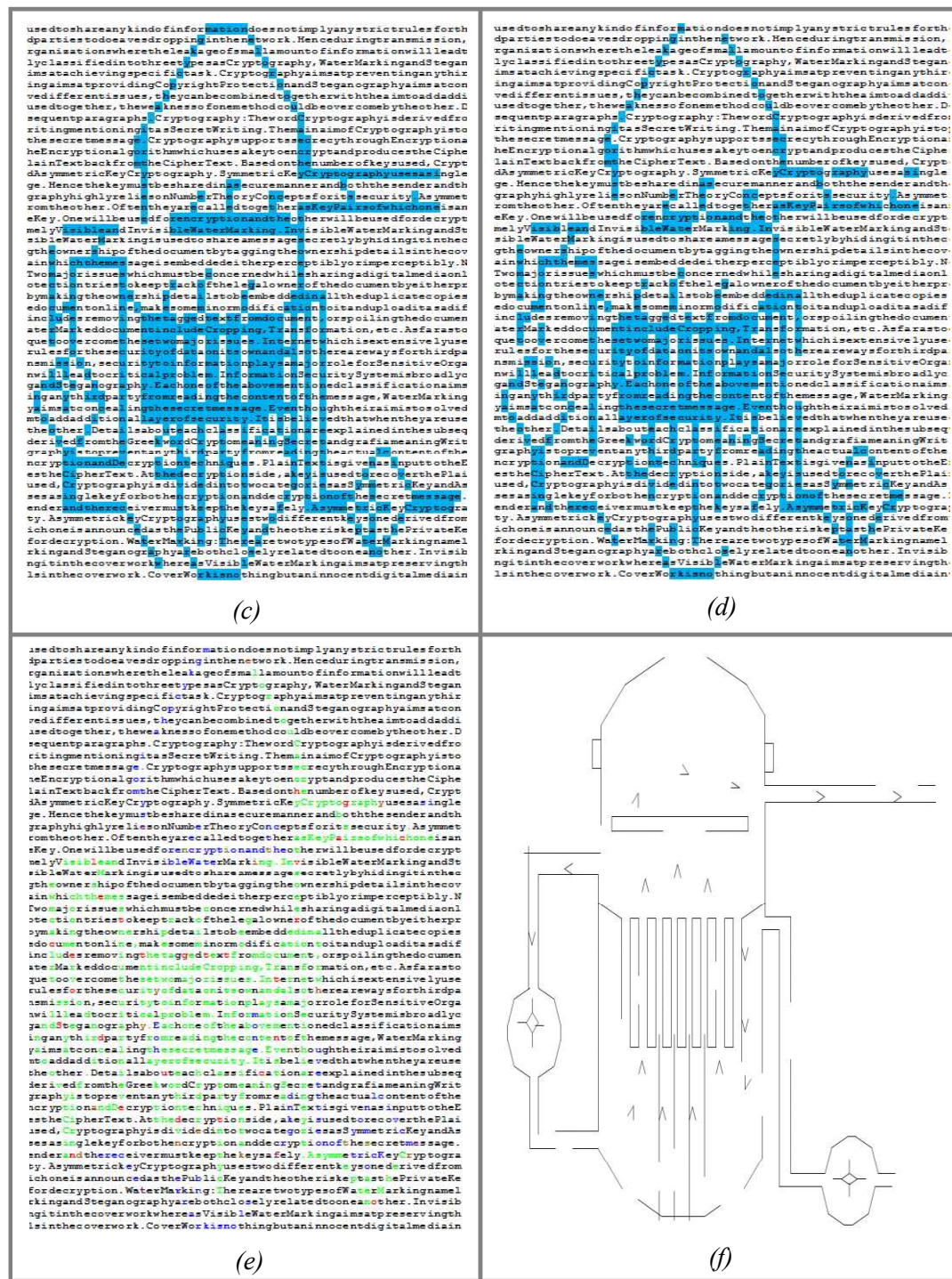
**Table 6.13** Used color patters to color the currently selected character

Execution status	Correctness of extracted code	Color pattern (R,G,B)
Before reporting error	Non-error	(0,255,0)
	$H_4$ mismatch with the number of identified neighbors (for a SP) or a non-stego character is selected whose neighbors carry the corresponding FPoB	(255,0,0)
	Appended tag “, X” or written (Y, 0000, X) in file	(155,74,7)
After reporting error	Non-error	(0,0,255)
	$H_4$ mismatch with the number of identified neighbors (for a SP) or a non-stego character is selected whose neighbors carry the corresponding FPoB	(117,117,5)
	Appended tag “, X” or written (Y, 0000, X) in file	(121,13,108)

$H_4$  – Hexadecimal value; FPoB – first point of a branch; SP – split point



**Figure 6.13 (A)** Illustration of handling transmission error: (a) Original image; (b) Transmitted image



**Figure 6.13 (B)** Illustration of handling transmission error: (c) Stego characters at sender side (highlighted for understanding purpose); (d) Identified stego characters after the occurrence of error (15% of stego characters have been distorted); (e) Applied color patterns for the stego characters in Fig. 6.13 (B) (d) during extraction; (f) Extracted image from Fig. 6.13 (B) (d)

## 6.6 Summary

In this chapter, the large cover document requirement of Method-B is addressed for a specific category of images. First, the size of the image is reduced to the extent possible through vector formats and then embedded them using Method-B. It is seen that the best vector format, SVGZ, considerably reduced the page requirement. However, even for single bit error, the method failed to extract a substantial portion of the image. To account for the same, error correcting codes has to be introduced as a part of embedding procedure, which increases the size of message that needs to be embedded. This is not an encouraging one. Hence, it is identified that an embedding algorithm that provides the necessary error handling capabilities to extraction algorithm, without adding redundancy, is a necessary one.

To achieve the same, a novel method (Method-C) has been developed to convert images into a custom format and embed the same along with the structure of image. This structural embedding nature facilitated the extraction algorithm to handle transmission errors. The error handling capabilities of the developed method are inspected and an illustration is, also, provided. The output file sizes of the custom and SVGZ formats are compared to verify the efficiency of the former format. Further evaluation led to the conclusion that the method has non-uniform embedding capacity and bits/distortion.

### SUMMARY AND SCOPE FOR FUTURE INVESTIGATIONS

---

*This chapter gives the summary of the work carried out, methods developed and conclusions arrived at, and future work that can be pursued.*

#### 7.1 Summary and Conclusions

Digital communication plays an important role in connecting the geographically distributed individuals as well as organizations. Today, even sensitive information is being communicated employing internet. As information security is of utmost priority for the organizations, achieving the same is a challenging task while using the public network. Though cryptography could be used to enhance the information security, it cannot hide its own presence from the attackers.

Alternatively, steganography, an information security measure, averts such attention by performing the communication in a stealthy manner. It hides the secret information in the redundant information of an innocent looking cover medium by making unnoticeable modifications. This characteristic attracted and narrowed down the research interest of the work, to be carried out as a part of the dissertation, to steganography.

Of the various cover types, documents containing texts are widely used by the organizations and are communicated using public networks. Also, text documents require low bandwidth during communication. Due to these advantages, the work carried out in the present study involves the development of efficient text steganographic techniques for a safer and stealthier communication.

A brief study on existing text steganographic methods was carried out and discussed in Chapter 2. A detailed comparison of the methods was presented and it led to the conclusion that UniSpaCh stands best among the rest. It was also noticed that, apart from the methods that generate the stego document directly, embedding capacity and bits/distortion of the existing methods are inversely proportional. As a result of this, embedding secret messages that are larger in size is not easy to be met by text steganographic techniques. This necessitated the development of novel methods, with high embedding capacity and bits/distortion, in the current scenario.

Based on a detailed analysis, it was understood that this limitation can be addressed by designing a method that: (i) embeds maximum number of data bits in a distortion; (ii) utilizes the available embedding space in an efficient manner. Bearing this in mind, features of various word processors were studied to identify the best word processor that supports steganography, along with the best feature that can carry a large number of bits per distortion. The possible ways to exploit the available font attributes was demonstrated in Chapter 3. The results led to the conclusion that the word processor Microsoft (MS) Word stands best with the attributes Color, Kerning and Spacing providing excellent opportunities to hide data.

After identifying the possible way to achieve high bits per distortion in Chapter 3, the potential techniques to improve the utilization of the available embedding space were explored and discussed in Chapter 4. Of the presented steganographic techniques, Cover\_Document\_Required (CDR) techniques consider the secret message as characters and embed them directly inside cover documents. This makes these methods an optimum choice for embedding text as they embed 8-bits/distortion. However, due to the non-uniform occurrence probabilities of characters in the cover document, the available embedding space gets wasted



whenever a low occurring character needs to be marked. This affected the overall embedding capacity of these techniques.

The necessary measures to handle this limitation were identified and reported, in Chapter 4, along with a novel method (Method-A) that uses the attribute Spacing to embed the secrets. The method introduced two new techniques Frequency Normalization Set (FNS) and Character & String Mapping (CSM). The combination (FNS and CSM) allowed single secret character to get embedded in multiple cover characters. This boosted the embedding probabilities of low and average occurring characters, and made the embedding probabilities of all the characters uniform. As a result, the method achieved an average embedding capacity of 2.22-bits/cover-character with 8-bits/distortion. This facilitated to, considerably, reduce the size of cover document and number of modifications that are required to embed the secrets. However, the method restricted the secret message to contain only English alphabets, Dot and Space (ADS) characters. This drawback limited the method from embedding messages that contains numbers and special characters viz. mobile, credit card, debit card, etc.

This limitation was addressed, in Chapter 5, by extending Method-A into a mixed-type embedding technique (Method-B) that embeds binary data. This was possible by converting the binary data into ADS characters, using a one-to-many mapping module called Binary to Character Converter (BCC). As the converted message looked almost identical to the secret message used in Chapter 4, the embedding and extraction procedures of Method-A successfully utilized them with ease. Experiments were conducted using various secret messages. The results depicted that the conversion procedure of Method-B has reduced the earlier attained embedding capacity and bits/distortion of Method-A by 12.5% and 50% respectively.



A case study on nuclear power plant related images concluded that Method-B has reduced the size of required cover document by 38.42%, when compared with UniSpaCh. But, it still requires more than hundred pages of cover document when the image size is  $\approx 40$  KB. This limitation is applicable to any type of binary data and handling such limitation differs from one data type to another.

In Chapter 6, this drawback was addressed for a specific category of images such as engineering drawings, roadmaps, graphs, etc. At first, an attempt was made to reduce the size of images through vector formats and then embedded them using the method explained in Chapter 5, Method-B. The vector format, SVGZ, considerably reduced the page requirement. However, even for single bit error, the method failed to extract a substantial portion of the image. It was observed that no existing method can provide the required error handling capabilities without increasing the size of secret message.

In Chapter 6, this issue was addressed by developing a novel method (Method-C) that represents an image in a custom format, with reduced size, and then embeds the same along with the structure of image. This structural embedding nature provided the extraction algorithm the expected error handling capabilities. The same has been inspected and an illustration is, also, provided in this chapter. Further evaluation concluded that, the method has non-uniform embedding capacity and bits/distortion.

The work carried out as part of the thesis has resulted in three novel methods that achieve high embedding capacity and bits/distortion:

- (i) Method-A embeds secret text, that contains ADS characters, using the attribute Spacing
- (ii) Method-B embeds binary data using the attribute Spacing

- (iii) Method-C embeds a specific category of images, along with their structure, using the attributes Color, Kerning and Spacing

Key findings of the thesis are summarized below:

- Character-level embedding techniques stand best, in terms of bits/distortion, by embedding 8-bits at a time
- Amongst the available methods, UniSpaCh stands best with an average embedding capacity of 1.046-bits/cover-character and  $\approx 2$ -bits/distortion (this is based on the literature survey)
- Except those methods that generate the stego document directly, no existing method was found to achieve both high embedding capacity and bits/distortion
- MS Word stands best for steganographic usage when compared with other word processors like OpenOffice, LibreOffice and WordPerfect
- Font attributes Color, Kerning and Spacing of MS Word achieves high imperceptibility level and embedding capacity
- Irrespective of the non-uniform occurrence probabilities of secret characters in cover documents, uniformity in embedding probability is possible in CDR techniques
- Size of cover document, required to embed images, can be reduced through vector formats
- Embedding images with error handling capabilities is achievable in text steganography

## **7.2 Scope for the Future Work**

The work carried out in the present study suggests that the following investigations could be taken up in future:

- A method to embed chemical equations and mathematical expressions could be developed
- Other possible objects of text documents such as table, graph, chart, equation, etc., could be explored
- The developed methods, Method-A and Method-B, could be extended to employ randomization in the embedding procedure by using a secret key and pseudo-random number generator
- Other document formats, like Portable Document Format (PDF), electronic publication (ePub), PostScript (PS), etc., could be explored to embed the secrets

## REFERENCES

- [1] Rob Kitchin, *The data revolution: Big data, open data, data infrastructures and their consequences.*: Sage, 2014.
- [2] Sarah Genner, *ON/OFF: Risks and rewards of the anytime-anywhere internet.*: vdf Hochschulverlag AG, 2017.
- [3] Lucy A. Tedd and J. A. Large, *Digital libraries: Principles and practice in a global environment.*: Walter de Gruyter, 2005.
- [4] Bhadoria et al., *Exploring enterprise service bus in the service-oriented architecture paradigm.*: IGI Global, 2017.
- [5] Miranda Walker, *Cambridge national level 1/2 child development.*: Hachette UK, 2017.
- [6] Constantine Photopoulos, *Managing catastrophic loss of sensitive data: A guide for IT and security professionals.*: Syngress, 2011.
- [7] Anne Rooney, *Computer science and IT: Investigating a cyber attack.*: Raintree, 2014.
- [8] Sean G. Lowther, *An employee's guide to safeguarding sensitive information properly: 12 keys every employee should know.*: BookBaby, 2012.
- [9] [Online]. <http://money.cnn.com/2017/09/01/technology/business/instagram-hack/index.html> [Last accessed on 27.8.2017].
- [10] [Online]. <https://www.cnn.com/2016/05/13/a-surprising-source-of-hackers-and-costly-data-breaches.html> [Last accessed on 27.8.2017].

- [11] [Online]. <http://www.13newsnow.com/news/local/employee-data-breach-report-ed-at-tcc/102256781> [Last accessed on 27.8.2017].
- [12] [Online]. <https://economictimes.indiatimes.com/small-biz/security-tech/security/the-worst-cyber-attacks-of-2016/articleshow/56212448.cms> [Last accessed on 27.8.2017].
- [13] [Online]. <http://www.computerworld.com/article/3088907/security/hacker-selling-655-000-patient-records-from-3-hacked-healthcare-organizations.html> [Last accessed on 27.8.2017].
- [14] [Online]. <http://www.encyclopedia.com/history/energy-government-and-defense-magazines/chemical-terrorism-threats> [Last accessed on 27.8.2017].
- [15] [Online]. <http://www.dailymail.co.uk/news/article-2524082/All-US-Armys-secret-bases-mapped-Google-maps.html> [Last accessed on 27.8.2017].
- [16] Sudhi R. Sinha and Youngchoon Park, *Building an effective IoT ecosystem for your business.*: Springer, 2017.
- [17] [Online]. [https://academlib.com/26640/computer\\_science/security\\_threats](https://academlib.com/26640/computer_science/security_threats) [Last accessed on 27.8.2017].
- [18] James M. Stewart, Mike Chapple, and Darril Gibson, *CISSP: Certified information systems security professional study guide.*: John Wiley & Sons, 2012.
- [19] P. Thiyagarajan, Prasanth Kumar Thandra, J. Rajan, S.A.V. Satyamurthy, and G. Aghila, "Shamir Secret Sharing Scheme with Dynamic Access Structure (SSSDAS): Case study on nuclear power plant," *Kerntechnik*, pp. 150-160, May 2015.
- [20] Rachael Lininger and Russell Dean Vines, *Phishing: Cutting the identity theft*

- line.*: John Wiley & Sons, 2005.
- [21] Wenke Lee, Cliff Wang, and David Dagon, *Botnet detection: Countering the largest security threat.*: Springer Science & Business Media, 2007.
- [22] Karen Scarfone, *Technical guide to information security testing and assessment: Recommendations of the national institute of standards and technology.*: Diane Publishing, 2009.
- [23] Mauricio Arregoces and Maurizio Portolani, *Data center fundamentals.*: Cisco Press, 2003.
- [24] Charles P. Pfleeger and Shari Lawrence Pfleeger, *Security in computing.*: Prentice Hall Professional, 2003.
- [25] *Information security and privacy in network environments.*: Diane Publishing, 1994.
- [26] Kevin Beaver and Peter T. Davis, *Hacking wireless networks For dummies.*: John Wiley & Sons, 2011.
- [27] [Online]. <https://www.technologyreview.com/s/600715/nsa-says-it-must-act-now-against-the-quantum-computing-threat/> [Last accessed on 27.8.2017].
- [28] [Online]. <https://ercim-news.ercim.eu/en90/special/advances-in-hash-function-cryptanalysis> [Last accessed on 27.8.2017].
- [29] [Online]. <https://www.scmagazineuk.com/freak-ssl-flaw-affects-mobile-browsers-thousands-of-websites/article/537461/> [Last accessed on 27.8.2017].
- [30] [Online]. <https://www.csoononline.com/article/3040534/security/latest-attack-against-tls-shows-the-pitfalls-of-intentionally-weakening-encryption.html> [Last accessed on 27.8.2017].

- [31] [Online]. <https://www.helpnetsecurity.com/2017/09/27/atm-network-based-attacks/> [Last accessed on 27.8.2017].
- [32] [Online]. <https://www.helpnetsecurity.com/2015/02/16/carbanak-cyber-gang-stole-hundreds-of-millions-from-banks/> [Last accessed on 27.8.2017].
- [33] [Online]. <https://www.helpnetsecurity.com/2016/11/22/cobalt-hackers-synchronized-atm-heists/> [Last accessed on 27.8.2017].
- [34] [Online]. [https://documents.trendmicro.com/assets/white\\_papers/wp-cashing-in-on-atm-malware.pdf](https://documents.trendmicro.com/assets/white_papers/wp-cashing-in-on-atm-malware.pdf) [Last accessed on 27.8.2017].
- [35] Nighat Mir, "Zero watermarking for text on WWW using semantic approach," in *Second International conference on software engineering and computer systems*, 2011, pp. 306-316.
- [36] Esra Satir and Hakan Isik, "A compression-based text steganography method," *The journal of systems and software*, pp. 2385–2394, 2012.
- [37] Joachim Eggers and Bernd Girod, *Informed watermarking.*: Springer Science & Business Media, 2012.
- [38] Behrouz A Forouzan and Debdeep Mukhopadhyay, *Cryptography and network security.*: Tata McGraw-Hill Education, 2011.
- [39] Dale Liu, *Next generation SSH2 implementation: Securing data in motion.*: Syngress, 2011.
- [40] Stefan Katzenbeisser and Fabien A. P. Petitcolas, *Information hiding techniques for steganography and digital watermarking.*: Artech House, 2000.
- [41] Zhi-Hui Wang, Chin-Chen Chang, Chia-Chen Lin, and Ming-Chu Li, "A reversible information hiding scheme using left–right and up–down chinese character representation," *The journal of systems and software*, pp. 1362–1369,

2009.

- [42] H. Berghel and L. O'Gorman, "Protecting ownership rights through digital watermarking," *Computer*, vol. 29, no. 7, pp. 101-103, 1996.
- [43] Laura Millar, *Archives: Principles and practices.*: Facet Publishing, 2010.
- [44] Chin-Chen Chang and The Duc Kieu, "A reversible data hiding scheme using complementary embedding strategy," *Information sciences*, pp. 3045–3058, 2010.
- [45] [Online]. <http://searchsecurity.techtarget.com/video/How-to-use-OpenPuff-steganography-to-send-sensitive-info-securely> [Last accessed on 27.8.2017].
- [46] [Online]. <https://threatpost.com/prime-diffie-hellman-weakness-may-be-key-to-breaking-crypto/115069/> [Last accessed on 27.8.2017].
- [47] [Online]. <https://www.wired.com/2013/09/nsa-backdoor/> [Last accessed on 27.8.2017].
- [48] [Online]. <https://www.safaribooksonline.com/library/view/web-security-privacy/0596000456/ch04s04.html> [Last accessed on 27.8.2017].
- [49] [Online]. [https://motherboard.vice.com/en\\_us/article/3dabbw/NIST-quantum-computers-can-crack-its-encryption-RSA](https://motherboard.vice.com/en_us/article/3dabbw/NIST-quantum-computers-can-crack-its-encryption-RSA) [Last accessed on 27.8.2017].
- [50] [Online]. <https://futurism.com/1-evergreen-how-quantum-computers-would-destroy-todays-encryption-methods/> [Last accessed on 27.8.2017].
- [51] Pratiksha Sethi and V. Kapoor, "A proposed novel architecture for information hiding in image steganography by using genetic algorithm and cryptography," in *International conference on computational science*, vol. 87, 2016, pp. 61-66.
- [52] Shouchao Song, Jie Zhang, Xin Liao, Jiao Du, and Qiaoyan Wen, "A novel



- secure communication protocol combining steganography and cryptography," *Procedia engineering*, vol. 15, pp. 2767-2772, 2011.
- [53] Gregory Kipper, *Investigator's guide to steganography*.: CRC Press, 2003.
- [54] [Online]. <https://usatoday30.usatoday.com/life/cyber/tech/2001-02-05-binladen.htm#more> [Last accessed on 27.8.2017].
- [55] [Online]. <http://edition.cnn.com/2001/US/09/20/inv.terrorist.search/> [Last accessed on 27.8.2017].
- [56] [Online]. <https://www.oneindia.com/feature/steganography-and-terrorism-why-i-sis-relies-on-it-so-much-1670728.html> [Last accessed on 27.8.2017].
- [57] [Online]. <https://www.atlasobscura.com/articles/how-a-kitten-video-can-transmit-secret-instructions-to-criminals> [Last accessed on 27.8.2017].
- [58] [Online]. <https://www.scmagazine.com/new-variant-of-zeus-banking-trojan-concealed-in-jpg-images/article/538677/> [Last accessed on 27.8.2017].
- [59] [Online]. <http://www.hackmageddon.com/2011/10/21/stuxnet-duqu-stars-and-galaxies/> [Last accessed on 27.8.2017].
- [60] [Online]. <https://www.virusbulletin.com/virusbulletin/2016/04/how-it-works-steganography-hides-malware-image-files/> [Last accessed on 27.8.2017].
- [61] P. Thiyagarajan, G. Aghila, and V. Prasanna Venkatesan, "Pixastic: Steganography based anti-phishing browser plug-in," *Journal of internet banking and commerce*, vol. 17, no. 1, pp. 1-19, 2012.
- [62] Abdelrahman Desoky and Mohamed Younis, "Chestega: Chess steganography methodology," *Security and Communication Networks*, pp. 555-566, 2009.
- [63] Jianhong Sun, Yingjiang Li, Xiaohui Zhong, and Junsheng Li, "A scheme of

- LSB steganography based on concept of finding optimization pixels selection," in *Software engineering and knowledge engineering: Theory and practice*, 2012, pp. 155-160.
- [64] Neil F. Johnson, Zoran Duric, and Sushil Jajodia, *Information hiding: Steganography and watermarking-Attacks and countermeasures.*: Springer Science & Business Media, 2012.
- [65] Lekha S. Nair and Lakshmi M. Joshy, "An improved image steganography method with SPIHT and arithmetic coding," in *3rd International conference on frontiers of intelligent computing: Theory and applications*, vol. 2, 2014, pp. 97-104.
- [66] Lip Yee Por, KosSheik Wong, and Kok Onn Chee, "UniSpaCh: A text-based data hiding method using unicode space characters," *The journal of systems and software*, pp. 1075-1082, 2012.
- [67] Chi-Kwong Chan and L.M. Cheng, "Hiding data in images by simple LSB substitution," *Pattern recognition*, vol. 37, no. 3, pp. 469-474, March 2004.
- [68] David Wheeler, Daryl Johnson, Bo Yuan, and Peter Lutz, "Audio steganography using high frequency noise introduction," , 2012.
- [69] Ravneet Kaur and Tanupreet Singh, "Hiding data in video sequences using LSB with elliptic curve cryptography," *International journal of computer applications*, vol. 117, no. 18, pp. 36-40, 2015.
- [70] Yugeshwari Kakde, Priyanka Gonnade, and Prashant Dahiwal, "Audio-video steganography," in *International conference on innovations in information, embedded and communication systems*, 2015.
- [71] Sandip Bobade and Rajeshawari Goudar, "Secure data communication using

- protocol steganography in IPv6," in *International conference on computing communication control and automation*, 2015, pp. 275-279.
- [72] Yuling Liu, Xingming Sun, Yongping Liu, and Chang-Tsun Li, "MIMIC-PPT: Mimicking-based steganography for Microsoft Power Point document," *Information technology journal*, vol. 7, no. 4, pp. 654-660, 2008.
- [73] Information Resources Management Association, *Big Data: Concepts, methodologies, tools, and applications.*: IGI Global, 2016.
- [74] Aliya Tabassum Abbasi, Syeda N. S. Naqvi, Aihab Khan, and Basheer Ahmad, "Urdu text steganography: Utilizing isolated letters," in *13th Australian information security management conference*, 2015, pp. 37-46.
- [75] Sunita Chaudhary, Meenu Dave, and Amit Sanghi, "Aggrandize text security and hiding data through text steganography," in *International conference on power india*, 2016.
- [76] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding," *IBM Systems Journal*, vol. 35, no. 384, pp. 313-336, 1996.
- [77] Zhangjie Fu, Xingming Sun, Yuling Liu, and Bo Li, "Text split-based steganography in OOXML format documents for covert communication," *Security and Communication Networks*, 2011.
- [78] M Agarwal, "Text steganographic approaches: A comparison," *International journal of network security & its applications*, vol. 5, pp. 91-106, January 2013.
- [79] Geoffrey Samuelsson-Brown, *A practical guide for translators*, 5th ed.: Multilingual Matters, 2010.
- [80] Patrick Dunleavy, *Authoring a PhD: How to plan, draft, write and finish a Doctoral thesis or dissertation.*: Palgrave Macmillan, 2003.

- [81] Li and Chang-Tsun , *Crime prevention technologies and applications for advancing criminal investigation.*: IGI Global, 2012.
- [82] John Sinard, *Practical pathology informatics: Demystifying informatics for the practicing anatomic pathologist.*: Springer Science & Business Media, 2006.
- [83] Tsung-Yuan Liu and Wen-Hsiang Tsai, "A new steganographic method for data hiding in microsoft word documents by a change tracking technique," *IEEE transactions on information forensics and security*, vol. 2, pp. 24-30, March 2007.
- [84] Xingming Sun, Gang Luo, and Huajun Huang, "Component-based digital watermarking of Chinese texts," in *3rd International conference on information security*, 2004, pp. 76–81.
- [85] Mohammad Shirali-Shahreza, "Text steganography by changing words spelling," in *10th International conference on advanced communication technology*, 2008.
- [86] A. Majumder and S. Changder, "A novel approach for text steganography: generating text summary using reflection symmetry," in *International conference on computational intelligence: Modeling techniques and application*, 2013, pp. 112-120.
- [87] Jack T. Brassil, Steven Low, and Nicholas maxemchuk F., "Copyright protection for the electronic distribution of text documents," *Proceedings of the IEEE*, vol. 87, no. 7, pp. 1181-1196, July 1999.
- [88] Sabu M. Thampi, "Information hiding techniques: A tutorial review," in *ISTE-STTP on Network Security & Cryptography, LBSCE*, 2004.
- [89] Fabien A. P. Petitcolas, Ross J. Anderson, and Markus G. Kuhn, "Information

- hiding—A survey," *Proceedings of the IEEE*, pp. 1062–1078, 1999.
- [90] Krista Bennett, "Linguistic steganography: Survey, analysis, and robustness concerns for hiding information in text," Purdue University, Technical 2004-13.
- [91] Mercan Topkara, Umut Topkara, and Mikhail J. Atallah, "Information hiding through errors: A confusing approach," in *SPIE International conference on security, steganography and watermarking of multimedia contents IX*, vol. 6505, 2007.
- [92] Jason Cranford Teague, *Fluid web typography*.: New Riders, 2009.
- [93] David Kahn, *The Code-Breakers: The comprehensive history of secret communication from ancient times to the internet*, 2nd ed.: Scribner, 1996.
- [94] Jibran Ahmed Memon, Kamran Khowaja, and Hameedullah Kazi, "Evaluation of steganography for Urdu/Arabic text," *Journal of theoretical and applied information technology*, pp. 232-237, 2005.
- [95] Cao Qi, Sun Xingming, and Xiang Lingyun, "A secure text steganography based on synonym substitution," in *13th IEEE joint International computer science and information technology conference*, 2013.
- [96] Russell Ogilvie and George R. S. Weir, "Genre-based information hiding," in *Global security, safety and sustainability*, vol. 99, 2012, pp. 104-111.
- [97] [Online]. <http://www.tysto.com/uk-us-spelling-list.html> [Last accessed on 27.8.2017].
- [98] Rajesh Kumar Tiwari and G. Sahoo, "Microsoft excel file: A steganographic carrier file," *International journal of digital crime and forensics*, vol. 3, no. 1, pp. 37-52, 2011.
- [99] Jack T. Brassil, Steven Low, and Nicholas F. Maxemchuk, "Electronic marking

- and identification techniques to discourage document copying," *IEEE journal on selected areas in communications*, vol. 13, no. 8, pp. 1495-1504, 1995.
- [100] Prem Singh, Rajat Chaudhary, and Ambika Agarwal, "A novel approach of text steganography based on null spaces," *IOSR journal of computer engineering*, vol. 3, no. 4, pp. 11-17, 2014.
- [101] L. Y. Por, T. F. Ang, and B. Delina, "WhiteSteg: A new scheme in information hiding using text steganography," *WSEAS transactions on computers*, vol. 7, no. 6, pp. 735-745, 2008.
- [102] Martin Cutts, *Oxford guide to plain English.*: Oxford University Press, 2013.
- [103] Janice R. Matthews, John M. Bowen, and Robert W. Matthews, *Successful scientific writing full Canadian binding: A step-by-step guide for the biological and medical sciences.*: Cambridge University Press, 2000.
- [104] L. Y. Por and B. Delina, "Information hiding: A new approach in text steganography," in *7th WSEAS International conference on applied computer & applied computational science*, 2008, pp. 689-695.
- [105] M. Hassan Shirali-Shahreza and Mohammad Shirali-Shahreza, "A new approach to Persian/Arabic text steganography," in *5th IEEE/ACIS International conference on computer and information science*, 2006.
- [106] Mohammed A. Aabed, Sameh M. Awaideh, and Abdul-Rahman M. Elshafei, "Arabic diacritics based steganography," in *International conference on signal processing and communications*, 2007.
- [107] Reem Alotaibi Ahmed and Lamiaa A. Elrefaei, "Arabic text watermarking: A review," *International journal of artificial intelligence & applications*, vol. 6, no. 4, pp. 1-16, 2015.

- [108] V. V. Muniswamy, *Design And analysis Of algorithms.*: I. K. International Pvt Ltd, 2009.
- [109] John R. Pierce, *An introduction to information theory: Symbols, signals and noise*, 2nd ed.: Dover Publications, 1980.
- [110] Isabelle de Ridder, *Reading from the screen in a second language: Empirical studies on the effect of marked hyperlinks on incidental vocabulary learning, text comprehension and the reading process.*: Garant, 2003.
- [111] David W. Beskeen, Carol Cram, Jennifer Duffy, Lisa Friedrichsen, and Elizabeth Eisner Reding, *Microsoft Office 2007-Illustrated introductory, windows XP edition.*: Cengage Learning, 2007.
- [112] ITL , Kamthane , and Rajkamal , *Computer programming and IT: For RTU.*: Pearson Education India, 2011.
- [113] Arne Mikalsen and Per Borgesen, *Local area networks: Includes data comm. network.*: John Wiley & Sons, 2002.
- [114] John C. Dean and Li Li, "Issues in developing security wrapper technology for COTS software products," in *International conference on COTS-Based software systems*, 2002, pp. 76-85.
- [115] Jesse Russell and Ronald Cohn, *History of Microsoft Word.*: Book on Demand, 2012.
- [116] LibreOffice Documentation Team, *Getting started with LibreOffice 5.0.*: Lulu.com, 2016.
- [117] Jean Hollis Weber, *Taming Apache OpenOffice: Getting started.*: Lulu.com, 2013.
- [118] Alegis Leon and Mathews Leon, *Introduction to computers.*: Vikas Publishing

- House Pvt. Ltd., 1999.
- [119] "The independent guide to personal computing," *PC Magazine*, vol. 15, no. 20, p. 410, 1996.
- [120] Roger Hersch, *Visual and technical aspects of type.*: Cambridge University Press, 1993.
- [121] [Online]. <http://mediamilitia.com/taking-type-to-the-next-level-with-alternate-characters/> [Last accessed on 27.8.2017].
- [122] [Online]. <http://www.will-harris.com/ligatures.htm> [Last accessed on 27.8.2017].
- [123] [Online]. [https://msdn.microsoft.com/en-us/library/microsoft.office.interop.word.\\_font.ligatures\(v=office.14\).aspx](https://msdn.microsoft.com/en-us/library/microsoft.office.interop.word._font.ligatures(v=office.14).aspx) [Last accessed on 27.8.2017].
- [124] [Online]. <http://www.magpiepaperworks.com/blog/using-opentype-fonts-in-microsoft-word/> [Last accessed on 27.8.2017].
- [125] [Online]. [https://msdn.microsoft.com/en-us/library/microsoft.office.interop.word.\\_font.numberform\(v=office.14\).aspx](https://msdn.microsoft.com/en-us/library/microsoft.office.interop.word._font.numberform(v=office.14).aspx) [Last accessed on 27.8.2017].
- [126] [Online]. [https://msdn.microsoft.com/en-us/library/microsoft.office.interop.word.\\_font.numberspacing\(v=office.14\).aspx](https://msdn.microsoft.com/en-us/library/microsoft.office.interop.word._font.numberspacing(v=office.14).aspx) [Last accessed on 27.8.2017].
- [127] Alex Fowkes, *Drawing type: An introduction to illustrating letterforms.*: Rockport Publishers, 2014.
- [128] Tova Rabinowitz, *Exploring typography*, 2nd ed.: Cengage Learning, 2015.
- [129] Muhammed N Kabir, Omar Tayan, and Yasser M Alginahi, "Evaluation of watermarking approaches for Arabic text documents," *International journal of computer science and information security*, pp. 49-54, 2013.





- [139] [Online]. <http://www.world-nuclear.org/gallery/reactor-diagrams/boiling-water-reactor.aspx> [Last accessed on 27.8.2017].
- [140] [Online]. <https://sosteneslekule.blogspot.in/2015/03/thermal-power-generation-plant-or.html> [Last accessed on 27.8.2017].
- [141] Bahman Zohuri, Jürgen K. Grunwald, and Takayuki Nakamura, *Nuclear energy: Perspectives, challenges and future directions*, Denver Morris, Ed.: Nova Science Publishers, 2017.
- [142] V. M. Mente et al., "Experimental studies in water for safety grade decay heat removal of prototype fast breeder reactor," *Annals of nuclear energy*, vol. 65, pp. 114-121, 2014.
- [143] [Online]. <http://www.dogdrip.net/84405179> [Last accessed on 27.8.2017].
- [144] Frederic P. Miller, Agnes F. Vandome, and John McBrewster, *Hamming code*: Alphascript Publishing, 2009.
- [145] Matt Doyle, *Beginning PHP 5.3*: John Wiley & Sons, 2011.
- [146] Ray-I Chang, Yachik Yen, and Ting-Yu Hsu, "An XML-based comic image compression," in *9th Pacific Rim conference on multimedia*, 2008, pp. 563-572.
- [147] V. Solachidis, N. Nikolaidis, and I. Pitas, "Fourier descriptors watermarking of vector graphics images," in *International conference on image processing*, 2000, pp. 9-12.
- [148] [Online]. <https://www.vecteezy.com/blog/2015/5/24/the-history-of-adobe-illustrator> [Last accessed on 27.8.2017].
- [149] "A short history of CorelDRAW," 2009.
- [150] [Online]. <https://inkscape.org/en/about/> [Last accessed on 27.8.2017].

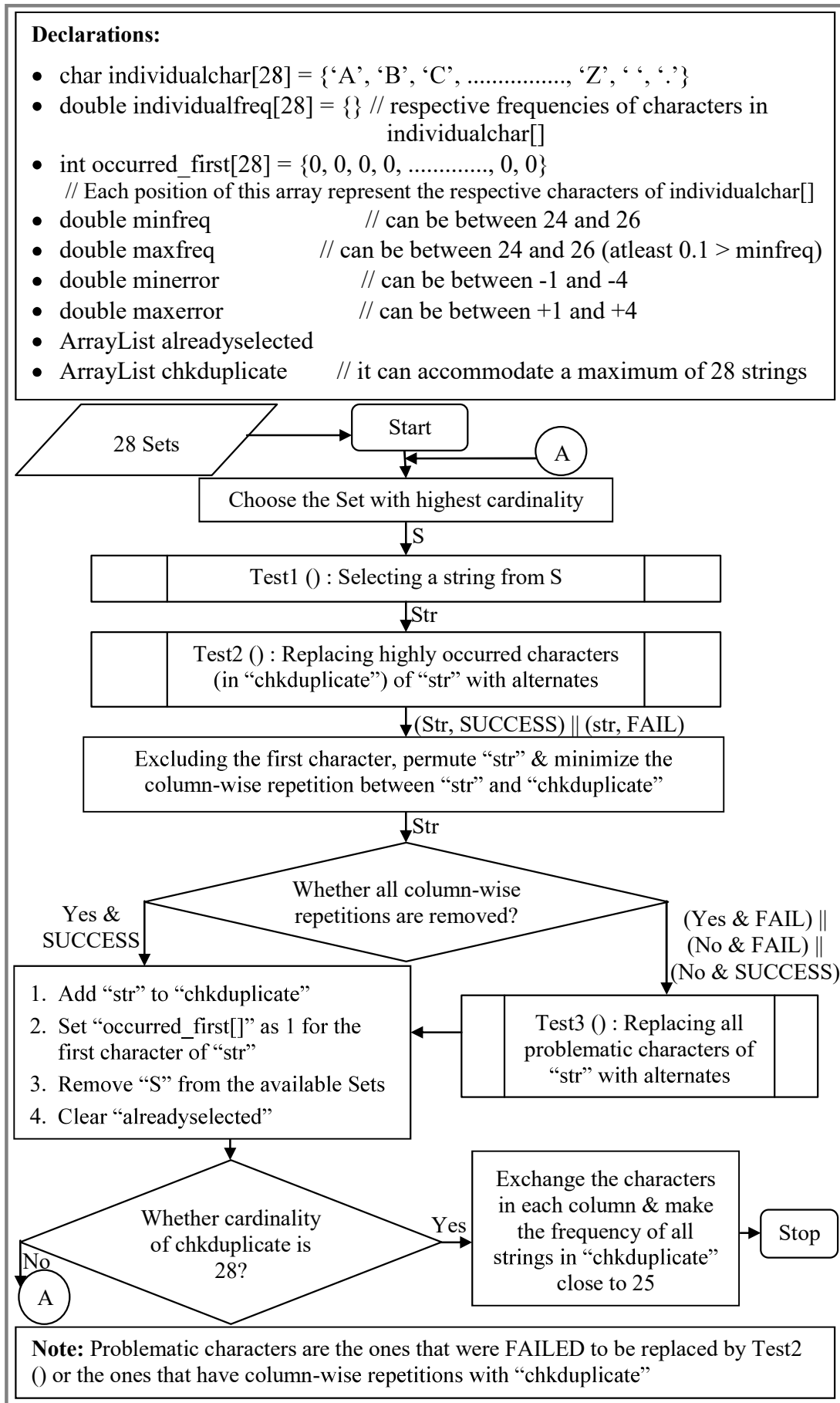
- [151] Daniel Schwarz, *Jump start Sketch: Master the tool made for UI designers.*: SitePoint, 2016.
- [152] [Online]. <https://www.khanacademy.org/math/trigonometry/trigonometry-right-triangles/trig-solve-for-a-side/a/unknown-side-in-right-triangle-w-trig> [Last accessed on 27.8.2017].
- [153] Patricia Law Hatcher, *Producing a quality family history.*: Ancestry Publishing, 1996.

## APPENDIX — A

### *Procedure to generate Frequency Normalization Set (FNS)*

Method-A described in Chapter 4, allows single secret character to get embedded in multiple cover characters. This facilitated to boost the embedding probability of secret characters and also made it uniform. It achieved the same through the construction of a Frequency Normalization Set (FNS) — refer Section 4.4 in page 76. The flowchart to generate such a set is provided below, in Fig. A.1, along with the necessary pseudo codes.

The procedure takes the occurrence frequencies of ADS characters, minimum and maximum allowed error (deviation from the value 25), etc., as inputs and generate a FNS.



**Figure A.1** Flowchart to generate Frequency Normalization Set (FNS)

**Pseudo code of Test 1:**

//Test 1 (): Chooses a string “str” from a Set of available strings.

---

**Input:** Set S, ArrayList chkduplicate

---

**Output:** string str

---

Int sim\_count2  $\leftarrow$  100

Int value2  $\leftarrow$  100

String str  $\leftarrow$  ""

For each string “str1” in “S” do

    Int sim\_count[cardinality of chkduplicate]  $\leftarrow$  No. of similar characters between  
    “str1” and each String in “chkduplicate” respectively

    Int max\_count  $\leftarrow$  Largest value in sim\_count[]

    If max\_count < sim\_count2

        Int value[7]  $\leftarrow$  No. of times each character of “str1” appeared in  
        “chkduplicate” respectively

        Int max\_value  $\leftarrow$  Largest value in value[]

        If max\_value < value2

            str  $\leftarrow$  str1

            sim\_count2  $\leftarrow$  max\_count

            value2  $\leftarrow$  max\_value

        End if

    End if

End for

Return str

---

**Pseudo code of sub-module: Count**

---

**Input:** char ch, ArrayList chkduplicate

---

**Output:** int no

---

Return Number of times “ch” has occurred in “chkduplicate”

---

### Pseudo code of Test 2:

//Test2 (): Tries to replace the characters of “str”, when its number of occurrences in “chkduplicate” crosses certain limit.

---

**Input:** string str, int occurred\_first[], char individualchar[], double individualfreq[], double minerror, double maxerror, ArrayList chkduplicate

**Output:** string str

---

String result ← “SUCCESS”

For each character “ch” in “str” do

    Int pos ← Position of “ch” in “str”

    If (Count(ch)==7) || (Count(ch)==6 && occurred\_first of “ch”==0 && str[0] != ch)

        Search for a character “ch1” in “individualchar[]” such that ch1 ∉ str && ch1 ∉ chkduplicate @ position “pos” && Frequency[ch1]-Frequency[ch] is between “minerror” and “maxerror” exclusive

        If Search == SUCCESSFUL then str[pos] ← ch1

        Else result ← “FAIL”

    End if

End for

Return (str, result)

---

### Pseudo code of Test 3:

//Test3 (): Handles the problematic characters of “str”.

---

**Input:** string str, ArrayList chkduplicate, ArrayList alreadyselected

**Output:** string str

---

For each problematic character “ch1” in “str” do

    Int pos ← Position of “ch1” in “str”

    L: Char ch ← Test\_3a ()

    If ch ∉ str then str[pos] ← ch

    Else

        String str1 ← Test\_3b ()

        If str1 != “”

            str[pos] ← str1[pos]

            chkduplicate ← Remove “str1” from “chkduplicate”

            str1[pos] ← ch

            chkduplicate ← Add “str1” in “chkduplicate”

        End if

---

```

        Else Add “ch” in “alreadyselected” & Goto L
    End else
End for
Return str

```

---

### Pseudo code of Test \_3a:

//Chooses a character from “individualchar” that satisfies certain properties

---

**Input:** char individualchar[], double individualfreq[], ArrayList chkduplicate, int pos, char ch1, ArrayList alreadyselected, string str

**Output:** char ch

---

Char ch ← ‘ ’

Int dupe ← 0

Int max ← 0

**L:** for each character “ch2” in “individualchar[]” do

    If dupe==0 && ch2 ∉ alreadyselected && Count(ch2) < 7 && ch2 ∉  
chkduplicate @ position “pos” && Frequency[ch2] ≤ Frequency[ch1] &&  
Frequency[ch2] > max

        ch ← ch2

        max ← Frequency[ch2]

    End if

    If ch==’ ’ && Last element of “individualchar[]” == ch2

        dupe ← 1

        Goto L

    End if

    If dupe==1 && (ch2 ∉ alreadyselected && Count(ch2) < 7 && ch2 ∉  
chkduplicate @ position “pos”) && ((Frequency[ch2] > Frequency[ch1]) || (ch2 ∈ str  
&& Count(ch2) ≤ 5))

        ch ← ch2

        Break

    End if

End for

Return ch

---



**Pseudo code of Test\_3b:**

//Chooses a string from “chkduplicate” that satisfies certain properties

---

**Input:** ArrayList chkduplicate, double individualfreq[], char ch1, int pos, char ch, string str

**Output:** string str2

---

String str2  $\leftarrow$  “”

Int dupe  $\leftarrow$  0

Int min  $\leftarrow$  100

L: For each string “str1” in “chkduplicate” do

    If dupe==0 && ch1  $\in$  str1 @ position “pos” && ch  $\notin$  str1

        str2  $\leftarrow$  str1

    End if

    If str2==”” && Last element of “chkduplicate[]” == str1

        dupe  $\leftarrow$  1

        Goto L

    End if

    If dupe == 1 && ch  $\notin$  str1 && str1[pos]  $\notin$  str && Frequency[str1[pos]]-Frequency[ch] < min && < 4.5

        str2  $\leftarrow$  str1

        min  $\leftarrow$  Frequency[str1[pos]]-Frequency[ch]

    End if

End for

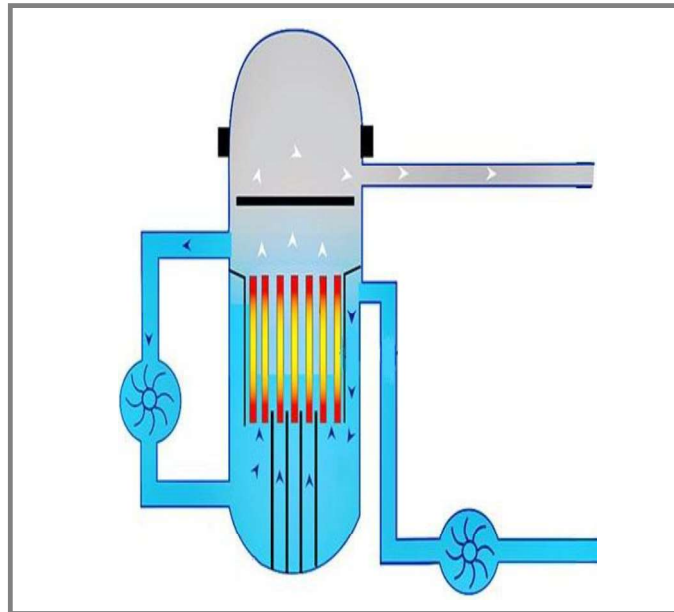
Return str2

---

## APPENDIX — B

### *Illustration of the imperceptibility level of Method-C*

Method-C described in Chapter 6, converts an image into a custom format and then embeds the same using the attributes Color, Kerning and Spacing. The imperceptibility level of the method was illustrated in Fig. 6.12 of Chapter 6 (see page 147). To verify the same, an additional illustration is provided below using Fig. B.1.



**Figure B.1** Boiling water reactor [139]

*Cover work (after formatting):*

Internet which is extensively used to share any kind of information does not imply any strict rules for the security of data on its own and also there are ways for third parties to do eavesdropping in the network. Hence during transmission, security to information plays a major role for Sensitive Organizations where the leakage of small amount of information will lead to critical problem. Information Security System is broadly classified into three types as Cryptography, WaterMarking and Steganography. Each one of the above mentioned classification aims at achieving specific task. Cryptography aims at preventing any third party from reading the content of the message, WaterMarking aims at providing Copyright Protection and Steganography aims at concealing the secret message. Even though their aim is to solve different issues, they can be combined together with the aim to add additional layer of security. It is believed that when they are used together, the weakness of one method could be overcome by the other. Details about each classification are explained in the subsequent paragraphs. Cryptography: The word Cryptography is derived from the Greek word Crypt meaning Secret and graphi meaning Writing mentioning it as Secret Writing. The main aim of Cryptography is to prevent any third party from reading the actual content of the secret message. Cryptography supports secrecy through Encryption and Decryption techniques. Plain Text is given as input to the Encryption algorithm which uses a key to encrypt and produces the Cipher Text. At the decryption side, a key is used to recover the Plain Text back from the Cipher Text. Based on the number of keys used, Cryptography is divided into two categories as Symmetric Key and Asymmetric Key Cryptography. Symmetric Key Cryptography uses a single key for both encryption and decryption of the secret message. Hence the key must be shared in a secure manner and both the sender and the receiver must keep the key safely. Asymmetric Key Cryptography highly relies on Number Theory Concepts for its security. Asymmetric Key Cryptography uses two different keys one derived from the other. Often they are called together as Key Pairs of which one is announced as the Public Key and the other is kept as the Private Key. One will be used for encryption and the other will be used for decryption. WaterMarking: There are two types of WaterMarking namely Visible and Invisible WaterMarking. Invisible WaterMarking and Steganography are both closely related to one another. Invisible WaterMarking is used to share a message secretly by hiding it in the cover work whereas Visible WaterMarking aims at preserving the ownership of the document by tagging the ownership details in the cover work. Cover Work is nothing but an innocent digital media in which the message is embedded either perceptibly or imperceptibly. Normally used cover works are Text, Audio, Video and Image. Two major issues which must be concerned while sharing a digital media online are Copyright Protection and Content Modification. Copyright protection tries to keep track of the legal owner of the document by either preventing from making copies from the original document or by making the ownership details to be embedded in all the duplicate copies. Content modification prevents user from downloading the document online, makes some minor modification to it and upload it as a different user in another website. Modification of document includes removing the tagged text from document, or spoiling the document and making the tag unrecognizable. Some of the attacks on WaterMarked document include Cropping, Transformation, etc. As far as today's technology Visible WaterMarking is used as a technique to overcome these two major issues. Internet which is extensively used to share any kind of information does not imply any strict rules for the security of data on its own and also there are ways for third parties to do eavesdropping in the network. Hence during transmission, security to information plays a major role for Sensitive Organizations where the leakage of small amount of information will lead to critical problem. Information Security System is broadly classified into three types as Cryptography, WaterMarking and Steganography. Each one of the above mentioned classification aims at achieving specific task. Cryptography aims at preventing any third party from reading the content of the message, WaterMarking aims at providing Copyright Protection and Steganography aims at concealing the secret message. Even though their aim is to solve different issues, they can be combined together with the aim to add additional layer of security. It is believed that when they are used together, the weakness of one method could be overcome by the other. Details about each classification are explained in the subsequent paragraphs. Cryptography: The word Cryptography is derived from the Greek word Crypt meaning Secret and graphi meaning Writing mentioning it as Secret Writing. The main aim of Cryptography is to prevent any third party from reading the actual content of the secret message. Cryptography supports secrecy through Encryption and Decryption techniques. Plain Text is given as input to the Encryption algorithm which uses a key to encrypt and produces the Cipher Text. At the decryption side, a key is used to recover the Plain Text back from the Cipher Text. Based on the number of keys used, Cryptography is divided into two categories as Symmetric Key and Asymmetric Key Cryptography. Symmetric Key Cryptography uses a single key for both encryption and decryption of the secret message. Hence the key must be shared in a secure manner and both the sender and the receiver must keep the key safely. Asymmetric Key Cryptography highly relies on Number Theory Concepts for its security. Asymmetric Key Cryptography uses two different keys one derived from the other. Often they are called together as Key Pairs of which one is announced as the Public Key and the other is kept as the Private Key. One will be used for encryption and the other will be used for decryption. WaterMarking: There are two types of WaterMarking namely Visible and Invisible WaterMarking. Invisible WaterMarking and Steganography are both closely related to one another. Invisible WaterMarking is used to share a message secretly by hiding it in the cover work whereas Visible WaterMarking aims at preserving the ownership of the document by tagging the ownership details in the cover work. Cover Work is nothing but an innocent digital media in which the message is embedded either perceptibly or imperceptibly. Normally used cover works are Text, Audio, Video and Image. Two major issues which must be



*Stego work (with formatting):*

Internet which is extensively used to share any kind of information does not imply any strict rules for the security of data on its own and also there are ways for third parties to do eavesdropping in the network. Hence during transmission, security to information plays a major role for Sensitive Organizations where the leakage of small amount of information will lead to critical problem. Information Security System is broadly classified into three types as Cryptography, Water Marking and Steganography. Each one of the above mentioned classification aims at achieving specific task. Cryptography aims at preventing any third party from reading the content of the message, Water Marking aims at providing Copyright Protection and Steganography aims at concealing the secret message. Even though their aim is to solve different issues, they can be combined together with the aim to add additional layer of security. It is believed that when they are used together, the weakness of one method could be overcome by the other. Details about each classification are explained in the subsequent paragraphs.

**Cryptography:** The word Cryptography is derived from the Greek word Crypt meaning Secret and graphi meaning Writing mentioning it as Secret Writing. The main aim of Cryptography is to prevent any third party from reading the actual content of the secret message. Cryptography supports secrecy through Encryption and Decryption techniques. Plain Text is given as input to the Encryption algorithm which uses a key to encrypt and produces the Cipher Text. At the decryption side, a key is used to recover the Plain Text back from the Cipher Text. Based on the number of keys used, Cryptography is divided into two categories as Symmetric Key and Asymmetric Key Cryptography. Symmetric Key Cryptography uses a single key for both encryption and decryption of the secret message. Hence the key must be shared in a secure manner and both the sender and the receiver must keep the key safely. Asymmetric Key Cryptography highly relies on Number Theory Concepts for its security. Asymmetric Key Cryptography uses two different keys one derived from the other. Often they are called together as Key Pairs of which one is announced as the Public Key and the other is kept as the Private Key. One will be used for encryption and the other will be used for decryption.

**Water Marking:** There are two types of Water Marking namely Visible and Invisible Water Marking. Invisible Water Marking and Steganography are both closely related to one another. Invisible Water Marking is used to share messages secretly by hiding it in the cover work whereas Visible Water Marking aims at preserving the ownership of the document by tagging the ownership details in the cover work. Cover Work is nothing but an innocent digital media in which the message is embedded either perceptibly or imperceptibly. Normally used cover works are Text, Audio, Video and Image. Two major issues which must be concerned while sharing a digital media online are Copy Protection and Content Modification. Copy protection tries to keep track of the legal owner of the document by either preventing from making copies from the original document or by making the ownership details to be embedded in all the duplicate copies. Content modification prevents user from downloading the document online, makes some minor modification to it and upload it as a different user in another website. Modification of document includes removing the tagged text from document, or spoiling the document and making the tag unrecognizable. Some of the attacks on Water Marked document include Cropping, Transformation, etc. As far as today's technology Visible Water Marking is used as a technique to overcome these two major issues. Internet which is extensively used to share any kind of information does not imply any strict rules for the security of data on its own and also there are ways for third parties to do eavesdropping in the network. Hence during transmission, security to information plays a major role for Sensitive Organizations where the leakage of small amount of information will lead to critical problem. Information Security System is broadly classified into three types as Cryptography, Water Marking and Steganography. Each one of the above mentioned classification aims at achieving specific task. Cryptography aims at preventing any third party from reading the content of the message, Water Marking aims at providing Copyright Protection and Steganography aims at concealing the secret message. Even though their aim is to solve different issues, they can be combined together with the aim to add additional layer of security. It is believed that when they are used together, the weakness of one method could be overcome by the other. Details about each classification are explained in the subsequent paragraphs.

**Cryptogr**aphy: The word Cryptography is derived from the Greek word Crypt meaning Secret and graphi meaning Writing mentioning it as Secret Writing. The main aim of Cryptography is to prevent any third party from reading the actual content of the secret message. Cryptogr<sup>aphy</sup>s supports secrecy through Encryption and Decryption techniques. Plain Text is given as input to the Encryption algorithm which uses a key to encrypt and produces the Cipher Text. At the decryption side, a key is used to recover the Plain Text back from the Cipher Text. Based on the number of keys used, Cryptography is divided into two categories as Symmetric Key and Asymmetric Key Cryptography. Symmetric Key Cryptography uses a single key for both encryption and decryption of the secret message. Hence the key must be shared in a secure manner and both the sender and the receiver must keep the keys safely. Asymmetric Key Cryptography highly relies on Number Theory Concepts for its security. Asymmetric Key Cryptography uses two different keys one derived from the other. Often they are called together as Key Pairs of which one is announced as the Public Key and the other is kept as the Private Key. One will be used for encryption and the other will be used for decryption.

**Water Marking:** There are two types of Water Marking namely Visible and Invisible Water Marking. Invisible Water Marking and Steganography are both closely related to one another. Invisible Water Marking is used to share messages secretly by hiding it in the cover work whereas Visible Water Marking aims at preserving the ownership of the document by tagging the ownership details in the cover work. Cover Work is nothing but an innocent digital media in which the message is embedded either perceptibly or imperceptibly. Normally used cover works are Text, Audio, Video and Image. Two major issues which must be



*Stego work with formatting (stego characters highlighted for understanding purpose):*

Internet which is extensively used to share any kind of information does not imply any strict rules for the security of data as it is shown and also there are ways for third parties to do eavesdropping in the network. Hence during transmission, security of information plays a major role for Sensitive Organizations where the leakage of small amount of information will lead to critical problem. Information Security System is broadly classified into three types as Cryptography, Water Marking and Steganography. Each one of the above mentioned classification aims at achieving specific task. Cryptography aims at preventing any third party from reading the content of the message, Water Marking aims at providing Copyright Protection and Steganography aims at concealing the secret message. Even though their aim is to solve different issues, they can be combined together with the aim to add additional layer of security. It is believed that when they are used together, the weakness of one method could be overcome by the other. Details about each classification are explained in the subsequent paragraphs.

**Cryptography:** The word Cryptography is derived from the Greek word Crypto meaning Secret and graphi meaning Writing mentioning it as Secret Writing. The main aim of Cryptography is to prevent any third party from reading the actual content of the secret message. Cryptography supports secrecy through Encryption and Decryption techniques. Plain Text is given as input to the Encryption algorithm which uses a key to encrypt and produces the Cipher Text. At the decryption side, a key is used to recover the Plain Text back from the Cipher Text. Based on the number of keys used, Cryptography is divided into two categories as Symmetric Key and Asymmetric Key Cryptography. Symmetric Key Cryptography uses a single key for both encryption and decryption of the secret message. Hence the key must be shared in a secure manner and both the sender and the receiver must keep the key safely. Asymmetric Key Cryptography highly relies on Number Theory Concepts for its security. Asymmetric Key Cryptography uses two different keys one derived from the other. Often they are called together as Key Pairs of which one is announced as the Public Key and the other is kept as the Private Key. One will be used for encryption and the other will be used for decryption. **Water Marking:** There are two types of Water Marking namely Visible and Invisible Water Marking. Invisible Water Marking and Steganography are both closely related to one another. Invisible Water Marking is used to share a message secretly by hiding it in the cover work whereas Visible Water Marking aims at preserving the ownership of the document by tagging the ownership details in the cover work. Cover Work is nothing but an innocent digital media in which the message is embedded either perceptibly or imperceptibly. Normally used cover works are Text, Audio, Video and Image. Two major issues which must be concerned while sharing a digital media online are Copy Protection and Content Modification. Copy protection tries to keep track of the legal owner of the document by either preventing from making copies from the original document or by making the ownership details to be embedded in all the duplicate copies. Content modification prevents user from downloading the document online, makes some minor modification to it and upload it as a different user in another website. Modification of document includes removing the tagged text from document, or spoiling the document and making the tag unrecoverable. Some of the attacks on Water Marked document include Cropping, Transformation, etc. As far as today's technology Visible Water Marking is used as a technique to overcome these two major issues. Internet which is extensively used to share any kind of information does not imply any strict rules for the security of data as it is shown and also there are ways for third parties to do eavesdropping in the network. Hence during transmission, security of information plays a major role for Sensitive Organizations where the leakage of small amount of information will lead to critical problem. Information Security System is broadly classified into three types as Cryptography, Water Marking and Steganography. Each one of the above mentioned classification aims at achieving specific task. Cryptography aims at preventing any third party from reading the content of the message, Water Marking aims at providing Copyright Protection and Steganography aims at concealing the secret message. Even though their aim is to solve different issues, they can be combined together with the aim to add additional layer of security. It is believed that when they are used together, the weakness of one method could be overcome by the other. Details about each classification are explained in the subsequent paragraphs.

**Cryptography:** The word Cryptography is derived from the Greek word Crypto meaning Secret and graphi meaning Writing mentioning it as Secret Writing. The main aim of Cryptography is to prevent any third party from reading the actual content of the secret message. Cryptography supports secrecy through Encryption and Decryption techniques. Plain Text is given as input to the Encryption algorithm which uses a key to encrypt and produces the Cipher Text. At the decryption side, a key is used to recover the Plain Text back from the Cipher Text. Based on the number of keys used, Cryptography is divided into two categories as Symmetric Key and Asymmetric Key Cryptography. Symmetric Key Cryptography uses a single key for both encryption and decryption of the secret message. Hence the key must be shared in a secure manner and both the sender and the receiver must keep the key safely. Asymmetric Key Cryptography highly relies on Number Theory Concepts for its security. Asymmetric Key Cryptography uses two different keys one derived from the other. Often they are called together as Key Pairs of which one is announced as the Public Key and the other is kept as the Private Key. One will be used for encryption and the other will be used for decryption. **Water Marking:** There are two types of Water Marking namely Visible and Invisible Water Marking. Invisible Water Marking and Steganography are both closely related to one another. Invisible Water Marking is used to share a message secretly by hiding it in the cover work whereas Visible Water Marking aims at preserving the ownership of the document by tagging the ownership details in the cover work. Cover Work is nothing but an innocent digital media in which the message is embedded either perceptibly or imperceptibly. Normally used cover works are Text, Audio, Video and Image. Two major issues which must be concerned while sharing a digital media online are Copy Protection and Content Modification. Copy protection tries to keep track of the legal owner of the document by either preventing from making copies from the original document or by making the ownership details to be embedded in all the duplicate copies. Content modification prevents user from downloading the document online, makes some minor modification to it and upload it as a different user in another website. Modification of document includes removing the tagged text from document, or spoiling the document and making the tag unrecoverable. Some of the attacks on Water Marked document include Cropping, Transformation, etc. As far as today's technology Visible Water Marking is used as a technique to overcome these two major issues.