資訊學院

資訊科學與工程研究所

博 士 論 文

# 在影像與文字檔案中進行資料隱藏的技術與應用之研究

# A Study on New Techniques of Data Hiding in Images and Text Documents and Their Applications

研 究 生: 李義溪

指 導 教 授: 蔡 文 祥 博士

中華民國九十七年七月

# 在影像與文字檔案中進行資料隱藏的技術與應用之研究

# A Study on New Techniques of Data Hiding in Images and Text Documents and Their Applications

研　究　生：李 義 溪　　Student: I-Shi Lee

指 導 教 授：蔡 文 祥 博士　　Advisor: Dr. Wen-Hsiang Tsai

國 立 交 通 大 學 資 訊 學 院
資 訊 科 學 與 工 程 研 究 所
博 士 論 文

A Dissertation Submitted to
Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy
in Computer and Information Science

July 2008
Hsinchu, Taiwan, 300
Republic of China

中華民國 九十七 年 七 月

# 國　立　交　通　大　學

## 博碩士論文全文電子檔著作權授權書

本授權書所授權之學位論文，為本人於國立交通大學資訊學院資訊科學與工程研究所，九十六學年度第二學期取得博士學位之論文。
論文題目：在影像與文字檔案中進行資料隱藏的技術與應用之研究
指導教授：蔡文祥 教授

■ 同意　□不同意

本人茲將本著作，以非專屬、無償授權國立交通大學與台灣聯合大學系統圖書館：基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會與學術研究之目的，國立交通大學及台灣聯合大學系統圖書館得不限地域、時間與次數，以紙本、光碟或數位化等各種方法收錄、重製與利用；於著作權法合理使用範圍內，讀者得進行線上檢索、閱覽、下載或列印。

| 論文全文上載網路公開之範圍及時間： | |
|---|---|
| 本校及台灣聯合大學系統區域網路 | ■ 立即公開 |
| 校外網際網路 | ■ 立即公開 |

**授 權 人：李義溪**

**親筆簽名：** 李義溪

**中華民國 97 年 7 月 28 日**

# 國　立　交　通　大　學

## 博碩士紙本論文著作權授權書

本授權書所授權之學位論文，為本人於國立交通大學資訊學院資訊科學與工程研究所，九十六學年度第二學期取得博士學位之論文。
論文題目：在影像與文字檔案中進行資料隱藏的技術與應用之研究
指導教授：蔡文祥 教授

■ 同意

本人茲將本著作，以非專屬、無償授權國立交通大學，基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會與學術研究之目的，國立交通大學圖書館得以紙本收錄、重製與利用；於著作權法合理使用範圍內，讀者得進行閱覽或列印。

本論文為本人向經濟部智慧局申請專利(未申請者本條款請不予理會)的附件之一，申請文號為：________________，請將論文延至____年____月____日再公開。

**授 權 人：李義溪**

**親筆簽名：**李 義 溪

**中華民國 97 年 7 月 28 日**

# 國立交通大學
## 資訊工程系博士班

### 論文口試委員會審定書

本校 資　訊　工　程　系　　李　義　溪 君

所提論文 在影像與文字檔案中進行資料隱藏的技術與應用之研究

合於博士資格水準、業經本委員會評審認可。

口試委員：蔡文祥

莊仁輝　　[illegible]

[illegible]　　范國清

陳永昇　　吳大鈞

指導教授：蔡文祥

系 主 任：[illegible]

中華民國 九十七 年 七 月 廿九 日

Department of Computer Science
College of Computer Science
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.

Date: July.29, 2008

*We have carefully read the dissertation entitled* **A Study on New Techniques of Data Hiding in Images and Text Documents and Their Applications** *submitted by* **I-Shi Lee** *in partial fulfillment of the requirements of the degree of* Doctor *of* Philosophy *and recommend its acceptance.*

[illegible]

[illegible] [illegible]

[illegible] [illegible]

[illegible] Da-chun Wu

Thesis Advisor: [illegible]

Chairman: [illegible]

# 在影像與文字檔案中進行資料隱藏的技術與應用之研究

研究生：李 義 溪　　　　　　　　　　指導教授：蔡文祥博士

國立交通大學資訊學院

資訊工程系

資訊科學與工程研究所

## 摘 要

本博士論文冀能領先全球研究，發展出將資料隱藏於影像及本文文字檔的技術及其應用。本論文總共提出了 10 種新方法，分別適用於黑白，灰階及彩色影像，以及電子郵件，$C^{++}$軟體程式，PDF和網頁等檔案類型。首先，本論文提出二種新方法分別針對黑白及灰階影像，基於人眼視覺模型及動態規劃技術去降低影像扭曲程度並增加資料被隱藏的容量。接著，本論文提出一種新方法可將大量資料藏於BMP彩色影像中，本方法係利用色彩立體方塊及色彩叢集的觀念來隱藏資料。然後，本論文提出一種利用特殊的ASCII控制碼將秘密訊息隱藏於電子郵件中的新方法，這些特殊的ASCII碼顯示在Outlook Express與IE 網路郵件的瀏覽視窗中是使用者看不見的。接著本論文提出二種將資料隱藏於原始程式的

新方法。其中一種方法係利用資訊分享與驗證的技巧及人眼看不見的ASCII控制碼來保護軟體程式的安全。另一種技術是應用於秘密通訊，同時可驗證隱藏訊息真偽的新方法。更進一步，本論文提出二種利用特殊的ASCII碼將資訊隱藏於通用的PDF檔中的新方法。其中之一應用於秘密通訊，另一個應用於驗證PDF檔的真偽。最後，本論文提出二種新方法將資訊隱藏於大家常瀏覽的網頁中。其中之一應用於秘密通訊，另一個應用於驗證網頁的真偽。二種方法皆是利用HTML檔中各種不同的編碼系統的特殊空白碼。以上本論文提出之 10 種方法, 皆為創新之作, 且已投稿於國內外重要期刊。實驗結果顯示本論文提出的方法皆具有可行性及實用性。

# A Study on New Techniques of Data Hiding in Images and Text Documents and Their Applications

**Student: I-Shi Lee** **Advisor: Dr. Wen-Hsiang Tsai**

**Institute of Computer Science and Engineering**
**College of Computer Science**
**National Chiao Tung University**

## Abstract

In this study, data hiding techniques for image files and text documents and their applications are investigated, and totally ten methods are proposed for binary, grayscale, and color images, as well as email, software $C^{++}$ program, PDF, and webpage files. First, two methods are proposed respectively for binary and grayscale images based on human vision modeling and dynamic programming to reduce the image distortion and increasing data hiding capacities. Also, a method is proposed for hiding large-volume data in BMP color images, based on the use of color cubes and the idea of color clustering. Then, a method is proposed for hiding secret messages in emails using some special ASCII codes which are invisible in the window of Outlook Express and IE Webmail browsers. Also proposed are two methods for data hiding in software programs. One is used for security protection of software programs by information sharing and authentication techniques using invisible ASCII control codes. And the other is applicable to covert communication with the additional capability of authenticating the hidden secret message. Furthermore, two methods are proposed for data hiding in PDF files which are popular nowadays. One is useful for covert communication and the other for PDF file authentication, both using certain special ASCII codes. Finally, two methods are proposed for data hiding in web pages which are browsed by lots of people in the world. One method is proposed for covert communication and the other for authentication of web pages, both utilizing certain space codes of various coding systems applicable in HTML files. Experimental results show the feasibility and practicality of all the proposed methods.

# Acknowledgements

I would like to express my sincere appreciation to my advisor, Professor Wen-Hsiang Tsai, for his patience and kind guidance throughout the course of this dissertation study. Appreciation is extended to Mr. Jiun-Tsung Wang for his programming support and helpful suggestions to my study on data hiding utilizing PDF files. Thanks are also extended to the colleagues in the Computer Vision Laboratory at National Chiao Tung University for their valuable help during this study.

Finally, I am so grateful to my wife and my parents for their love, support, and endurance. This dissertation is dedicated to them.

# Table of Contents

摘 要 ........................................................................................ viii
Abstract ........................................................................................ x
Acknowledgements ........................................................................................ xii
Table of Contents ........................................................................................ xiii
List of Tables ........................................................................................ xvi
List of Figures ........................................................................................ xviii
Chapter 1 Introduction ........................................................................................ 1
1.1 Scope of Data Hiding Research ........................................................................................ 1
1.2 Motivation of Study ........................................................................................ 2
1.3 Contributions of This Study ........................................................................................ 4
1.4 Dissertation Organization ........................................................................................ 6
Chapter 2 Surveys of Related Studies and Brief Descriptions of Proposed Methods 7
2.1 Survey of Related Studies ........................................................................................ 7
2.2 Brief Descriptions of Proposed Methods ........................................................................................ 14
Chapter 3 Data Hiding in Binary Images with Distortion-Minimizing Capabilities by Optimal Block Pattern Coding and Dynamic Programming Techniques ........................................................................................ 21
3.1 Idea of Proposed Method ........................................................................................ 21
3.2 Proposed Data Embedding Process ........................................................................................ 22
3.3 Experimental Results ........................................................................................ 37
3.4 Concluding Remarks ........................................................................................ 46
Chapter 4 Data Hiding in Grayscale Images by Dynamic Programming Based on A Human Visual Model ........................................................................................ 49
4.1 Idea of Proposed Method ........................................................................................ 49
4.2 Cost Function for Distortion Measurement ........................................................................................ 50
4.3 Proposed Horizontal Data Hiding Process ........................................................................................ 55
4.4 Proposed Data Recovery Process ........................................................................................ 64
4.5 Experimental Results ........................................................................................ 66
4.6 Concluding Remarks ........................................................................................ 73
Chapter 5 Data Hiding in Color Images by Color Replacements with Reduction of Image Distortion and Change Noticeability ........................................................................................ 76
5.1 Idea of Proposed Method ........................................................................................ 76
5.2 Detailed Algorithms of Proposed Data Embedding and Extraction ........................................................................................ 89
5.3 Experiment Results and Discussions ........................................................................................ 92

5.4 Concluding Remarks....96
Chapter 6 Data Hiding in Emails and Applications by Unused ASCII Control Codes 98
6.1 Idea of Proposed Method....98
6.2 Properties of Email Systems....99
6.3 Embedding ASCII Control Codes into Emails....101
6.4 Proposed Data Hiding Process for Emails....104
6.5 Proposed Data Recovery Process for Emails....106
6.6 Proposed Authentication Process for Email Documents....107
6.7 Experimental Results....109
6.8 Concluding Remarks....117
Chapter 7 Security Protection of Software Programs by Information Sharing and Authentication Techniques Using Invisible ASCII Control Codes....119
7.1 Idea of Proposed Method....119
7.2 Proposed Program Sharing Scheme....121
7.3 Secret Program Recovery Scheme....125
7.4 Discussions on Security Protection....128
7.5 Experimental Results....129
7.6 Concluding Remarks....130
Chapter 8 Covert Communication with Authentication via Software Programs Using Invisible ASCII Codes....135
8.1 Idea of Proposed Method....135
8.2 Data Hiding Using Invisible Codes....137
8.3 Secret Hiding, Recovery and Authentication....138
8.4 Experimental Results....140
8.5 Concluding Remarks....140
Chapter 9 Covert Communication via PDF Files and PDF File Authentication by Invisible Codes....144
9.1 Idea of Proposed Methods....144
9.2 Principle of Encoding Message Data....146
9.3 Message Hiding and Recovery for Covert Communication and Experimental Results....149
9.4 PDF Authentication Process and Experimental Results....153
9.5 Concluding Remarks....158
Chapter 10 Secret Communication through Web Pages and Automatic Authentication of Web Pages Using Special Space Codes in HTML Files....160
10.1 Idea of Proposed Method....160
10.2 Secret Message Coding Using Space Characters in HTML....162

10.3 Message Hiding and Experimental Results ........................................ 165
10.4 Automatic Authentication of Web Page Text Contents ........................ 172
10.5 Security Consideration and Experimental Results ............................... 174
10.6 Concluding Remarks ............................................................................... 178
Chapter 11 Conclusions and Suggestions for Future Research ............................... 180
11.1 Conclusions ............................................................................................ 180
11.2 Suggestions for Future Research ........................................................... 181
References ........................................................................................................
183
Publication List ................................................................................................ 192
Vita ....................................................................................................................
194

# List of Tables

Table 3.1 Proposed block pattern encoding table……………………………………….25

Table 3.2 An extraction table (table index *B*=0)………………………………......36

Table 3.3 Statistics of three stego-images for proposed algorithms…………………47

Table 3.4 Statistics of 19 stego-images processed by proposed DPA……………..…48

Table 4.1 A block pattern encoding table proposed in this study…………………..…59

Table 4.2 An extraction table (table number *T*=0)……………………………………66

Table 4.3 Statistics of stego-images yielded by DPA using optimal encoding table…71

Table 4.4 Comparison of run times for four methods for grayscale images (in unit of sec.)……………………………………………………………………..71

Table 4.5 Comparison of PSNR values of the four methods for grayscale images (in unit of dB)……………………………………………………………………72

Table 4.6 Comparison of RS analysis results of the four methods for color images...73

Table 5.1 The colors in the (0, 0, 0)-th color cube with base color (*r*, *g*, *b*) = (0, 0, 0)……………………………………………………………………...81

Table 5.2 Color encoding table for the (0, 0, 0)-th color cube with base color (0, 0, 0)……………………………………………………………………...87

Table 5.3 Statistics of experimental results……………………………………………..94

Table 6.1 ASCII control codes and description……………………………………100

Table 7.1. ASCII control codes and descriptions………………………………….....120

Table 7. 2 Invisible character coding table…………………………………………….121

Table 8.1. Invisible codes under various environments……………………………136

Table 9.1 ASCII codes selected for message representations in this study…………145

Table 9.2 Null space coding table for message “This is a covert communication method”…………………………………………………………………..151

Table 10.1 Character representations in HTML…………………………………….164

## List of Figures

Figure 1.1 Classification of data hiding techniques…………………………………...3

Figure 3.1 Illustration of block patterns and corresponding binary values………..…23

Figure 3.2 Division of input image into 2×2 blocks with separating lines (grids with bold boundaries are 2×2 blocks for data embedding)……………………27

Figure 3.3 An example of proposed data embedding process………………………..29

Figure 3.4 An example of proposed data embedding process………………………..32

Figure 3.5 Flowchart of the proposed data embedding process……………………...34

Figure 3.6 Flowchart of the proposed extraction process……………………………38

Figure 3.7 Input binary images, output stego-images with message data, and difference images. (a) Binary image “NCTU”. (b) Stego-image using greedy search and optimal encoding table. (c) Stego-image using DPA and optimal encoding table. (d) The difference image between (a) and (c) in which the white spots are difference pixels………………………………39

Figure 3.8 Input binary images, output stego-images with message data, and the difference images. (a) Binary image “Lena”. (b) The stego-image using the greedy search algorithm and the optimal encoding table. (c) The stego-image using the DPA and the optimal encoding table. (d) The difference image between (a) and (c) in which the white spots are difference pixels………………………………………………………41

Figure 3.9 Input binary images, output stego-images with message data, and difference images. (a) Binary image “Patent.” (b) Stego-image resulting from greedy search and optimal encoding table. (c) Stego-image using DPA and optimal encoding table. (d) An enlarged part of difference image between (a) and (c) in which the white spots are difference pixels……..43

Figure 4.1 Three grayscale images and their 8 corresponding bit planes (from left to right, original images, $bp_0$, $bp_1$, $bp_2$, …, and $bp_7$, respectively)…………50

Figure 4.2 Division of input image into 2×2 blocks with separating lines (grids with bold boundaries are 2×2 blocks for data embedding)……………………58

Figure 4.3 An example of proposed data embedding process………………………..61

Figure 4.4 A cover image "House" with the size of 256×256 and its stego-image with 16440-bit message data embedded. (a) The cover image. (b) The stego-image………………………………………………………67

Figure 4.5 Experimental results of three images. (a) The original images and their corresponding bit planes (repeated from Figure 4.1). (b) The resulting three stego-images and their corresponding bit planes (from left, $bp_0$, $bp_1$, $bp_2$, …, $bp_7$)………………………………………………………………68

Figure 4.6 PSNR values of stego-image "Lena" using DPA……..…………………72

Figure 5.1 An illustration of range sets of color cubes………………………………78

Figure 5.2 A color cube with 8 colors divided into four groups with base color (0, 0, 0)……82

Figure 5.3 Illustration of a 3D spherical coordinate system for use in even color distribution……..…………………………………………………………86

Figure 5.4 An example of color distribution in a color cube --- the 8 colors in group 3……………………………………………………………………...…..87

Figure 5.5 An experimental result of message data embedding applied to Figure 5.1(a) with a 256×256 cover image and a 22900-byte message data…………....95

Figure 5.6 A second experimental result with a 256×256 cover image and a 22900-byte message……………………………………………………...95

Figure 5.7 A third experimental result of data embedding with a 512×512 cover image and an 88200-byte message……………………………………………96

Figure 5.8 A fourth experimental result of data embedding with a 512×512 cover image and an 88200-byte message……………………………………96

Figure 6.1 Partial content of a cover email………………………………………110

Figure 6.2 Partial content of the stego-email generated from Figure 6.1…………...111

Figure 6.3 Partial content of an embedded secret data file………………………111

Figure 6.4 Partial content of the extracted secret data file…………………………112

Figure 6.5 Partial content of a cover email………………………………………112

Figure 6.6 Partial content of the stego-email generated from Figure 6.5 before being transmitted……………………………………………………………113

Figure 6.7 Partial content of the stego-email received and displayed in IE………...113

Figure 6.8 Content of the original secret file……………………………………..114

Figure 6.9 Content of the extracted secret file……………………………………...114

Figure 6.10 Content of a stego-email for authentication before transmission……...115

Figure 6.11 Authentication result of "pass" after receiving a stego-email by Outlook Express…………………………………………………………115

Figure 6.12 Authentication result of "pass" after receiving a stego-email by IE…..116

Figure 6.13 Authentication result of "fail" after receiving the stego-email by IE. The word "Lee" in the content has been modified to be "lee"………………116

Figure 7.1 Illustration of invisible ASCII control codes in a comment of a source program……………………………………………………………..123

Figure 7.2 Experimental results of sharing a secret program……………………131

Figure 7.3 An experimental result of authenticating a destructed stego-program….134

Figure 8.1 An experimental result………………………………………………...141

Figure 8.2 An example of authentication results…...……………………………143

Figure 9.1 Display of all ASCII codes in Adobe Reader 8.1.2, in which only 20 and A0 appear to be white spaces (the first spaces in the 3rd and the 11th lines)……….146

Figure 9.2 Display of all ASCII codes in the window of Adobe Reader 8.1.2, in which the width of A0 was set to be zero so that A0 becomes nonexistent (i.e., there is no space before the first comma in the 11th row, as compared with Figure 9.1)…..147

Figure 9.3 Invisibility of multiple A0's at between-character locations……………………148

Figure 9.4 An experimental result of null space coding……………………………..152

Figure 9.5 An experimental result for authenticating a PDF file……………………………157

Figure 10.1 Appearances of nine space codes as white spaces in the window of the IE…………………………………………………………………………164

Figure 10.2 Invisibility of space codes for the message "sky" in an HTML text…..169

Figure 10.3 The embedded secret data…………………………………………...…171

Figure 10.4 An experimental result of authentication of a modified web page…….177

# Chapter 1
# Introduction

## 1.1 Scope of Data Hiding Research

Data hiding is a type of information hiding, emphasizing the purpose of embedding digital data behind multimedia of various forms. The multimedia into which data are hidden are called *cover media*, like cover image, cover text, etc., and the results are called *stego-media*, like stego-image, stego-text, etc. Applications of data hiding include at least the following.

(1) Copyright protection --- the data hidden are of the forms of *watermarks* like logos of companies, series numbers of products, etc.

(2) Covert communication --- the data hidden are *secret messages* sent from one site to another. Data hiding for the purpose of covert communication is sometimes called *steganography*. The goal of steganography is to arouse as little notice from observers of the stego-media as possible.

(3) Multimedia authentication --- the hidden data are *authentication signals* in various forms, created for the purposes of checking cover media's fidelity, integrity, utilization rights, etc.

(4) Secret sharing --- the hidden data are parts of certain multimedia forms like text or image documents, and are taken as *secret messages* which are embedded into several *shares* in other forms of multimedia. Only a sufficient number of shares are collected can the secret message be recovered for inspection.

(5) Data association --- the hidden data are various information, like metadata, history, identification, etc., about the cover media. Data hiding in this way

facilitates close association of the data with the cover image for convenient preservation or transmission of the cover media.

(6) Digital rights management --- the hidden data need not always be invisible; on the contrary, we may embed a *visible* watermark on a video (like a movie) to prevent it form being watched by a customer paying no fee. More generally, data hiding may be used in applications of digital rights management, like pay movie control, video distribution management, limitation of watch times, etc.

A classification of the techniques of data hiding which are related to this dissertation study is illustrated in Figure 1.1.

In the following sections, the motivation of study is given in Section 1.2. The contributions of this study and the organization of this dissertation are reported in Sections 1.3 and 1.4, respectively.

## 1.2 Motivation of Study

Many data hiding techniques with images as cover media have been proposed. Most of the techniques were proposed for color and grayscale images because pixels in such images take a wide range of values and so are more proper for data hiding. Only a few techniques were proposed for binary images. In this study, we will investigate more efficient methods for hiding more data in binary images.

On the other hand, the cover media need not always be images. On the Internet, so many documents of formats other than images are being transmitted or displayed, like e-mails, web pages, freeware, etc. If we can hide data behind e-mails, for example, covert communication will be easily implemented. Authentication of e-mails is also possible to prevent receiving false or illegally altered messages. Furthermore, it is also desired to protect software from being stolen or illegal distributed. If we can hide data into the source programs, then possibly protection of software copyrights is achievable. It is noted by the way that studies of data hiding in text contents are very few so far.

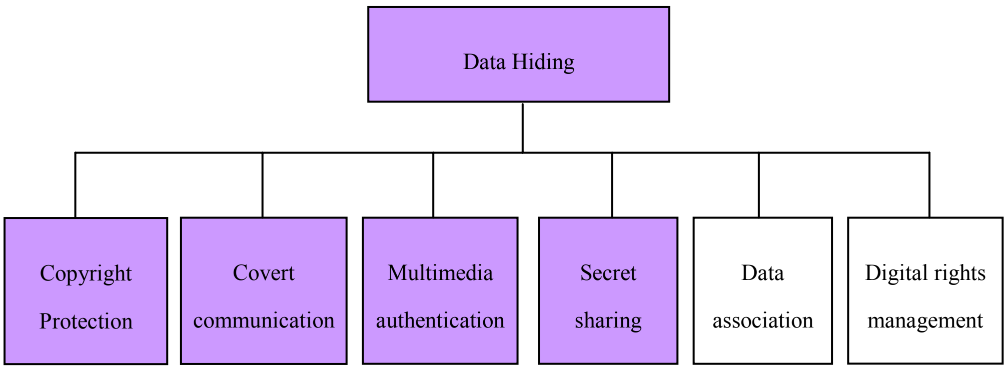


Figure 1.1. Classification of data hiding techniques.

Most researches about data hiding in images lack serious considerations of image distortion reduction in stego-images. It is desired in this study to design new techniques of data hiding emphasizing optimality in image distortion reduction. In doing so, it is also hoped that human vision modeling may be considered, so that changes in the resulting stego-image can be less noticeable. It is noted here that most existing data hiding methods are conducted in the frequency domain and thus are useful for images compressed in the frequency domain like JPEG. For images of other types like BMP, appropriate data hiding methods need be developed. And this is also part of the goal of our study on data hiding in images.

On the other hand, it is also a goal of this study to devise new techniques for data hiding in text documents, which are still few so far. Such techniques will be very useful for daily uses because text documents like e-mails are popular and used or watched every day by humans worldwide, especially for the purpose of steganography.

## 1.3 Contributions of This Study

In this study, we propose ten data hiding techniques for various applications of copyright protection, covert communication, authentication, and secret sharing. The processed file types include two major categories, namely, image and document. The former category includes binary, grayscale, and color images, and the latter type includes email, software program, PDF, and HTML (web page). The contribution of each of the ten techniques is described in the following.

(1) Data hiding in binary images --- the proposed technique has distortion-minimizing capabilities by optimal block pattern coding and dynamic programming techniques. Accordingly, not only more data bits can be embedded in an image block on the average, but the resulting image distortion is also reduced in an optimal way.

(2) Data hiding in grayscale images --- the proposed technique is based on dynamic programming and a human visual model with distortion-minimizing capabilities. The proposed method can predict the PSNR value of the resulting image according to the size of the data to be embedded before the embedding process starts.

(3) Data hiding in color images --- the proposed technique is based on color replacements with capabilities of reducing image distortion and change noticeability. Color cubes and the idea of color clustering are used for large-volume data hiding.

(4) Data hiding in emails and applications --- the proposed technique embeds data in emails via Outlook Express and IE by some unused and invisible ASCII control codes. Also described are two applications of the proposed data hiding technique,

covert communication via emails and authentication of emails.

(5) Security protection of software programs --- the proposed technique is based on information sharing and authentication using invisible ASCII control codes. These invisible codes are hidden in the camouflage program, resulting in a stego-program for a participant to keep. To enhance security, three security measures via the use of a secret random key are also proposed to prevent the secret program from being recovered illegally, authenticate the stego-program and check the stego-program whether it has been tampered with or not.

(6) Covert communication with authentication via software programs --- the proposed technique is also based on the use of invisible ASCII codes. Each binary message, after being encoded by certain ASCII codes and inserted at specific C++ program locations, becomes invisible in the source code editors. A scheme for tamper-proof authentication of the embedded message has also been proposed.

(7) Covert communication via PDF files --- the proposed technique is based on the use of special ASCII codes. A secret message, after being encoded by a special ASCII code and embedded at between-word and between character locations in the text of a PDF file, becomes invisible in the window of a common PDF reader, creating a steganographic effect for secret transmission through the PDF file.

(8) Authentication of PDF files --- the proposed technique is based on the use of invisible ASCII codes. To authenticate each word in a PDF file, a authentication signal composed of multiple non-breaking space codes is generated from the characters in the word and a random number. The authentication signal is invisible for common PDF readers, thus reducing the probability for the authentication signal to be tampered with.

(9) Secret communication via web pages --- the proposed technique is based on the use of some special space codes in HTML. These codes, like the ASCII code 20,

appear to be white spaces as well. Message hiding and recovery with security enhancement are also proposed.

(10) Automatic authentication of web pages --- the proposed technique is based on the use of multiple special space codes in HTML. The propose method is useful for checking automatically the integrity of the text content of a web page at the word level. Special space codes are used again as authentication signals with steganographic effects. Security enhancement techniques using secret keys and multiple word encoding are also proposed.

## 1.4 Dissertation Organization

In the remainder of this dissertation, a survey of related studies and a more detailed description of the ten proposed methods are given in Chapter 2. The proposed methods are described one by one in the subsequent chapters. In Chapter 3, the proposed method for data hiding in binary images is described. In Chapter 4, the proposed method for data hiding in grayscale images is presented. In Chapter 5, the proposed method for data hiding in color images is described. In Chapter 6, the proposed method data hiding in emails and some applications are described. In Chapter 7, the proposed method for security protection of software programs is presented. In Chapter 8, the proposed method for covert communication with authentication via software programs is described. In Chapter 9, the proposed methods for covert communication via PDF files and authentication of PDF files are described. In Chapter 10, the proposed methods for secret communication via web pages and automatic authentication of web pages are described. Finally, in the last chapter, conclusions of this study and some suggestions for future research are included.

# Chapter 2

# Surveys of Related Studies and Brief Descriptions of Proposed Methods

## 2.1 Survey of Related Studies

Many data hiding techniques have been proposed while this dissertation study is dedicated to develop new data hiding techniques for various applications. Surveys of related studies on data hiding are described first in the following, followed by brief descriptions of the proposed methods.

### 2.1.1. Survey of Data Hiding in Binary Images

Many data hiding techniques have been proposed for a variety of applications of digital images in recent years [1-22]. Most of the techniques were proposed for color and grayscale images because pixels in such images take a wide range of values and so are more proper for data hiding. One simple method to data hiding in grayscale images is to use the LSB replacement technique to hide secret data or authentication signals. However, data hiding in binary images is a more challenging work. Because binary image pixels have drastic contrast, it is easier for humans' eyes to find out pixel value changes in binary images. Therefore, it is more difficult to hide data into binary images than into color or grayscale images. Wu et al. [12] embedded secret data in specific image blocks that are selected with higher "flippability" scores by pattern matching. Manipulated flippable pixels on the image region boundary are then used to embed a significant amount of data without causing noticeable artifacts. Pan et al. [6] changed pixel values in image blocks, mapped block contents into the secret data, and used a secret key and a weight matrix to protect the hidden data. Given an image block of size $m \times n$, the scheme can conceal up to $\lfloor \log_2(m \times n + 1) \rfloor$ bits of data in the image by changing, at most, two bits in an image block**.** Tseng and Pan [8] proposed a technique to alter an image bit into a new value identical to a neighboring one. It can yield better hiding effect within a binary image. Koch and Zhao [2] embedded a bit 0 or 1 in a block by changing the number of black pixels in the block to be larger or smaller than that of white ones, respectively. In [5, 11], secret data are concealed into dithered images by maneuvering dithering patterns. Tzeng and Tsai [9] encoded the edge features of binary images into 4×4 block patterns, and authenticated the images by pattern matching. Tzeng and Tsai [10] also proposed a new feature, called surrounding edge count, for measuring the structural randomness in a 3×3 image block, and defined "pixel embeddability" from the viewpoint of minimizing image distortion. Accordingly, embeddable image pixels suitable for hiding secret data can be selected. Wu et al. [14] used even-odd relationships of lengths of run pairs to embed information in binary images, and adjusted the length of each run to an even or odd value to represent the embedded bit value.

### 2.1.2. Survey of Data Hiding in Grayscale Images

Wang et al. [15] embedded an image in the fifth LSB bit plane of a cover grayscale image, and employed an optimal substitution process based on a genetic algorithm and a local pixel adjustment method to lower the distortion in the stego-image. Chang et al. [16] used dynamic programming to obtain an optimal solution for the LSB substitution method. Chan and Cheng [17, 18] presented an optimal pixel adjustment process to improve the image quality of the stego-image acquired by Wang's schemes. Thien and Lin [19] proposed a method for hiding data in images digit by digit using a modulus function. The method is better than simple LSB substitution not only in eliminating false contours but also in reducing image distortion. Lee and Chen [20] applied variable-sized LSB insertion to estimate the maximum embedding capacity by a human visual system (HVS) property, and to maintain image fidelity by removing false contours in smooth image regions. Liu et al. [21] presented a novel bit plane-wise data hiding scheme using variable-depth LSB substitution and employed post-processing to eliminate the resulting noticeable artifacts.

Most of the above methods lack consideration of using precise human visual models in improving the data hiding effect. Instead, Wu and Tsai [13] presented a method based on the HVS by modifying quantization scales according to variation insensitivity from smooth to contrastive to improve stego-image quality. And Lie and Chang [22] presented an adjusted LSB technique with the number of LSBs adapting to the pixels of different grayscales.

On the other hand, some steganalysis techniques were developed to detect secret messages among stego-images. Lyu and Farid [23] developed a universal blind detection scheme to detect hidden messages in stego-images, which uses wavelet-like decomposition to build higher-order statistical models of natural images and adopts the support vector machine as an optimal classifier to separate stego-images from cover images. The method demonstrates good performance on JPEG images and the selected statistics is rich enough to detect hidden data in the results yielded by a very wide range of steganographic methods. In addition, to detect data hidden in LSBs in the spatial domain, it is observed that the basic LSB substitution method changes pixel values only between $2i$ and $2i + 1$ in the $i$-th bit plane of the pixel value. This leads to an effective steganalytic technique, the RS method proposed by Fridrich, et al. [24], which not only can expose the presence of secret data but also can estimate the length of the embedded data.

### 2.1.3. Survey of Data Hiding in Color Images

Many techniques for data hiding in color images have been proposed in the past decade [1, 7, 27] which may be categorized into two major methods: the spatial-domain method and the frequency-domain method. In the former, secret data are directly embedded in the characteristics of the pixels of the cover image, and in the latter, the cover image is transformed first into frequency-domain coefficients, into which secret data are embedded. In general, the frequency-domain method is more robust against attacks while the spatial-domain method can hide more data. The previously-surveyed methods for data hiding in binary and grayscale images are conducted in the spatial domain. For the other method, related papers are very few unless the previously-surveyed methods are adapted to be applicable to color images, for example, by considering each color channel as a grayscale image. Tsai and Wang [28] proposed a data hiding technique for color images using a binary space partitioning tree, which partitions the RGB color space into voxels and embeds three message bits into each voxel.

### 2.1.4. Survey of Data Hiding in Text Documents

In contrast with other multimedia, digital texts contain less redundant information for embedding data. Most data hiding methods for digital text documents try to encode information directly *into the text itself* or *into the text format*. One way of into-text hiding is to exploit the natural redundancy of languages, and one way of into-format hiding is to adjust inter-word or inter-line space [29]. On the other hand, from the steganographic point of view, digital text documents can be classified into two types: hard-copy and soft-copy [27]. A hard-copy text may be treated as a binary image resulting from scanning a text document, while a soft-copy text may be regarded as an American Standard Code for Information Interchange (ASCII) text that can be edited by a text editing software like Notepad.

For a hard-copy text, which is interpreted as a highly-structured image, information can be embedded into the layout or format of the image. Low et al. and Brassil et al. [30-31] presented text-based steganographic methods which use the distances between consecutive lines of texts or between consecutive words to hide information. If the space between two lines is smaller than a threshold, a “0” is represented; otherwise, a “1.”

In contrast with hard-copy texts and other digital media, soft-copy texts are more difficult to hide data due to the lack of redundant information. Even a slight modification, like rewriting a letter, may be noticed by a reader. However, huge amounts of text documents that people deal with daily on the Internet are essentially soft-copy texts in nature. Thus, the protection of digital rights of this type of text document becomes more and more important.

Bender et al. [27] proposed the use of infrequent additional spaces to form secret data and transmit them in soft-copy texts, including inter-sentence spacing, end-of-line spacing, and inter-word spacing in texts. For example, one space between words is taken to represent a “0” and two spaces a “1.” Wayner [32] proposed a method to use the context-free grammar to create secret text messages in cover files for covert communication; the secret message is not embedded in the cover file directly. And a receiver extracts the hidden message by parsing. A constraint is that the cover text should be a meaningful message; otherwise, a reader will doubt it.

Cantrell and Dampier [33] proposed to embed data into unused spaces in file headers. These spaces are invisible to usual users because they are disregarded when the files are opened. The spaces can be seen when examined at the byte level, but few users would do so. Johnson et al. [34] proposed another way to embed information in unused spaces that are imperceptible to an observer, which is based on the fact that usually operating systems allocate more space than the need of a file and the result leaves some unused space to hide information. A third method is to create a hidden partition in a file system to embed information. The partition is not viewed normally. This concept has been expanded in a steganographic file system [35]. If a user knows the file name and the password, access to the file will be granted; otherwise, no evidence of the file will be revealed in the system of the hidden files.

Characteristics inherent in network protocols may also be taken advantage of to hide information [36]. For example, TCP/IP packets can be used to transmit secret messages across the Internet by embedding unused spaces in the packet header. Finally, Chang and Tsai [37] proposed a special space encoding to embed copyright information into the HTML text content. The bit "1" is encoded by inserting a so-called *pseudo-space string* "&nbsp;" before a real space, while the bit "0" is represented by a normal space between two words or sentences.

### 2.1.5. Survey of Data Hiding and Sharing in Software Programs

A survey about watermarking in programs can be found in Zhu, et al. [54]. Two methods have been identified: static and dynamic. The former inserts and extracts watermarks in program codes without running the program while the latter does the same in the execution state of a software object. Two respective examples are Venkatesan, et al. [55] and Collberg and Thomborson [56]. There exist other methods with digital text, sentence syntax, text typos, e-mails [1, 27, 53, 57-59] as cover media.

The concept of secret sharing was proposed first by Shamir [46]. By a so-called ($k$, $n$)-threshold scheme, the idea is to encode a secret data item into $n$ shares for $n$ participants to keep, and any $k$ or more of the shares can be collected to recover the original secret, but any ($k$ – 1) or fewer of them will gain no information about it. A similar scheme, called *visual cryptography*, was proposed by Naor and Shamir [46] for sharing an image. The scheme provides an easy and fast decryption process consisting of xeroxing the shares onto transparencies and stacking them to reveal the original image for visual inspection. This technique has been investigated further in [48-50], though it is suitable for binary images only. Verheul and van Tilborg [51] extended the visual cryptography technique for processing images with small numbers of gray levels or colors. Lin and Tsai [52] proposed a digital version of the visual cryptography scheme for color images with no limit on the number of colors. The $n$ shares obtained from a color image are hidden in $n$ camouflage images which may be selected to have well-known contents, like famous characters or paintings, to create additional steganographic effects for security protection of the shares.

### 2.1.6. Survey of Data Hiding in PDF Documents

Portable Document Format (PDF) files [63] are popular nowadays, and so using them as carriers of secret messages for covert communication is convenient. Though there are some techniques of embedding data in text files [57-58], studies of using PDF files as cover media are very few, except Zhong et al. [64] in which integer numerals specifying the positions of the text characters in a PDF file are used to embed secret data.

For security, it is necessary to verify the authenticity of a file received from another party or kept for a long time in a certain environment, before the file is used for various purposes. This is the *authentication* problem of the file, which should be solved for protection of the file against unintentional changes and malicious manipulations. In the past, the information hiding method [1] has been adopted to solve this problem but most studies were about images [10, 66-71]. There is yet no investigation on the authentication of PDF files, though a related study about data hiding in PDF files can be found in Zhong et al. [64]. Hiding data in documents other than PDF files have also been investigated [61-62].

### 2.1.7. Survey of Data Hiding in HTML Documents

About hiding data in the HTML, Shirali-Shahreza [72] protects a Java applet in an HTML file from being copied by hiding a special 8-character string with a key within the Java applet. Wu and Lai [60] hide binary data in HTML files using various properties of tags, like attributes. Wu, et al. [73] use hash functions to compute digests of web page contents as watermarks. Chang and Tsai [37] insert extra white spaces in HTML text to encode bits for watermarking, as done by some commercial software [74].

There are very few studies on web page authentication using data hiding techniques so far. Zhao and Lu [75] generated watermarks of web pages based on principal component analysis and embed them by upper and lower cases of letters in HTML tags. The watermark was used to check the integrity of the entire web page. Wu et al. [73] designed fast fragile watermarks for web pages based on hash functions which generate digests of web pages quickly with case insensitivity. Two related studies can be found in [37, 60] which utilize properties of spaces, tabs, tags, attributes, etc., to encode and hide data bits into the HTML for purposes other than web page authentication. And some more general studies about data hiding can be found in [1, 76].

## 2.2 Brief Descriptions of Proposed Methods

In this dissertation study, we have developed totally ten methods, three for data hiding in various images with distortion reduction capability, one for data hiding in emails with capabilities of authenticating the hidden data, two for data hiding in source programs, two for data hiding in PDF files, and finally two for data hiding in web pages. They are briefly described in the remainder of this section

### 2.2.1. Data Hiding in Binary Images with Distortion-Minimizing Capabilities by Optimal Block Pattern Coding and Dynamic Programming Techniques

The first method we propose is a new technique which embeds data into a binary image and minimizes the resulting image distortion in an optimal way. In a binary image, there are two distinct pixel values, 0 and 1, corresponding to *black* and *white* pixels, respectively. When data are embedded into a binary image, some image pixels used for data hiding will be changed from black to white or reversely. The pixel value changes will be called *bit flippings* in the sequel. To embed more data, more bit flippings may be conducted; however, the quality of the resulting image will also get worse. The bit flipping rates of most data hiding methods for binary images are about 50%. We propose a new data hiding method which has the capability to conceal up to *three* data bits in a 2×2 block, resulting in bit flipping rates lower than 50%. The method can thus be used to embed more data. This is achieved by a block pattern coding technique. On the other hand, while it is desirable to embed more data, the resulting image quality should be maintained in the mean time. For this purpose, two optimization techniques are proposed. The first is to use multiple block pattern encoding tables, from which an optimal one is selected for each input image. The second technique is to use a dynamic programming algorithm to divide the message data stream into appropriate bit segments for optimal data embedding in the image blocks in the sense of minimizing the number of bit flippings. As a result, the proposed method can achieve the goals of both increasing the embedded data volume and reducing the resulting image distortion. Furthermore, the method can be used to extract embedded data without referencing the original image.

### 2.2.2. Data Hiding in Grayscale Images by Dynamic Programming Based on A Human Visual Model

The second method we propose is a new technique which embeds data into a grayscale image, based on the use of a new HVS model, to estimate the number of usable bits of each pixel in the cover image. Furthermore, a block pattern encoding method is proposed to embed up to three data bits in a 2×2 block of the bit planes without visible degrading of the stego-image quality. This is achieved by using two optimization techniques. The first technique utilizes multiple block pattern encoding tables, from which an optimal one is chosen for each input image; and the second technique uses dynamic programming to divide the message data stream into appropriate bit segments for optimal data bit embedding in the image blocks to minimize a cost function. Especially, the proposed method can predict the PSNR value of the stego-image according to the embedded data size before the embedding process is started. Moreover, the proposed method can extract embedded data without referencing the original image, and does not require post-processing to refine the stego-image quality.

### 2.2.3. Data Hiding in Color Images by Color Replacements with Reduction of Image Distortion and Change Noticeability

The third proposed method is a new one for hiding data in RGB color images using *color space partitioning* and *color encoding*. The RGB color space is partitioned into non-overlapping, equal-sized color clusters, each being cubic in shape, called a *color cube*. The colors in each cube are used to represent fixed-length codes. Message data hiding is accomplished by replacing selected image pixels' colors with closest ones in color cubes to embed corresponding codes representing the message bits. And data extraction is a reverse process of data embedding. To reduce image distortion, each color cube is designed to include a number of color groups, with all colors in each group representing an identical code. The colors in each group are distributed as separately as possible in the cube, and color replacement at an image pixel is conducted by choosing as the replacing color the one in a group, which is *closest* to the pixel's color in the sense of Euclidean color distance. And to reduce the noticeability of the resulting color changes, we select adaptively for use in data embedding those cubes whose colors are *more scattered* in the cover image (that is, the pixels whose colors are in these cubes are more separated mutually in the cover image), so that the color changes on these pixels will arouse less notice from the observer.

### 2.2.4. Data Hiding in Emails and Applications by Unused ASCII Control Codes

The fourth proposed method is a new technique for data hiding in emails via Outlook Express and IE under the operating system of the traditional Chinese version of Microsoft Windows XP, service pack 2, 2002. The idea is based on the use of unused ASCII codes. Secret data are encoded by special ASCII control codes and embedded into cover emails by inserting the data into the text line ends in the body of a given email. These ASCII control codes, when displayed both by Outlook Express and IE, are invisible to the user, achieving the effect of steganography. Such invisible ASCII control codes were found out in this study by a systematic test of all the ASCII codes on various email server software systems and standards. The proposed data encoding technique is a combination of five coding rules found in this study, which insert special ASCII control codes into different places in email texts. The inserted codes will not change the meanings of the sentences in the cover email, neither causing any noticeable difference to the reader. Furthermore, hidden data can be extracted from a stego-email completely to recover the original email text content.

Also described in this study are two applications of the proposed data hiding technique, namely, covert communication via emails and authentication of emails. In the former application, security is enhanced by the use of a secret key, and in the latter, an authentication signal is generated from the cover email for email fidelity checking.

### 2.2.5. Security Protection of Software Programs by Information Sharing and Authentication Techniques Using Invisible ASCII Control Codes

The fifth proposed method is a new technique based on the use of some specific ASCII control codes *invisible* in certain software editors. By the use of the logic operation of "exclusive-OR," each source program to be shared is transformed into a number of shares, say *N* ones, which are then hidden respectively into *N* pre-selected *camouflage source programs*, resulting in *N stego-programs*. Each stego-program still can be compiled and executed to perform the function of the original camouflage program, and each camouflage program may be selected arbitrarily, thus enhancing the steganographic effect.

To improve the security protection effect further, we propose additionally an authentication scheme for verifying the correctness of the contents of the stego-programs brought by the participants to join the process of secret program recovery. This is advantageous to prevent any of the participants from accidental or intentional provision of a false or destructed stego-program. The verified contents include the share data and the camouflage program contained in each stego-program.

### 2.2.6. Covert Communication with Authentication via Software Programs Using Invisible ASCII Codes

The sixth proposed method is a new one for covert communication by embedding messages in source programs. A binary message, after being encoded into some ASCII codes and embedded into certain C++ program locations, becomes *invisible* in the source code editors of Visual C++ and C++ Builder under some Windows OS environments, creating a steganographic effect. A tamper-proof authentication scheme for the embedded message is also proposed.

### 2.2.7. Covert Communication via PDF Files and PDF File Authentication by Invisible Codes

The seventh proposed method is a new technique for covert communication, which embeds secret messages in PDF files. A message is regarded as a string of bits or characters, which are then encoded with a special ASCII code by binary or unitary coding. The results, after being embedded at the between-word or between-character locations in the text of a PDF file, are found in this study to be *invisible* in the windows of common PDF readers, creating a steganographic effect and achieving the purpose of secret communication.

The eighth method is proposed for authenticating PDF files using a special ASCII code A0. For each word in the text of a PDF file to be protected, an authentication signal composed of repeating A0's is generated from the 8-bit ASCII codes of the characters composing the word as well as a random number. The signal is then embedded to the right of the word. These A0's are invisible in the window of common PDF readers, enhancing the security of the embedded authentication signals. Without the key for use in generating the random numbers, malicious creation of a fake file is nearly impossible.

### 2.2.8. Secret Communication through Web Pages and Automatic Authentication of Web Pages Using Special Space Codes in HTML Files

The ninth proposed method is a new technique for secret communication by embedding special space codes in the HTML files of web pages. These codes appear as white spaces in the web page, and so may be used to encode secret message bits with steganographic effects. The codes are the result of a thorough investigation of all possible coding systems which can be applied in the HTML file. There are many of such codes, and each of them may be used to encode at least three message bits, increasing the data hiding capability.

The last proposed method is a new automatic authentication technique for checking the integrity of web page text contents. The method, aiming to check the authenticity of each single word, is based on a data hiding technique which uses some special space codes as authentication signals. Such codes, which are found in this study to be multiple and appear identical to normal white spaces in web pages, are used to encode certain binary mapping results from the word contents. These codes are then taken to replace the between-word spaces in the HTML codes, resulting in good steganographic effects. Security enhancement has also been considered, and related problems are solved by the use of secret keys and a multiple word encoding scheme.

# Chapter 3
# Data Hiding in Binary Images with Distortion-Minimizing Capabilities by Optimal Block Pattern Coding and Dynamic Programming Techniques

## 3.1 Idea of Proposed Method

In a binary image, there are only two pixel values, 0 and 1, and the corresponding pixels may be called *black* and *white* ones, respectively. When data are embedded in a binary image, the image pixels will be changed from black to white or from white to black. The distortion rate is 50% in general data hiding methods for binary images. The method which we propose in this study for data hiding in binary images is based on a block pattern coding technique and a dynamic programming algorithm. The method can be used to embed more data in a block of a binary image, and minimize the resulting stego-image distortion simultaneously.

In order to embed more data in a binary image, more pixels need be changed; however, the quality of the resulting stego-image will get worse. On the contrary, in order to maintain the quality of the resulting image, the amount of the embedded data should be limited. The proposed method is designed to be a compromise between the embedded data volume and the resulting image distortion. The method can extract embedded data without referencing the original image. It also has the merit of concealing up to *three* data bits in a 2×2 block by changing the smallest number of bits in a block. Contrastively, most existing methods for hiding data in binary images can embed only one or two data bits in a 2×2 image block [7, 10, 12].

In the remainder of this chapter, the proposed method for dealing with 2×2 image blocks is first described in Section 3.2. Some experimental results are shown in Section 3.3, followed by some cluding remarks in Section 3.4.

## 3.2 Proposed Data Embedding Process

The proposed method is designed to hide secret data behind binary images in random fashions controlled by secret keys. The method consists of a data embedding process and a data extraction process. In this section, the principles behind the proposed method are presented first, followed by the details of the proposed data embedding and extraction processes.

### A. Encoding Block Patterns for Secret Data Embedding

In order to embed secret data into a binary cover image, every 2×2 block of the cover image is regarded as a *pattern* with a corresponding 4-bit *binary value* in this study, with each black pixel representing a bit 0 and each white one representing 1. An illustration is shown in Figure 3.1. Therefore, in a 2×2 block, possible binary values of the block pattern are $0000_2$ through $1111_2$, where “$0000_2$” means an entirely black block while “$1111_2$” means an entirely white one.

The main idea of the proposed data hiding method is based on the use of a *block pattern encoding table* which maps each block pattern into a certain code for use as hidden data with the code being up to three bits in length. And data embedding is accomplished by changing the block patterns so that the codes of the resulting blocks become just the input secret data to be embedded. A *block pattern encoding table* designed for use in this study is shown in Table 3.1. The idea behind the design of this table is described as follows. It is emphasized, by the way, that such a table is just one of the many possible ones usable for data hiding, and the proposed data embedding process will choose from them an *optimal* one for each specific input image, as described later.

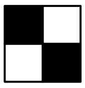
| 2×2 block pattern | Corresponding binary value |
|---|---|
| $\begin{array}{\|c\|c\|}\hline b_1 & b_2 \\ \hline b_3 & b_4 \\ \hline\end{array}$ | $b_1b_2b_3b_4$ |
|  | 0101 |

Figure 3.1 Illustration of block patterns and corresponding binary values.

The number of possible patterns in a 2×2 block are 16. This number is much larger than the need to represent the two secret bits ‘0’ and ‘1’, so we may use multiple block patterns to represent a single secret value, allowing the possibility of choosing among the patterns an optimal one to replace the original image block in the data embedding process, thus reducing the resulting image distortion in the replaced block. Furthermore, we wish to embed more data in a block, and for this goal we may use a block pattern to represent more than one bit of secret data.

For example, we may use both the block pattern $t_1 = 1101_2$ and the pattern $t_2 = 1110_2$ to represent the two-bit secret value $s = 00_2$. In this way, when we want to embed, for example, the secret value $s = 00_2$ into a block $B$ with pattern $v = 0110_2$, we have the *two* choices of block patterns $t_1 = 1101_2$ and $t_2 = 1110_2$ instead of the conventional case of just *one*, from which we can choose $t_2 = 1110_2$ to replace the pattern $v = 0110_2$ of the block $B$, resulting in the smaller distortion of just a 1-bit error. Note that if only the choice of $t_1 = 1101_2$ is allowed, then the error will be 3 bits which mean a larger distortion in the replaced block. It is such allowance of multiple choices for block pattern replacement that achieves distortion reduction in the proposed method.

More generally, we group in this study the 16 possible block patterns in a 2×2 block $B$ into distinct sets according to the *entropy* values $E$ of the block patterns, where an entropy value $E$ of a block pattern $P$ is defined as follows:

$$E = -\sum_{k} p_k \log_2 p_k = -p_0 \log_2 p_0 - p_1 \log_2 p_1$$

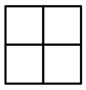
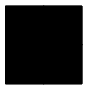
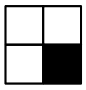
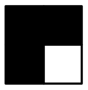
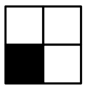
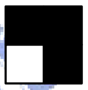
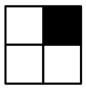
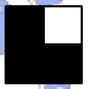
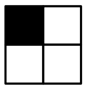
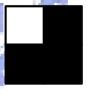
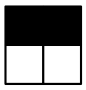
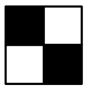
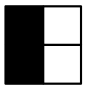
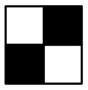
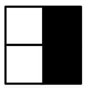
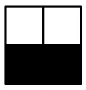
with $p_0$ and $p_1$ being the occurrence probability values of black and white pixels appearing in $P$ computed as

$$p_0 = (\text{number of black pixels in } P)/4;\ p_1 = (\text{number white pixels in } P)/4.$$

A pattern $P$ in a set with a higher entropy value $E$ is presumably more random in its black and white arrangement, and so is more suitable for hiding more secret data without causing a noticeable change. There are three possible entropy values 0, 0.811, and 1 in a 2×2 block by the above definition, so we divide the 16 possible block patterns into three sets. The first set with the entropy value 0 has two distinct block patterns, one being the entirely white block, the other the entirely black. They are denoted as A and F in Table 3.1 and are used to represent the secret data of 1 and 0, respectively. That is, they *encode* the secret data of 0 and 1, respectively.

The second set with the entropy value 0.811 includes eight distinct block patterns, which can be classified into two classes, one class with each pattern including one black pixel and three white ones and the other class with each pattern including three black pixels and one white one. The first class, denoted as B in Table 3.1, includes four block patterns, and we use two block patterns of them to encode the secret value $00_2$, and the other two to encode the secret value $01_2$. When deciding which two patterns should be selected to encode an identical secret value, we adopted the "mismatch reduction criterion" of making the two selected patterns less different in the number of mismatching pixel values when one of the two selected patterns is superimposed on the other. We use the four block patterns of the other class, denoted as E in Table 3.1, to encode the secret values $10_2$ and $11_2$ in a similar way.

Table 3.1 Proposed block pattern encoding table.

| Type | Block pattern | Entropy value | Corresponding binary value | Encoded secret data | Type | Block pattern | Entropy value | Corresponding binary value | Encoded secret data |
|---|---|---|---|---|---|---|---|---|---|
| A |  | 0 | 1111 | 1 | F |  | 0 | 0000 | 0 |
| B1 |  | 0.811 | 1110 | 00 | E1 |  | 0.811 | 0001 | 11 |
| B2 |  | 0.811 | 1101 | 00 | E2 |  | 0.811 | 0010 | 11 |
| B3 |  | 0.811 | 1011 | 01 | E3 |  | 0.811 | 0100 | 10 |
| B4 |  | 0.811 | 0111 | 01 | E4 |  | 0.811 | 1000 | 10 |
| C1 |  | 1 | 0011 | 011 | D1 |  | 1 | 0110 | 100 |
| C2 |  | 1 | 0101 | 011 | D2 |  | 1 | 1001 | 101 |
| C3 |  | 1 | 1010 | 010 | | | | | |
| C4 |  | 1 | 1100 | 010 | | | | | |

The last set with the entropy value 1 has six distinct block patterns. So far we have completed the encoding of all possible one-bit and two-bit secret values with ten patterns. So the remaining six patterns in the 16 ones may be used to encode three-bit secret values. But six patterns are not enough to encode all the eight three-bit secret values, so we can only take care of some of them, following the aforementioned mismatch reduction criterion. In particular, we use two block patterns to encode each of the two 3-bit secret values $011_2$ and $010_2$, and finally, the last two patterns to encode the secret values $100_2$ and $101_2$, respectively. The six patterns are denoted as C1 through C4 and D1 and D2 in Table 3.1.

**B. Sketch of proposed idea of data hiding**

In the proposed data embedding process, the more data we embed in a 2×2 block, the worse the resulting image quality becomes. Therefore, we must control the number of destructed pixels in a block to reduce the resulting image distortion. The idea of the proposed data embedding process is sketched as four major steps in the following, which includes two folds of distortion minimization.

(1) *Dividing the input image into blocks:* We divide the input image into 2×2 blocks with every two neighboring blocks being separated by a 1-pixel-wide line, as shown in Figure 3.2. The 1-pixel-wide band around each 2×2 block is said to be the *neighborhood* of the block.

(2) *Selecting a random list of embeddable blocks for data embedding:* We then use a secret key $K$ as well as a random number generator $f$ to select randomly a sequential list of *embeddable blocks*. A block $B$ is said to be *embeddable* in this study if the following two conditions are satisfied: (a) the neighborhood of $B$ is not entirely black or white, (b) $B$ has not been selected for data embedding yet. The way we adopt to generate the random list of embeddable blocks is as follows: (a) concatenate all blocks obtained in Step (1) above in sequence; (b) use $K$ and $f$ to generate sequentially a random number $f(K)$, divide it by the total number of blocks, and take the remainder as a block number, denoted by $N$; (c) check block $N$ to see if it is an embeddable block; if not, then perform the same process until

an embeddable block is obtained; (d) append the obtained embeddable block to the end of the desired random list; (e) stop the process when a sufficient number of embeddable blocks for data embedding are obtained.

(3) *Using multiple block pattern encoding tables for the first-fold distortion reduction:* We generate all possible block pattern encoding tables and select *an optimal one* for use in the data embedding process, in the sense of introducing *the least distortion*.

(4) *Applying optimal search techniques for the second-fold distortion reduction:* Finally we apply the dynamic programming technique to segment the input message data stream *optimally* into a series of codes and embed them in the input image, according to a *cost function* designed in advance for measuring the degree of the pattern change in each image block. This reduces the resulting distortion further in *a global sense*.

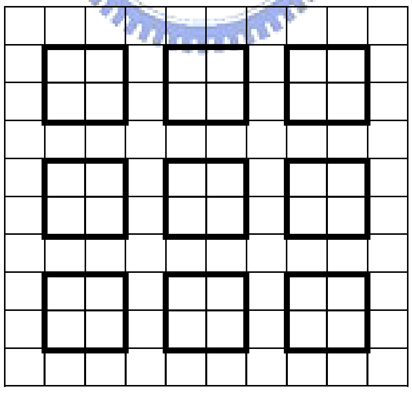


Figure 3.2. Division of input image into 2×2 blocks with separating lines (grids with bold boundaries are 2×2 blocks for data embedding).

## C. Use of Multiple Block Pattern Encoding Tables

The first distortion-reduction technique using multiple block pattern encoding tables, as mentioned previously in the third major step of the proposed data; embedding process, is based on the idea that a single encoding table will not be suitable for every binary image in the embedding process. If a binary image is destroyed very seriously in the data embedding process using Table 3.1, it will be necessary to use another table with other combinations of block patterns and encoded hidden data. For example, assume that a binary secret value $v$ = $101_2$ is to be embedded into a sequence of three randomly selected image blockswith patterns 0000,0100, and 1111 by Table 3.1. The data embedding process using Table 3.1, as illustrated in Figure 3.3(a), will select optimally the block pattern type D2 = 1001, which encodes the three-bit secret value $v$ = $101_2$, to replace the first selected block with pattern 0000, resulting in reversing two bits. However, if we replace the encoded secret data of type A in Table 3.1 with those of type F, and replace those of all of types B1 through B4 with those of all of types E1 through E4, respectively, then we will get a new block pattern encoding table and the use of it to hide the secret value $v$ = $101_2$ will result in no bit reversing because here we can, as illustrated in Figure 3.3(b), select in sequence optimally the new pattern type F = 0000 (encoding the secret data of $1_2$) and the new pattern type E3 = 0100 (encoding the secret data of $01_2$) to encode together the secret data $v$ = $101_2$. This means that adaptive table generations and selections for use in data embedding help distortion reduction indeed. More generally, by enumerating all possible ways for exchanging the encoded secret data of certain types in Table 3.1 with those of the other types, we can get 128 distinct block pattern encoding tables for selection in the data embedding process to minimize the distortion.

## D. Proposed Distortion-Minimizing Cost Function and Search Techniques for Optimal Solutions

The cost function proposed in this study for use in the proposed data embedding process to minimize image distortion is *the total number of reversed bits* in the resulting stego-image. In Table 3.1, block patterns can be used to encode one, two, or three secret bits. Correspondingly, we hide a binary secret value *v* by embedding the first one, two, or three bits in the prefix of *v* into a block.

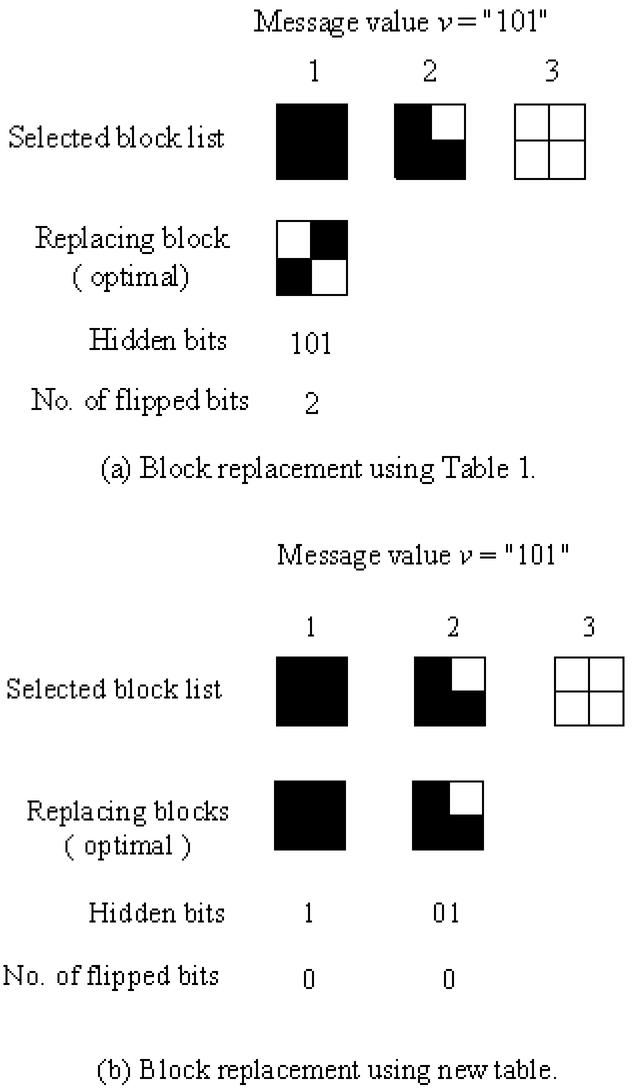


Figure 3.3. An example of proposed data embedding process

To determine how many bits should be embedded, we may calculate first the cost function value for each of the three cases, and then replace the currently selected block with the block pattern which corresponds to the case with the minimum cost function value. This method provides a quick way for data embedding. However, it is actually a *greedy search* and *not* an optimal solution.

To see this, for example, for the previously-mentioned example in which the secret value $v$ of $101_2$ is embedded in three selected blocks with patterns 0000, 0100, and 1111 by Table 3.1, by the above-mentioned greedy algorithm we first replace the block with pattern 0000 by the block pattern E3 = 0100 to embed two bits 10. The computed cost function value is 1 because a bit is reversed here. This cost is a local minimal one. Next, we replace the block with pattern 0100 by the block pattern A= 1111 to embed the last bit 1of $v$, and get a local minimal cost value 3. The total cost value is 4. Now, if we do not use the greedy algorithm from the beginning, and replace instead the first block with pattern 0000 by the block pattern D2 = 1001 to embed three bits 101 directly, then the total cost value will be reduced to 2 which is smaller than the previous total cost 4. This shows that there indeed exist at least one solution better than that found by the greedy method. Figure 3.4 illustrates the data embedding process for this example. This is also true for many other examples, as found by this study. And so the search of an optimal solution is meaningful, for which the proposed method is dynamic programming.

In the proposed dynamic programming algorithm (abbreviated as DP in the sequel), certain edit distances are defined to minimize the cost function, as described in the following. Assume that the input secret data value to be hidden is in the form of an $n$-bit string $S_1$ with $S_1[i]$ denoting its $i$th bit. Also, let the randomly selected blocks for embedding the secret value be expressed as a list in the form of another string $S_2$ with $S_2[i]$ denoting its $i$th block. Only one type of edit operation, namely, *replacement*, is needed for use in the proposed algorithm to represent the image block replacement operations involving $S_1$ and $S_2$ in the proposed secret data embedding process. The edit distance of $S_1$ and $S_2$ is defined, according to the previous discussions, as the minimum cost to transform $S_2$ into $S_1$ by edit operations according to an optimal block pattern encoding table used in the data embedding process. Let C be a two-dimensional *cost matrix* with its element C[$i$, $j$] denoting the minimum cost to transform a substring of $S_2$ with bits $S_2[j]$ through $S_2[n]$ into a substring of $S_1$ with bits $S_1[i]$ through $S_1[n]$. Then C[1, 1] is the value of the minimum cost to transform $S_2$ into $S_1$. Also, let RC be a three-dimensional *replacement cost matrix* with its element RC($L$, $i$, $j$) denoting the cost for replacing the ($j$+1)th block in $S_2$, denoted by $S_2[j]$, with the block patterns encoding the initial $L$ bits of a substring of $S_1$ with bits $S_1[i]$ through $S_1[n-1]$, where $L$ may be 1, 2, or 3. By the above definitions, the value C[$i$, $j$] is recursively just the value of the minimum of all possible values of RC($L$, $i$, $j$)+C[$i$+$L$, $j$+1], where $L$ = 0, 1, and 3. And because of this, the size of C must be expanded to $n+2 \times n$. Furthermore, those elements of C with indices larger than $n-1$ should be given certain values (0 or $\infty$) to specify their correspondences to "*boundary conditions*". Then, according to the dynamic programming technique, the minimum edit distance may be computed by the following recursive formulas:

*set initial values*

$$C[n, j] = 0, \quad j = 0, 1, 2, \ldots n,$$

$$C[n+1, j] = 0, \quad j = 0, 1, 2, \ldots n,$$

$$C[n+2, j] = 0, \quad j = 0, 1, 2, \ldots n,$$

$$C[i, n] = \infty, \quad i = 0, 1, 2, \ldots n-1,$$

$$C[i, j] = \infty, \quad i, j = 0, 1, 2, \ldots n-1;$$

*and then for all* $i = 0, 1, \ldots n-1, j = 0, 1, \ldots n-1$, *compute*

$$C[i, j] = \min\{RC(1, i, j)+C[i+1, j+1], RC(2, i, j)+C[i+2, j+1], RC(3, i, j)+C[i+3, j+1]\}.$$

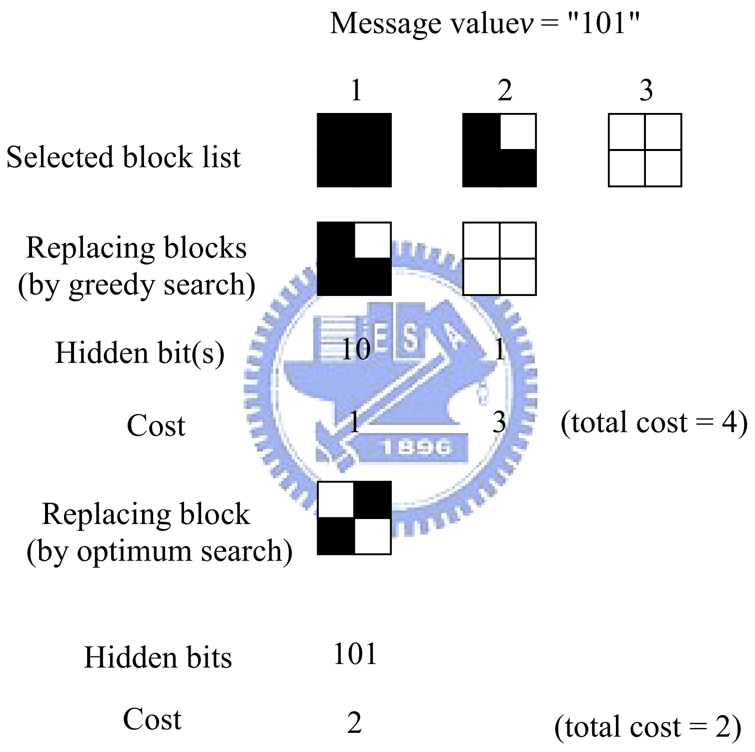


Figure 3.4. An example of proposed data embedding process.

**Algorithm 3.1** ***Computing minimum cost for minimizing distortion in data embedding process by DP*.**

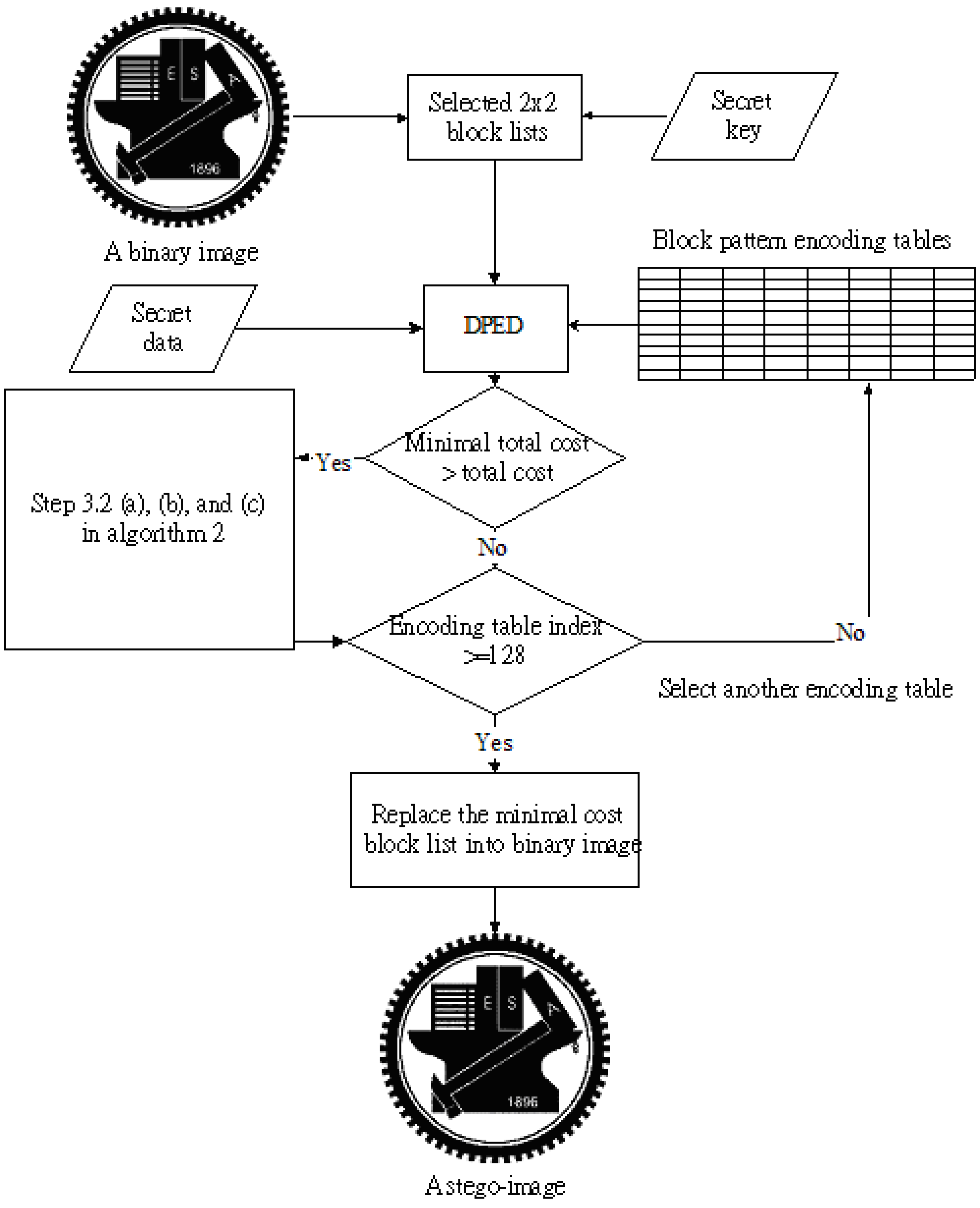
***Input***: an $n$-bit secret code string $S_1$, a string of $n$ randomly selected blocks $S_2$, a block pattern encoding table, a two-dimensional cost matrix $C[i, j]$, for $i = 0, 1, \ldots, n+2$, $j = 0, 1, \ldots, n$ with the initial values specified in the above recursive formulas, a two-dimensional *index matrix* I[$i$, $j$], for $i$ = 0, 1, … $n$–1, $j$ = 0, 1, … $n$–1, for recording the relative indices in the block pattern encoding table after calculating C[$i$, $j$], and a two-dimensional matrix N[$i$, $j$], for $i$ = 0, 1, … $n$-1, $j$ = 0, 1, … $n$–1, for recording the relative next step after calculating C[$i$, $j$] with each element given an initial value of minus one.

***Output***: C[$i$, $j$], the minimum cost to change the substring $S_2$[$j$] through $S_2$[$n$] into $S_1$[$i$ … $n$], I[$i$, $j$], and N[$i$, $j$].

***Steps***:

1. If C[$i$, $j$] is equal to an infinitive value ∞, continue the next step; else go to Step 4.
2. Calculate three temporary cost functions T[1], T[2], and T[3], record every next step and the corresponding value as the indices index1, index2, and index3 of the block pattern encoding table which is used in calculating the minimal cost in RC(1, $i$, $j$), RC(2, $i$, $j$), and RC(3, $i$, $j$), respectively, in the following way:
   2.1 T[1] = RC(1, $i$, $j$) + C($i$+1, $j$+1), next_step[1] = $i$+1, and acquire index1.
   2.2 T[2] = RC(2, $i$, $j$) + C($i$+2, $j$+1), next_step[2] = $i$+2, and acquire index2.
   2.3 T[3] = RC(3, $i$, $j$) + C($i$+3, $j$+1), next_step[3] = $i$+3. and acquire index3.
3. Take C($i$, $j$) to be the minimum of the three temporary cost functions, record the corresponding relative next step in N[$i$, $j$], and record the relative index in the block pattern encoding table in I[$i$, $j$]
4. Return C[$i$, $j$].

Because every next step and the used indices of the block pattern encoding table have been recorded, we can reconstruct the embedding process easily. The space complexity and time complexity are both O($n^2$) for the DP. Now, the proposed data embedding process is described in detail as an algorithm in the following. Figure 3.5 illustrates a flowchart of the data embedding process.



A binary image
Selected 2x2 block lists
Secret key
Block pattern encoding tables
Secret data
DPED
Yes
Minimal total cost > total cost
Step 3.2 (a), (b), and (c) in algorithm 2
No
Encoding table index >=128
No
Select another encoding table
Yes
Replace the minimal cost block list into binary image
A stego-image

Figure 3.5 Flowchart of the proposed data embedding process.

**Algorithm 3.2** ***Data embedding process using block pattern encoding tables and DP.***

***Input***: a binary image *I*, a secret data string $S_1$ with *n* bits, a secret key *K* as well as a random number generator *f*, and 128 block pattern encoding tables.

***Output***: a stego-image *S*, an optimal block pattern encoding table *B*, a length of block list $L$, and a minimal total cost $C_{min}$.

***Steps***:

1. Get a list of embeddable 2×2 blocks from the input image $I$ in a way as described previously.
2. Set the value of the desired minimal total cost $C_{min}$ to be infinitive.
3. For each block pattern encoding table $B_i$ among the 128 possible ones, perform the following operations.

   3.0 Calculate a total cost $C_i$ using $B_i$ and the DP.

   3.1 If $C_{min}$ is larger than $C_i$, perform the following operations.

   a. Take $C_i$ as the minimal total cost $C_{min}$.

   b. Set the optimal block pattern encoding table $B$ as $B_i$.

   c. Sequentially, record every index obtained from Step 3.1 according to the next-step matrix N and index matrix I, until an element of N is equal to −1. Meanwhile, calculate $L$, the length of the block list.
4. Replace the minimal-cost block list with the selected block list of binary image $I$ by the recorded index sequence of block pattern encoding table $B$ and the length of the block list $L$.
5. Take the final result as the desired stego-image $S$.

## E. Data recovery process

The goal of the proposed data recovery process is to extract the embedded bit stream from a stego-image. In the proposed data extraction process, Table 3.1 is first simplified as an extraction table as shown in Table 3.2. It is easier to use this table to finish the extraction process, as follows. Figure 3.6 illustrates a flowchart of the data recovery process.

**Algorithm3.3** ***Secret data recovery process***.

***Input***: a stego-image *I*' presumably including a secret bit stream *S*; and the secret key *K* as well as the random number generator *f* used in the data embedding process; the index table *B* **t**hat points outs which table is used in the embedding process, and the length of the block list *L*.

***Output***: the secret bit stream *S* or a report of failure to recover the secret.

***Steps***:

1. Extract a list of 2×2 embeddable blocks from the stego-image *I*' by the secret key *K* , the random number generator *f*, and the length *L*.
2. For each 2×2 embeddable block in *I*', compute the corresponding block pattern *P*, and look *P* up in the table *B* to decode the data bits embedded in the block.
3. Take all the extracted data bits in sequence as the desired secret bit stream *S*.

Table 3.2 An extraction table (table index *B*=0).

| Corresponding binary value of block pattern | Encoded secret data | Corresponding binary value of block pattern | Encoded secret data |
|---|---|---|---|
| 1111 | 1 | 0111 | 01 |
| 1110 | 00 | 0110 | 100 |
| 1101 | 00 | 0101 | 011 |
| 1100 | 010 | 0100 | 10 |
| 1011 | 01 | 0011 | 011 |
| 1010 | 010 | 0010 | 11 |
| 1001 | 101 | 0001 | 11 |
| 1000 | 10 | 0000 | 0 |

## 3.3 Experimental Results

Some experimental results of applying the proposed method are shown in Figures 3.7, 3.8, and 3.9. Figures 3.7(a), 3.8(a) and 3.9(a) show three binary cover images of the sizes 687×534, 512×512, and 2320×3408, respectively. Two streams of message data were generated by a random fashion. One is a stream of 2432 bits, which was embedded into each of the binary images shown in Figures 3.7(a), and 3.9(a). The other is 992-bit long, which was embedded into the binary image shown in Figure 3.8(a). The stego-images obtained by embedding the message data using the greedy search algorithm and the optimal encoding table among the 128 ones are shown in Figures 3.7(b), 3.8(b) and 3.9(b), respectively. And the stego-images after embedding the message using the DPA and the optimal encoding table among the 128 ones are shown in Figures 3.7(c), 3.8(c) and 3.9(c), respectively. Figure 3.8(d) shows the difference between Figures 3.8(a) and 3.8(c) in terms of white pixels. And Figure 3.9(d) show similarly enlarged version of parts of the differences between Figures 3.9(a) and 3.9(c), for better inspection effects.

Note that the original input images are included in Figures 3.7(d), 3.8(d) and 3.9(d) in gray values as the backgrounds to show more clearly the difference spots. Tables 3.3 shows the statistical data of the stego-images of Figures 3.7(a), 3.8(a), and 3.9(a) for the proposed algorithms, in which we list the numbers of the selected table index, the used blocks, the minimum cost values and the length of secret data. The minimum cost values show that the DP is the best, the greedy algorithm using an optimal encoding table among the 128 possible ones is the next, and the greedy algorithm using just an encoding table is the worst. For other images, similar results can be observed. For the images shown here, the average number of secret data embedded in a block, using the DP algorithm, is about 1.7 bits. And the distortion rate computed as the ratio of the number of reversed bits to the length of the secret data, using the DP algorithm, is about in the range from 0.37 to 0.39, which is smaller than 0.5 yielded by most existing data hiding methods for binary images.

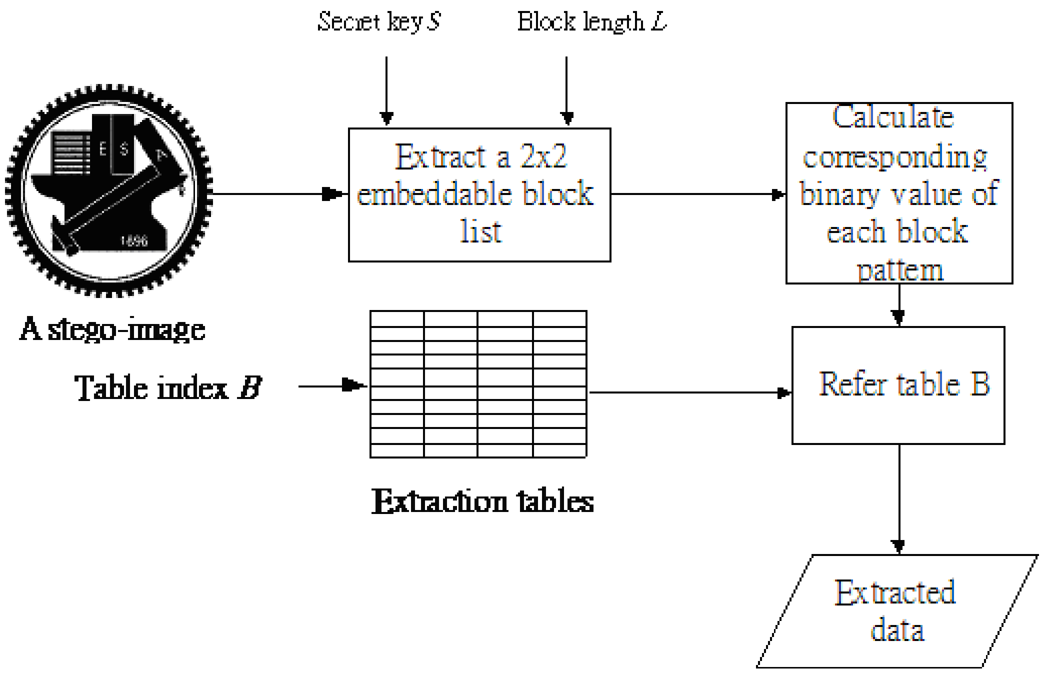
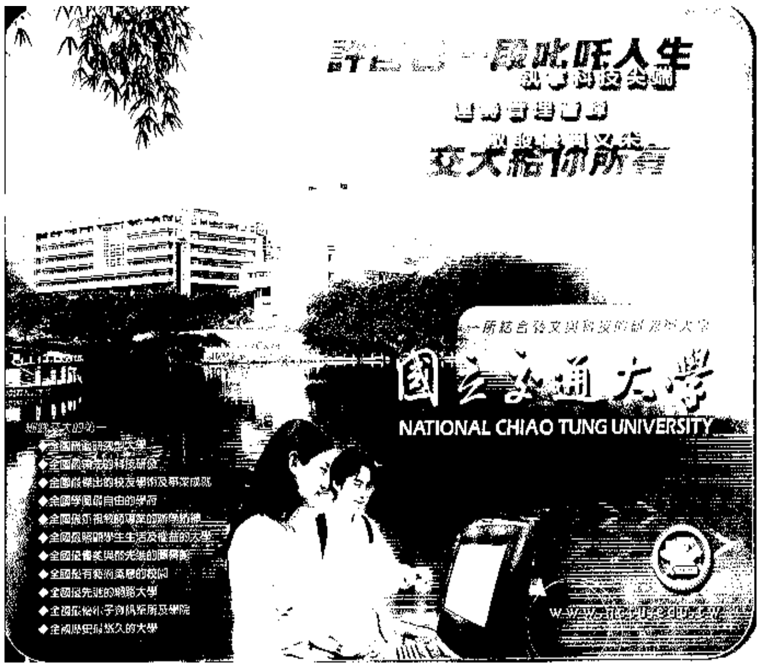
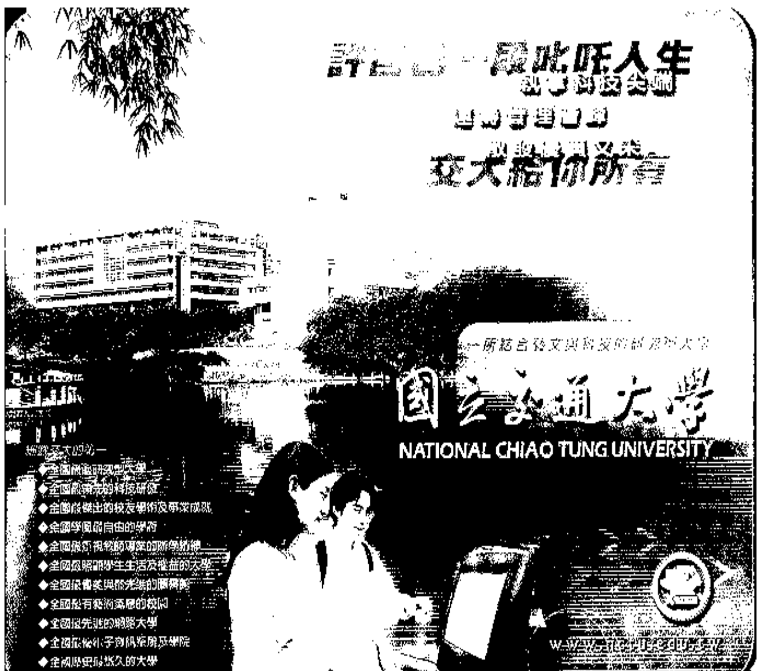


Figure 3.6 Flowchart of the proposed extraction process.

Furthermore, we tested 17 images that are obtained from an image database of the USC, and the results are listed in Table 3.4. As shown there, the average number of message data embedded in a block, using the DPA, is about 1.9388 bits. And the average distortion rate using the DPA is 35.53%, which is smaller than 50% yielded by most existing data hiding methods for binary images.

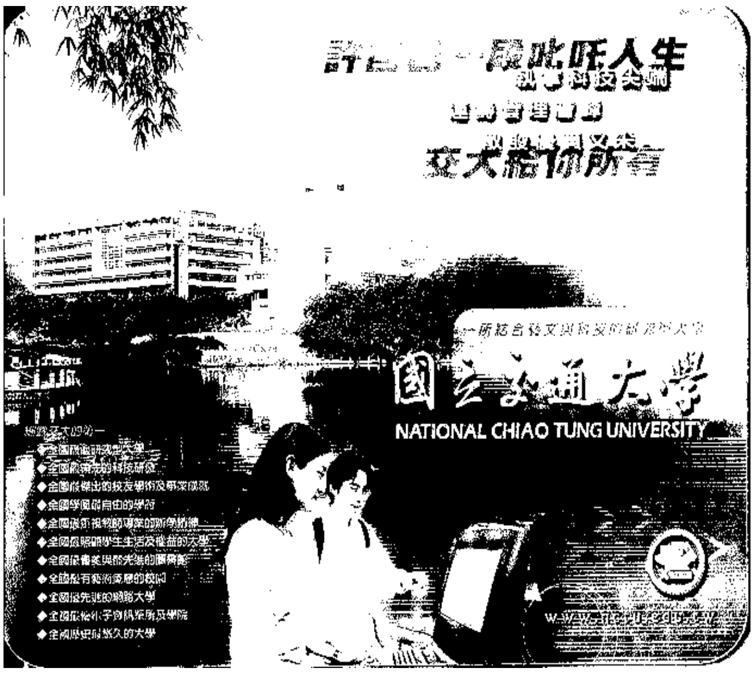


(a)

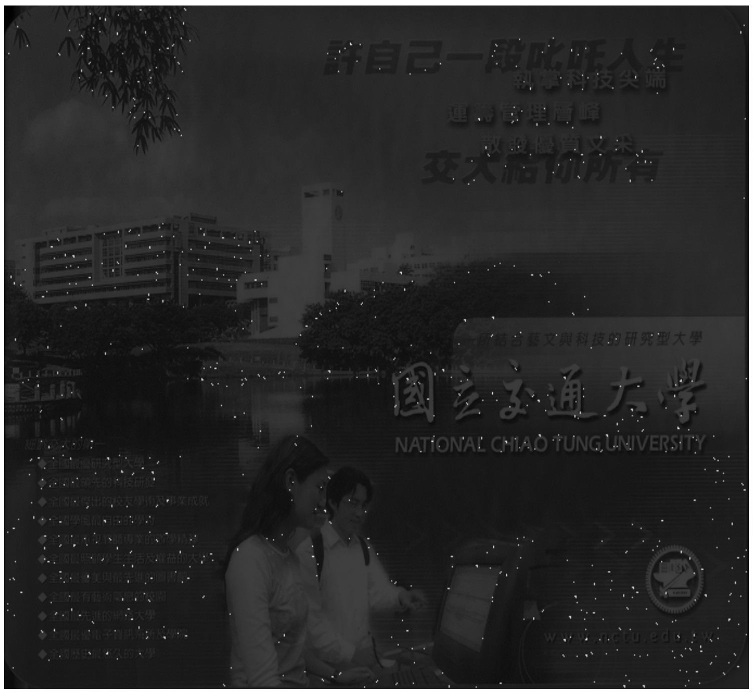


(b)

Figure 3.7 Input binary images, output stego-images with secret data, and the differences. (a) Binary image "NCTU". (b) Stego-images after embedding secret data using greedy algorithm. (c) Stego-images after embedding secret data using DP algorithm. (d) The difference image after embedding secret data.

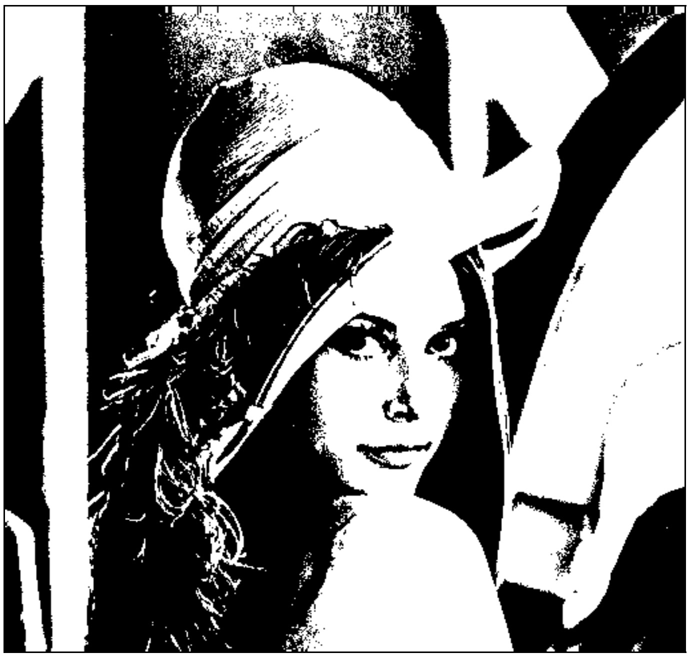


(c)

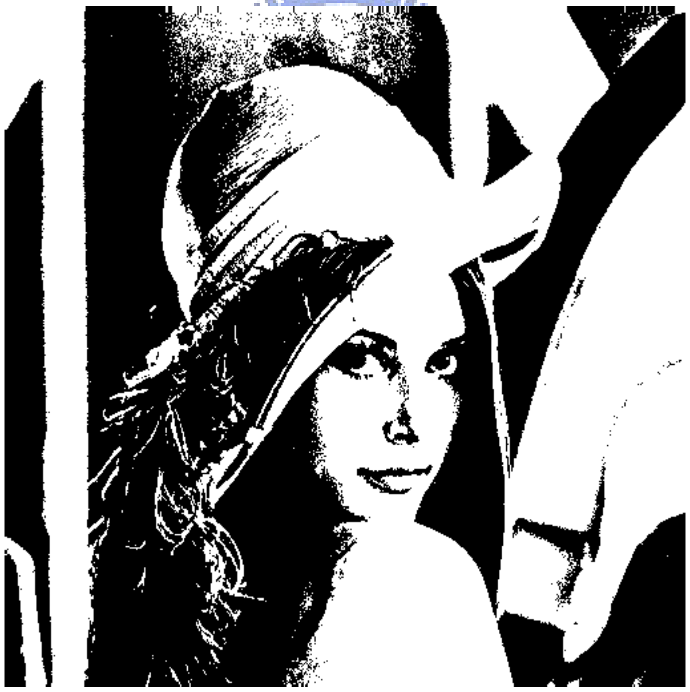


(d)

Figure 3.7 Input binary images, output stego-images with secret data, and the differences. (a) Binary image "NCTU". (b) Stego-images after embedding secret data using greedy algorithm. (c) Stego-images after embedding secret data using DP algorithm. (d) The difference image after embedding secret data (continued).

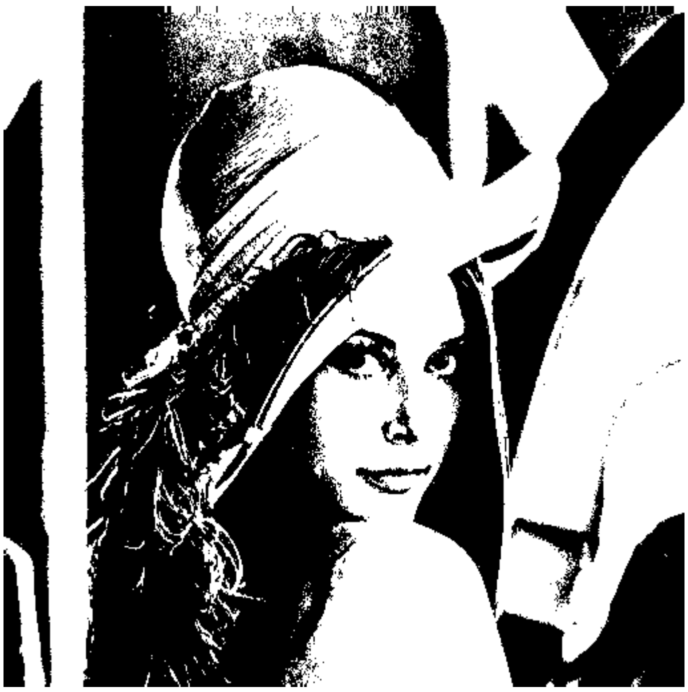


(a)

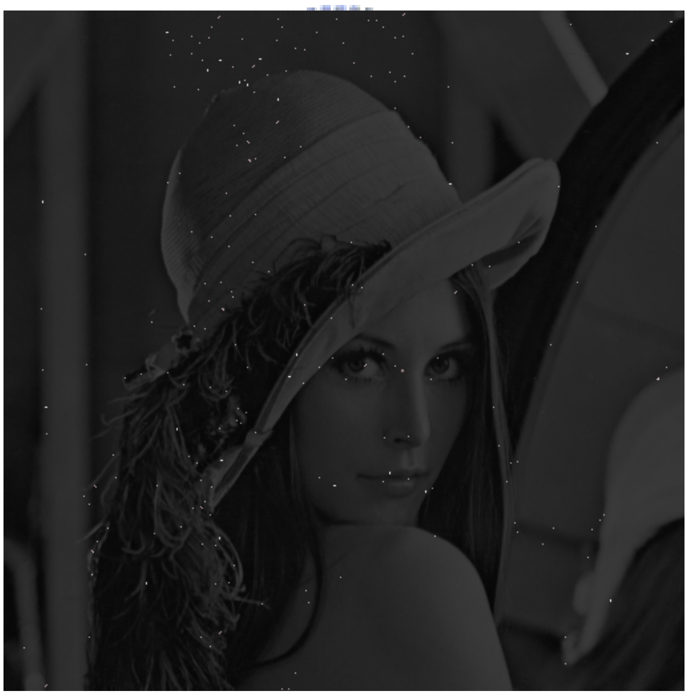


(b)

Figure 3.8 Input binary images, stego-images with secret data, and differences. (a) Binary image “Lena”. (b) Stego-images after embedding secret data using greedy algorithm. (c) Stego-images after embedding secret data using DP algorithm. (d) Difference image after embedding secret data.

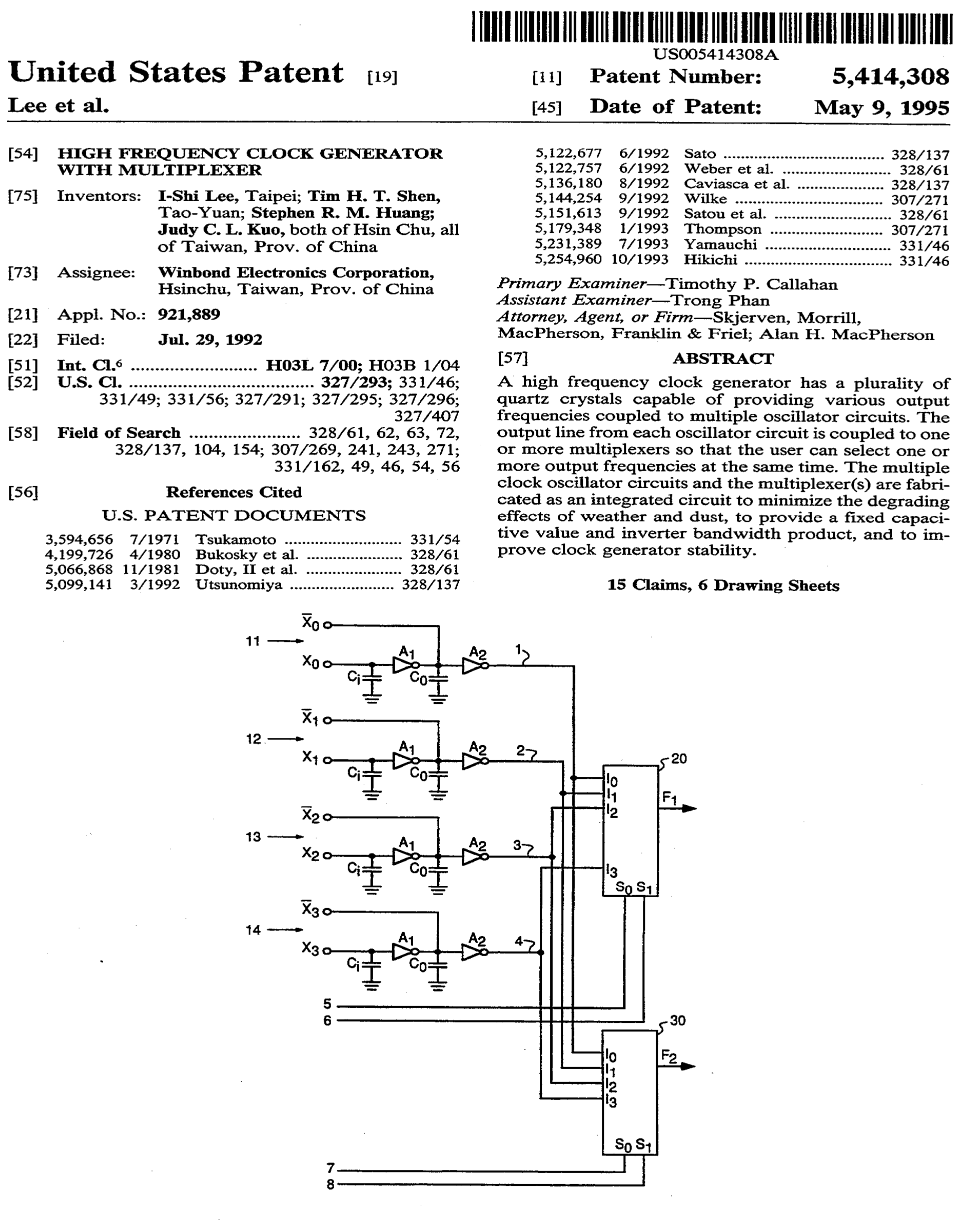


(c)



(d)

Figure 3.8 Input binary images, stego-images with secret data, and differences. (a) Binary image "Lena". (b) Stego-images after embedding secret data using greedy algorithm. (c) Stego-images after embedding secret data using DP algorithm. (d) Difference image after embedding secret data (continued).



US005414308A

United States Patent [19]

Lee et al.

[11] Patent Number: 5,414,308

[45] Date of Patent: May 9, 1995

[54] HIGH FREQUENCY CLOCK GENERATOR WITH MULTIPLEXER

[75] Inventors: I-Shi Lee, Taipei; Tim H. T. Shen, Tao-Yuan; Stephen R. M. Huang; Judy C. L. Kuo, both of Hsin Chu, all of Taiwan, Prov. of China

[73] Assignee: Winbond Electronics Corporation, Hsinchu, Taiwan, Prov. of China

[21] Appl. No.: 921,889

[22] Filed: Jul. 29, 1992

[51] Int. Cl.$^6$ .......................... H03L 7/00; H03B 1/04

[52] U.S. Cl. ...................................... 327/293; 331/46; 331/49; 331/56; 327/291; 327/295; 327/296; 327/407

[58] Field of Search ....................... 328/61, 62, 63, 72, 328/137, 104, 154; 307/269, 241, 243, 271; 331/162, 49, 46, 54, 56

[56] References Cited

U.S. PATENT DOCUMENTS

| | | | |
|---|---|---|---|
| 3,594,656 | 7/1971 | Tsukamoto | 331/54 |
| 4,199,726 | 4/1980 | Bukosky et al. | 328/61 |
| 5,066,868 | 11/1981 | Doty, II et al. | 328/61 |
| 5,099,141 | 3/1992 | Utsunomiya | 328/137 |
| 5,122,677 | 6/1992 | Sato | 328/137 |
| 5,122,757 | 6/1992 | Weber et al. | 328/61 |
| 5,136,180 | 8/1992 | Caviasca et al. | 328/137 |
| 5,144,254 | 9/1992 | Wilke | 307/271 |
| 5,151,613 | 9/1992 | Satou et al. | 328/61 |
| 5,179,348 | 1/1993 | Thompson | 307/271 |
| 5,231,389 | 7/1993 | Yamauchi | 331/46 |
| 5,254,960 | 10/1993 | Hikichi | 331/46 |

*Primary Examiner*—Timothy P. Callahan
*Assistant Examiner*—Trong Phan
*Attorney, Agent, or Firm*—Skjerven, Morrill, MacPherson, Franklin & Friel; Alan H. MacPherson

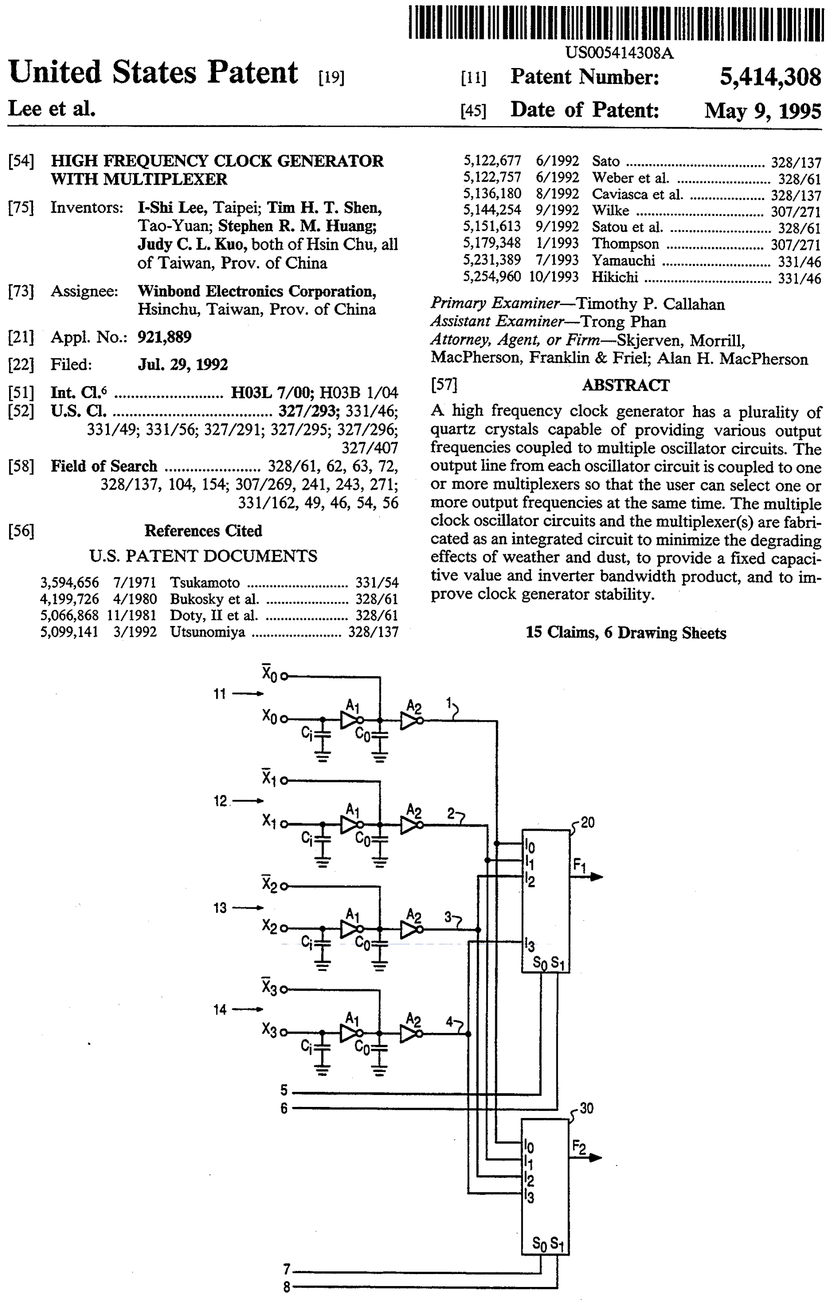
[57] ABSTRACT

A high frequency clock generator has a plurality of quartz crystals capable of providing various output frequencies coupled to multiple oscillator circuits. The output line from each oscillator circuit is coupled to one or more multiplexers so that the user can select one or more output frequencies at the same time. The multiple clock oscillator circuits and the multiplexer(s) are fabricated as an integrated circuit to minimize the degrading effects of weather and dust, to provide a fixed capacitive value and inverter bandwidth product, and to improve clock generator stability.

15 Claims, 6 Drawing Sheets

(a)

Figure 3.9 Input binary images, output stego-images with secret data, and the differences. (a) Binary image "Patent". (b) Stego-images after embedding secret data using greedy algorithm. (c) Stego-images after embedding secret data using DP algorithm. (d) An enlarged part of difference image between (a) and (c) in which the white spots are difference pixels.



US005414308A

**United States Patent** [19]

**Lee et al.**

[11] **Patent Number: 5,414,308**

[45] **Date of Patent: May 9, 1995**

[54] **HIGH FREQUENCY CLOCK GENERATOR WITH MULTIPLEXER**

[75] Inventors: **I-Shi Lee**, Taipei; **Tim H. T. Shen**, Tao-Yuan; **Stephen R. M. Huang**; **Judy C. L. Kuo**, both of Hsin Chu, all of Taiwan, Prov. of China

[73] Assignee: **Winbond Electronics Corporation**, Hsinchu, Taiwan, Prov. of China

[21] Appl. No.: **921,889**

[22] Filed: **Jul. 29, 1992**

[51] **Int. Cl.**$^6$ .......................... **H03L 7/00**; H03B 1/04

[52] **U.S. Cl.** ...................................... **327/293**; 331/46; 331/49; 331/56; 327/291; 327/295; 327/296; 327/407

[58] **Field of Search** ........................ 328/61, 62, 63, 72, 328/137, 104, 154; 307/269, 241, 243, 271; 331/162, 49, 46, 54, 56

[56] **References Cited**

U.S. PATENT DOCUMENTS

| | | | |
|---|---|---|---|
| 3,594,656 | 7/1971 | Tsukamoto | 331/54 |
| 4,199,726 | 4/1980 | Bukosky et al. | 328/61 |
| 5,066,868 | 11/1981 | Doty, II et al. | 328/61 |
| 5,099,141 | 3/1992 | Utsunomiya | 328/137 |
| 5,122,677 | 6/1992 | Sato | 328/137 |
| 5,122,757 | 6/1992 | Weber et al. | 328/61 |
| 5,136,180 | 8/1992 | Caviasca et al. | 328/137 |
| 5,144,254 | 9/1992 | Wilke | 307/271 |
| 5,151,613 | 9/1992 | Satou et al. | 328/61 |
| 5,179,348 | 1/1993 | Thompson | 307/271 |
| 5,231,389 | 7/1993 | Yamauchi | 331/46 |
| 5,254,960 | 10/1993 | Hikichi | 331/46 |

*Primary Examiner*—Timothy P. Callahan
*Assistant Examiner*—Trong Phan
*Attorney, Agent, or Firm*—Skjerven, Morrill, MacPherson, Franklin & Friel; Alan H. MacPherson

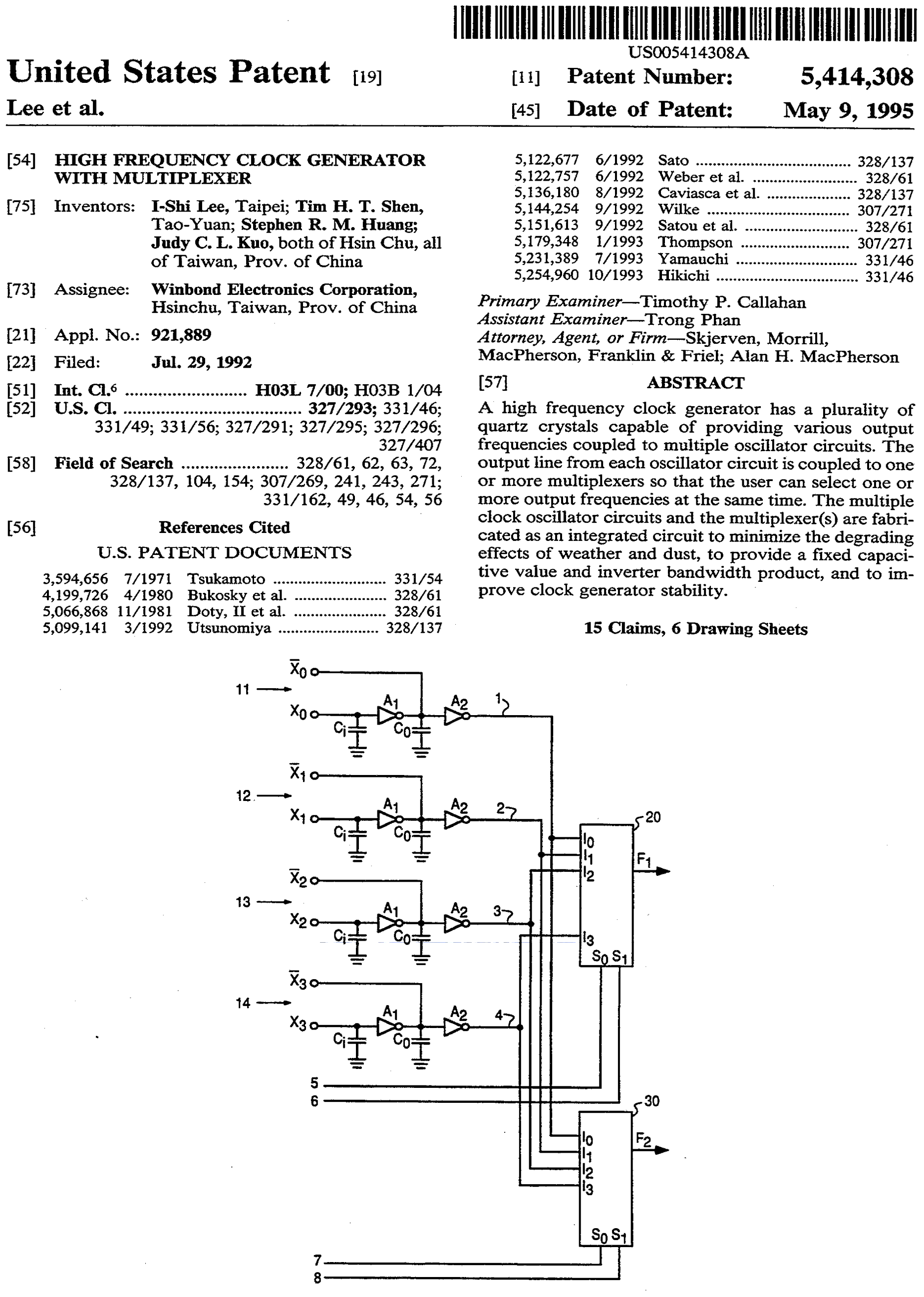
[57] **ABSTRACT**

A high frequency clock generator has a plurality of quartz crystals capable of providing various output frequencies coupled to multiple oscillator circuits. The output line from each oscillator circuit is coupled to one or more multiplexers so that the user can select one or more output frequencies at the same time. The multiple clock oscillator circuits and the multiplexer(s) are fabricated as an integrated circuit to minimize the degrading effects of weather and dust, to provide a fixed capacitive value and inverter bandwidth product, and to improve clock generator stability.

**15 Claims, 6 Drawing Sheets**

(b)

Figure 3.9 Input binary images, output stego-images with secret data, and the differences. (a) Binary image "Patent". (b) Stego-images after embedding secret data using greedy algorithm. (c) Stego-images after embedding secret data using DP algorithm. (d) An enlarged part of difference image between (a) and (c) in which the white spots are difference pixels (continued).



US005414308A

**United States Patent** [19]
**Lee et al.**

[11] **Patent Number: 5,414,308**
[45] **Date of Patent: May 9, 1995**

[54] **HIGH FREQUENCY CLOCK GENERATOR WITH MULTIPLEXER**

[75] Inventors: **I-Shi Lee**, Taipei; **Tim H. T. Shen**, Tao-Yuan; **Stephen R. M. Huang**; **Judy C. L. Kuo**, both of Hsin Chu, all of Taiwan, Prov. of China

[73] Assignee: **Winbond Electronics Corporation**, Hsinchu, Taiwan, Prov. of China

[21] Appl. No.: **921,889**

[22] Filed: **Jul. 29, 1992**

[51] **Int. Cl.**$^6$ .......................... **H03L 7/00**; H03B 1/04

[52] **U.S. Cl.** ...................................... **327/293**; 331/46; 331/49; 331/56; 327/291; 327/295; 327/296; 327/407

[58] **Field of Search** ........................ 328/61, 62, 63, 72, 328/137, 104, 154; 307/269, 241, 243, 271; 331/162, 49, 46, 54, 56

[56] **References Cited**

U.S. PATENT DOCUMENTS

| | | | |
|---|---|---|---|
| 3,594,656 | 7/1971 | Tsukamoto | 331/54 |
| 4,199,726 | 4/1980 | Bukosky et al. | 328/61 |
| 5,066,868 | 11/1981 | Doty, II et al. | 328/61 |
| 5,099,141 | 3/1992 | Utsunomiya | 328/137 |
| 5,122,677 | 6/1992 | Sato | 328/137 |
| 5,122,757 | 6/1992 | Weber et al. | 328/61 |
| 5,136,180 | 8/1992 | Caviasca et al. | 328/137 |
| 5,144,254 | 9/1992 | Wilke | 307/271 |
| 5,151,613 | 9/1992 | Satou et al. | 328/61 |
| 5,179,348 | 1/1993 | Thompson | 307/271 |
| 5,231,389 | 7/1993 | Yamauchi | 331/46 |
| 5,254,960 | 10/1993 | Hikichi | 331/46 |

*Primary Examiner*—Timothy P. Callahan
*Assistant Examiner*—Trong Phan
*Attorney, Agent, or Firm*—Skjerven, Morrill, MacPherson, Franklin & Friel; Alan H. MacPherson

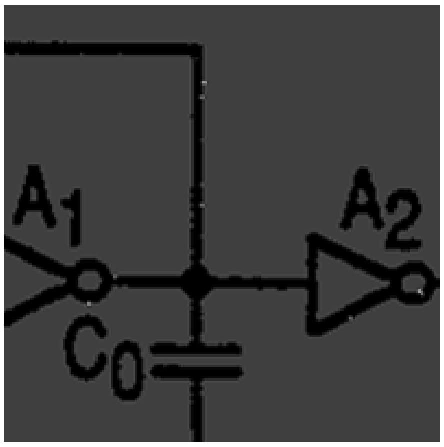
[57] **ABSTRACT**

A high frequency clock generator has a plurality of quartz crystals capable of providing various output frequencies coupled to multiple oscillator circuits. The output line from each oscillator circuit is coupled to one or more multiplexers so that the user can select one or more output frequencies at the same time. The multiple clock oscillator circuits and the multiplexer(s) are fabricated as an integrated circuit to minimize the degrading effects of weather and dust, to provide a fixed capacitive value and inverter bandwidth product, and to improve clock generator stability.

**15 Claims, 6 Drawing Sheets**

(c)

Figure 3.9 Input binary images, output stego-images with secret data, and the differences. (a) Binary image "Patent". (b) Stego-images after embedding secret data using greedy algorithm. (c) Stego-images after embedding secret data using DP algorithm. (d) An enlarged part of difference image between (a) and (c) in which the white spots are difference pixels (continued).



(d)

Figure 3.9 Input binary images, output stego-images with secret data, and the differences. (a) Binary image “Patent”. (b) Stego-images after embedding secret data using greedy algorithm. (c) Stego-images after embedding secret data using DP algorithm. (d) An enlarged part of difference image between (a) and (c) in which the white spots are difference pixels (continued).

## 3.4 Concluding Remarks

A novel optimal method for hiding secret data into binary images with a distortion minimization effect and a larger data embedding capability has been proposed. An optimal block pattern encoding table is chosen from 128 alternative ones for use in the proposed data embedding process to minimize distortion in the stego-image. The method can minimize further the distortion using the dynamic programming technique and can embed up to three bits in a 2×2 image block. Therefore, by our method, not only more data can be embedded in a binary image, but also the distortion rate of the stego-image can be effectively reduced.

The proposed method is based on the use of 2×2 blocks in data embedding process. It may be extended by processing larger-sized blocks, because when the block size is larger, the number of the block patterns which can be selected to encode a certain secret value becomes larger as well, resulting possibly in a greater reduction of image distortion.

Table 3.3 Statistics of three stego-images for proposed algorithms.

| stego-image | Algorithm | Table index | Used blocks | Cost value | Length of secret data |
|---|---|---|---|---|---|
| NCTU | Greedy Algorithm (using just a fixed encoding table) | 0 | 1528 | 1153 | 2432 |
| | Greedy Algorithm (using the optimal one among 128 encoding tables) | 16 | 1526 | 1115 | |
| | DP | 26 | 1418 | 954 | |
| Lena | Greedy Algorithm (using just a fixed encoding table) | 0 | 621 | 431 | 992 |
| | Greedy Algorithm (using the optimal one among 128 encoding tables) | 30 | 637 | 401 | |
| | DP | 41 | 582 | 369 | |
| Patent | Greedy Algorithm (using just a fixed encoding table) | 0 | 1439 | 1037 | 2432 |
| | Greedy Algorithm (using the optimal one among 128 encoding tables) | 70 | 1530 | 1007 | |
| | DP | 24 | 1433 | 924 | |

Other future works may be directed to designing a better cost function for the human visual system, constraining certain conditions for the cost function to find a better image quality, and finding a better encoding table for replacing selected blocks to reduce stego-image distortion further.

Table 3.4 Statistics of 19 stego-images processed by proposed DPA.

| Image No. of USC | Table number | No. of used blocks | Cost value | Message data length | Distortion rate | Embedding density |
|---|---|---|---|---|---|---|
| 4.2.03 | 22 | 482 | 369 | 992 | 0.37 | 2.06 |
| 4.2.06 | 66 | 516 | 351 | 992 | 0.35 | 1.92 |
| 5.2.08 | 57 | 500 | 356 | 992 | 0.36 | 1.98 |
| 5.2.09 | 41 | 527 | 352 | 992 | 0.35 | 1.88 |
| 5.2.10 | 6 | 523 | 336 | 992 | 0.34 | 1.90 |
| 7.1.01 | 24 | 508 | 361 | 992 | 0.36 | 1.95 |
| 7.1.03 | 70 | 521 | 346 | 992 | 0.35 | 1.90 |
| 7.1.04 | 70 | 507 | 362 | 992 | 0.36 | 1.96 |
| 7.1.05 | 6 | 530 | 343 | 992 | 0.35 | 1.87 |
| 7.1.06 | 8 | 525 | 349 | 992 | 0.35 | 1.89 |
| 7.1.07 | 18 | 533 | 345 | 992 | 0.35 | 1.86 |
| 7.1.08 | 66 | 508 | 353 | 992 | 0.36 | 1.95 |
| 7.1.09 | 57 | 507 | 353 | 992 | 0.36 | 1.96 |
| 7.1.10 | 18 | 514 | 352 | 992 | 0.35 | 1.93 |
| boat.512 | 57 | 507 | 360 | 992 | 0.36 | 1.96 |
| elain.512 | 6 | 499 | 358 | 992 | 0.36 | 1.99 |
| house | 6 | 495 | 355 | 992 | 0.36 | 2.00 |
| **average** | | | | | **0.3553** | **1.9388** |

# Chapter 4
# Data Hiding in Grayscale Images by Dynamic Programming Based on A Human Visual Model

## 4.1 Idea of Proposed Method

Eight bits represent a pixel's intensity in a grayscale image. The *bit plane* formed by the same bit of each pixel in the grayscale image is a binary image. Figure 4.1 shows the eight bit planes of each of three given 128×128 grayscale images. The image of each bit plane is zoomed out for comparison. It is observed that the LSB plane $bp_0$ is almost fully randomized. If the message is embedded in $bp_0$, the result will appear almost unaltered to human eyes. On the contrary, random noise areas are less in a more significant bit plane. The most-significant-bit plane $bp_7$ contains almost no noise, and data cannot be embedded easily in it without causing significant visual changes. We may embed message data into bit planes in the order of $bp_0$, $bp_1$, …, $bp_7$. This scheme is termed *horizontal data hiding*, to be contrastive with traditional *vertical data hiding* methods which embed data into the bits $b_7$, $b_6$, …, $b_0$ of each pixel in the order of $b_0$ through $b_7$, where $b_0$ is the LSB of the pixel. Compared with the vertical data hiding method, horizontal data hiding can reduce more distortion in the stego-image, as revealed in the results of this study.

On the other hand, embedding data directly in bit planes will cause visible damages to the edges in the bit planes. To overcome this difficulty, in this study we design a new cost function which considers certain perception characteristics of the HVS, and adopt a method proposed in Lee and Tsai [25] for data embedding. Each bit plane is regarded to have a different weight in its capability for data hiding, and the new cost function is designed accordingly to measure the degree of distortion resulting from pixel value changes. The details are discussed in the following.

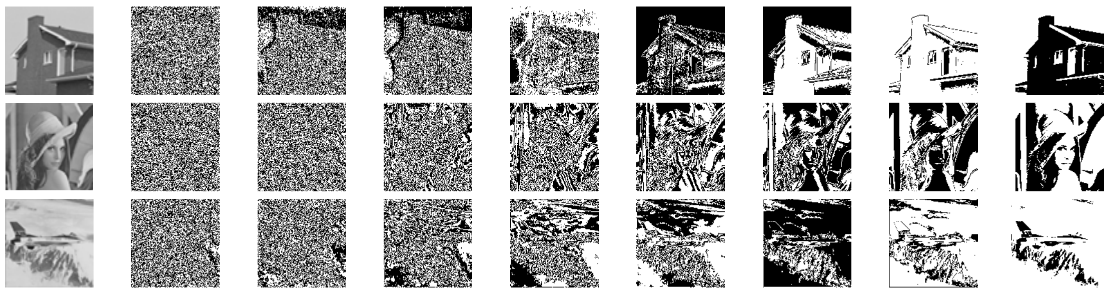


Figure 4.1 Three grayscale images and their 8 corresponding bit planes (from left to right, original images, $bp_0$, $bp_1$, $bp_2$, …, and $bp_7$, respectively).

In the following sections, the proposed cost function for distortion measurement is given first in Section 4.2. The proposed horizontal data hiding and recovery processes are described in Section 4.3 and Section 4.4, respectively. The experimental results are shown in Section 4.5, followed by concluding remarks in Section 4.6.

## 4.2 Cost Function for Distortion Measurement

Since stego-images are viewed by human vision, the characteristics of the HVS must be exploited in designing a data embedding process. Two of such characteristics are useful here. First, human perception is more sensitive to grayscale changes in smooth areas than in texture areas in a grayscale image. Second, human perception is sensitive to relative luminance rather than absolute one. Designing the cost function for distortion measurement for data embedding must take these two characteristics into consideration, as elaborated in the following.

## A. Computing Number of Data-Embeddable Bits with Consideration of Neighborhood Grayscale Value Change

For the first consideration, assume that a pixel $P$ with grayscale value $g$ is to be used to embed message data. Let $MAX$ denote the maximum grayscale value, and $MIN$ the minimum, in the 3×3 block with $P$ as the center, which we call the *neighborhood* of $P$. Then, the maximum *between-pixel grayscale range* in this block is $\Delta = MAX - MIN$. According to the previous discussions, to avoid a significant change of the smoothness degree with respect to the neighborhood of $P$, the new grayscale value $g'$ resulting from the data embedding is restricted in this study to remain in the range of $g \pm \Delta/2$. Then, we define a *maximum number D of data-embeddable bits* at $P$ as

$$D = \lfloor \log_2(\Delta/2) \rfloor = \lfloor (\log_2\Delta - 1 \rfloor = \lfloor \log_2(MAX - MIN) - 1 \rfloor. \tag{1}$$

## B. Computing Number of Data-Embeddable Bits with Consideration of Pixel's Luminance Change

For the second consideration mentioned above, let $f$ denote the luminance of a pixel $P$ with grayscale value $g$ where $1 \le f \le 100$. According to the Fechner law [26], the relative luminance property perceived by the HVS may be expressed as a contrast value $c$ computed by

$$c = 50 \times \log_{10} f$$

where $0 \le c \le 100$. Moreover, according to the Weber law [26], the maximum allowable change $\Delta c$ of the contrast value $c$ according to the principle of "just noticeable difference (JND)" about the pixel's luminance change is about 2. That is, if the luminance of a pixel is changed too much so that $\Delta c$ is larger than 2, the change will be noticeable to the HVS. Accordingly, we can compute in another way a maximum number of data-embeddable bits in the 8 bits of a pixel's grayscale value, as described next.

First, we want to compute the maximum luminance change $(\Delta f)_{max}$ in accordance with the maximum allowable contrast change $(\Delta c)_{max} = 2$. With $c$ being the contrast of pixel $P$, let $c_{max}$ denote the maximum possible contrast value. Then, we have

$$2 = (\Delta c)_{max} = c_{max} - c = 50\times\log_{10} f_{max} - 50\times\log_{10} f = 50\times\log_{10}\frac{f_{max}}{f},$$

which can be reduced to be

$$\frac{f_{max}}{f} = 10^{(2/50)} = 10^{0.04}.$$

So, the maximum allowable luminance change can be expressed as

$$(\Delta f)_{max} = f_{max} - f = (\frac{f_{max}}{f} - 1)\times f = (10^{0.04} - 1)\times f \approx 0.0965 \times f.$$

And so we may impose the following constraint to the value of $f$:

$$(\Delta f)_{max}/f \leq 0.0965. \qquad (2)$$

On the other hand, in a monochrome image the luminance $f$ in the range of [1 100] is represented by the grayscale value $g$ in the range [0 255], such that $g$ may be computed by the mapping $g = (f - 1) \times (255/99) \approx 2.576(f - 1)$, or equivalently, the mapping $f \approx 0.3882g + 1$, which specifies a linear relation between $f$ and $g$. Hence, from Constraint (2), we can, after some derivations, get the following new constraint for grayscale changes according to the principle of JND:

$$0.0965 \geq (\Delta f)_{max}/f = (\Delta g)_{max}/(g + 2.576) \qquad (3)$$

where $(\Delta g)_{max}$, corresponding to $(\Delta f)_{max}$, denotes the maximum grayscale change in the pixel's neighborhood. That is, if the above constraint (3) is set for data embedding, the changes of grayscales in the stego-image will not be detectable by human eyes according to the principle of JND.

Now, we discuss how many bits can be utilized for data embedding for each possible grayscale value $g$. If 5 bits of the pixel's grayscale are used for embedding message data, the maximum grayscale change at the pixel will be $(\Delta g)_{max} = 2^5 - 1 = 31$. And according to Constraint (3), $g$ must be larger than 319, which, however, is out of the grayscale range [0, 255]. This means that embedding 5 or more bits of message data into a pixel is impractical according to the principle of JND. As a result, $bp_4$, $bp_5$, $bp_6$, and $bp_7$ are not used for data embedding in this study. If 4 LSBs of $g$ are changed, then $(\Delta g)_{max} = 2^4 - 1 = 15$, and by Constraint (3) we get $g > 153$. That is, when the constraint $g > 153$ is satisfied, we can embed data into the 4 LSBs of $g$ without causing a noticeable luminance change according to the principle of JND.

However, the binary value of 153 is $10011001_2$. After the 4 LSBs of $g$ are changed, the new value of $g$ might become a value in the range of $10010000_2$ through $10011000_2$, which is smaller than 153, causing a violation of Constraint (3). Therefore, we must change the above constraint $g > 153$ to be $g > 160$ where $160 = 10100000_2$ such that after any 4-bit data are embedded into the 4 LSBs of $g$, the resulting new value $g'$ of $g$ will always be larger than 160, thus satisfying Constraint (3). In other words, to meet Constraint (3), only when a given pixel's grayscale $g$ satisfies $g \geq 160$ can the 4 LSBs of $g$ be replaced by 4-bit message data. And in short, 4 bits are the upper limit to be embedded in a pixel's grayscale according to the principle of JND.

Similarly, if 3 bits are changed, then $(\Delta g)_{max} = 2^3 - 1 = 7$, and by Constraint (3) as well as a similar reasoning process, the constraint $g \geq 72$ should be satisfied, where $72 = 01001000_2$. If 2 bits are changed, the constraint $g \geq 32$ is required, where $32 = 00100000_2$. Finally, if 1 bit is changed, $g \geq 10$ is necessary, where $10 = 00001010_2$. In summary, we embed an appropriate number $B$ of message bits in a pixel's grayscale $g$ according to the following rule to satisfy the principle of JND:

$$\begin{aligned}&\text{if } g \geq 160, \text{ then } B = 4;\\ &\text{if } g \geq 72, \text{ then } B = 3;\\ &\text{if } g \geq 32, \text{ then } B = 2;\\ &\text{if } g \geq 10, \text{ then } B = 1;\\ &\text{otherwise, } B = 0.\end{aligned} \tag{4}$$

### C. Combining Results of Two Considerations

To combine the results of the above two considerations, it is not difficult to figure out that the maximum number of data-embeddable bits at a pixel should be taken to be $E = \min(D, B)$ where $D$ and $B$ are as specified in (1) and (4), respectively.

Let the grayscale value $g$ of a pixel $P$ in binary form be denoted as $g = (g_7\ g_6\ g_5\ g_4\ g_3\ g_2\ g_1\ g_0)_2$, and the *replacement cost* of $g_i$ in the $i$-th bit plane be denoted as $C_i$ where $0 \leq i \leq 3$. According to the previous discussions, $C_i$ is defined in this study as:

$$\text{if } i \leq (E-1), \text{ then } C_i = 8/2^{(E-1)-i}; \text{ otherwise, } C_i = \infty.$$

The above definition of cost function gives more penalties to replacements of more significant bits. In more detail, we have the following results:

$$\begin{aligned}&\text{if } E = 4, \text{ then } C_0 = 1, C_1 = 2, C_2 = 4, C_3 = 8, \text{ and } C_4 \text{ through } C_7 = \infty;\\ &\text{if } E = 3, \text{ then } C_0 = 2, C_1 = 4, C_2 = 8, \text{ and } C_3 \text{ through } C_7 = \infty;\\ &\text{if } E = 2, \text{ then } C_0 = 4, C_1 = 8, \text{ and } C_2 \text{ through } C_7 = \infty;\\ &\text{if } E =1, \text{ then } C_0 = 8, \text{ and } C_1 \text{ through } C_7 = \infty;\\ &\text{if } E = 0, \text{ then } C_0 \text{ through } C_7 = \infty.\end{aligned}$$

## 4.3 Proposed Horizontal Data Hiding Process

The proposed method is implemented as an algorithm which can be divided into two stages: (1) embedding of some *control data*, followed by (2) embedding of message data. The control data include the necessary information for use in the data recovery process. All data are embedded in the bit planes by the block pattern encoding method. As mentioned previously, each of the bit planes $bp_0$ through $bp_3$ can be viewed as a binary image and they together can be regarded as concatenated into a sequence for data embedding. In this section, the idea to deal with the binary image is presented first, followed by the proposed process.

### A. Block Pattern Encoding for Data Embedding

In order to embed a message into a binary image, every 2×2 image block is regarded as a *pattern* with a 4-bit *binary value* in which each bit of 0 corresponds to a black pixel and each 1 a white one. The proposed data embedding process is based on the use of a *block pattern encoding table* which maps each block pattern into a certain code with each code being one, two, or three bits of the message data to be hidden. And data embedding is accomplished by changing the block bit values so that the corresponding code of the resulting block pattern become just some bits of the input message data to be embedded. A possible block pattern encoding table designed for use in this study is shown in Table 4.1. It is emphasized, by the way, that such a table is just one of the many possible tables which may be used for data hiding, and the proposed data embedding process will choose from them an *optimal one* for each specific input binary image, as described later.

Suppose that we want to embed one bit in a 2×2 block. The number of possible patterns in a 2×2 block are 16. This number is much larger than the required number of 2 to represent the two different message bits ‘0’ and ‘1’ in a block, so we may use *more than one* block pattern to represent a single message bit (0 or 1), allowing the possibility of choosing among the block patterns an *optimal one* to replace the original block in the data embedding process and thus reducing more distortion in the resulting block. On the other hand, we wish to embed more data in a block, not just a bit as just mentioned; and for this we may use a block pattern to represent more than one bit, as is done in this study. In short, we want to achieve both minimum-cost bit replacement and maximum-volume data embedding.

As an illustration, we may use either the block pattern $t_1 = 1011_2$ or the pattern $t_2 = 0111_2$ to represent the two-bit message value $s = 01_2$. In this way, when we want to embed, for example, the message value $s = 01_2$ into a block $B$ with value $v = 1010_2$, we have the *two alternative* block patterns $t_1 = 1011_2$ and $t_2 = 0111_2$ to choose to replace $v = 1010_2$, instead of the conventional case of *just one*. And if we choose $t_1 = 1011_2$ to replace $v = 1010_2$, then less distortion of just a 1-bit error (occurring at the LSB position) will result. Contrastively, if only one block pattern, say, $t_2 = 0111_2$ is available, then an error of 3 bits will result, causing more distortion in the resulting block. It is such an allowance of multiple choices for block pattern replacement that achieves more distortion reduction in the proposed method. By the way, the previously-mentioned *bit errors* are used just for convenience of illustrating the advantage of multiple choices of replacing blocks; they in fact should be the *replacement cost*s defined previously.

### B. Data Embedding in Binary Images

The proposed data embedding process in binary bit-plane images consists of four major steps and includes two folds of distortion minimization, as described in the following.

(I) *Computing bit costs for data embedding:* We calculate the replacement cost value for each bit in the image according to the cost function defined in Section 4.2.

(II) *Dividing the input image into blocks:* We first divide each of the bit planes $bp_0$ through $bp_3$ into non-overlap 2×2 blocks with every two neighboring blocks separated by a 1-pixel-wide line of pixels in between, as shown in Figure 4.2. And next, we select the first $n$ "embeddable" blocks and concatenate them sequentially, where $n$ is the length of the message data string to be embedded. A block is said to be *embeddable* in this study if the replacement cost value of any bit of the block is not infinite.

(III) *Using multiple block pattern encoding tables for the first-fold distortion reduction:* We generate all possible block pattern encoding tables and select an *optimal* one for use in the data embedding process, in the sense of introducing the least distortion. The reason is that a single block pattern encoding table will not be suitable for every input binary image; if an image is destroyed seriously after data embedding using a specific table like Table 4.1, it will be appropriate to use another table with other combinations of block patterns to encode the message data. Specifically, we exchange the encoded message data of certain types in Table 4.1 with those of the other types in the following way:

exchange the message data "0" with the message data "1";

exchange the message data "00" with the message data "01";

exchange the message data "10" with the message data "11";

exchange the message data "010" with the message data "011";

exchange the message data "100" with the message data "101";

exchange the message data "00" and "01" with the message data "10" and "11," respectively;

exchange the message data "010" and "011" with the message data "100" and “101,” respectively.

By enumerating all possible cases in the above way, we can get the 128 distinct tables (numbered from 0 to 127) for selection to minimize the distortion.

(IV) *Applying search techniques for the second-fold distortion reduction:* Finally, we apply the dynamic programming technique to segment the input message data stream *optimally* into a series of codes and embed them in the input image, according to the cost function proposed previously. This reduces the resulting distortion further in a global sense.

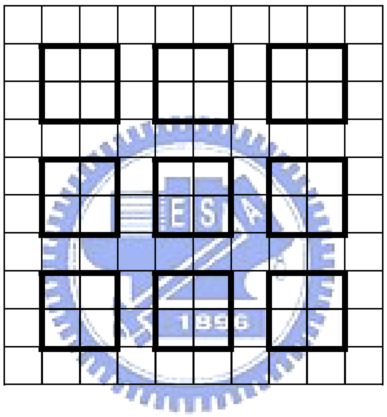
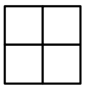
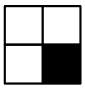
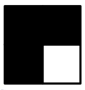
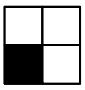
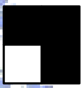
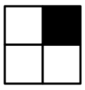
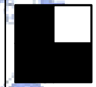
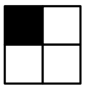
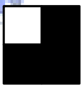
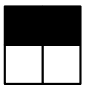
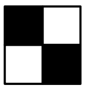
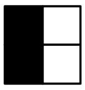
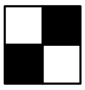
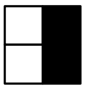
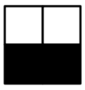


Figure 4.2 Division of input image into 2×2 blocks with separating lines (grids with bold boundaries are 2×2 blocks for data embedding).

## C. Search for Optimal Solutions

The *search cost* proposed in this study for use in the adopted search technique is the *total replacement cost* in the resulting stego-image, computed from the summation of the replacement costs of all the bit changes in the replaced blocks. In Table 4.1, block patterns can be used to encode one, two, or three message bits. Accordingly, when we embed a binary message value $v$, we have the three choices of embedding one, two, and three initial bits of $v$ into a block. To determine how many bits should be embedded in a selected block, we may calculate first the cost for each of the three cases, and replace the selected block with the block pattern corresponding to the minimum cost. This method provides a quick way for data embedding; however, it is just a *greedy search* algorithm and in general *does not* yield an optimal solution.

Table 4.1 A block pattern encoding table proposed in this study.

| Type | Block pattern | Corresponding binary value | Encoded message data | Type | Block pattern | Corresponding binary value | Encoded message data |
|---|---|---|---|---|---|---|---|
| 0 |  | 1111 | 1 | 1 |  | 0000 | 0 |
| 2 |  | 1110 | 00 | 3 |  | 0001 | 11 |
| 4 |  | 1101 | 00 | 5 |  | 0010 | 11 |
| 6 |  | 1011 | 01 | 7 |  | 0100 | 10 |
| 8 |  | 0111 | 01 | 9 |  | 1000 | 10 |
| 10 |  | 0011 | 011 | 11 |  | 0110 | 100 |
| 12 |  | 0101 | 011 | 13 |  | 1001 | 101 |
| 14 |  | 1010 | 010 | 15 |  | 1100 | 010 |

To see this, for example, suppose that the message value $v$ of $011_2$ is to be embedded in three selected blocks with patterns $B_1 = 0100$, $B_2 = 0100$, and $B_3 = 1100$ according to Table 4.1. And as illustrated in Figure 4.3, suppose also that the costs of replacing the four bits are computed to be 2, 1, 1, and 2 for $B_1$; to be 1, 4, 4, and 1 for $B_2$; and to be 4, 4, 1, and 1 for $B_3$. By the above-mentioned greedy search algorithm, we replace $B_1$ = 0100 with the block pattern 0000 of type 1 to embed the initial bit 0 of *v*. The replacement cost for this block is 2×0 + 1×1 + 1×0 + 2×0 = 1 because a bit (the second bit) is flipped here with its corresponding cost being 1 and the other bits in the original block are not changed. This cost is a local minimum. Next, we replace $B_2$ = 0100 with the block pattern 0001 of type 3 to embed the last two bits $11_2$ of *v*, and the replacement cost is 1×0 + 4×1 + 4×0 + 1×1 = 5. Therefore the total replacement cost for embedding *v* is 1 + 5 = 6.

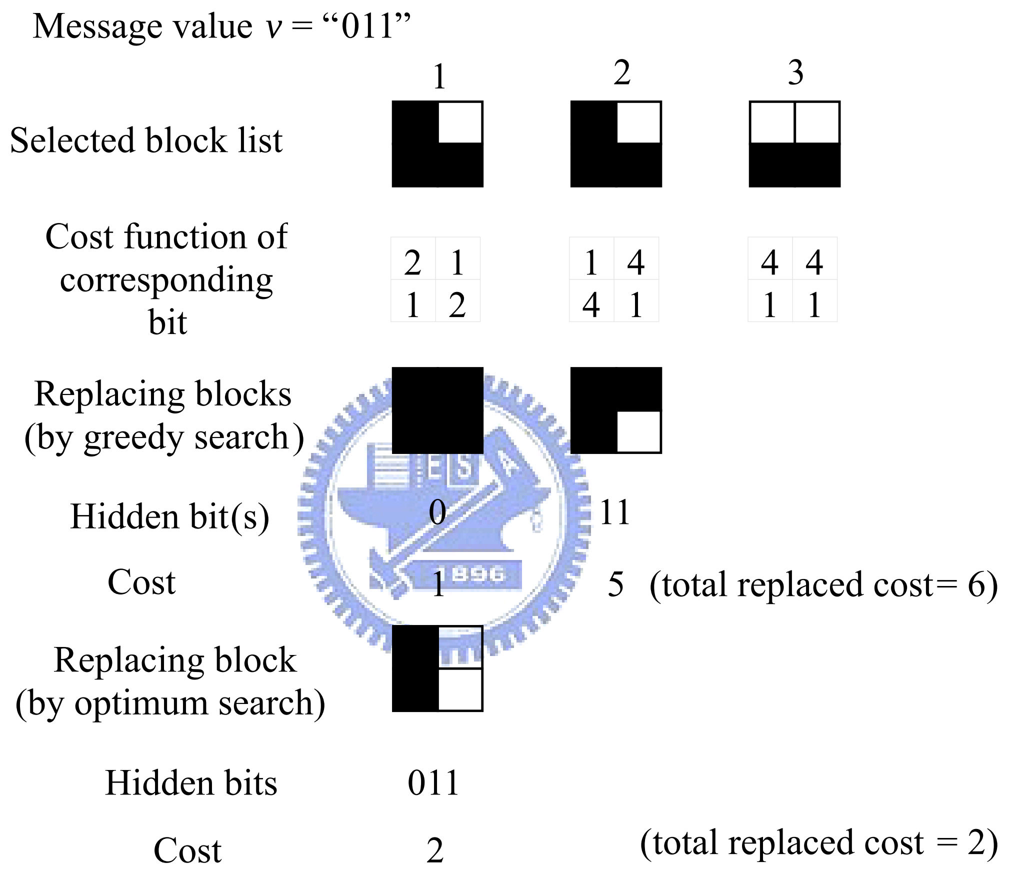
Now, if we do not use the greedy search algorithm at the beginning, and replace instead $B_1$ = 0100 by the block pattern 0101 of type 12 in Table 4.1 to embed the three bits $011_2$ of *v* directly, then the total replacement cost value will be reduced to be 2×0 + 1×0 + 1×0 + 2×1 = 2 which is *smaller* than the previously-computed total replacement cost of 6. This shows that there indeed exists at least one solution better than that found by the greedy search algorithm. Figure 4.3 illustrates the data embedding process for this example. This is also true for many other examples, as found by this study. And so the search of an optimal solution is meaningful, for which the proposed method is *dynamic programming*.

## D. Dynamic Programming for Data Embedding

In the proposed dynamic programming algorithm (abbreviated as DPA hereafter), *edit distances* are defined for cost minimization in the search. Assume that the input message data to be embedded are in the form of an *n*-bit string $S_1$ with $S_1[i]$ denoting its *i*-th bit. Also, let *n* 2×2 embeddable blocks be selected as a list in advance for data embedding and expressed as another string $S_2$ with $S_2[i]$ denoting its *i*-th *block*. For convenience, let $S_k[i \sim j]$ denote a substring of $S_k$ with bits or blocks $S_k[i]$ through $S_k[j]$, where $k$ =1, 2 and $i, j$ = 1, 2, …, *n*.

Only one type of edit operation, namely, *replacement*, is used in the proposed

DPA to specify the image block replacement operations involving $S_1$ and $S_2$ in the proposed data embedding process. The edit distance between $S_1$ and $S_2$ is defined, according to the previous discussions, as the minimum total replacement cost to transform $S_2$ into $S_1$ by editing operations according to a certain block pattern encoding table.



Message value v = "011"
1
2
3
Selected block list
Cost function of corresponding bit
2 1
1 2
1 4
4 1
4 4
1 1
Replacing blocks (by greedy search)
Hidden bit(s)
0
11
Cost
1
5 (total replaced cost= 6)
Replacing block (by optimum search)
Hidden bits
011
Cost
2
(total replaced cost = 2)

Figure 4.3 An example of proposed data embedding process.

Let C be an $n \times n$ *cost matrix* with its element C[$j$, $i$] denoting the minimum total replacement cost to transform a substring $S_2[j \sim m]$ of $S_2$ into a substring $S_1[i \sim n]$ of $S_1$, where $m \leq n$. Then C[1, 1] is the minimum total replacement cost to transform $S_2[1 \sim m]$ into $S_1[1 \sim n]$ (i. e., to transform the substring of $S_2$ into the entire string of $S_1$), where $1 \leq m \leq n$. Also, let RC be a cost function with each of its element RC($j$, $i$, $L$)

denoting the minimum replacement cost for replacing the $j$-th block $S_2[j]$ of $S_2$ with the block pattern which encodes the initial $L$ bits of the substring $S_1[i \sim n]$ of $S_1$ with = 1, 2, or 3. We define RC($j$, $i$, $L$) = ∞ if $i + L > n + 1$.

By the above definitions, the value C[$j$, $i$] is recursively just the minimum of all the possible values of RC($j$, $i$, $L$) + C[$j$+1, $i$+ $L$], where $L$ = 1, 2 or 3. Also, we define C[$j$, $i$] = 0 if $i > n$ or $j > n$. Then, according to dynamic programming, the *minimum search cost* and its corresponding solution may be computed by the following algorithm.

**Algorithm 4.1** ***Computing minimum search cost for minimizing distortion by the DPA.***

***Input***: (1) an $n$-bit message data string $S_1$; (2) a string $S_2$ of $n$ selected blocks; (3) a block pattern encoding table $T$; (4) an $n \times n$ cost matrix C[$j$, $i$], for $i, j$ = 1, 2, …, $n$; (5) an $n \times n$ *type matrix* I with its element I[$j$, $i$] used for recording the block pattern in $T$ used for replacing $S_2[j]$ in calculating C[$j$, $i$]; and (6) an $n \times n$ *segmentation matrix* N with its element N[$j$, $i$] used for recording the number of initial bits of $S_1[i \sim n]$ used in calculating C[$j$, $i$].

***Output***: C[$j$, $i$], I[$j$, $i$], and N[$j$, $i$] for all $i, j$ = 1, 2, …, $n$.

***Steps***:

3. Set all C[$j$, $i$] initially to be ∞ for all $i, j$ = 1, 2, …, $n$.
4. Starting from $i = n$ and $j = n$, for each pair of ($j$, $i$) with $i, j$ = 1, 2, …, $n$, perform the following steps.
   2.1 If C[$j$, $i$] is equal to ∞, continue the next step (Step 2.2); else increment $i$ and $j$ to calculate the next C[$j$, $i$].
   2.2 Take C[$j$, $i$] to be the minimum of the three replacement costs, RC($j$, $i$, 1) + C[$j$+1, $i$+1], RC($j$, $i$, 2) + C[$j$+1, $i$+2], and RC($j$, $i$, 3) + C[$j$+1, $i$+3]; and record the corresponding number of the processed initial bits (1, 2, or 3) of $S_1[i \sim n]$

in N[$j$, $i$], and the corresponding type of the used block pattern of $T$ in I[$j$, $i$].

In the above algorithm, the number of initial bits of $S_1[i \sim n]$ and the used block pattern type in each recursive step are recorded in matrices N and I, respectively, which are used in the data embedding process, as described in the next algorithm.

**Algorithm 4.2** ***Data embedding using block pattern encoding tables and the DPA.***

***Input***: (1) a grayscale image $G$; (2) a secret message data string $S_1$ with $n$ bits; (3) a control message data string $S_c$ with $m$ bits, including a *table number* $T_{\text{opt}}$ (specifying the block pattern encoding table used) with seven bits, followed by a value $L_{\text{opt}}$ (specifying the number of selected blocks used) with $m - 7$ bits; and (4) 128 block pattern encoding tables.

***Output***: a stego-image $G'$.

***Steps***:

1. Compute the cost of each bit of $G$ as mentioned previously.
2. Get a list $B_m$ of $m$ 2×2 embeddable blocks sequentially from the bit planes $bp_0$ through $bp_3$ of $G$ in order for embedding the $m$ bits of $S_c$. Following $B_m$, get also a list $B_n$ of $n$ 2×2 embeddable blocks sequentially for the $n$ bits of $S_1$. Let $B_m$ and $B_n$ also include the position information of each selected block.
3. For each block pattern encoding table $T$ among the input 128 ones, with $S_1$, $B_n$, and $T$ as input, apply Algorithm 4.1 to calculate the cost matrix C[$j$, $i$], the type matrix I[$j$, $i$], and the segmentation matrix N[$j$, $i$] for all $i, j = 1, 2, \ldots, n$.
4. Find the minimum $C_{\text{min}}$ of the 128 values of C[1, 1] computed in the last step, and set $T_{\text{opt}}$ to be the table number of the corresponding block pattern encoding table used in computing $C_{\text{min}}$.
5. Use the block pattern encoding table $T_{\text{opt}}$, the type matrix $I_{\text{min}}$, and the

segmentation matrix $N_{\min}$ corresponding to $C_{\min}$, and the position information of each block in $B_n$, to embed the string $S_1$ into $bp_0$ through $bp_3$ of $G$ to get an *initial* stego-image $G_i$.

6. Set the value $L_{\text{opt}}$ to be the number of the blocks used for embedding $S_1$ in the last step.

7. Using $S_c$ (including $T_{\text{opt}}$ and $L_{\text{opt}}$), $B_m$, and $T= 1$ as input, apply Algorithm 4.1 to calculate the cost matrix C[$j$, $i$], the type matrix I[$j$, $i$], and the segmentation matrix N[$j$, $i$] for all $i, j = 1, 2, \ldots, m$.

8. Use the block pattern encoding table Table 4.1, the type matrix I and the segmentation matrix N in the last step, and the position information of each block in $B_m$, to embed the substring $S_c$ into $bp_0$ through $bp_3$ of $G_i$ to get the *final* stego-image $G'$.

Simply speaking, the above algorithm embeds the control message and the secret message data sequentially into the first $m$ and $n$ embeddable blocks in Steps 8 and 5, respectively.

## 4.4 Proposed Data Recovery Process

The goal of data recovery is to extract the embedded message data from a stego-image. Before the proposed data recovery process is started, Table 4.1 is simplified in advance as an *extraction table* as shown in Table 4.2. The other 127 encoding tables are converted similarly. It is easier to use this type of table to carry out the recovery process described in the following.

**Algorithm 4.3** ***Message data recovery***.

***Input***: a stego-image $G'$ including a message bit stream $S$.

***Output***: the message bit stream $S$.

***Steps***:

1. Calculate the cost of every bit of $G'$ as mentioned previously.
2. Get $m$ 2×2 embeddable blocks sequentially from $bp_0$ through $bp_3$ of $G'$ as a list $L_m$.
3. For each 2×2 block $P$ of $L_m$, compute the binary value $v$ corresponding to the block pattern, and decode $v$ by looking $v$ up in the block pattern encoding table Table 1 to get the corresponding encoded message data bits as the data recovery result of $P$.
4. Concatenate the initial $m$ data bits extracted in the last step into a sequence as a desired control message data $S_c$.
5. Get the initial 7 data bits of $S_c$ as $T_{opt}$, and the remaining $m - 7$ data bits of $S_c$ as $L_{opt,}$ which specify respectively (1) the optimal block pattern encoding table $T_{opt}$ used in data embedding; and (2) the number of 2×2 blocks of $G'$ used in embedding $S_c$ in the $bp_0$ through $bp_3$ of $G'$.
6. Also, get $L_{opt}$ 2×2 selected blocks sequentially from $bp_0$ through $bp_3$ of $G'$ as a list $L$.
7. For each 2×2 block $P$ of $L$, compute the binary value $v$ corresponding to the block pattern, and decode $v$ by looking $v$ up in the block pattern encoding table $T_{opt}$ to get the corresponding encoded message data bits as the data recovery result of $P$.
8. Concatenate all the data bits extracted in the last step into a sequence as the desired message bit stream $S$ and exit.

For security consideration, we encrypt further the control message by a secret key before the data embedding process, and embed the result into $bp_0$ through $bp_3$ at bit positions randomly generated with a distinct secret key as well as a random number generator. The reverse process can be easily performed to get the original control message. The same method is also applied to the message data to get a higher degree of data protection.

## 4.5 Experimental Results

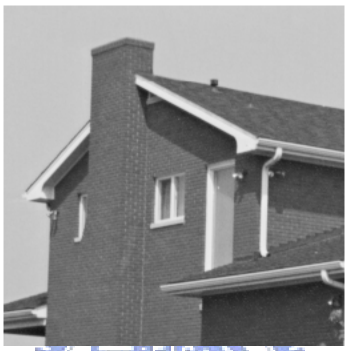
Figures 4.4 and 4.5 illustrate some experimental results of applying the proposed method. The bit streams of message data in Figures 4.4 and 4.5 were generated randomly. The stego-image "House" of size 256×256 with a high PSNR value of 56.88 dB obtained by embedding 16440 bits (about 2KB) message data using the DPA and the optimal block pattern encoding table among the 128 ones is shown at the right side of Figure 4.4. The cover image is depicted in the left side of Figure 4.4 for comparison. The result shows that the proposed method can be applied to embed message data in a grayscale image and obtain a good-quality stego-image without noticeable artifacts in the smooth regions.

Table 4.2 An extraction table (table number $T$=0).

| Corresponding binary value of block pattern | Encoded message data | Corresponding binary value of block pattern | Encoded message data |
|---|---|---|---|
| 1111 | 1 | 0111 | 01 |
| 1110 | 00 | 0110 | 100 |
| 1101 | 00 | 0101 | 011 |
| 1100 | 010 | 0100 | 10 |
| 1011 | 01 | 0011 | 011 |
| 1010 | 010 | 0010 | 11 |
| 1001 | 101 | 0001 | 11 |
| 1000 | 10 | 0000 | 0 |

Figure 4.5(b) illustrates three grayscale stego-images "House", "Lena" and "Jet,"

and their 8 corresponding bit planes. For comparison, Figure 4.1 is repeated as Figure 4.5(a) here. The three stego-images of size 128×128 were obtained by embedding 1000 bytes of message data using the proposed DPA and the optimal encoding table. The PSNR values are 46.90 dB, 49.33dB and 48.80 dB, respectively. Compared with the cover images in Figure 4.1 and their 8 corresponding bit planes, it can be seen that the stego-images retain most significant textures.


(a)

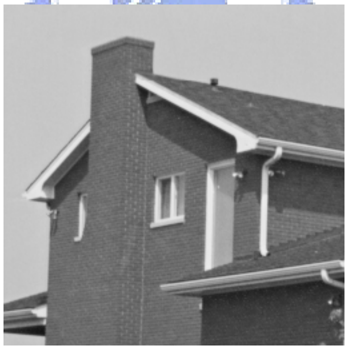

(b)

Figure 4.4 A cover image "House" with the size of 256×256 and its stego-image with 16440-bit message data embedded. (a) The cover image. (b) The stego-image.

Table 4.3 summarizes the statistical data of the stego-image "Lena" using the DPA and the optimal encoding table, including the message data length, the PSNR value, the selected block pattern encoding table, the numbers of used blocks, the minimum replacement cost values, and the average of the numbers of embedded bits per block. The message data bit stream in Tables 3 was generated randomly.

In more detail, the result from Table 4.3 is transformed into Figs. 6. When the amount of the embedded data is smaller than 1000B, the PSNR values in Table 4.3 are all larger than 49dB. And the differences in the stego-images cannot be noticed by human eyes.

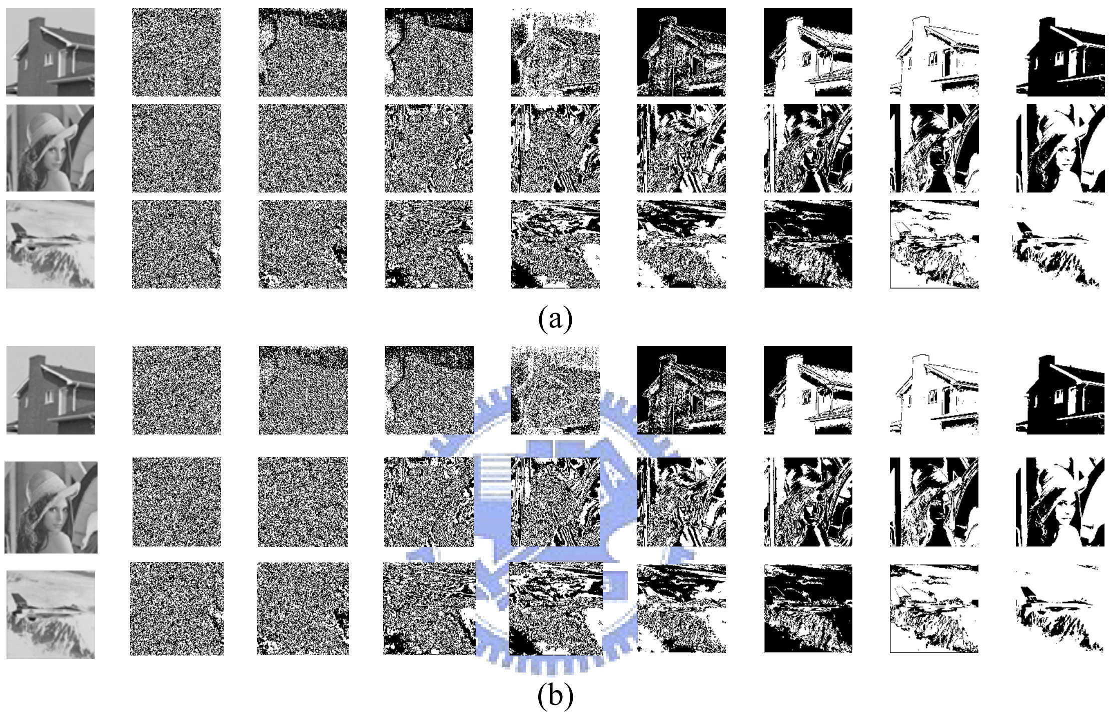


Figure 4.5 Experimental results of three images. (a) The original images and their corresponding bit planes (repeated from Figure 4.1). (b) The resulting three stego-images and their corresponding bit planes (from left, $bp_0$, $bp_1$, $bp_2$, …, $bp_7$).

Figure 4.6 also reveals that the relation between the PSNR value yielded by the proposed method and the embedded data amount is approximately *linear*. The PSNR value of the DPA decreases about 3.3225 dB when the embedded data size increases 200 bytes. Thus, the proposed DPA method can predict the PSNR value before the data embedding process starts according to the message data size. From Table 4.3, the PSNR value of the DPA can be estimated by a simple line fitting method to be

$$\text{PSNR} = 62.5870 - (m - 1) \times 3.3225 \text{ (dB)}, \ 1 \leq m \leq 5,$$

where $m$ denotes the size of message data in the unit of 200 bytes for 128×128 grayscale images. Similar results can be observed for the other images. Moreover, the equation of the PSNR value can be extended and used for grayscale images of any sizes by applying the proposed DPA. For a grayscale image of size $H \times W$, if the above value of $m$ denotes the size of the message data in the unit of $(200 \times H \times W)/(128 \times 128)$ bytes, then the resulting PSNR value still can be estimated using the above equation. Note that this merit of predictable PSNR values enables a user of the proposed method to determine how large a cover image should be selected for a certain given amount of message data.

Furthermore, we may compute a *distortion rate* for each stego-image. This rate is computed in this study as the ratio of the number of bit flippings (changing bit 0 to 1 or 1 to 0 in data embedding) to the length of the message data. Most existing vertical data hiding methods yield distortion rates of about 50% for grayscale images because of the characteristic of randomness in bit flippings. In most existing *vertical* data hiding methods, when 200-byte secret message data are embedded into a 128×128 grayscale image, these data will be divided into pieces of 4 bits and each piece is embedded into the bits $b_3$, $b_2$, $b_1$, and $b_0$ of a pixel. Then the *average* grayscale change of the pixel, measured in terms of the number of flipped bits, may be computed to be $1 \times 50\% + 2 \times 50\% + 4 \times 50\% + 8 \times 50\% = 7.5$. Consequently, the corresponding mean square-error value *MSE* of the stego-image may be computed to be $MSE = [(200 \times 8)/4] \times (7.5 \times 7.5)]/(128 \times 128)$ where $(200 \times 8)/4$ is the number pixels required for embedding the 200-byte message data, $7.5 \times 7.5$ is the square error incurred at each pixel, and $128 \times 128$ is the image size. Finally, the PSNR value of the stego-image may be computed to be $10 \times \log(255^2/\text{MSE}) = 46.75$ dB. If the secret data is embedded by a *horizontal* data hiding method without distortion optimization, the corresponding average grayscale change of the pixel may be computed to be 1×50% + 1×50% + 1×50% + 1×50% = 2. And the corresponding *MSE* of the stego-image may be computed to be *MSE* = [(200×8)/4]×(2×2)]/(128×128). Finally, the corresponding PSNR value of the stego-image may be computed to be $10\times\log(255^2/\text{MSE})$ = 58.23 dB, which is larger than that of the vertical method. However, as seen in Table 4.3 the PSNR value of our method is an even larger value of 62.48 dB, which means the proposed method is superior to the conventional vertical data hiding method in distortion reduction.

The proposed DPA method takes long computation time to obtain the optimal solution when the volume of the message data is large. If time is a major concern, then the greedy search method mentioned previously may be used. As a comparison, we list in Table 4.4 the run times spent by the proposed methods (DPA and greedy search) and two others on a PC with a 3.4G Pentium 4 CPU for some grayscale images with two typical image sizes and three input message lengths. One of the two other methods is the simplest "1-LSB" which embeds message data in the LSB of each pixel. The other is "Hide4PGP" whose program was downloaded from the website http://www.heinz-repp.onlinehome.de/Hide4PGP.htm. As can been see from the table, the DPA takes about a minute to embed a message of 200 bytes and more than 35 minutes to embed 1200-byte data, while all the other three methods takes little times to accomplish the works. Therefore, the DPA can only be used for non-real-time applications with the need of distortion reduction, though the greedy search method may be used as a suboptimal substitute of it.

We also conduct an additional comparison of image distortion caused by the above-mentioned four methods for the same set of images of Table 4.4. The result is shown in Table 4.5 from which we see clearly that the proposed DPA method yields the largest PSNR values for all the tested images, indicating its effectiveness for reducing image distortion.

Table 4.3 Statistics of stego-images yielded by DPA using optimal encoding table.

| Stego-image | Message data length (bytes) | PSNR (dB) | Table number | No. of used blocks | Cost value | Embedded bit number per block |
|---|---|---|---|---|---|---|
| Lena (128×128) | 200 | 62.48 | 8 | 779 | 1774 | 2.054 |
| | 400 | 59.37 | 57 | 1636 | 3243 | 1.956 |
| | 600 | 55.57 | 57 | 2297 | 5680 | 2.09 |
| | 800 | 52.96 | 57 | 3101 | 8355 | 2.064 |
| | 1000 | 49.33 | 8 | 3826 | 11699 | 2.091 |
| | 1200 | 46.74 | 57 | 4295 | 16248 | 2.235 |

Table 4.4 Comparison of run times for four methods for grayscale images (in unit of sec.).

| Size | Stego- image | DPA | Greedy search | 1-LSB | Hide4PGP |
|---|---|---|---|---|---|
| 256×256 | Lena256+200B | 60 | 0.079 | 0.00016 | 0.0068 |
| | Lena256+1200B | 2112 | 0.437 | 0.00097 | 0.0072 |
| | House256+200B | 60 | 0.079 | 0.00016 | 0.0068 |
| | House256+1000B | 1473 | 0.366 | 0.00081 | 0.0071 |
| | Jet256+200B | 59 | 0.079 | 0.00016 | 0.0068 |
| | Jet256+1200B | 1082 | 0.439 | 0.00097 | 0.0072 |

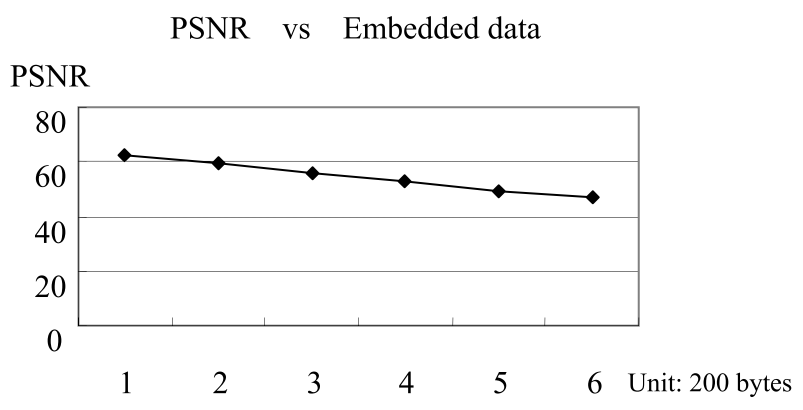


Figure 4.6 PSNR values of stego-image "Lena" using DPA.

Table 4.5 Comparison of PSNR values of the four methods for grayscale images (in unit of dB).

| Size | Stego- image | DPA | Greedy search | 1-LSB | Hide4PGP |
|---|---|---|---|---|---|
| 256×256 | Lena256+200B | 68.45 | 67.51 | 67.23 | 67.28 |
| | Lena256+1200B | 60.70 | 59.73 | 59.49 | 59.46 |
| | House256+200B | 68.64 | 67.60 | 67.07 | 67.29 |
| | House256+1000B | 61.53 | 60.60 | 60.21 | 60.21 |
| | Jet256+200B | 68.34 | 67.52 | 67.15 | 67.16 |
| | Jet256+1200B | 60.74 | 59.72 | 59.44 | 59.58 |

Finally, we applied steganalysis to the four the methods using a software tool available at http://diit.sourceforge.net, which is an open-source implementation of RS analysis developed by Fridrich, et al. [24]. We made a comparison of the analysis results shown in Table 4.6. Because the tool was designed for 24-bit color images with three color channels, we apply the proposed DPA and the greedy search method by embedding the message data evenly into image blocks of the three channels of R, G, and B alternatively, with the first block selected from R, the second from G, the third from B, the fourth from R again, and so on. For the 1-LSB method, we embed the data similarly except that bits instead of blocks are selected from the three channels alternatively. The third column in Table 4.6 specifies the detected message length of the cover image. This length value may be regarded as a “bias” of the RS detector, which supposedly should be zero because no message is hidden in the cover image. The last four columns specify the detected message lengths of the four methods, from whose contents we see that the DPA method is more robust against steganalysis than the greedy search method, and is not obviously so than the other two methods.

Table 4.6 Comparison of RS analysis results of the four methods for color images.

| Size | Cover image *C* | Detected message length of *C* | Stego-image *S* | Detected message length of *S* yielded by DPA | Detected message length of *S* yielded by greedy search | Detected message length of *S* yielded by 1-LSB | Detected message length of *S* yielded by Hide4PGP |
|---|---|---|---|---|---|---|---|
| 256× 256 | Lena256 | 379.58B | Lena256+200B | 458.42B | 497.65B | 529.13B | 510.38B |
| | Lena256 | 379.58B | Lena256+1200B | 1174.06B | 1378.88B | 1737.28B | 999.68B |
| | House256 | 508.05B | House256+200B | 596.93B | 626.86B | 561.83B | 552.40B |
| | House256 | 508.05B | House256+1000B | 1330.27B | 1646.94B | 1607.94B | 1310.35B |
| | Jet256 | 731.17B | Jet256+200B | 751.60B | 839.44B | 819.19B | 772.44B |
| | Jet256 | 731.17B | Jet256+1200B | 1341.43B | 1403.21B | 1244.48B | 1406.40B |

## 4.6 Concluding Remarks

A data hiding method for hiding message data into grayscale images with distortion reduction effects have been proposed. Two novel techniques for reducing distortions in resulting stego-images have been adopted, one being an optimal dynamic programming algorithm, and the other the use of multiple block pattern encoding tables. First, a cost function has been proposed to estimate the weight of each bit in each pixel to be replaced according to an HVS model. Next, a horizontal data hiding scheme in which message data are embedded in a sequence of bit planes has also been proposed to decrease possible distortions in stego-images. Also, an optimal block pattern encoding table is chosen from 128 alternative ones for use in data embedding to minimize image distortion. The encoding tables are designed in such a way that up to three bits in a 2×2 image block can be embedded. Finally, the proposed method minimizes further the distortion using dynamic programming based on the proposed cost function. The proposed method can predict the PSNR value of a sego-image before the embedding process starts according to the size of the data to be embedded.

The space and time complexities of the proposed dynamic programming algorithm are both quadratic. The algorithm costs more time to embed a long secret message. But in certain applications there is no need of real-time processing, and optimality in data embedding volumes or minimization in image distortion is the main concern. In such cases, the proposed method is good to use. On the other hand, if time is really concerned, then one can alternatively use the proposed greedy search algorithm, that takes only linear computation time and still minimize distortion in the stego-image in a suboptimal way. If high-speed processing is necessary, our method can be adapted to run on a parallel computer. In particular, each of the 128 block pattern encoding tables may be processed separately, and the dynamic programming process may be parallelized, too.

At least two methods may be adopted to make the proposed method more robust. First, multiple copies of a secret message may be embedded in the input image randomly with control by a key, so that an attack will not entirely destroy the secret information. And after the data are extracted by the proposed method, we may apply a voting scheme to recover the secret. The second method is to try to place secret data in the more significant bits of the cover image, for example, in $bp_2$ and $bp_3$ in the proposed method, assuming that most attacks to BMP images are conducted to the LSBs. Because the information encoded in these bit-planes cannot be removed in most applications (otherwise, the image will be seriously distorted or destructed), hopefully this method will work in real applications.

The proposed method processes 2×2 blocks in the data embedding process. It may be extended to process larger-sized blocks because when the block size is larger, the number of the block patterns which can be selected to encode a certain message value becomes larger as well, resulting possibly in greater reduction of image distortion. Other future works may be directed to embed multiple message data in a grayscale image for protecting the intellectual property right and authenticating multimedia data, to define more general cost functions for other HVS models, and to design better encoding tables to reduce image distortion further.

# Chapter 5
# Data Hiding in Color Images by Color Replacements with Reduction of Image Distortion and Change Noticeability

## 5.1 Idea of Proposed Method

The basic idea of the proposed method for data hiding in RGB color images is to encode certain colors in the color space, and embed given message bits into selected scattered image pixels by replacing these pixels' colors by the encoded colors. And extraction of the message is a reverse process, consisting of finding image pixels with encoded colors and decoding these colors to get the embedded message bits.

Appropriate techniques must be devised for the above simple idea of data embedding and extraction to be carried out effectively. The concern of reducing image content distortion and color change noticeability should be taken into consideration in these techniques. Also, the common requirement of data recoverability in data extraction need be met.

The techniques proposed in this study satisfy these aims and are described in the following.

In the remainder of this chapter, the detailed algorithms of the proposed data embedding and extraction are given in Section 5.2. In Section 5.3, some experiment results and discussions are described, followed by concluding remarks in Section 5.4.

### A. Proposed technique for reduction of color change noticeability

It is unnecessary to use all of the huge number of colors in the color space for data embedding by color replacements. Instead, we partition them into *non-overlapping* cubic-shaped clusters, called *color cubes*, and find out those cubes *better* for use in data embedding. More specifically, we find out image pixels with their colors “falling” in each color cube, and check the *scattering* degree of these pixels. Presumably, image pixels located more separately in the cover image are more suitable for data embedding because the changes of their colors, appearing to be farther way from one another, will attract less notice from observers. On the contrary, color changes at less scattered pixels tend to create visual artifacts and arouse more suspicion. Based on this idea, we propose in this study the following scheme of reducing the noticeability caused by image pixels’ color changes.

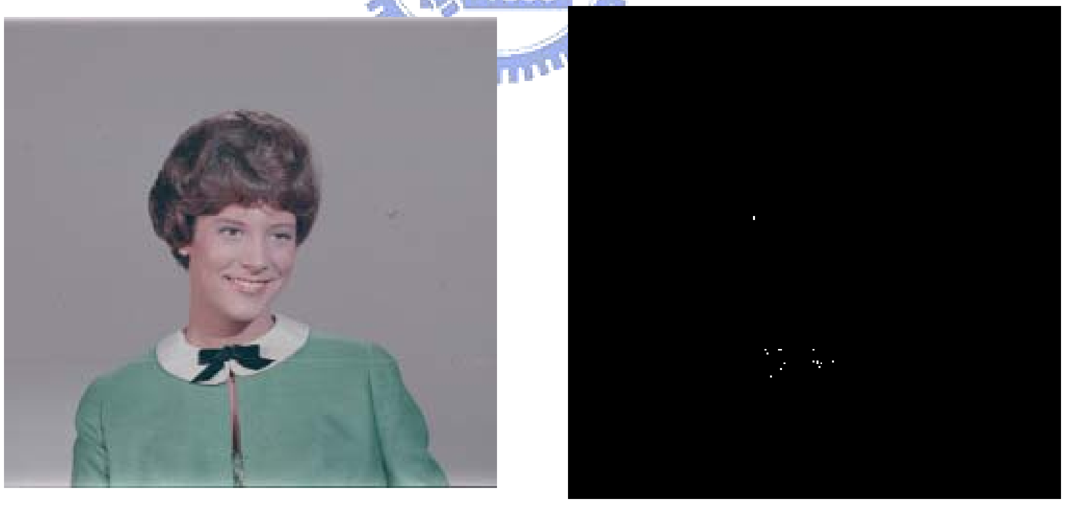
1. Partition the RGB color space into color cubes.
2. Collect the set of pixels in the cover image with their colors “falling” in each color cube, called *the range set* of the color cube.
3. Define the degree of *pixel scattering* of each color cube by a certain scatter measure of the pixels in the range set of the color cube.
4. Sort into a list the color cubes with *nonempty* range sets by their pixel scattering degrees, with the color cube with the largest scattering degree on the top of the list.
5. Sort further those color cubes with *equal* pixel scattering degrees by their range set sizes, meaning that color cubes with larger range sets will be used first for data embedding.
6. According to the length of the message to be hidden, select from the top of the color cube list a *sufficient* number of color cubes for use in data embedding.
7. Use the pixels of the range sets of the selected color cubes as the locations for data embedding by color replacements.

Let $S = \{P_1, P_2, \ldots, P_n\}$ denote the range set of a color cube $C$ with $n$ pixels. The scatter measure mentioned in Step 3 above for $C$, denoted as $M$, is defined as the mean of the Euclidean distances of all the pixel pairs in $P$, i.e., is defined as

$$M = \frac{\sum_{i,j} | P_i - P_j |}{n} \qquad (1)$$

where the Euclidean distance $|P_i - P_j|$ between any two pixels $P_i$ and $P_j$ at image coordinates $(u_i, v_i)$ and $(u_j, v_j)$, respectively, is computed as $|P_i - P_j| = [(u_i - u_j)^2 + (v_i - v_j)^2]^{1/2}$. A larger value of $M$ means higher pixel separateness of $S$ in the cover image.

As an illustration of the range sets of color cubes, Figure 5.1(a) shows a cover image and Figure 5.1(b) is the range set of a color cube found in (a), shown as a binary image with each white dot indicating a pixel in the set. The range set may be seen to include pixels with some dark green colors.



(a) Cover image. (b) Range set of a color cube.

Figure 5.1. An illustration of range sets of color cubes.

### B. Proposed technique for reduction of image content distortion

For convenience of data processing, the number of colors included in each color cube is taken to be a power of 2 in this study. If all the colors in a color cube, say with $2^m$ ones, are used for data embedding, each color may be used to represent $m$ message bits. The embedding work of an $m$-bit message segment then is to replace the color of an image pixel in the range set of a color cube by the color of the $2^m$ ones in the color cube, which corresponds to the value of the $m$ message bits.

However, to reduce the image distortion resulting from such color replacements, we propose in this study to allow *multiple* colors, instead of just a *single* one, to represent an identical message segment. For example, if we allow, say, $2^n$ colors as a group to represent a message segment, then whenever an image pixel's color is to be replaced by one in the color cube, there will be $2^n$ choices, and the one *closest* to the pixel's color may be taken as the replacing color, thus achieving the purpose of reducing image distortion due to the color replacement.

Consequently, each color cube as discussed above should be expanded to have $2^n \times 2^m = 2^{m+n}$ colors, instead of just $2^m$ ones, if embedding of $m$-bit message segments is still desired. And because of the property of having three color channels in an RGB image, $m+n$ must be a multiple of 3 for $2^{m+n}$ to be the cube of an integer $M$ (i.e., $M^3$), meaning that each color cube has the side length of $M$. That is, it must be true that $m + n = 3k$ for some positive integer $k$ such that $2^{m+n} = 2^{3k} = (2^k)^3 = (2^k) \times (2^k) \times (2^k) = M^3$ with $M = 2^k$. If not, then the color cluster will not form a cube; instead, it becomes a rectangular parallelepiped (also called a cuboid), which is less convenient to handle due to side asymmetry.

For example, if we take $m = 2$ and $n = 1$, then each color cube has $2^{2+1} = 8$ colors, divided into 4 groups with each group including two colors. One of such color cubes is shown in Figure 5.2, in which the four color groups are $G_1 = \{(0, 0, 0), (1, 1, 1)\}$, $G_2 = \{(1, 0, 0), (0, 1, 1)\}$, $G_3 = \{(1, 1, 0), (0, 1, 1)\}$, $G_4 = \{(0, 1, 0), (1, 0, 1)\}$ and the two colors in each group are located diagonally in opposite directions, where each color is expressed as a 3-tuple ($r$, $g$, $b$) with $r$, $g$, and $b$ being the values of the R, G, and B channels, respectively. Such color cubes are too small to be useful. The color cube adopted for use in the experiment of this study is taken to include $2^{m+n} = 2^{3+3} = 64$ colors with $m = 3$ and $n = 3$, i.e., with 8 groups of 8 colors. Therefore, for 8-bit R, G, and B color channels, there are totally $(256/4) \times (256/4) \times (256/4) = 64^3$ color cubes.

For convenience of discussions, we define a *base color* for each color cube as the one in the cube with the smallest of the summation of the $r$, $g$, and $b$ values. By identifying color cubes with three indexes $i$, $j$, $k$ for the three dimensions of $R$, $G$, and $B$, respectively, it is not difficult to figure out that the base color ($r_i^{\mathrm{b}}$, $g_j^{\mathrm{b}}$, $b_k^{\mathrm{b}}$) for the ($i$, $j$, $k$)-th color cube for $m = 3$ and $n = 3$ may be computed by

$$r_i^{\mathrm{b}} = 4i, \ g_j^{\mathrm{b}} = 4j, \ b_k^{\mathrm{b}} = 4k, \qquad (2)$$

where $i$, $j$, $k$ = 0, 1, …, 63, and the values of the 64 colors in the cube may be computed by

$$\begin{aligned} r &= r_i^{\mathrm{b}},\ r_i^{\mathrm{b}}+1,\ r_i^{\mathrm{b}}+2,\ r_i^{\mathrm{b}}+3; \\ g &= g_j^{\mathrm{b}},\ g_j^{\mathrm{b}}+1,\ g_j^{\mathrm{b}}+2,\ g_j^{\mathrm{b}}+3; \\ b &= b_k^{\mathrm{b}},\ b_k^{\mathrm{b}}+1,\ b_k^{\mathrm{b}}+2,\ b_k^{\mathrm{b}}+3. \end{aligned} \qquad (3)$$

For example, the (0, 0, 0)-th color cube with base color (0, 0, 0) is shown in Table 5.1. Simply adding the base color values ($r_i^{\mathrm{b}}$, $g_j^{\mathrm{b}}$, $b_k^{\mathrm{b}}$) respectively to the color channel values in the table, we can get the table for the ($i$, $j$, $k$)-th color cube.

According to the above idea, we propose the following scheme for reduction of image distortion.

Table 5.1 The colors in the (0, 0, 0)-th color cube with base color ($r$, $g$, $b$) = (0, 0, 0).

| No. | Color | No. | Color | No. | Color | No. | Color |
|---|---|---|---|---|---|---|---|
| 1 | (0, 0, 0) | 17 | (1, 0, 0) | 33 | (2, 0, 0) | 49 | (3, 0, 0) |
| 2 | (0, 0, 1) | 18 | (1, 0, 1) | 34 | (2, 0, 1) | 50 | (3, 0, 1) |
| 3 | (0, 0, 2) | 19 | (1, 0, 2) | 35 | (2, 0, 2) | 51 | (3, 0, 2) |
| 4 | (0, 0, 3) | 20 | (1, 0, 3) | 36 | (2, 0, 3) | 52 | (3, 0, 3) |
| 5 | (0, 1, 0) | 21 | (1, 1, 0) | 37 | (2, 1, 0) | 53 | (3, 1, 0) |
| 6 | (0, 1, 1) | 22 | (1, 1, 1) | 38 | (2, 1, 1) | 54 | (3, 1, 1) |
| 7 | (0, 1, 2) | 23 | (1, 1, 2) | 39 | (2, 1, 2) | 55 | (3, 1, 2) |
| 8 | (0, 1, 3) | 24 | (1, 1, 3) | 40 | (2, 1, 3) | 56 | (3, 1, 3) |
| 9 | (0, 2, 0) | 25 | (1, 2, 0) | 41 | (2, 2, 0) | 57 | (3, 2, 0) |
| 10 | (0, 2, 1) | 26 | (1, 2, 1) | 42 | (2, 2, 1) | 58 | (3, 2, 1) |
| 11 | (0, 2, 2) | 27 | (1, 2, 2) | 43 | (2, 2, 2) | 59 | (3, 2, 2) |
| 12 | (0, 2, 3) | 28 | (1, 2, 3) | 44 | (2, 2, 3) | 60 | (3, 2, 3) |
| 13 | (0, 3, 0) | 29 | (1, 3, 0) | 45 | (2, 3, 0) | 61 | (3, 3, 0) |
| 14 | (0, 3, 1) | 30 | (1, 3, 1) | 46 | (2, 3, 1) | 62 | (3, 3, 1) |
| 15 | (0, 3, 2) | 31 | (1, 3, 2) | 47 | (2, 3, 2) | 63 | (3, 3, 2) |
| 16 | (0, 3, 3) | 32 | (1, 3, 3) | 48 | (2, 3, 3) | 64 | (3, 3, 3) |

1. Define each color cube to have $2^{m+n}$ colors, divided into $2^m$ groups with each group including $2^n$ colors, where $m + n = 3k$ for some positive integer $k$.
2. Assign the colors in the color cube into groups such that the colors in each group are distributed *evenly*, for the purpose of achieving more effectively the goal of reducing image distortion due to color replacements as discussed above.
3. Encode identically all the $2^n$ colors in each group into an $m$-bit message segment, i.e., represent the $m$ message bits identically by any color in the group.
4. When an image pixel $P$ in the range set of a color cube $C$ is to be used for embedding an $m$-bit message segment $H$, find the group $G$ in $C$ whose colors represent the value of $H$.
5. Find the color $c'$ in $G$ which is closest to $c$ in the sense of Euclidean color distance.
6. Replace $c$ by $c'$ to complete the data embedding work at pixel $P$.

In Step 5 above, the Euclidean color distance between the two colors $c = (r, g, b)$ and $c' = (r', g', b')$ are defined to be $|c - c'| = [(r - r')^2 + (g - g')^2 + (b - b')^2]^{1/2}$.

## C. Proposed technique for extraction of embedded data

A merit of the previously-proposed technique of data hiding (including partitioning the color space into non-overlapping color cubes as well as replacing a pixel's color with another one, both in an identical color cube) is the resulting assurance of *data recoverability* in the data extraction stage. There are two reasons which guarantee this merit, as described in the following.

(1) Although some original colors in the cover image have been replaced, each of the replacing colors is in the same color cube as that of the replaced one at an image pixel. This ensures that if we use the pixels' colors in the stego-image to find the range set of each color cube, as is done in the data extraction process, the result will be the same as that found in the data hiding process. This means that the pixels where data were hidden will not be missed in the data extraction process.

(2) Only color cubes with more-scattered range sets are utilized for data embedding,

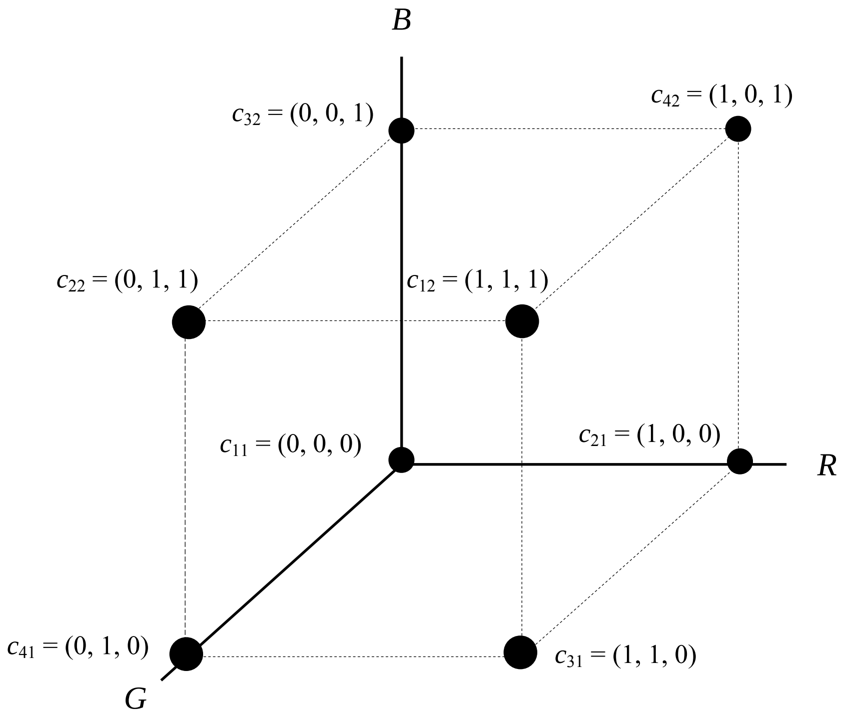


Figure 5.2 A color cube with 8 colors divided into four groups with base color (0, 0, 0).

and so if we select similarly color cubes with more-scattered range sets in the data extraction process, then the same set of color cubes will be found, from whose range sets we can extract exactly the previously-embedded message bits.

The proposed scheme for data extraction is described in the following.

1. Partition the color space in the same way as done in the data embedding process.
2. Collect the range set of each color cube from the pixels in the given stego-image.
3. Compute the scattering degree of the range set of each color cube.
4. Sort the color cubes into a list in the same way as done in the data embedding process described previously.
5. Select a sufficient number of color cubes from the top of the list according to the length of the embedded message.
6. Follow the color encoding rule used in data embedding to decode as a message segment the color of each pixel in the range set of each color cube selected in the last step.
7. Concatenate all the decoded message segments in order into a message as the extraction result.

In Step 5 above, to decide how many color cubes should be selected, the length of the message (in the unit of bit) should be known in advance. For this, we take the message length as part of the data to be hidden and append it to the message data as the prefix, in the form of a *fixed* number of bytes. If the value of the message length, expressed as a bit sequence, is shorter than the length of all the bytes allocated for it, then we pad sufficient leading 0's to it to fill up the bytes. In this way, the message length will be embedded first as a fixed number of bytes into the image, and in the data extraction process it can be extracted first as well from a fixed number of bytes hidden in the stego-image, from which the total number of remaining data bits can be decided, and the message bits extracted properly.

## D. Even distribution of cube colors into groups for image distortion reduction

As mentioned previously, we assign the colors in the color cube into groups such that the colors in each group are distributed *evenly*. Consequently, a color in the group closest to an image pixel's color can be selected for color replacement, in order to reduce the resulting image distortion. Here we describe the technique we use for achieving such a goal of even distribution of colors in groups. First, it is not difficult to see that the desired distributions in the groups should be *symmetric* to each other. To accomplish this, we adopt the following steps, using the first color cube with base color (0, 0, 0) as an example for explanation of the detail. For other color cubes, the corresponding steps are the same except the base color. Table 2 shows the details of the involved computation results in the steps.

1. Take the 64 color values of the color cube as Euclidean coordinates, and compute its centroid, which is (1.5, 1.5, 1.5).
2. Transform the Euclidean coordinates into new ones through a translation of (1.5, 1.5, 1.5).
3. Transform the new Euclidean coordinates $(r, g, b)$ into 3D spherical coordinates $(\rho, \theta, \phi)$ by the following formula:

$$\rho = (r^2 + g^2 + b^2)^{1/2}, \qquad \phi = \tan^{-1}(g/b), \qquad \theta = \tan^{-1}[b/(r^2+g^2)^{1/2}]$$

   where $\rho$ is the distance from the origin to a point in the Euclidean space, $\theta$ is the zenith angle with respect to the $R$-axis, $\phi$ is the azimuth angle with respect to the $B$-axis, as shown in Figure 5.3, and the function $\tan^{-1}$ has values in the range from $-90^\circ$ to $+90^\circ$.
4. To facilitate the purpose of even distribution of group colors, modify the range

of $\tan^{-1}$ such that the computed values of $\theta$ lie in the range $0^{o} \leq \theta < 360^{o}$ with $0^{o}$ indicating the direction of the $R$-axis.

5. Use in order the values of $\rho$, $\phi$, and $\theta$ to sort the 64 colors into a list.
6. Assign the 64 colors of the color cube evenly into the 8 groups using the list according to the following criteria to achieve the goal of even distribution of group colors:

   (1) each group has an equal number of colors which have a certain value of $\rho$;

   (2) the colors of each group have as many angles of $\theta$ as possible;

   (3) the 8 color groups, when seen as grid points, are symmetric to one another.
7. Regard all the 8 colors in each color group to be identical, and encode each group to represent one of the eight 3-bit segments 000 through 111, as mentioned previously.

In Step 6 above, to satisfy Criteria (2) and (3) we normalize the angle values of $\theta$ of all the grid points with respect to each of the angles of "8 selected symmetric points" and listed them for easier selection of appropriate colors into the groups. For the 64-color cubes, these 8 symmetric points may be selected to be the 8 corners of the cube, as done in our experiment. The result of color distribution for the first color cube with base color (0, 0, 0) is shown in Table 2. And an example of the color distribution result for group 3, which includes the corner of (0, 0, 0), is shown in . The assigned 8 colors in the group are (1, 2, 2), (3, 2, 2), (1, 0, 2), (2, 2, 3), (2, 3, 0), (0, 1, 0), (3, 0, 1), (0, 0, 0).

The above process is designed for color cubes with 64 colors. It is not difficult to modify the process to fit more general cases of color cubes with $2^{m+n}$ colors mentioned before.

Furthermore, as an example of data embedding at image pixels, let $P$ be a pixel with color $c = (r, g, b) = (1, 3, 2)$ and assume that the 3-bit message segment we want to embed is 010. The color cube used is that described in Table 2 and the group of colors involved is the third shown in Figure 5.4. The color in the group *closest* to $c$ is $c' = (1, 2, 2)$ with a distance of 1 to $c$. Therefore, the color $c = (1, 3, 2)$ of $P$ is replaced by $c' = (1, 2, 2)$ in the data embedding process.

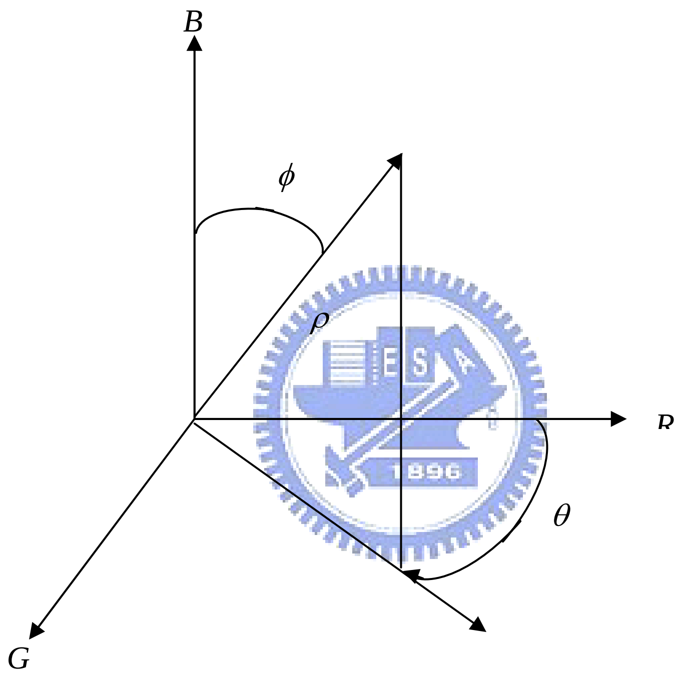


Figure 5.3 Illustration of a 3D spherical coordinate system for use in even color distribution.

As a deeper investigation of the effect of the above even distribution of group colors in a color cube, we tried to compute the value of the peak of the signal-to-noise ratio (PSNR) for the worst case of color replacements, which occurs when the colors of all image pixels are replaced with the most dissimilar colors in color cubes. For this, we have two cases. One is when the colors of each group in a color cube are *not* evenly distributed. Then, the largest Euclidean color distance resulting from a color replacement obviously will be $|(3, 3, 3) - (0, 0, 0)| = (3\times3^2)^{1/2} = \sqrt{27}$. The other case

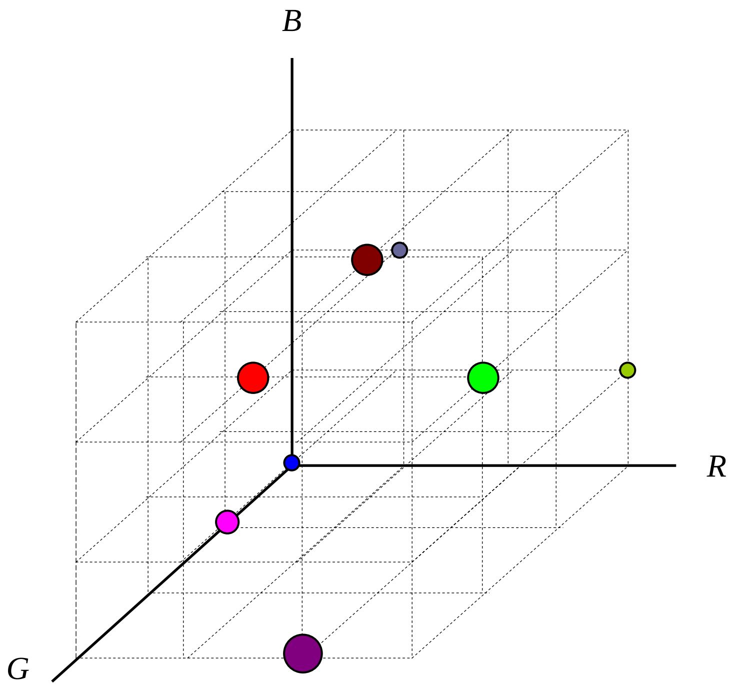


Figure 5.4 An example of color distribution in a color cube --- the 8 colors in group 3.

is when the even distribution is done as shown in Table 5.2. Then, according to a computer program written in this study which computes exhaustively the Euclidean color distances between every pair of colors in the color cube based on the groups of Table 3, the largest Euclidean color distance is $d = \sqrt{4}$ .

Accordingly, for the 2nd case the maximum mean-square error (MSE) for the stego-image may be computed to be $MSE_{\max} = d^2/3 = 4/3$, and the corresponding worst PSNR value is $PSNR_{\min} = 10 \times \log[255^2/MSE_{\max}] = 10\times\log[65025/(4/3)] \approx 46.88$ dB which is quite high. In contrast, the former case has $PSNR_{\min} = 10\times\log[65025/(27/3)] \approx 38.59$ dB which is lower. In short, the 2nd case, which is what we have implemented in this study, has less image distortion.

Table 5.2 Color encoding table for the (0, 0, 0)-th color cube with base color (0, 0, 0).

| $r$ | $g$ | $b$ | $\rho$ | $\phi$ (degree) | $\theta$ (degree) | group | code |
|---|---|---|---|---|---|---|---|

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 1 | 2 | 0.9 | 35 | 315 | 1 | 000 |
| 2 | 3 | 2 | 1.7 | 18 | 72 | | |
| 0 | 1 | 2 | 1.7 | 18 | 198 | | |
| 1 | 1 | 3 | 1.7 | 65 | 225 | | |
| 3 | 2 | 0 | 2.2 | -43 | 18 | | |
| 1 | 0 | 0 | 2.2 | -43 | 252 | | |
| 0 | 3 | 1 | 2.2 | -13 | 135 | | |
| 3 | 3 | 0 | 2.6 | -35 | 45 | | |
| 2 | 2 | 2 | 0.9 | 35 | 45 | 2 | 001 |
| 0 | 2 | 2 | 1.7 | 18 | 162 | | |
| 2 | 0 | 2 | 1.7 | 18 | 288 | | |
| 2 | 1 | 3 | 1.7 | 65 | 315 | | |
| 1 | 3 | 0 | 2.2 | -43 | 108 | | |
| 3 | 1 | 0 | 2.2 | -43 | 342 | | |
| 0 | 0 | 1 | 2.2 | -13 | 225 | | |
| 0 | 3 | 0 | 2.6 | -35 | 135 | | |
| 1 | 2 | 2 | 0.9 | 35 | 135 | 3 | 010 |
| 3 | 2 | 2 | 1.7 | 18 | 18 | | |
| 1 | 0 | 2 | 1.7 | 18 | 252 | | |
| 2 | 2 | 3 | 1.7 | 65 | 45 | | |
| 2 | 3 | 0 | 2.2 | -43 | 72 | | |
| 0 | 1 | 0 | 2.2 | -43 | 198 | | |
| 3 | 0 | 1 | 2.2 | -13 | 315 | | |
| 0 | 0 | 0 | 2.6 | -35 | 225 | | |
| 1 | 1 | 2 | 0.9 | 35 | 225 | 4 | 011 |
| 1 | 3 | 2 | 1.7 | 18 | 108 | | |
| 3 | 1 | 2 | 1.7 | 18 | 342 | | |
| 1 | 2 | 3 | 1.7 | 65 | 135 | | |
| 0 | 2 | 0 | 2.2 | -43 | 162 | | |
| 2 | 0 | 0 | 2.2 | -43 | 288 | | |
| 3 | 3 | 1 | 2.2 | -13 | 45 | | |
| 3 | 0 | 0 | 2.6 | -35 | 315 | | |
| 2 | 1 | 1 | 0.9 | -35 | 315 | 5 | 100 |
| 1 | 1 | 0 | 1.7 | -65 | 225 | | |
| 2 | 3 | 1 | 1.7 | -18 | 72 | | |
| 0 | 1 | 1 | 1.7 | -18 | 198 | | |
| 0 | 3 | 2 | 2.2 | 13 | 135 | | |
| 3 | 2 | 3 | 2.2 | 43 | 18 | | |
| 1 | 0 | 3 | 2.2 | 43 | 252 | | |
| 3 | 3 | 3 | 2.6 | 35 | 45 | | |
| 2 | 2 | 1 | 0.9 | -35 | 45 | 6 | 101 |
| 2 | 1 | 0 | 1.7 | -65 | 315 | | |
| 0 | 2 | 1 | 1.7 | -18 | 162 | | |
| 2 | 0 | 1 | 1.7 | -18 | 288 | | |
| 0 | 0 | 2 | 2.2 | 13 | 225 | | |
| 1 | 3 | 3 | 2.2 | 43 | 108 | | |
| 3 | 1 | 3 | 2.2 | 43 | 342 | | |
| 0 | 3 | 3 | 2.6 | 35 | 135 | | |
| 1 | 2 | 1 | 0.9 | -35 | 135 | 7 | 110 |
| 2 | 2 | 0 | 1.7 | -65 | 45 | | |
| 3 | 2 | 1 | 1.7 | -18 | 18 | | |
| 1 | 0 | 1 | 1.7 | -18 | 252 | | |
| 3 | 0 | 2 | 2.2 | 13 | 315 | | |
| 2 | 3 | 3 | 2.2 | 43 | 72 | | |
| 0 | 1 | 3 | 2.2 | 43 | 198 | | |
| 0 | 0 | 3 | 2.6 | 35 | 225 | | |
| 1 | 1 | 1 | 0.9 | -35 | 225 | 8 | 111 |
| 1 | 2 | 0 | 1.7 | -65 | 135 | | |
| 1 | 3 | 1 | 1.7 | -18 | 108 | | |
| 3 | 1 | 1 | 1.7 | -18 | 342 | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 3 | 2 | 2.2 | 13 | 45 | | |
| 0 | 2 | 3 | 2.2 | 43 | 162 | | |
| 2 | 0 | 3 | 2.2 | 43 | 288 | | |
| 3 | 0 | 3 | 2.6 | 35 | 315 | | |

## 5.2 Detailed Algorithms of Proposed Data Embedding and Extraction

We now describe the detailed algorithms for data embedding and extraction. We assume that the maximum length of given messages to be embedded is $B$ bytes ($8B$ bits) long.

**Algorithm 5.1** ***Data embedding process.***

**Input:** a cover image $I$, a message $G$ in the form of a bit string, and the color encoding tables (like Table 2) for color cubes with 64 colors defined by Eqs. (2) and (3).

**Output:** a stego-image $I'$ with $G$ embedded.

**Steps:**

***A. Finding the range sets of the color cubes ---***

1. Find the range set $S_i$ from the cover image $I$ for each color cube $C_i$.
2. Compute the scattering degree $M_i$ of each $C_i$ by Eq. (1).
3. Sort all non-empty $S_i$ into a list $L$ according to their values of $M_i$ with the top of the list corresponding to the largest $M_i$.

***B. Creating extended message data ---***

4. Pad 0's, if necessary, to the front of the bit string representing the length of message $G$ so that the resulting bit string, $T$, occupies $B$ bytes.
5. Concatenate $T$ and $G$ in order, to form a third string $T'$.
6. Count the number of bits in $T'$, append 0's to the end of $T'$, if necessary, to

make the total number $N$ of bits a multiple of 3, call the resulting bit string an *extended message*, and denote it by $G'$.

***C. Embedding of message data ---***

7. Regard all the pixels in each range set $S_j$ in $L$ in the raster-scan order as a sequence $Q_j$, and concatenate all sequences of $Q_j$ in order into a longer one $Q$.
8. Embed sequentially every 3-bit segment $H$ of $G'$ into pixels in $Q$ in order in the following way, until all bits of $G'$ are exhausted:
   (1) take sequentially an unprocessed pixel $P$ in $Q$ with color $c$;
   (2) find out the color cube $C$ whose range set includes $P$;
   (3) find out the color group $p$ of $C$, whose corresponding code is equal to $H$;
   (4) find out the color $c'$ in $p$ which is *closest* to $c$ in the sense of Euclidean color distance;
   (5) replace $c$ of $P$ by $c'$ in the cover image.

The data extraction process is described as an algorithm in the following. We assume the embedded data in the given stego-image is the extended message $G'$ mentioned in the previous algorithm, which includes the original message $G$ preceded by the value of the length of $G$ in the form of $B$ bytes.

**Algorithm 5.2** ***Data extraction process.***

**Input:** a stego-image $I'$, and the color encoding tables (like Table 2) for color cubes with 64 colors defined by Eqs. (2) and (3).

**Output:** the message $G$.

**Steps:**

***A. Finding the range sets of the color cubes ---***

1. Find the range set $S_i$ from the stego image $I'$ for each color cube $C_i$.
2. Compute the scattering degree $M_i$ of each $C_i$ by Eq. (1).

3. Sort all non-empty $S_i$ into a list $L$ according to their values of $M_i$ with the top of the list corresponding to the largest $M_i$.

***B. Extracting the length of the message***

4. Regard all the pixels in each range set $S_j$ in $L$ in the raster-scan order as a sequence $Q_j$, and concatenate all sequences of $Q_j$ in order into a longer one $Q$.
5. Extract $B$ bytes of data from $Q$ first to obtain the length $N$ of the message $G$ in the following way:
   (1) take sequentially an unprocessed pixel $P$ in $Q$ with color $c'$;
   (2) find out the color cube $C$ whose range set includes $P$;
   (3) find out the color group $p$ of $C$, which includes $c'$;
   (4) find out the 3-bit code corresponding to $p$;
   (5) repeat the above steps until the concatenation of all the found 3-bit codes in order, denoted as $K$, is *just more than* $B$ bytes long;
   (6) take the first $B$ bytes of $K$ and convert it into an integer as the message length $N$, and the tail portion $R$ in $K$ as the leading bits of the message $G$.

***C. Extracting the message data***

6. Compute $N' = \lceil N/3 \rceil$ where $\lceil \cdot \rceil$ means the ceiling function.
7. Repeating the following steps $N'$ times:
   (1) take sequentially an unprocessed pixel $P$ in $Q$ with color $c'$;
   (2) find out the color cube $C$ whose range set includes $P$;
   (3) find out the color group $p$ of $C$, which includes $c'$;
   (4) find out the 3-bit code of $p$;
8. Concatenate $R$ extracted in Step 5 and all the codes extracted in Step 7 in order as a bit string, and take the first $N$ bits of it as the desired message $G$.

## 5.3 Experiment Results and Discussions

A series of experiments have been conducted in this study on BMP images. Some experimental results are shown in Figs. 5 through 8. Figure 5.5 is a continuation of Figure 5.1. Figure 5.5(a) shows the stego-image resulting from embedding 22900 bytes of message data into the cover image shown in Figure 5.1(a) which is of the size 256×256. And Figure 5.5(b) shows the difference between Figure 5.1(a) and Figure 5.5(a) as a color image $I''$ (called a *difference image*), which is produced in the following way, assuming that ($r$, $g$, $b$) is a color in the cover image $I$, ($r'$, $g'$, $b'$) the corresponding color in the stego-image $I'$, and ($r''$, g'', $b''$) the computed difference color:

$$\begin{aligned} x'' &= |x - x'| + 128 && \text{if } |x - x'| \neq 0; \\ &= 255 && \text{if } |x - x'| = 0, \end{aligned}$$

where $x = r$, $g$, or $b$. The concept behind the above computation is to set a difference value of 0 to be 255 and a non-zero one to be around 128. Consequently, an unprocessed pixel with three zero difference values will become a white pixel in the difference image $I''$, while a processed pixel will have a color ($r''$, $g''$, $b''$) with all the three color channel values around 128. As can be seen from Figure 5.5(b), most of the pixels in the cover image have been utilized for data embedding, but the stego-image looks almost identical to the cover image of Figure 5.1(a) due to the effectiveness of image distortion and change noticeability reduction. It can also be observed from Figure 5.5(b) that the processed pixels are quite random in their locations, and more uniform regions, like those on the clothes, yield range sets with smaller scatter measures, as expected, which are not used for data embedding (seen as white-pixel clusters in the figure). The rate of processed pixels (called *processed pixel rate* in the sequel) is (22900×8) ÷ 3 ÷ (256×256) ≈ 0.932 and the PSNR value was computed to be 48.59 dB which is better than the worse-case value 46.88 dB, as it should be. Totally, 1628 color cubes have been utilized.

Figure 5.6 shows another experimental result with a 256×256 cover image. The processed pixel rate is again 0.932, the computed PSNR value is 48.23 dB, and the number color cubes used is 5242. A similar phenomenon of leaving uniform regions unused for data embedding is observed (most on the flowers at the lower part of the cover image). For illustrations, we also include the range set of a color cube as Figure 5.6(b). Two more examples of experimental results with 512×512 cover images are shown in Figs. 7 and 8. The message data embedded are 88200 bytes long, and the processed pixel rates are (88200×8) ÷ 3 ÷ (512×512) ≈ 0.897, for both cases. The PSNR values are 48.70 dB and 48.27 dB, respectively.

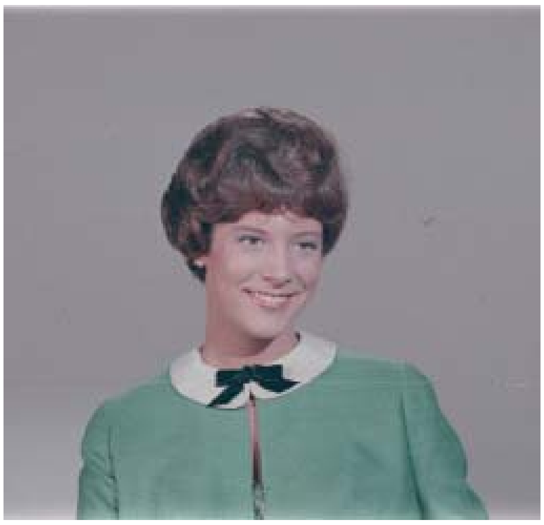
More statistics data about our experiments are shown in Table 5.3, in which images 4.1.03, 4.1.01, 4.2.04, 4.2.07 are those in Figs. 5.5 through 5.8, respectively. All the images come from the USC image database. From the table, we see that the PSNR values of all the stego-images are over 48 dB.

The experiments were conducted for color cubes with 64 colors and color groups of 8 colors. Color cubes and color groups of sizes other than those used in the experiments of this study may also be applied for various application needs. In general, larger-sized color cubes will lead to larger embedding capacity of each color replacement (that is, more bits are encoded by each replacing color) if the size of each color group is fixed.

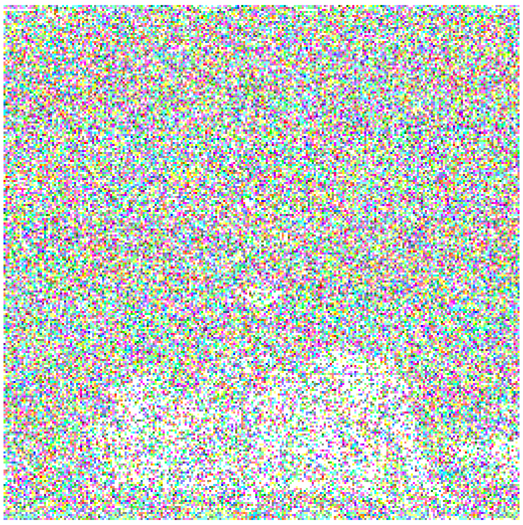
Table 5.3 Statistics of experimental results.

| No. | Image | Size of image (pixels) | Size of message data (bytes) | Processed pixel rate | No. of used color cubes | PSNR (dB) |
|---|---|---|---|---|---|---|
| 1 | 4.1.01 | 256×256 | 22900 | 0.932 | 5242 | 48.23 |
| 2 | 4.1.02 | 256×256 | 22900 | 0.932 | 3329 | 48.49 |
| 3 | 4.1.03 | 256×256 | 22900 | 0.932 | 1628 | 48.59 |
| 4 | 4.1.05 | 256×256 | 22900 | 0.932 | 3840 | 48.68 |
| 5 | 4.2.01 | 512×512 | 88200 | 0.879 | 5514 | 48.36 |
| 6 | 4.2.02 | 512×512 | 88200 | 0.879 | 5446 | 49.19 |
| 7 | 4.2.04 | 512×512 | 88200 | 0.879 | 9908 | 48.70 |
| 8 | 4.2.05 | 512×512 | 88200 | 0.879 | 6626 | 48.61 |
| 9 | 4.2.06 | 512×512 | 88200 | 0.879 | 17093 | 48.60 |
| 10 | 4.2.07 | 512×512 | 88200 | 0.879 | 17110 | 48.27 |
| 11 | House | 512×512 | 88200 | 0.879 | 16048 | 48.68 |

On the other hand, with the size of the color cube being fixed, larger-sized color groups, though reducing more distortion caused by color replacements, will lead to less embedding capability (that is, less bits are encoded by each color group). The original cover image is not needed in data recovery, so the proposed method is a blind scheme. The PSNR values of the stego-images constructed in the experiments are high, showing that the aim of image distortion reduction carried out by the use of color groups is accomplished. The stego-images look almost identical to the cover images, showing that another aim of reducing color change noticeability is also reached. Furthermore, secret keys may be used to randomize the message data before they are embedded into the cover image or/and randomize the sequence of pixels (sequence *Q* in Algorithms 5.1 and 5.2) into which the data are embedded, in order to enhance data security. Illegal recovery of the embedded data will so obtain just a sequence of noise. The proposed method is thus appropriate for uses in steganographic applications.

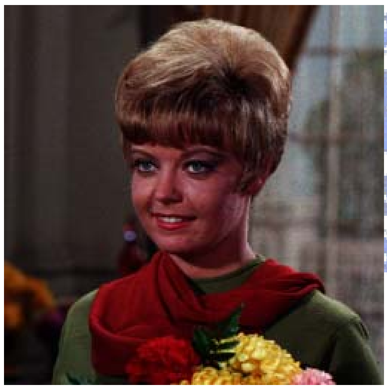


(a) Stego-image.



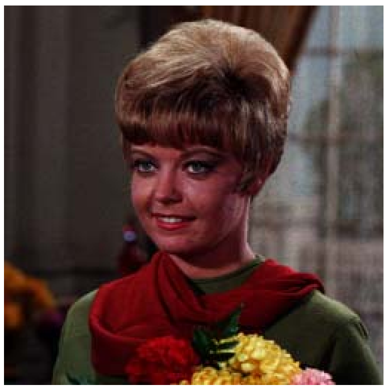
(b) Difference image.

Figure 5.5 An experimental result of message data embedding applied to Figure 5.1(a) with a 256×256 cover image and a 22900-byte message data.

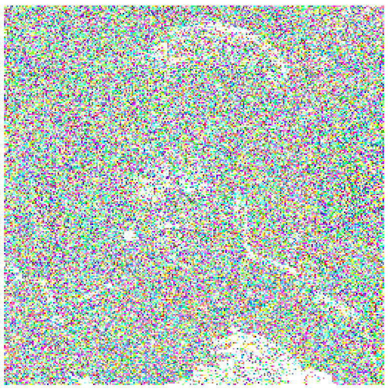


(a) Cover image.

(b) Range set of a color cube.

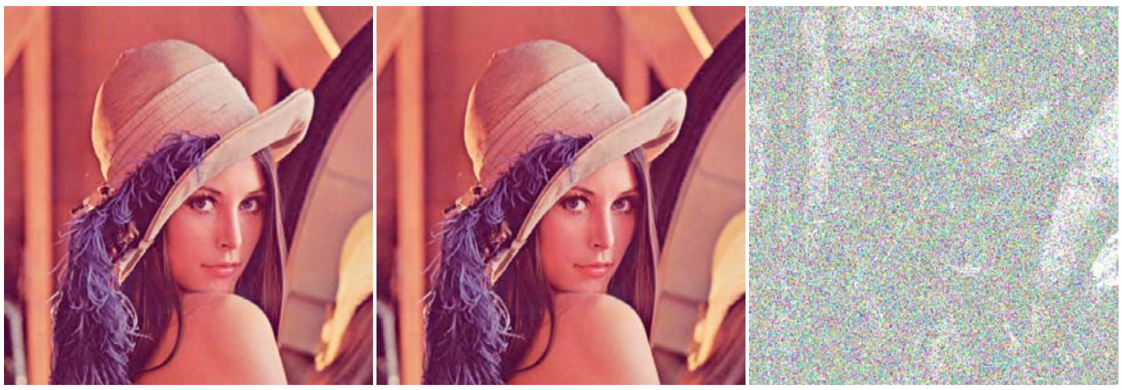


(c) Stego-image.



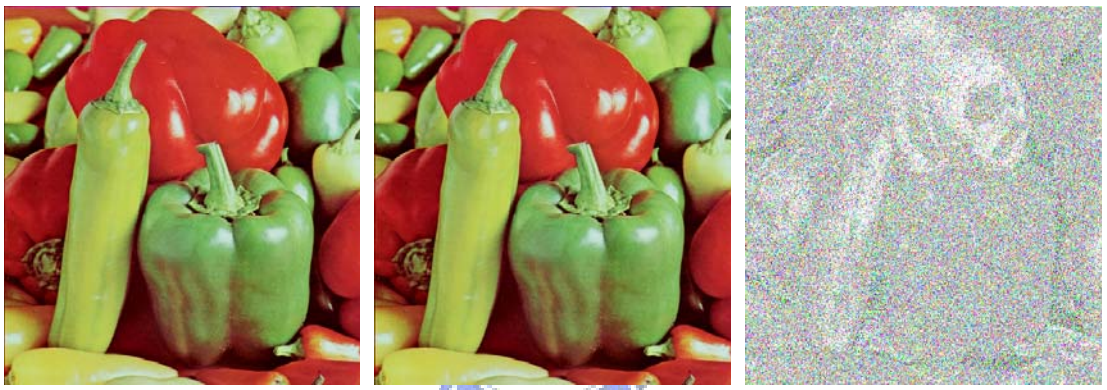
(d) Difference image.

Figure 5.6 A second experimental result with a 256×256 cover image and a 22900-byte message.



(a) Cover image. (b) Stego-image. (c) Difference image.

Figure 5.7 A third experimental result of data embedding with a 512×512 cover image and an 88200-byte message.



(a) Cover image. (b) Stego-image. (c) Difference image.

Figure 5.8 A fourth experimental result of data embedding with a 512×512 cover image and an 88200-byte message.

## 5.4 Concluding Remarks

A novel method for hiding large-volume message data in RGB images has been proposed. The method is based on the idea of changing selected image pixels' colors by similar ones which encode the message bits. The replacing colors come from some selected color cubes in the color space, and the image pixels come from the range sets of the color cubes. Data recoverability is ensured by the use of color cubes and range sets. The color cubes are selected in such a way that the pixels in their range sets are as separated as possible. This reduces the noticeability caused by the color changes. Each replacing color comes from the choice of an optimal one from a group of evenly distributed colors in a color cube. This reduces the resulting image distortion due to the color replacements.

Experimental results show the feasibility of the proposed method for large-volume data hiding as well as the effectiveness of reducing image distortion and change noticeability. The method is a blind data hiding technique; the original cover image is not required in the data extraction process. Future researches may be directed to dynamic uses of variable-sized color cubes, random distributions of groups' colors in color cubes, uses of the proposed method for various applications, etc.

# Chapter 6
# Data Hiding in Emails and Applications by Unused ASCII Control Codes

## 6.1 Idea of Proposed Method

ASCII codes, usually expressed as hexadecimal numbers, are used very commonly to represent text for information interchange on computers. Parts of the ASCII codes, namely, from 00 through 1F, are used as *control codes* which are listed in Table 1. They were originally designed to control computer peripheral devices like printers, tape drivers, teletypes, etc. But now they are rarely used for their original purpose because of the rapid development of new peripheral hardware technologies, except those codes for text display control, such as 0A with the meaning of *line feed* and 08 with the meaning of *backspace*. Besides, some of the control codes, when displayed by a text editing program or a browser on monitors, are *invisible*; and some others are shown as *spaces* under certain software environments, just like the function of the original ASCII space code 20. These two types of ASCII codes may be utilized to increase secret data encoding variability in the data hiding process. For convenience of reference, we say that the former type displays a *null space*, in contrast with the *white space* displayed by the latter type.

On the other hand, as computer technology spreads throughout the world, many coding standards have been developed to facilitate the expression of non-English alphabets. But these alphabet coding standards, such as the Unicode and the Big 5, all include the ASCII codes as the kernel set. For example, the popular Unicode standard, UTF-8, equates exactly to the ASCII codes for code values below 128. Therefore, the good property of the ASCII control codes for embedding secret data in text documents is still preserved in various coding standards.

In this study, it is desired to use the white-space and null-space codes to embed data in text documents of the Unicode UTF-8 format without causing noticeable artifacts under the popular software environments of Outlook Express, IE, and the operating system of the traditional Chinese version of Microsoft Windows XP, service pack 2, 2002.

In the remainder of this chapter, some properties of email systems and embedding ASCII control codes into emails are described in Sections 6.2, and 6.3, respectively. The proposed methods for data hiding and recovery processes for emails are introduced in Sections 6.4 and 6.5 respectively. Some experimental results are shown in Section 6.7, followed by some concluding remarks in Section 6.8.

## 6.2 Properties of Email Systems

In this study, it is assume that all emails are transmitted through the popular Simple Mail Transfer Protocol (SMTP) [38-40] and that users retrieve their emails from remote server systems of the Post Office Protocol version 3 (POP3) standard [41]. In addition, most emails nowadays are of the Multipurpose Internet Mail Extensions (MIME) format [42-44] which is compatible with the SMTP standard.

However, some mail server systems do not follow the SMTP standard precisely [44]. Therefore, before we make use of an email document for data embedding, we must find out servers which do not change the content of an email body, or must set up a new SMTP server. Otherwise, data embedded in the email might be destroyed before being read and retrieved on the server of the receiver end.

According to the SMTP standard [40], According to the SMTP standard The codes 0D for carriage return (CR) and 0A for line feed (LF) must appear together as 0D0A (denoted as CRLF in the sequel) for use at the end of each line. A text line, if *folded*, should be limited to be 78 characters in length, excluding CRLF. Here, by *folding* we mean to split a long text line into multiple shorter ones. A folding will occur when a CRLF is inserted in a line to replace a space, separating the line into two parts.

Table 6.1 ASCII control codes and description.

| **Dec** | **Hex** | **Char** | **Description** | **Dec** | **Hex** | **Char** | **Description** |
|---|---|---|---|---|---|---|---|
| 0 | 0 | NUL | null character | 16 | 10 | DLE | data link escape |
| 1 | 1 | SOH | start of header | 17 | 11 | DC1 | device control 1 |
| 2 | 2 | STX | start of text | 18 | 12 | DC2 | device control 2 |
| 3 | 3 | ETX | end of text | 19 | 13 | DC3 | device control 3 |
| 4 | 4 | EOT | end of transmission | 20 | 14 | DC4 | device control 4 |
| 5 | 5 | ENQ | enquiry | 21 | 15 | NAK | negative acknowledge |
| 6 | 6 | ACK | acknowledge | 22 | 16 | SYN | synchronize |
| 7 | 7 | BEL | bell (ring) | 23 | 17 | ETB | end transmission block |
| 8 | 8 | BS | backspace | 24 | 18 | CAN | cancel |
| 9 | 9 | HT | horizontal tab | 25 | 19 | EM | end of medium |
| 10 | A | LF | line feed | 26 | 1A | SUB | substitute |
| 11 | B | VT | vertical tab | 27 | 1B | ESC | escape |
| 12 | C | FF | form feed | 28 | 1C | FS | file separator |
| 13 | D | CR | carriage return | 29 | 1D | GS | group separator |
| 14 | E | SO | shift out | 30 | 1E | RS | record separator |
| 15 | F | SI | shift in | 31 | 1F | US | unit separator |

Outlook Express, after being opened, often has a smaller window for viewing the mail content. The window width is about 70 characters. In this study, we propose to hide secret data in an email by adding ASCII control codes at the end of each text line with the resulting line being *of this width*, such that when the resulting stego-email is opened by Outlook Express, the mail body can fit the window width, thus increasing the steganographic effect. For this aim, we fold the original email lines into shorter ones, each being 65 characters in length, leaving 5 characters at each line end as a data embedding *slot*.

Another popular protocol by which emails are accessed on a server is the Internet Message Access Protocol version 4 (IMAP4) [45]. The IMAP4 supports single web-mail servers and permits manipulations of mailboxes as remote message folders in a way that is functionally equivalent to local folders. Web mails enjoy its popularity because people can use the same client software to both surf the Internet and transmit/receive emails. And IE is probably the most popular browser for manipulating web mails. In this study, we assume that Outlook Express 6.0 and IE 6.0 are used as the client software to manipulate emails.

## 6.3 Embedding ASCII Control Codes into Emails

In this study, we identify five possible ways for secret data embedding in emails by use of ASCII control codes. They are listed as follows.

(1) *White-space coding* --- As mentioned previously, there are many different white-space codes, each of which, when displayed, appears to be a white space, yielding the same effect as the original ASCII space code 20. For example, under the environment of the Big 5 standard using Outlook Express, each of the three ASCII codes, 07, 09, and 0C, will be displayed as a white space, as found in this study. Therefore, we can use each of them to replace a white space in an email text in a data hiding process, with the resulting stego-email bringing no reader's notice.

(2) *Inserting multiple white-space codes at text line ends* --- We may place multiple white-space codes before the CRLF at the end of a text line. Since no character but *background* white spaces are shown after the CRLF, these additionally inserted white-space codes, though displayed as *visible* white spaces, will be connected to the background white spaces and thus bring no noticeable effect to

the reader.

(3) *Null-space coding* --- As mentioned previously, there are many null-space codes, which are displayed as *nothing*. We can thus insert them *at any position* in a line *for any repetitions* in a data hiding process without causing the reader's notice. For example, under the environment of the UTF-8 standard using IE, the four null-space codes 1C, 1D, 1E, and 1F, as found in this study, are invisible.

(4) *Inserting multiple null-space codes at text line ends* --- We may place null-space codes repetitively at the end of a text line without causing noticeable effect because they are invisible when displayed, as in the case of (2) above.

(5) *Combining techniques of the above* --- We may combine the above techniques in arbitrary ways if both white-space and null-space coding are applicable in the environment.

In the above discussions, we see that the ASCII control codes usable for embedding secret data are *variant* for different kinds of servers, browsers, and character sets. In order to have a systematic investigation in this aspect, in this study we created an email file which includes all ASCII control codes shown in Table 1 to find out SMTP server software suitable for data embedding, as well as the corresponding appearances of the ASCII control codes after they are processed and displayed in the environment of such server software. The investigation results are described as follows.

First, we have found four SMTP email servers which do not change the text contents of emails, and so can be used as *standard* SMTP servers for the purpose of data embedding in this study. Their uniform resource locators (URLs) are http://cis.nctu.edu.tw, http://mis.tsint.edu.tw, http://tw.yahoo.com and http://www.hotmail.com. The first is located in the Department of Computer Science at National Chiao Tung University in Taiwan, with an SMTP software of Twig 2.7.7. The department has additionally another SMTP server system, Horde, for web mails. The second server is located at the Department of Management Information at Technology and Science Institute of Northern Taiwan. The SMTP software is SendMail 8.12. The third server is located in Taiwan and deals with web mails with the name Yahoo! Mail. The last server is Hotmail, a web mail server of Microsoft Corporation. After registering at any of these four servers, a user may read, transmit, or receive emails by Outlook Express or IE.

In this study, the email format we use is MIME 1.0, the content-type is text/plain, and the character set is UTF-8. These formats are very commonly used and so are adopted in this study for data hiding applications.

After a systematic test of the ASCII character set on the above-mentioned four servers, we found that the hexadecimal ASCII control codes appropriate for data embedding *under both the Outlook Express and the IE environments* are 1C, 1D, 1E, and 1F. These four codes *all* appear to be invisible on the IE browser, and *all* are shown as white spaces in the Outlook Express window. They can so be used for data embedding *respectively* according to the techniques of (2) and (4) mentioned above. However, our goal is to take into account *simultaneously*, instead of respectively, the techniques of (2) and (4), resulting in a method of repeatedly placing these four ASCII control codes *at the ends of email text lines*. The displayed result of the stego-email will be of no difference from the appearance of the original cover email, thus achieving the steganographic effect.

More specifically, we use the following encoding rules to embed secret data into the text line ends of a cover email.

1. Encode 2-bit binary secret data “00,” “01,” “10,” and “11” with the four ASCII codes 1C, 1D, 1E, and 1F, respectively.

2. Put the unique combined ASCII codes 201E in front of a sequence of secret data as its *start signal*, and append another copy of it at the sequence tail as the *end signal*.
3. Use the unique combined ASCII codes 201C to encode the 1-bit data ‘0,’ and the combined codes 201D to encode ‘1.’
4. Use the unique combined ASCII codes 201F as a *separator* to stop the underline display that starts from a special lexical token of the network protocol, like http, ftp, email, …, etc.

Rule 4 above is necessary because otherwise the extra white-space codes we insert at the end of a text line, when happening to be connected to the end of a network protocol text line, will appear to be *underlined* white spaces, like in http://cis.nctu.edu.tw                    , which obviously are against the purpose of steganography.

Based on the above rules, we describe the proposed data hiding algorithm for the purpose of covert communication and authentication in the next section.

## 6.4 Proposed Data Hiding Process for Emails

We first describe the technique we propose to embed secret data into an email as Algorithm 6.1 below, and then describe how to transmit the stego-email by Outlook Express or IE. In the following, when we refer to an email, we mean its text body, excluding the header.

**Algorithm 6.1** ***Data embedding in an email***.

***Input***: a secret data file *S* and a cover email *E* long enough to hide *S*.

***Output***: a stego-email *E'*.

***Steps***:

5. Set the format of the cover email $E$ to MIME 1.0, the content-type to text/plain, and the character set to UTF-8.
6. Fold sequentially each long text line in $E$ with over 65 characters into a 65-character line by inserting a CRLF to replace the first space code 20 *found backward from the 65th character breakpoint* in the line.
7. Check every line in the resulting $E$ to see if there exists in it any special lexical token of the network protocol right before the CRLF; if so, insert a separator code 201F before the CRLF so that we can insert secret data in between the separator code and the CRLF, as described next.
8. Get a text line from $E$, starting from the first, and perform the following operations.

   4.1 Insert the start signal 201E before the CRLF which appears at the line end.

   4.2 Compute the *embedding capacity EC* between the start signal and the CRLF in the following way:

   $$EC = 70 - \textit{position of CRLF in the text line},$$

   which means the number of secret data bits we can insert before the CRLF until the line becomes 70 characters long and should not be made longer, as discussed previously.

   4.3 Perform one of the following three cases (assuming that $|S|$ means the length of $S$):

   (1) if $EC \neq 0$ and $|S| > 1$, then get a pair of bits from the prefix of $S$, encode it with the corresponding code (one of 1C, 1D, 1E, 1F), insert the result before the CRLF, decrement $EC$ by 1, decrement $|S|$ by 2, and perform Step 4.3 again;

   (2) if $EC = 0$ and $|S| > 1$, then get the next text line in $E$ and perform Step 4.2;

(3) if $|S| \leq 1$, then continue.

9. Check $S$ to see if there still remains a single bit $B$ in $S$. If so, then:

   (1) if $EC \neq 0$, insert the code 201C before the CRLF if $B$ is '0' or the code 201D if $B$ is '1';

   (2) if $EC = 0$, then get a text line in $E$ with nonzero embedding capacity $EC$ and conduct the insertion as in Step 5(1) above.

10. Append the end signal 201E at the end of all the codes inserted in the previous steps.

11. Output the result as the desired stego-email $E'$.

After a stego-mail $E'$ is obtained, we want to send it to the receiver site through Outlook Express or IE as a traditional email or a web mail, respectively. For the former way using Outlook Express, we open a new email, denoted as $E_n$, set the character set of $E_n$ to UTF-8, expand the window size of $E_n$ to the maximum, copy the text body of $E'$ into $E_n$, and finally send the result to the receiver without encrypting it. For the latter way using IE, we use IE to log in the selected web mail server, and do all the same to complete the mail transmission.

## 6.5 Proposed Data Recovery Process for Emails

At the receiver end, after a stego-mail is received by the use of Outlook Express or IE, its content of ASCII codes is checked for secret data extraction. The algorithm for this purpose is described as follows.

**Algorithm 6.2** ***Data Recovery from a stego-email text body***.

***Input***: a stego-email text $E'$, presumably including a secret data file $S$.

***Output***: the file $S$.

***Steps***:

1. Scan separator signals 201F in $E'$ and remove all of them, if there exists any.
2. Scan the resulting $E'$ to find the start signal 201E in $E'$ and remove it
3. Perform the following steps.
   3.1 Get a pair of ASCII codes in order from $E$.
   3.2 If the code pair $P$ is the end signal of 201E, then perform Step 4; otherwise:
      (1) if $P$ is either 201C or 201D, then decode $P$ to be the bit 0 or 1, respectively;
      (2) if $P$ is neither 201C nor 201D, then check each code $Q$ in $P$ and if $Q$ is one of 1C, 1D, 1E, and 1F, then decode $Q$ to get the corresponding secret bit pair (one of 00, 01, 10, and 11) and remove $Q$.
   3.3 Go to Step 3.1.
4. Remove the end signal.
5. Concatenate all the decoded secret data bits extracted in the previous steps into a sequence as the desired secret data file $S$ and exit.

## 6.6 Proposed Authentication Process for Email Documents

The data embedding and extraction techniques proposed previously, in addition to being useful for the purpose of covert communication, may be used for the purpose of email authentication. More specifically, by embedding appropriately-designed codes as an *authentication signal*, the signal, when extracted, can be used to check the *fidelity* of a received email, proving that it was transmitted by a specified server and not tampered with before received. In this study, we achieve this goal by embedding an authentication signal into an email by Algorithm 6.1 to generate an *authenticable* stego-email. The signal is generated by the use of the content of an email by a division operation. The fidelity verification work is accomplished by matching the authentication signal extracted from a given authenticable stego-email with that computed directly from the original text content of the email. The details are described as two algorithms below.

**Algorithm 6.3** ***Generation of an authenticable email***.

***Input***: a cover email $E$ and a secret key $K$.

***Output***: an authenticable email $E'$.

***Steps***:

1. Fold each long text line in $E$ with over 65 characters into a 65-character line by inserting a CRLF code to replace the first space code found backward from the 65th character breakpoint.
2. Compute a value $M$ by summing up all the ASCII code values in the resulting $E$ after excluding all the special codes of 1C, 1D, 1E, 1F, 201C, 201D, 201E, and 201F.
3. Compute an authentication signal $A$ as the remainder of dividing $M$ by the secret key $K$.
4. Use Algorithm 1 to embed $A$ into $E$ to obtain an authenticable email as the desired output $E'$.

In Step 2 above, the reason of excluding the special codes is that these codes are to be used for embedding the authentication signal $A$ in Step 4.

**Algorithm 6.4** ***Authentication of an email***.

***Input***: a stego-email $E'$, presumably including an authentication signal; and a secret key $K$.

***Output***: an authentication message about the fidelity of the displayed text content of $E'$.

***Steps***:

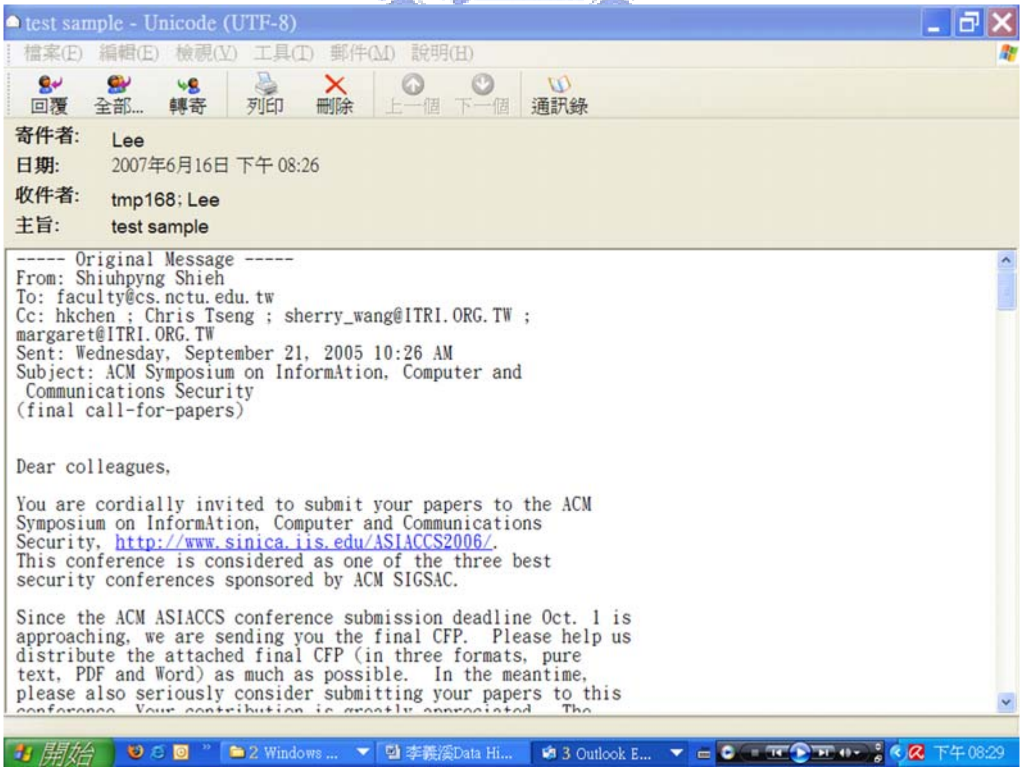
1. Compute a value $M$ by summing up all the ASCII code values in $E'$ after excluding all the special codes of 1C, 1D, 1E, 1F, 201C, 201D, 201E, and 201F.
2. Compute an authentication signal $A$ as the remainder of dividing $M$ by the secret key $K$.
3. Extract the hidden authentication signal $A'$ from $E'$ by Algorithm 2.
4. Compare $A'$ with $A$, and if they are identical, then output the authentication message "pass," meaning the displayed text content of $E'$ is genuine; else, the message "fail," meaning the reverse.

## 6.7 Experimental Results

Figures 6.1 through 6.4 illustrate some experimental results of applying Algorithms 6.1 and 6.2 for covert communication using Outlook Express. Figure 6.1 shows part of the content of a 9.3KB cover email. Figure 6.2 shows part of the content of the stego-email (12.7KB) obtained by applying Algorithm 6.1 with the cover email as the input. This content was displayed with Outlook Express by a receiver with email address tmp168@mis.tsint.edu.tw, to whom the stego-email was sent. From Figure 6.2, we see that no difference can be seen in the stego-email, when it is compared with the cover email. Figure 6.3 shows the content of the 1.07KB secret data file embedded in the stego-email. And Figure 6.4 shows the content of the 1.07KB secret data file extracted from the stego-email shown in Figure 6.2 by applying Algorithm 6.2. The two file contents can be seen to be the same. These results show that the proposed method of data hiding and recovery is feasible.

Figures 6.5 through 6.9 illustrate some additional experimental results of applying the proposed algorithms using IE. All password portions in the emails in these figures were blackened for protecting the privacy of the mail owners. Figure 6.5 shows the content of a 2.42KB cover email. Figure 6.6 shows the content of the corresponding stego-email (2.54KB) generated by Algorithm 1. Figure 6.7 shows part of the content of the stego-email seen as a web mail in IE at a receiver site with address gis87809@cis.nctu.edu.tw. Figure 6.8 shows the content of the original secret data file with 27 bytes. Figure 6.9 shows the content of the secret data file that was extracted from the stego-email shown in Figure 6.7. Again, the original and the extracted secret data are seen identical.

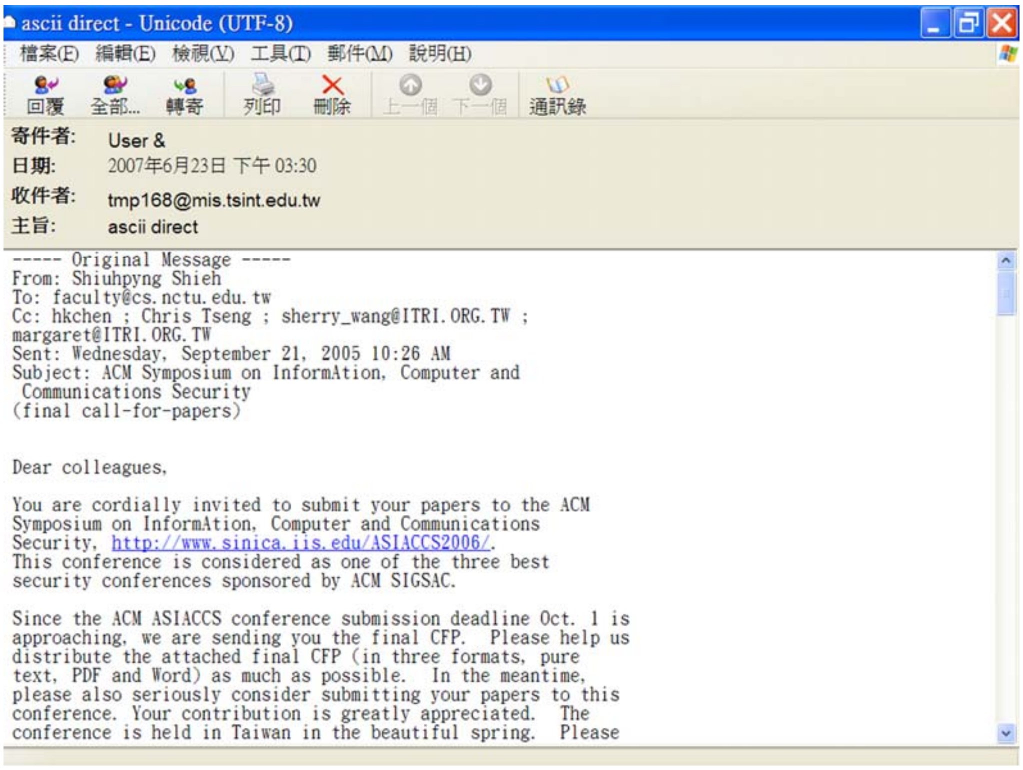
The experiments presented above were conducted under the condition that the transmitter's and the receiver's operations were performed on the same server. Actually, we also conducted experiments in which the transmitter's and receiver's operations were performed on difference servers. For example, one server we used was the mail server at Yahoo! in Taiwan, and the other a mail server in the Department of Computer Science at National Chiao Tung University in Taiwan. The results remained unchanged.


test sample - Unicode (UTF-8)
檔案(F) 編輯(E) 檢視(V) 工具(T) 郵件(M) 說明(H)
回覆 全部... 轉寄 列印 刪除 上一個 下一個 通訊錄
開始
2 Windows ...
李義溪Data Hi...
3 Outlook E...
下午 08:29

寄件者: Lee
日期: 2007年6月16日 下午 08:26
收件者: tmp168; Lee
主旨: test sample

```
----- Original Message -----
From: Shiuhpyng Shieh
To: faculty@cs.nctu.edu.tw
Cc: hkchen ; Chris Tseng ; sherry_wang@ITRI.ORG.TW ;
margaret@ITRI.ORG.TW
Sent: Wednesday, September 21, 2005 10:26 AM
Subject: ACM Symposium on InformAtion, Computer and
 Communications Security
(final call-for-papers)


Dear colleagues,

You are cordially invited to submit your papers to the ACM
Symposium on InformAtion, Computer and Communications
Security, http://www.sinica.iis.edu/ASIACCS2006/.
This conference is considered as one of the three best
security conferences sponsored by ACM SIGSAC.

Since the ACM ASIACCS conference submission deadline Oct. 1 is
approaching, we are sending you the final CFP.  Please help us
distribute the attached final CFP (in three formats, pure
text, PDF and Word) as much as possible.  In the meantime,
please also seriously consider submitting your papers to this
conference.  Your contribution is greatly appreciated.  The
```

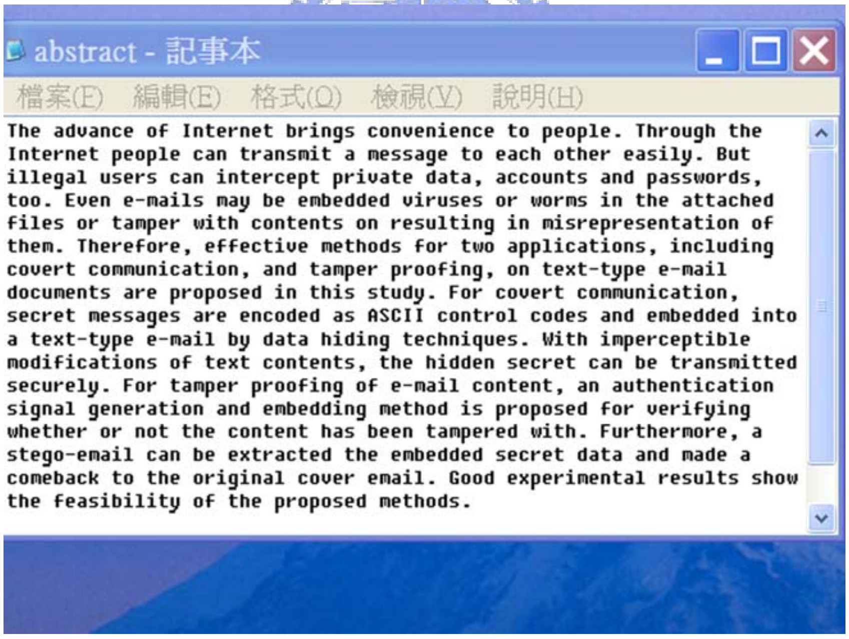
Figure 6.1 Partial content of a cover email.


ascii direct - Unicode (UTF-8)
檔案(F) 編輯(E) 檢視(V) 工具(T) 郵件(M) 說明(H)
回覆 全部... 轉寄 列印 刪除 上一個 下一個 通訊錄

寄件者: User &
日期: 2007年6月23日 下午 03:30
收件者: tmp168@mis.tsint.edu.tw
主旨: ascii direct

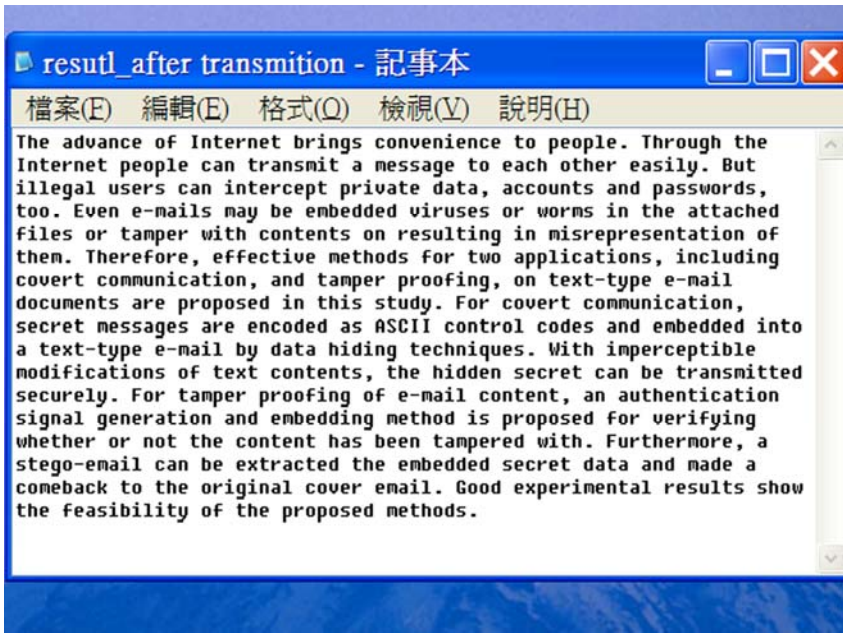
```
----- Original Message -----
From: Shiuhpyng Shieh
To: faculty@cs.nctu.edu.tw
Cc: hkchen ; Chris Tseng ; sherry_wang@ITRI.ORG.TW ;
margaret@ITRI.ORG.TW
Sent: Wednesday, September 21, 2005 10:26 AM
Subject: ACM Symposium on InformAtion, Computer and
 Communications Security
(final call-for-papers)


Dear colleagues,

You are cordially invited to submit your papers to the ACM
Symposium on InformAtion, Computer and Communications
Security, http://www.sinica.iis.edu/ASIACCS2006/.
This conference is considered as one of the three best
security conferences sponsored by ACM SIGSAC.

Since the ACM ASIACCS conference submission deadline Oct. 1 is
approaching, we are sending you the final CFP.  Please help us
distribute the attached final CFP (in three formats, pure
text, PDF and Word) as much as possible.  In the meantime,
please also seriously consider submitting your papers to this
conference. Your contribution is greatly appreciated.  The
conference is held in Taiwan in the beautiful spring.  Please
```

Figure 6.2 Partial content of the stego-email generated from Figure 6.1.



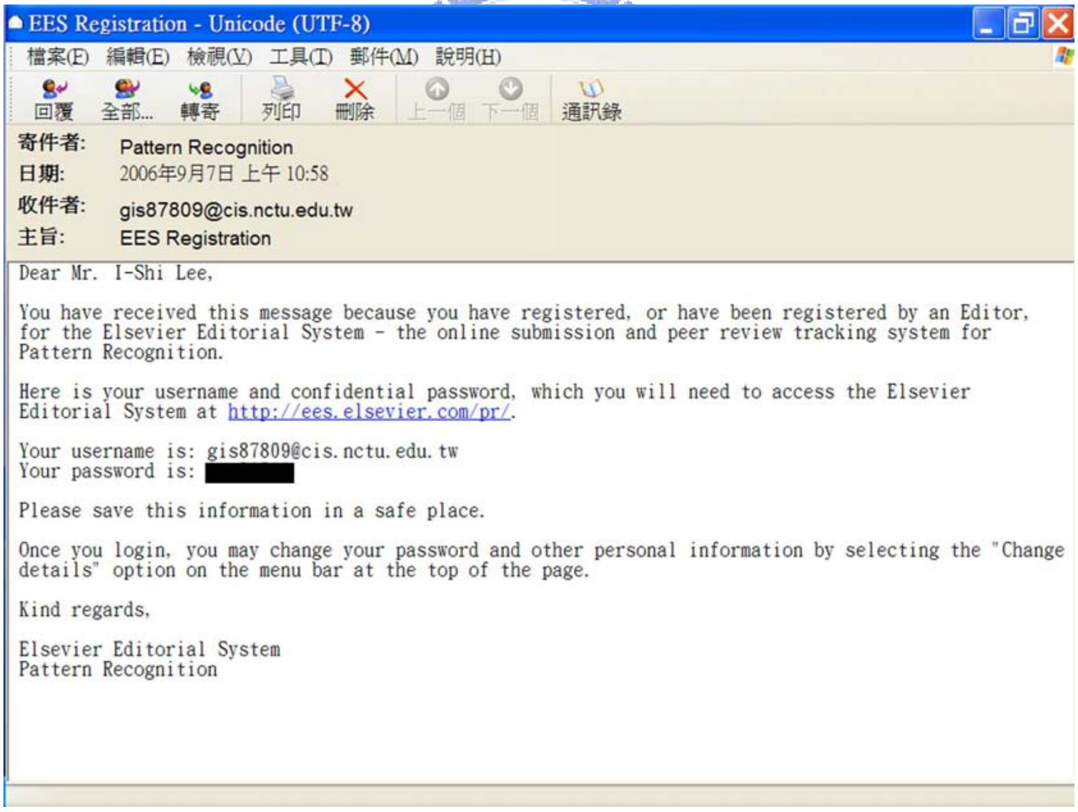
```
The advance of Internet brings convenience to people. Through the
Internet people can transmit a message to each other easily. But
illegal users can intercept private data, accounts and passwords,
too. Even e-mails may be embedded viruses or worms in the attached
files or tamper with contents on resulting in misrepresentation of
them. Therefore, effective methods for two applications, including
covert communication, and tamper proofing, on text-type e-mail
documents are proposed in this study. For covert communication,
secret messages are encoded as ASCII control codes and embedded into
a text-type e-mail by data hiding techniques. With imperceptible
modifications of text contents, the hidden secret can be transmitted
securely. For tamper proofing of e-mail content, an authentication
signal generation and embedding method is proposed for verifying
whether or not the content has been tampered with. Furthermore, a
stego-email can be extracted the embedded secret data and made a
comeback to the original cover email. Good experimental results show
the feasibility of the proposed methods.
```

Figure 6.3 Partial content of an embedded secret data file.


resutl_after transmition - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

```
The advance of Internet brings convenience to people. Through the
Internet people can transmit a message to each other easily. But
illegal users can intercept private data, accounts and passwords,
too. Even e-mails may be embedded viruses or worms in the attached
files or tamper with contents on resulting in misrepresentation of
them. Therefore, effective methods for two applications, including
covert communication, and tamper proofing, on text-type e-mail
documents are proposed in this study. For covert communication,
secret messages are encoded as ASCII control codes and embedded into
a text-type e-mail by data hiding techniques. With imperceptible
modifications of text contents, the hidden secret can be transmitted
securely. For tamper proofing of e-mail content, an authentication
signal generation and embedding method is proposed for verifying
whether or not the content has been tampered with. Furthermore, a
stego-email can be extracted the embedded secret data and made a
comeback to the original cover email. Good experimental results show
the feasibility of the proposed methods.
```

Figure 6.4 Partial content of the extracted secret data file.


EES Registration - Unicode (UTF-8)
檔案(F) 編輯(E) 檢視(V) 工具(T) 郵件(M) 說明(H)
回覆
全部...
轉寄
列印
刪除
上一個
下一個
通訊錄

寄件者: Pattern Recognition
日期: 2006年9月7日 上午 10:58
收件者: gis87809@cis.nctu.edu.tw
主旨: EES Registration

Dear Mr. I-Shi Lee,

You have received this message because you have registered, or have been registered by an Editor, for the Elsevier Editorial System - the online submission and peer review tracking system for Pattern Recognition.

Here is your username and confidential password, which you will need to access the Elsevier Editorial System at http://ees.elsevier.com/pr/.

Your username is: gis87809@cis.nctu.edu.tw
Your password is: [redacted]

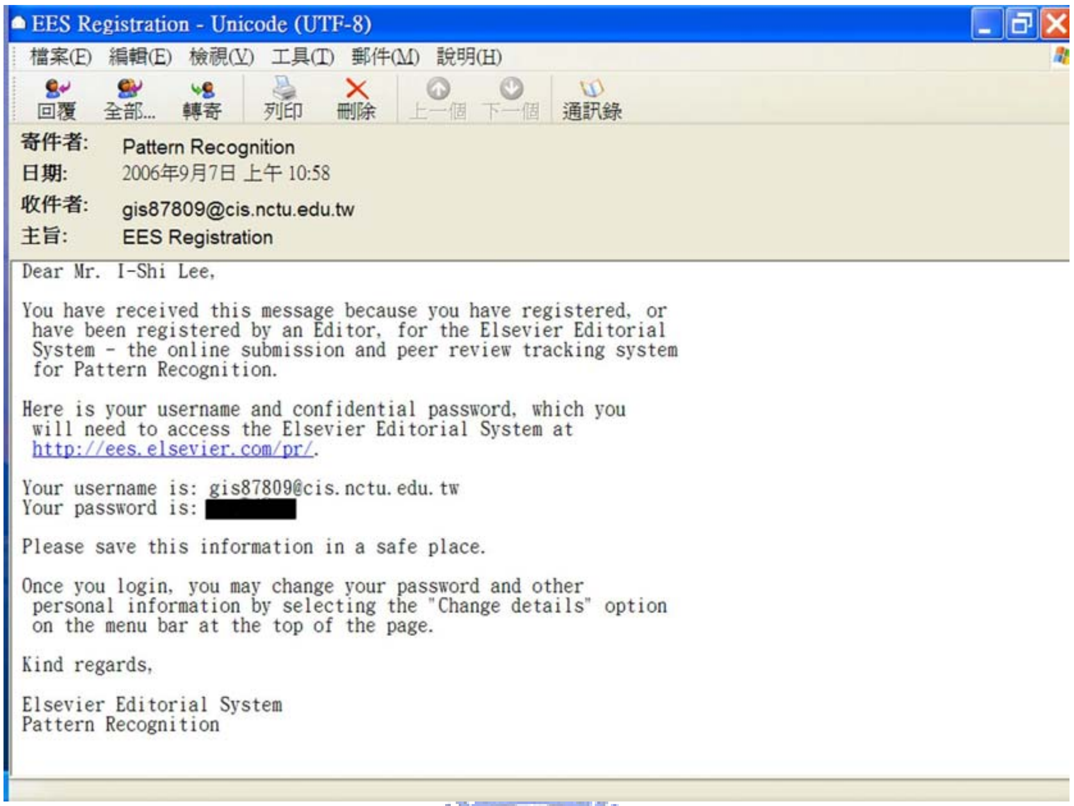
Please save this information in a safe place.

Once you login, you may change your password and other personal information by selecting the "Change details" option on the menu bar at the top of the page.

Kind regards,

Elsevier Editorial System
Pattern Recognition

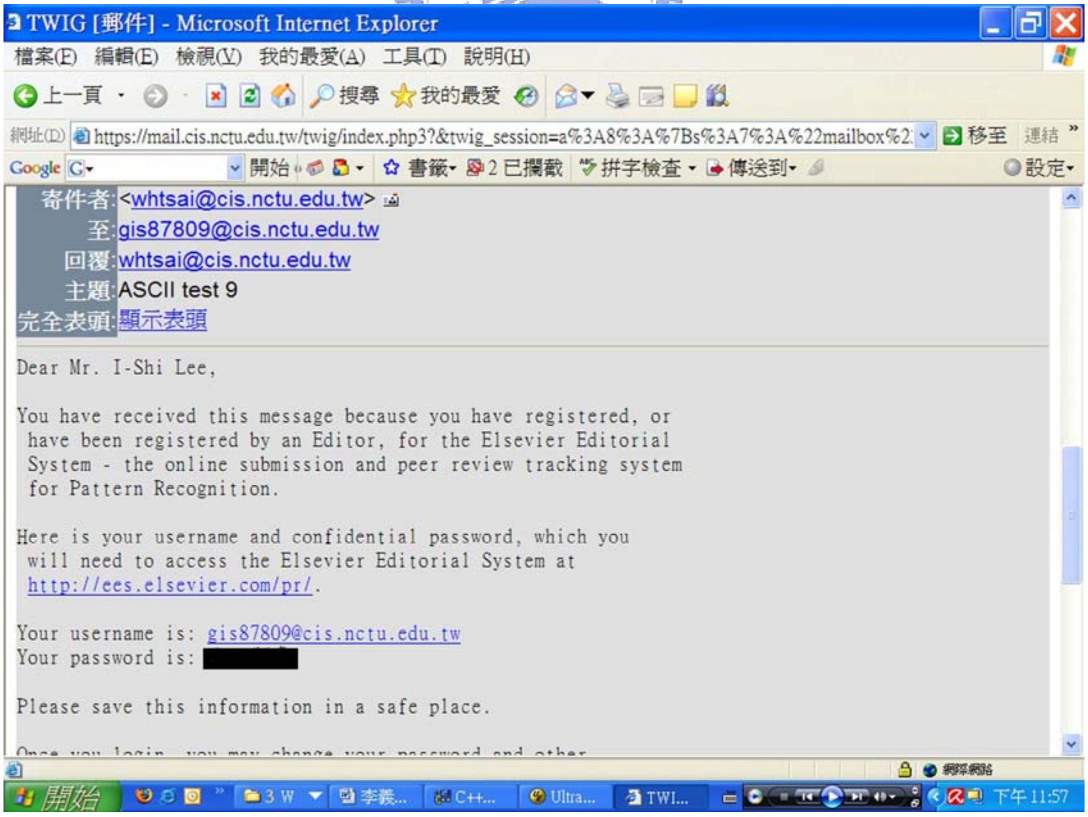
Figure 6.5 Partial content of a cover email.


EES Registration - Unicode (UTF-8)
檔案(F) 編輯(E) 檢視(V) 工具(T) 郵件(M) 說明(H)
回覆 全部... 轉寄 列印 刪除 上一個 下一個 通訊錄

寄件者: Pattern Recognition
日期: 2006年9月7日 上午 10:58
收件者: gis87809@cis.nctu.edu.tw
主旨: EES Registration

```
Dear Mr. I-Shi Lee,

You have received this message because you have registered, or
 have been registered by an Editor, for the Elsevier Editorial
 System - the online submission and peer review tracking system
 for Pattern Recognition.

Here is your username and confidential password, which you
 will need to access the Elsevier Editorial System at
 http://ees.elsevier.com/pr/.

Your username is: gis87809@cis.nctu.edu.tw
Your password is: [redacted]

Please save this information in a safe place.

Once you login, you may change your password and other
 personal information by selecting the "Change details" option
 on the menu bar at the top of the page.

Kind regards,

Elsevier Editorial System
Pattern Recognition
```

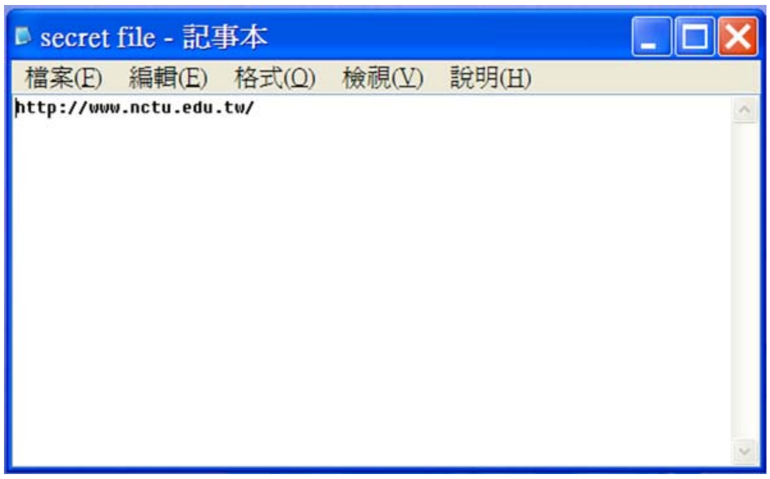
Figure 6.6 Partial content of the stego-email generated from Figure 6.5 before being transmitted.


TWIG [郵件] - Microsoft Internet Explorer
檔案(F) 編輯(E) 檢視(V) 我的最愛(A) 工具(T) 說明(H)
上一頁 搜尋 我的最愛
網址(D) https://mail.cis.nctu.edu.tw/twig/index.php3?&twig_session=a%3A8%3A%7Bs%3A7%3A%22mailbox%2
移至 連結
Google 開始 書籤 2 已攔截 拼字檢查 傳送到 設定
開始 下午 11:57

寄件者: <whtsai@cis.nctu.edu.tw>
至: gis87809@cis.nctu.edu.tw
回覆: whtsai@cis.nctu.edu.tw
主題: ASCII test 9
完全表頭: 顯示表頭

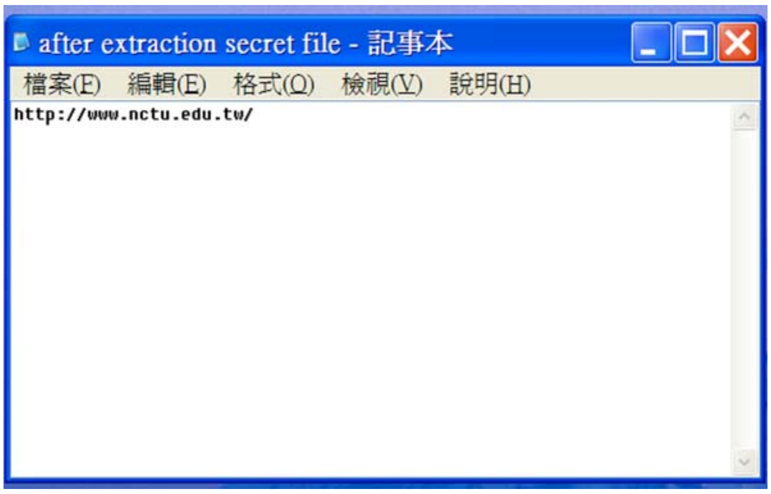
```
Dear Mr. I-Shi Lee,

You have received this message because you have registered, or
 have been registered by an Editor, for the Elsevier Editorial
 System - the online submission and peer review tracking system
 for Pattern Recognition.

Here is your username and confidential password, which you
 will need to access the Elsevier Editorial System at
 http://ees.elsevier.com/pr/.

Your username is: gis87809@cis.nctu.edu.tw
Your password is: [redacted]

Please save this information in a safe place.
```

Figure 6.7 Partial content of the stego-email received and displayed in IE.



```
http://www.nctu.edu.tw/
```

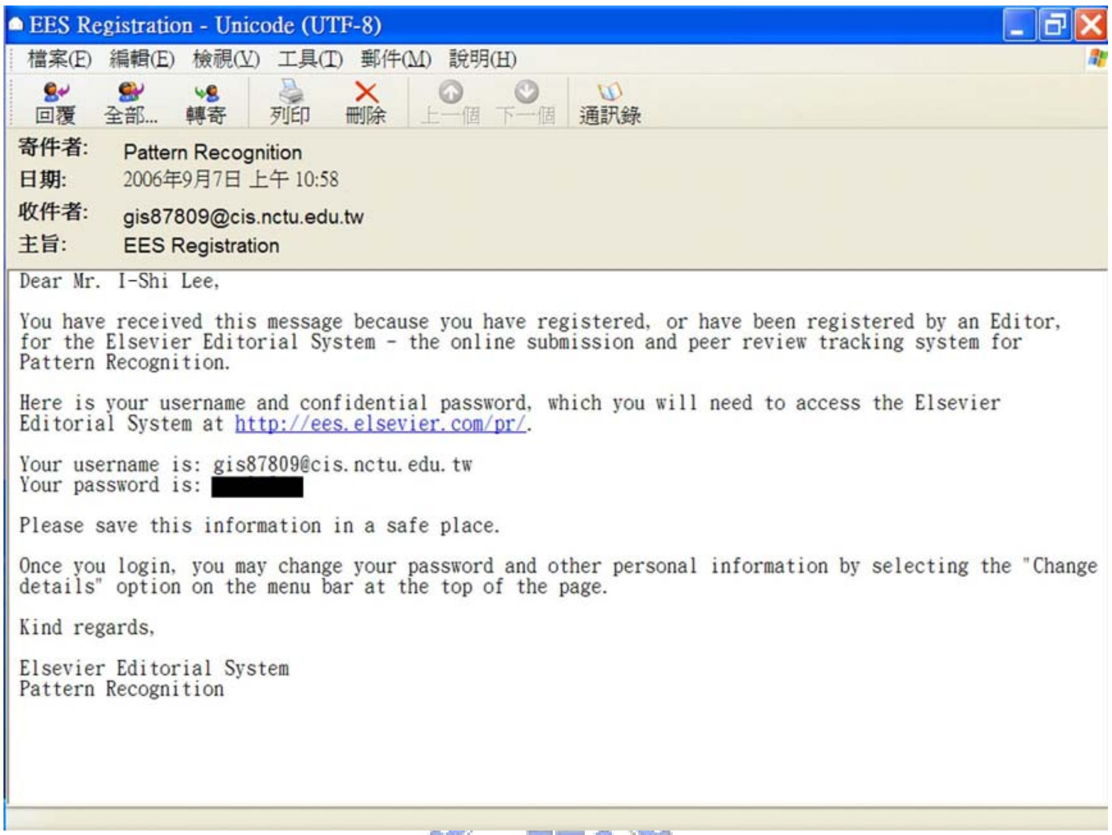
Figure 6.8 Content of the original secret file.


after extraction secret file - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

```
http://www.nctu.edu.tw/
```

Figure 6.9 Content of the extracted secret file.

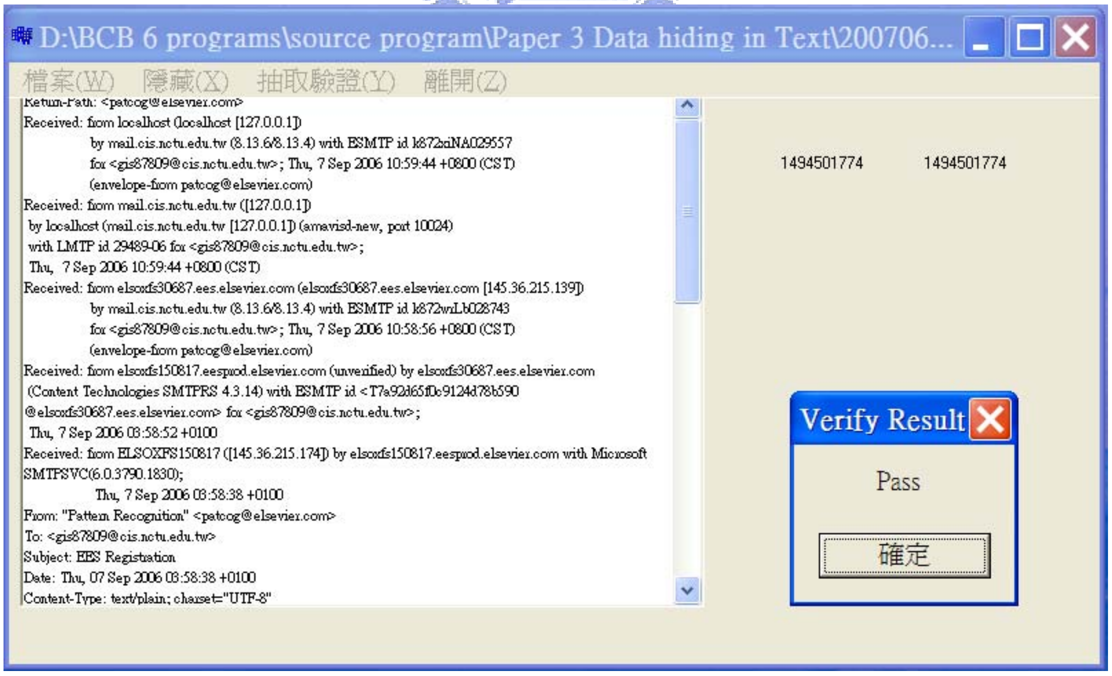
Figures 6.10 to 6.13 illustrate some experimental results of applying the proposed email authentication method. Figure 6.10 shows the content of a stego-email which was generated by Algorithm 6.3. The password portion in the stego-email was also blackened for protecting privacy. The embedded secret data are invisible to a casual reader. Figure 6.11 shows part of the content of the stego-email file after being received by Outlook Express, and the authentication result of "pass".


EES Registration - Unicode (UTF-8)
檔案(F) 編輯(E) 檢視(V) 工具(T) 郵件(M) 說明(H)
回覆 全部... 轉寄 列印 刪除 上一個 下一個 通訊錄

寄件者: Pattern Recognition
日期: 2006年9月7日 上午 10:58
收件者: gis87809@cis.nctu.edu.tw
主旨: EES Registration

```
Dear Mr. I-Shi Lee,

You have received this message because you have registered, or have been registered by an Editor,
for the Elsevier Editorial System - the online submission and peer review tracking system for
Pattern Recognition.

Here is your username and confidential password, which you will need to access the Elsevier
Editorial System at http://ees.elsevier.com/pr/.

Your username is: gis87809@cis.nctu.edu.tw
Your password is: [redacted]

Please save this information in a safe place.

Once you login, you may change your password and other personal information by selecting the "Change
details" option on the menu bar at the top of the page.

Kind regards,

Elsevier Editorial System
Pattern Recognition
```

Figure 6.10 Content of a stego-email for authentication before transmission.

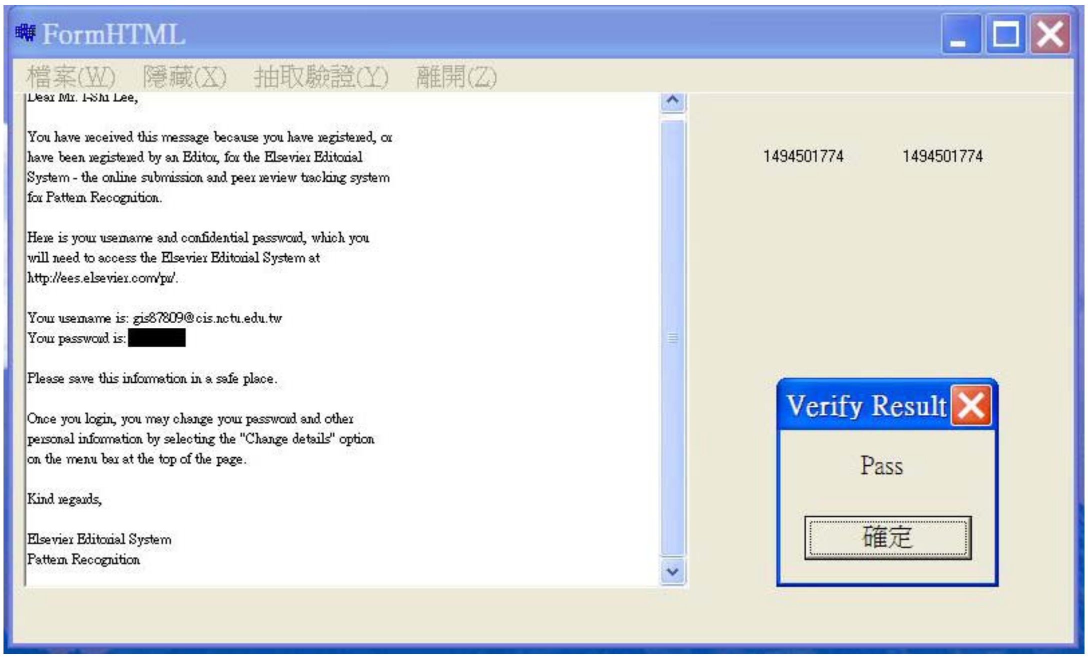

D:\BCB 6 programs\source program\Paper 3 Data hiding in Text\200706...
檔案(W) 隱藏(X) 抽取驗證(Y) 離開(Z)
Verify Result
確定

```
Return-Path: <patcog@elsevier.com>
Received: from localhost (localhost [127.0.0.1])
            by mail.cis.nctu.edu.tw (8.13.6/8.13.4) with ESMTP id k872xiNA029557
            for <gis87809@cis.nctu.edu.tw>; Thu, 7 Sep 2006 10:59:44 +0800 (CST)
            (envelope-from patcog@elsevier.com)
Received: from mail.cis.nctu.edu.tw ([127.0.0.1])
 by localhost (mail.cis.nctu.edu.tw [127.0.0.1]) (amavisd-new, port 10024)
 with LMTP id 29489-06 for <gis87809@cis.nctu.edu.tw>;
 Thu,  7 Sep 2006 10:59:44 +0800 (CST)
Received: from elsoxfs30687.ees.elsevier.com (elsoxfs30687.ees.elsevier.com [145.36.215.139])
            by mail.cis.nctu.edu.tw (8.13.6/8.13.4) with ESMTP id k872wLb028743
            for <gis87809@cis.nctu.edu.tw>; Thu, 7 Sep 2006 10:58:56 +0800 (CST)
            (envelope-from patcog@elsevier.com)
Received: from elsoxfs150817.eesprod.elsevier.com (unverified) by elsoxfs30687.ees.elsevier.com
 (Content Technologies SMTPRS 4.3.14) with ESMTP id <T7a92d65f0c9124d78b590
@elsoxfs30687.ees.elsevier.com> for <gis87809@cis.nctu.edu.tw>;
 Thu, 7 Sep 2006 03:58:52 +0100
Received: from ELSOXFS150817 ([145.36.215.174]) by elsoxfs150817.eesprod.elsevier.com with Microsoft
SMTPSVC(6.0.3790.1830);
             Thu, 7 Sep 2006 03:58:38 +0100
From: "Pattern Recognition" <patcog@elsevier.com>
To: <gis87809@cis.nctu.edu.tw>
Subject: EES Registration
Date: Thu, 07 Sep 2006 03:58:38 +0100
Content-Type: text/plain; charset="UTF-8"
```

```
1494501774        1494501774
```

Pass

Figure 6.11 Authentication result of “pass” after receiving a stego-email by Outlook Express.



Dear Mr. I-Shi Lee,

You have received this message because you have registered, or have been registered by an Editor, for the Elsevier Editorial System - the online submission and peer review tracking system for Pattern Recognition.

Here is your username and confidential password, which you will need to access the Elsevier Editorial System at http://ees.elsevier.com/pr/.

Your username is: gis87809@cis.nctu.edu.tw
Your password is: [redacted]

Please save this information in a safe place.

Once you login, you may change your password and other personal information by selecting the "Change details" option on the menu bar at the top of the page.

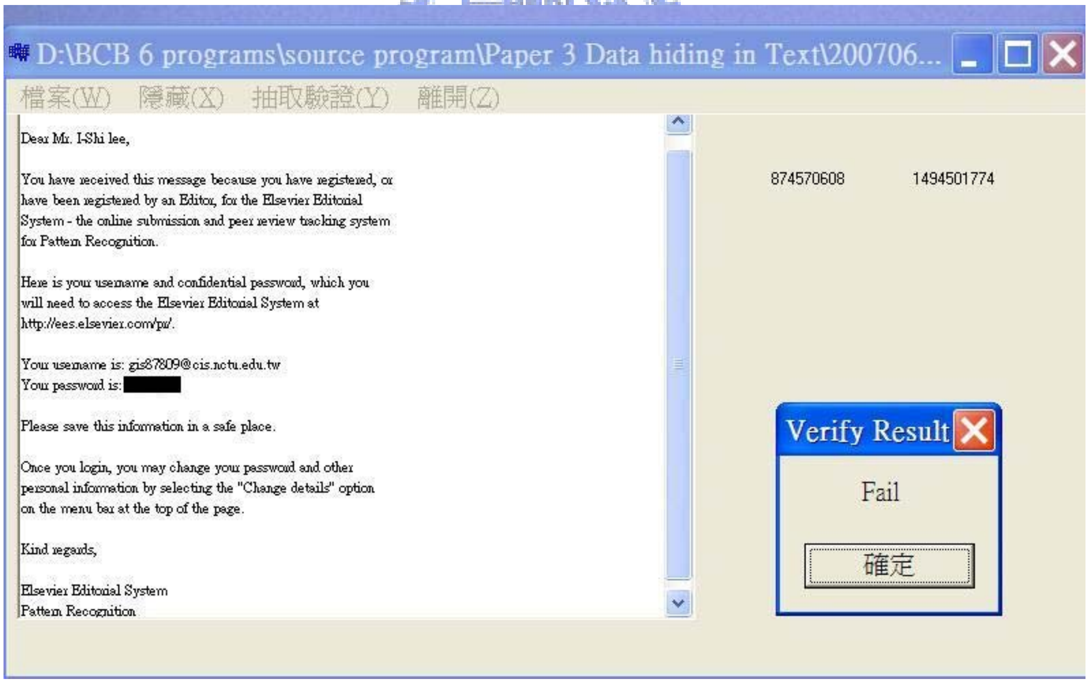
Kind regards,

Elsevier Editorial System
Pattern Recognition

```
1494501774        1494501774
```

Pass

Figure 6.12. Authentication result of “pass” after receiving a stego-email by IE.


D:\BCB 6 programs\source program\Paper 3 Data hiding in Text\200706...
檔案(W) 隱藏(X) 抽取驗證(Y) 離開(Z)
Verify Result
確定

Dear Mr. I-Shi lee,

You have received this message because you have registered, or have been registered by an Editor, for the Elsevier Editorial System - the online submission and peer review tracking system for Pattern Recognition.

Here is your username and confidential password, which you will need to access the Elsevier Editorial System at http://ees.elsevier.com/pr/.

Your username is: gis87809@cis.nctu.edu.tw
Your password is: [redacted]

Please save this information in a safe place.

Once you login, you may change your password and other personal information by selecting the "Change details" option on the menu bar at the top of the page.

Kind regards,

Elsevier Editorial System
Pattern Recognition

```
874570608        1494501774
```

Fail

Figure 6.13. Authentication result of “fail” after receiving the stego-email by IE. The word “Lee” in the content has been modified to be “lee.”

Figure 6.12 shows part of the content of the stego-email after being received by

IE, and the authentication result of “pass”, too. Figure 6.13 shows part of the content of the stego-email file after being received by IE, and the authentication result of “fail,” since the content has been tampered with (the word “Lee” has been changed to “lee”). These results show that the proposed email authentication method is effective.

## 6.8 Concluding Remarks

In this study, we propose a method to embed secret data into emails via the use of the ASCII codes under the operating system of the traditional Chinese version of Microsoft Windows XP, service pack 2, 2002. After a systematic test of all the ASCII codes on various email server software systems and standards, four special ASCII control codes 1C, 1D, 1E, and 1F have been found to be invisible at the line ends of email texts on the SMTP email server in the environment of Outlook Express or IE. A technique has been proposed to utilize these special codes to encode secret data, which is a combination of five coding rules found in this study. Each stego-email can be transmitted to a receiver, and read as a normal email. Extra long lines of emails are folded to be of a proper length for normal displays on email servers to increase steganographic effects. The experiment results prove the feasibility of the proposed method.

In this study, 2-bit secret data are embedded into a white space of a text email. Comparing to other methods proposed by Bender et al. [27] and Chang and Tsai [37] in which on average each secret bit needs 1.5 white spaces to encode (one white space representing a “0,” and two white spaces representing a “1,” leading to the average of $1\times0.5+2\times0.5 = 1.5$ spaces for a secret bit), the proposed method needs only 0.5 white space for each secret bit (one ASCII code representing 2 secret bits), which is an increase of the embedding capacity for three times.

The proposed methods may put into practice in the four servers as listed previously. However, not all mail servers fully follow the SMTP standard. Instead, some mail servers have their own ways of management, like Gmail and Yahoo! Mail, which delete redundant spaces and undefined characters. So, the proposed method is inapplicable to these two servers. Other applicable techniques should be investigated, and are left for further study. Another topic worth future investigation is to apply the proposed data hiding technique to check the integrity of an email, in addition to the fidelity check scheme proposed in this study. Finally, we may extend both the convert communication and authentication works of this study to dealing with web pages.

# Chapter 7
# Security Protection of Software Programs by Information Sharing and Authentication Techniques Using Invisible ASCII Control Codes

## 7.1 Idea of Proposed Method

ASCII codes, usually expressed as hexadecimal numbers, are used very commonly to represent texts for information interchanges on computers. Some of the ASCII codes of 00 through 1F were used as *control codes* to control computer peripheral devices like printers, tape drivers, teletypes, etc. (see Table 7. 1). But now they are rarely used for their original purposes because of the rapid development of new peripheral hardware technologies, except those codes for text display controls, such as 0A and 08 with the meanings of “line feed” and “backspace,” respectively. It is found in this study that some of the ASCII control codes, when displayed by certain text editors under some OS environments, are *invisible*. Such ASCII codes may be utilized for various secret data hiding purposes [53].

The finding of such invisible codes resulted from a systematic test of all the ASCII control codes in the environment of the VC$^{++}$ editor of Microsoft Visual Studio .NET 2003, Service Pack 1. Four of such codes so found are 1C, 1D, 1E, and 1F, which are invisible in the *comments* or *character strings* of VC$^{++}$ programs (see Table 7. 2). Such codes will simply be said *invisible* in subsequent discussions.

As an illustrative example, in Figure 7.1 we show a simple source program in Figure 7.1(a) with a short comment “test a file.” In the comment, we inserted consecutively the four codes 1C, 1D, 1E, and 1F between the letters “s” and “t” in the word “test.” Their existences can be checked with the text editor UltraEdit 32, as can be seen from Figure 7.1(b). But the four codes are invisible in the VC$^{++}$ editor, as can be seen from Figure 7.1(a). Such invisibility usually will arouse no suspicion and so achieve a steganographic effect, since, unless necessary, people will always use the VC$^{++}$ editor for program inspection and development. We utilize such an “invisibility phenomenon” for hiding both share data and authentication signals in source programs in this study, as described in the following.

Table 7.1. ASCII control codes and descriptions.

| Dec | Hex | Char | Description | Dec | Hex | Char | Description |
|---|---|---|---|---|---|---|---|
| 0 | 0 | NUL | null character | 16 | 10 | DLE | data link escape |
| 1 | 1 | SOH | start of header | 17 | 11 | DC1 | device control 1 |
| 2 | 2 | STX | start of text | 18 | 12 | DC2 | device control 2 |
| 3 | 3 | ETX | end of text | 19 | 13 | DC3 | device control 3 |
| 4 | 4 | EOT | end of transmission | 20 | 14 | DC4 | device control 4 |
| 5 | 5 | ENQ | enquiry | 21 | 15 | NAK | negative acknowledge |
| 6 | 6 | ACK | acknowledge | 22 | 16 | SYN | synchronize |
| 7 | 7 | BEL | bell (ring) | 23 | 17 | ETB | end transmission block |
| 8 | 8 | BS | backspace | 24 | 18 | CAN | cancel |
| 9 | 9 | HT | horizontal tab | 25 | 19 | EM | end of medium |
| 10 | A | LF | line feed | 26 | 1A | SUB | substitute |
| 11 | B | VT | vertical tab | 27 | 1B | ESC | escape |
| 12 | C | FF | form feed | 28 | 1C | FS | file separator |
| 13 | D | CR | carriage return | 29 | 1D | GS | group separator |
| 14 | E | SO | shift out | 30 | 1E | RS | record separator |
| 15 | F | SI | shift in | 31 | 1F | US | unit separator |

For the purpose of program sharing among several participants, after a given secret source program is transformed into shares, each share is transformed further into a string of the above-mentioned invisible ASCII control codes, which is then embedded into a corresponding camouflage source program held by a participant. And for the purpose of security protection, authentication signals, after generated, are transformed as well into invisible ASCII control codes before embedded. These two data transformations are based on a binary-to-ASCII mapping proposed in this study, which is described as a table as shown in Table 7. 2, called *invisible character coding table* by regarding each ASCII code as a character.

Table 7. 2 Invisible character coding table.

| Bit pair | Corresponding invisible ASCII code |
|---|---|
| 00 | 1C |
| 01 | 1D |
| 10 | 1E |
| 11 | 1F |

Specifically, after the share and the authentication signal data are transformed into binary strings, the bit pairs 00, 01, 10, and 11 in the strings are encoded into the hexadecimal ASCII control codes 1C, 1D, 1E, and 1F, respectively. To promote security, a secret random key is also used in generating the authentication signal. The details are described in the next section.

In the remainder of this chapter, the secret program sharing and recovery schemes are introduced in Sections 7.2 and 7.3, respectively. The security protection problem is discussed in Section 7.4. Some experimental results are shown in Section 7.5, followed by concluding remarks in Section 7.6.

## 7.2 Proposed Program Sharing Scheme

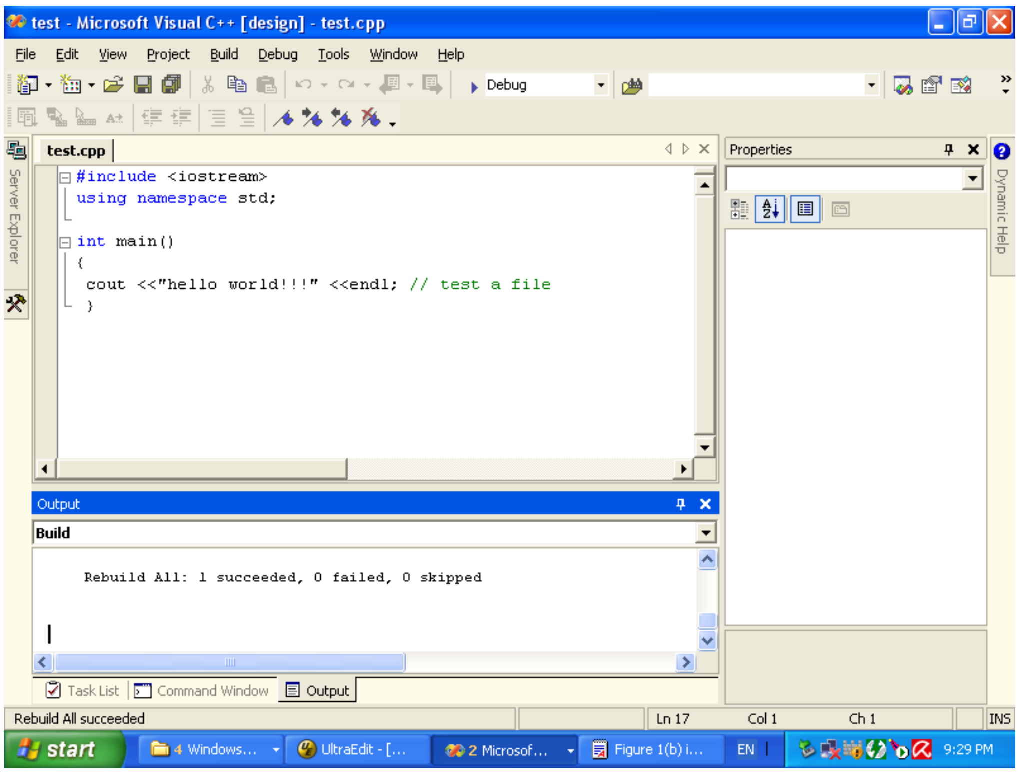
In the sequel, by a program we always mean a *source* program. A sketch of the proposed process for sharing a secret program is described as follows.

(1) *Creating shares* --- Apply exclusive-OR operations to the contents of the secret program and all the camouflage programs, and divide the resulting string into $N$ segments as shares, with the one for the $k$-th participant to keep being denoted as $E_k$.

(2) *Generating authentication signals* --- For each camouflage program $P_k$, use the random key value $Y$ to compute two modulo-$Y$ values from the binary values of the contents of $P_k$ and $E_k$, respectively; and concatenate them as the authentication signal $A_k$ for $P_k$.

(3) *Encoding and hiding shares and authentication signals* --- Encode $E_k$ and $A_k$ respectively into invisible ASCII control codes by the invisible character coding table (Table 7. 2) and hide them evenly at the right sides of all the characters of the comments of camouflage program $P_k$, resulting in a stego-program for the $k$-th participant to keep.

A detailed algorithm for the above scheme is given in the following. We assume that the length of a program is measured as the number of the ASCII characters in it. Also, given two ASCII characters $C$ and $D$, each with 8 bits, denoted as $C = c_0c_1...c_7$ and $D = d_0d_1...d_7$, we define the result of "exclusive-ORing" the two characters as $E = C\oplus D = e_0e_1...e_7$ with $e_i = c_i\oplus d_i$ for $i = 0, 1, ..., 7$ where $\oplus$ denotes the bitwise exclusive-OR operation. Note that $E$ has eight bits, too. And given two equal-lengthed character strings $S$ and $T$, we define the result of exclusive-ORing them, $U = S\oplus T$, as that resulting from exclusive-ORing the corresponding characters in the two strings.

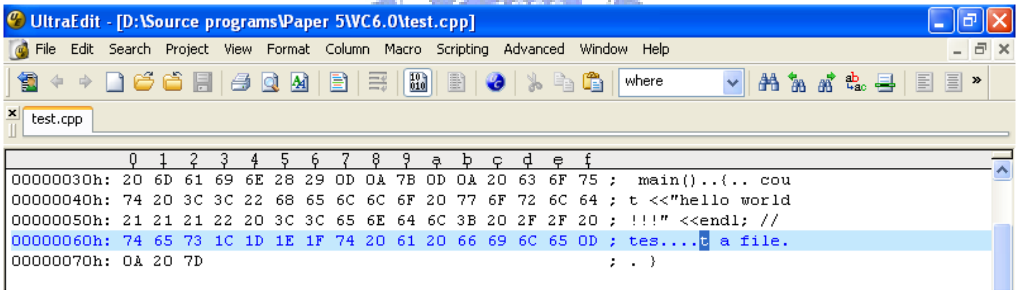

test - Microsoft Visual C++ [design] - test.cpp
File Edit View Project Build Debug Tools Window Help
Debug
test.cpp
Properties
Server Explorer
Dynamic Help
Output
Build
Task List
Command Window
Output
start

```
#include <iostream>
using namespace std;

int main()
{
 cout <<"hello world!!!" <<endl; // test a file
 }
```

```
Rebuild All: 1 succeeded, 0 failed, 0 skipped
```

```
Rebuild All succeeded          Ln 17     Col 1     Ch 1          INS
```

(a) A source program with four invisible ASCII control codes inserted in the comment “test a file.”


UltraEdit - [D:\Source programs\Paper 5\VC6.0\test.cpp]
File Edit Search Project View Format Column Macro Scripting Advanced Window Help
where
test.cpp

```
             0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000030h: 20 6D 61 69 6E 28 29 0D 0A 7B 0D 0A 20 63 6F 75 ;  main()..{.. cou
00000040h: 74 20 3C 3C 22 68 65 6C 6C 6F 20 77 6F 72 6C 64 ; t <<"hello world
00000050h: 21 21 21 22 20 3C 3C 65 6E 64 6C 3B 20 2F 2F 20 ; !!!" <<endl; //
00000060h: 74 65 73 1C 1D 1E 1F 74 20 61 20 66 69 6C 65 0D ; tes....t a file.
00000070h: 0A 20 7D                                        ; . }
```

(b) The program seen in the window of the text editor UltraEdit with the four ASCII control codes visible between the letters “s” and “t” of the word “test” in the comment.

Figure 7.1 Illustration of invisible ASCII control codes in a comment of a source program.

**Algorithm 7.1** *Program sharing and authentication.*

**Input:** (1) a secret program $P_s$ of length $\lambda_s$; (2) $N$ pre-selected camouflage programs $P_1$, $P_2$, ..., $P_N$ of lengths $\lambda_1$, $\lambda_2$, ..., $\lambda_N$, respectively; and (3) a secret key which is a random binary number $Y$ with length $\lambda_Y$ (in the unit of bit).

**Output:** $N$ stego-programs, $P_1'$, $P_2'$, ..., $P_N'$, in each of which a share and an authentication signal are hidden.

**Steps:**

**Stage 1. Creating shares from the secret program.**

1. Create $N + 1$ character strings, all of the length $\lambda_s$ of $P_s$, from the secret program and the camouflage programs in the following way.
   1.1 Scan the characters (including letters, spaces, and ASCII codes) in the secret program $P_s$ line by line, and concatenate them into a character string $S_s$.
   1.2 Do the same to each camouflage program $P_k$, $k = 1, 2, ..., N$, to create a character string $S_k$ of length $\lambda_s$ (not $\lambda_k$) either by discarding the extra characters in $P_k$ if $\lambda_k > \lambda_s$ or by repeating the characters of $P_k$ at the end of $S_k$ if $\lambda_k < \lambda_s$, when $\lambda_k \neq \lambda_s$.
2. Compute the new string $E = S_s \oplus S_1 \oplus S_2 \oplus ... \oplus S_N$.
3. Divide $E$ into $N$ segments $E_1$, $E_2$, ..., $E_N$ as *shares*.

**Stage 2. Generating authentication signals from the contents of the shares and the camouflage programs.**

4. Generate an authentication signal $A_k$ for each camouflage program $P_k$, $k = 1, 2, ..., N$, using the data of $S_k$ and $E_k$ as follows.
   4.1 Regarding $S_k$ as a sequence of 8-bit integers with each character in $S_k$ composed of 8 bits, compute the sum of the integers, take the modulo-$Y$ value of the sum as $A_{S_k}$, transform $A_{S_k}$ into a binary number, and adjust its length to be $\lambda_Y$, the length of the key $Y$, by padding leading 0's if necessary.
   4.2 Do the same to $E_k$ to obtain a binary number $A_{E_k}$ with length $\lambda_Y$, too.
   4.3 Concatenate $A_{S_k}$ and $A_{E_k}$ to form a new binary number $A_k$ with length $2\lambda_Y$ as the authentication signal of $P_k$.

**Stage 3. Encoding and hiding the share data and authentication signals.**

5. For each camouflage program $P_k$, $k = 1, 2, ..., N$, perform the following tasks.
   5.1 Concatenate the share $E_k$ and the authentication signal $A_k$ as a binary string $F_k$.
   5.2 Encode every bit pair of $F_k$ into an invisible ASCII control code according to the invisible coding table (Table 7. 2), resulting in a code string $F_k'$.
   5.3 Count the number $m$ of characters in all the comments of $P_k$.
   5.4 Divide $F_k'$ evenly into $m$ segments, and hide them in order into $P_k$, with each segment hidden to the right of a character in the comments of $P_k$.
6. Take the final camouflage programs $P_1'$, $P_2'$, ..., $P_N'$ as the output stego-programs.

In Step 3, we assume that the number of characters in the secret program is a multiple of $N$, the number of participants, for simplicity of algorithm description; if not, it can be made so by appending a sufficient number of blank spaces at the end of the original secret program. In Steps 4.1 and 4.2, the purpose we compute the signals $A_{S_k}$ and $A_{E_k}$ from the contents of the camouflage program $P_k$ and the share $E_k$, respectively, for use in generating the authentication signal $A_k$ is to prevent any participant from intentionally or accidentally changing the contents of the original camouflage program or the hidden share; illegal tampering with them will be found out in the process of secret program recovery described in the next section. It is also noted that each stego-program yielded by the algorithm still can be compiled and executed to perform the function of the original camouflage program.

## 7.3 Secret Program Recovery Scheme

A sketch of the proposed process for recovering the secret source program is described as follows, for which it is assumed that the stego-program brought to the recovery activity by participant $k$ is denoted as $P_k'$. Also, the original key with value $Y$

used in Algorithm 7.1 is provided.

(1) *Extracting hidden shares and authentication signals* --- Scan the comments of each stego-program $P_k'$ to collect the invisible ASCII control codes hidden in them and concatenate the codes as a character string; decode the string into a binary one by the invisible character coding table (Table 7. 2); and divide the string into two parts, the share data $E_k$ and the authentication signal $A_k$. Also, remove the hidden codes from $P_k'$ to get the original camouflage program $P_k$.

(2) *Authenticating the shares and the camouflage programs* --- Use the authentication signal $A_k$ as well as the key $Y$ to check the correctness of the contents of the extracted share data $E_k$ and the camouflage program $P_k$ by decomposing $A_k$ into two signals and matching them with the modulo-$Y$ values of the binary values of $P_k$ and $E_k$, respectively. Issue warning messages if either or both authentications fail.

(3) *Recovering the secret program* --- Apply exclusive-OR operations to the extracted share data $E_1$ through $E_N$ and the camouflage programs $P_1$ through $P_N$ to reconstruct the secret program $P_s$.

The secret program recovery process is described as a detailed algorithm in the following.

**Algorithm 7.2** ***Authentication of the stego-programs and recovery of the secret program.***

**Input:** $N$ stego-programs $P_1'$, $P_2'$, ..., $P_N'$ provided by the $N$ participants and the secret key $Y$ with length $\quad_{Y}$ used in secret program sharing (Algorithm 7.1).

**Output:** the secret program $P_s$ hidden in the $N$ stego-programs if the shares and the camouflage programs in the stego-programs are authenticated to be correct.

**Steps:**

**Stage I. extracting hidden shares and authentication signals.**

1. For each stego-program $P_k'$, $k = 1, 2, ..., N$, perform the following tasks to get the contents of the camouflage programs and the authentication signals.

   1.1 Scan the comments in $P_k'$ line by line, and collect the invisible ASCII codes located to the right of the comment characters as a character string $F_k'$.

   1.2 Remove all the collected characters of $F_k'$ from $P_k'$, resulting in a program $P_k$ with length $_k$, which presumably is the original camouflage program.

   1.3 Decode the characters in $F_k'$ using the invisible character coding table (Table 7. 2) into a sequence of bit pairs, denoted as $F_k$.

   1.4 Regarding $F_k$ as a binary string, divide it into two segments $E_k$ and $A_k$ with the length of the latter being fixed to be $2\lambda_Y$, which presumably are the hidden share and the authentication signal, respectively.

   1.5 Divide $A_k$ into two equal-lengthed binary numbers $A_{S_k}$ and $A_{E_k}$.

**Stage II. Authenticating share data and camouflage programs.**

2. Concatenate all $E_k$, $k$ =1, 2, ..., $N$, in order, resulting in a string $E$ with length $\lambda_E$ which presumably equals $\lambda_s$, the length of the secret program to be recovered.

3. For each $k = 1, 2, ..., N$, perform the following authentication operations.

   3.1 Create a character string $S_k$ of length $\lambda_E$ from the characters in $P_k$ either by discarding extra characters in $P_k$ if $\lambda_k > \lambda_E$ or by repeating the characters of $P_k$ at the end of $S_k$ if $\lambda_k < \lambda_E$, when $\lambda_k \neq \lambda_E$.

   3.2 Regarding $S_k$ as a sequence of 8-bit integers with each character in $S_k$ composed of 8 bits, compute the sum of the integers, take the modulo-$Y$ value of the sum as $A_{S_k}'$, transform $A_{S_k}'$ into a binary number, and adjust its length to be $\lambda_Y$, the length of the key $Y$, by padding leading 0's if necessary.

   3.3 Do the same to $E_k$, resulting in a binary number $A_{E_k}'$.

   3.4 Compare $A_{S_k}'$ with the previously extracted $A_{S_k}$; if mismatching, issue the message “the camouflage program is not genuine,” and stop the algorithm.

3.5 Compare $A_{E_k}'$ with the previously extracted $A_{E_k}$; if mismatching, issue the message “the share data have been changed,” and stop the algorithm.

**Stage III. Recovering the secret program.**

4. Compute $S_s = E \oplus S_1 \oplus S_2 \oplus \ldots \oplus S_N$, and regard it as a character string.
5. Use the ASCII codes 0D and 0A (“carriage return” and “line feed”) in $S_s$ as separators, break $S_s$ into program lines to reconstruct the original secret program $P_s$ as output.

Note that in Step 4 above, we conduct the exclusive-OR operations of $E \oplus S_1 \oplus S_2 \oplus \ldots \oplus S_N$. This will indeed result in the desired $S_s$ because $E$ was computed as $E = S_s \oplus S_1 \oplus S_2 \oplus \ldots \oplus S_N$ in Step 2 of Algorithm 7.1, and so

$$\begin{aligned} E \oplus S_1 \oplus S_2 \oplus \ldots \oplus S_N &= (S_s \oplus S_1 \oplus S_2 \oplus \ldots \oplus S_N) \oplus S_1 \oplus S_2 \oplus \ldots \oplus S_N \\ &= S_s \oplus (S_1 \oplus S_1) \oplus \ldots \oplus (S_N \oplus S_N) \\ &= S_s \oplus \mathbf{0} \oplus \mathbf{0} \oplus \ldots \oplus \mathbf{0} = S_s \end{aligned}$$

by the commutative and associative laws of the exclusive-OR operation and the facts that $X \oplus X = 0$ and $X \oplus 0 = X$ for any bit $X$, where the bold character **0** is used to represent 8 consecutive bits of zero, i.e., $\mathbf{0} = 00000000$.

## 7.4 Discussions on Security Protection

In the previous discussions, we assume that the proposed algorithms of secret sharing and recovery (Algorithms 7.1 and 7.2) are known to the public, and that the key $Y$ is held by a supervisor other than any of the $N$ participants. The key is provided by the supervisor as an input to the secret program sharing and recovery processes described by Algorithms 7.1 and 7.2; it is not available to any participant. Under these assumptions and by Algorithm 7.2 above, if any participant changes the content of the camouflage program or that of the share contained in the stego-program which he/she holds before the secret program recovery process, such illegal tampering will be found out and warnings issued during the recovery process.

However, there still exists in the two algorithms another kind of weakness in security protection of the secret program. That is, the secret program may be recovered illegally if *all* the stego-programs are stolen by a person who knows the algorithms, because then he/she may run Algorithm 7.2 to extract the secret program without performing Step 3, as can be figured out!

One way to remove this weakness is to use the secret key to randomize the result of $E = S_s \oplus S_1 \oplus S_2 \oplus \ldots \oplus S_N$ computed in Step 2 in Algorithm 7.1 before $E$ is divided into shares in the next step. We implement this by letting the secret key $Y$ join the exclusive-OR operation of Step 2 after expanding $Y$ repeatedly to have a length equal to that of the secret program $S_s$. That is, in Step 2 of Algorithm 7.1 we repeat the key $Y$'s and concatenate them until the length of the expanded key $Y'$ in the unit of character (8 bits for a character) is equal to $\lambda_s$, the length of $S_s$, and then compute $E$ instead as $E = S_s \oplus S_1 \oplus S_2 \oplus \ldots \oplus S_N \oplus Y'$. Correspondingly, in Step 4 of Algorithm 7.2 we expand $Y$ similarly to get $Y'$, and then compute $S_s$ instead as $S_s = E \oplus S_1 \oplus S_2 \oplus \ldots \oplus S_N \oplus Y'$. The properties of the exclusive-OR operation assure that the $S_s$ so computed is the desired secret program in its string form. In this way, without the key $Y$, $S_s$ obviously cannot be recovered, and so the previously-mentioned weakness is removed.

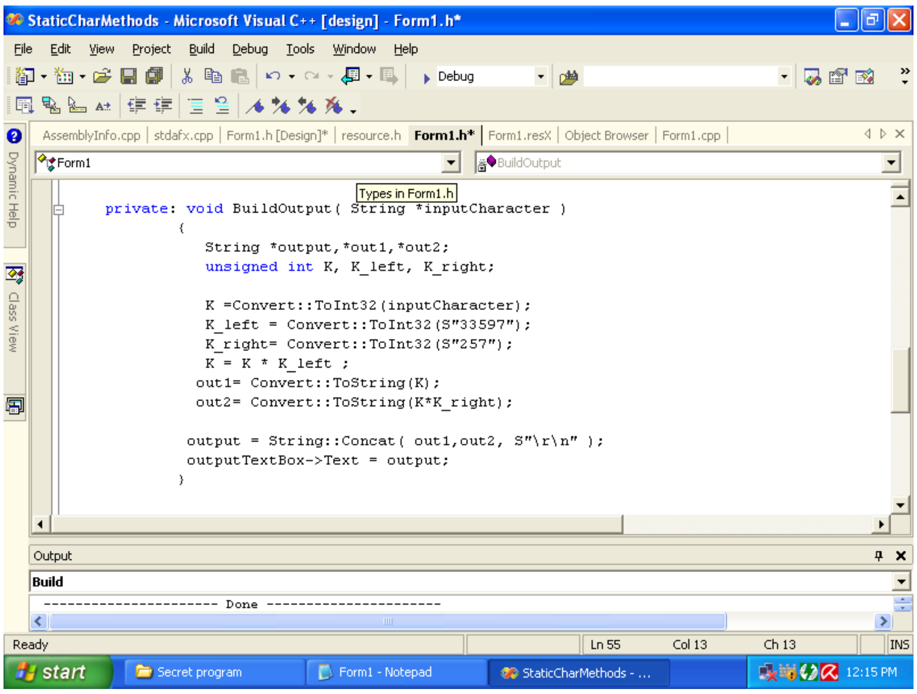
## 7.5 Experimental Results

In one of our experiments, we applied the proposed schemes described previously to share a secret program among three participants. The main part of the secret program seen in the window of the Microsoft VC$^{++}$ editor is shown in Figure 7.2(a), which has the function of generating a secret key from an input seed. And part of one of the three camouflage programs is shown in Figure 7.2(b). After hiding the shares and the authentication signals in the comments of each camouflage programs, the stego-program resulting from Figure 7.2(b) appears to be the upper part of Figure 7.2(c) which is not different from that of Figure 7.2(b). The real content of the stego-program seen in the window of the UltraEdit 32 editor is shown in the lower part of Figure 7.2(c) which includes the ASCII codes representing the program on the left and the appearance of the codes as characters on the right. The recovered secret program is shown in Figure 7.2(d), which is identical to that shown in Figure 7.2(a).

We also tested the case of recovery with one of the stego-images (the second one) being damaged, as shown in Figure 7.3(a). The proposed scheme issued a warning message, as shown in Figure 7.3(b).

## 7.6 Concluding Remarks

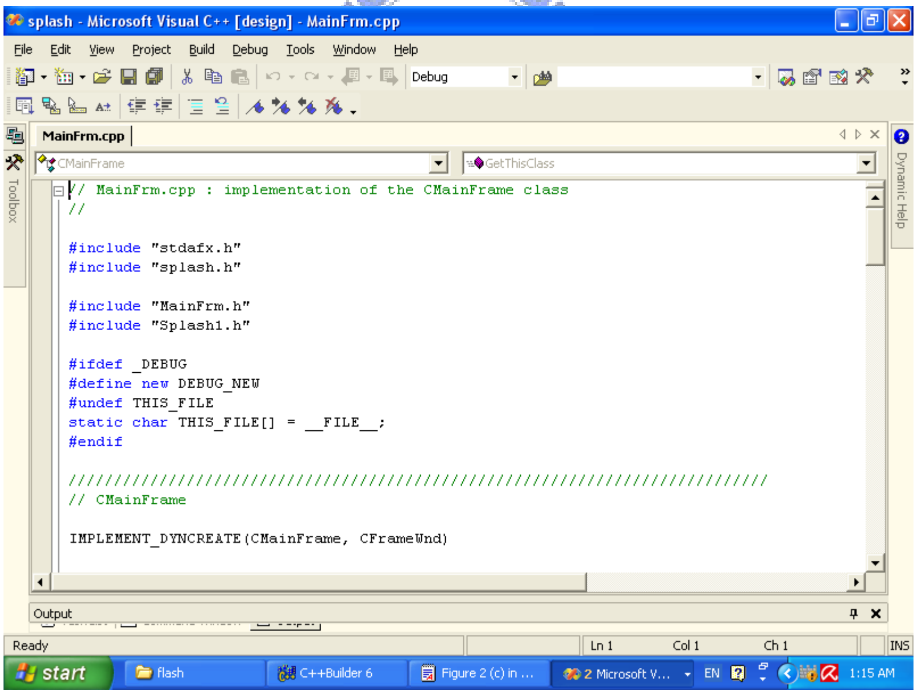
For the purpose of protecting software programs, new techniques for sharing secret source programs and authentication of resulting stego-programs using four special ASCII control codes invisible in the window of the Microsoft VC$^{++}$ editor have been proposed. The proposed sharing scheme divides the result of exclusive-ORing the contents of the secret program and a group of camouflage programs into shares, each of which is then encoded into a sequence of invisible ASCII control codes before being embedded into the comments of the corresponding camouflage program. The resulting stego-programs are kept by the participants of the sharing process. The original function of each camouflage program is not destroyed in the corresponding stego-program. The sharing of the secret program and the invisibility of the special ASCII codes as share data provides two-fold security protection of the secret program.



```
private: void BuildOutput( String *inputCharacter )
         {
            String *output,*out1,*out2;
            unsigned int K, K_left, K_right;

            K =Convert::ToInt32(inputCharacter);
            K_left = Convert::ToInt32(S"33597");
            K_right= Convert::ToInt32(S"257");
            K = K * K_left ;
           out1= Convert::ToString(K);
           out2= Convert::ToString(K*K_right);

          output = String::Concat( out1,out2, S"\r\n" );
          outputTextBox->Text = output;
         }
```

```
---------------------- Done ----------------------
```

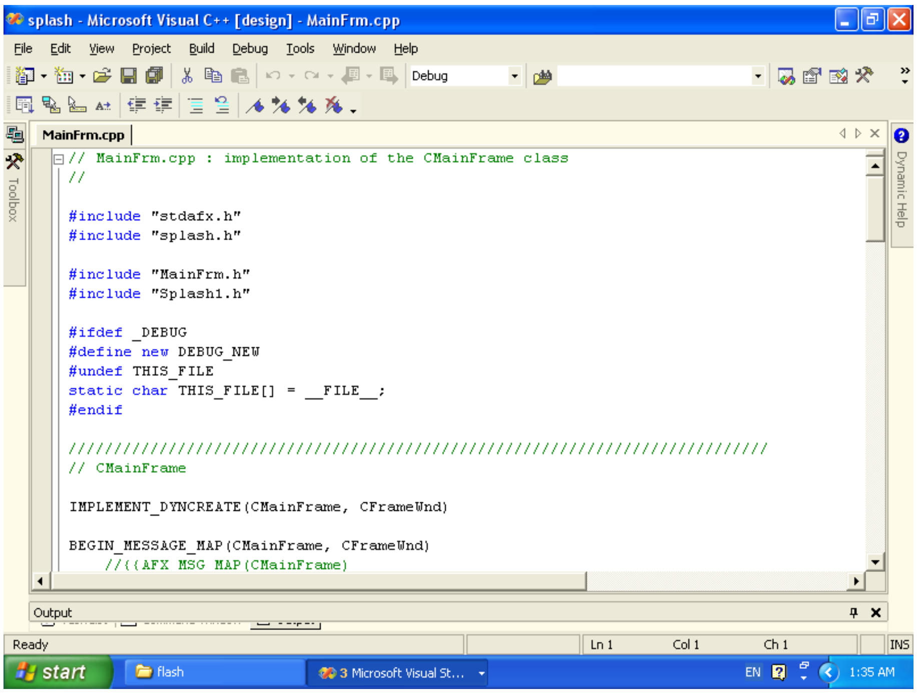
(a) Main part of the secret source program seen in the window of the Microsoft VC$^{++}$ editor.



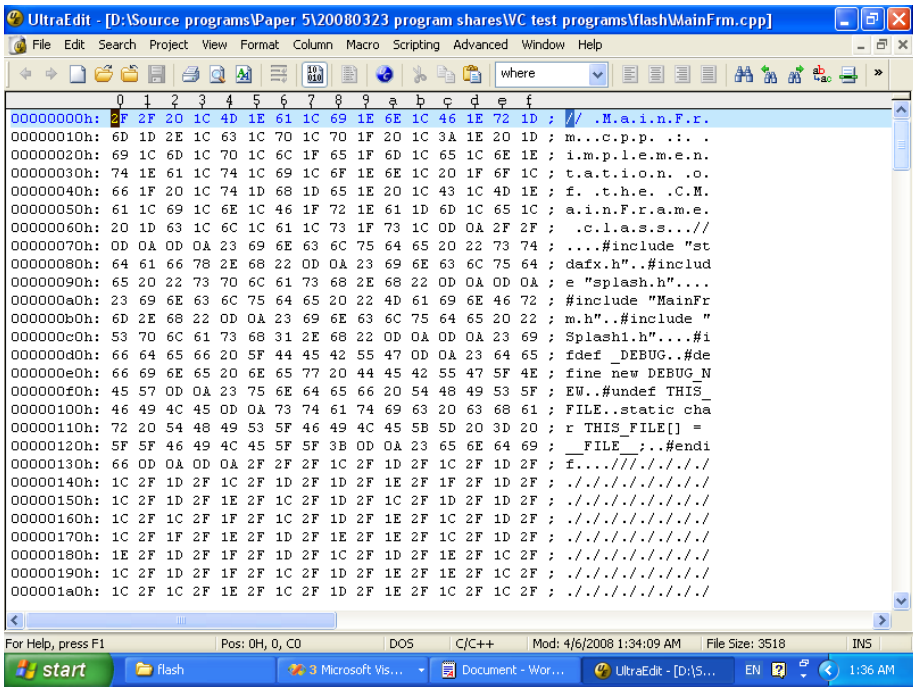
```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "splash.h"

#include "MainFrm.h"
#include "Splash1.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
```

(b) Part of one camouflage program seen in the window of Microsoft Visual C$^{++}$ editor.

Figure 7.2 Experimental results of sharing a secret program.



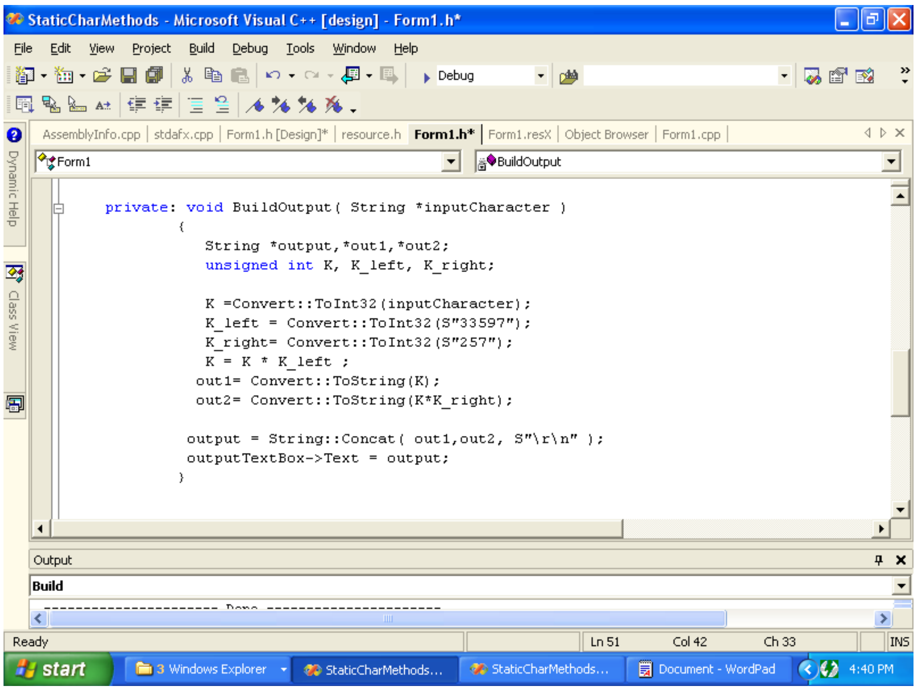
```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "splash.h"

#include "MainFrm.h"
#include "Splash1.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
```


UltraEdit - [D:\Source programs\Paper 5\20080323 program shares\VC test programs\flash\MainFrm.cpp]
For Help, press F1
Pos: 0H, 0, C0
DOS
C/C++
Mod: 4/6/2008 1:34:09 AM
File Size: 3518
INS

```
           0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 2F 2F 20 1C 4D 1E 61 1C 69 1E 6E 1C 46 1E 72 1D ; // .M.a.i.n.F.r.
00000010h: 6D 1D 2E 1C 63 1C 70 1C 70 1F 20 1C 3A 1E 20 1D ; m...c.p.p. .:. .
00000020h: 69 1C 6D 1C 70 1C 6C 1F 65 1F 6D 1C 65 1C 6E 1E ; i.m.p.l.e.m.e.n.
00000030h: 74 1E 61 1C 74 1C 69 1C 6F 1E 6E 1C 20 1F 6F 1C ; t.a.t.i.o.n. .o.
00000040h: 66 1F 20 1C 74 1D 68 1D 65 1E 20 1C 43 1C 4D 1E ; f. .t.h.e. .C.M.
00000050h: 61 1C 69 1C 6E 1C 46 1F 72 1E 61 1D 6D 1C 65 1C ; a.i.n.F.r.a.m.e.
00000060h: 20 1D 63 1C 6C 1C 61 1C 73 1F 73 1C 0D 0A 2F 2F ;  .c.l.a.s.s...//
00000070h: 0D 0A 0D 0A 23 69 6E 63 6C 75 64 65 20 22 73 74 ; ....#include "st
00000080h: 64 61 66 78 2E 68 22 0D 0A 23 69 6E 63 6C 75 64 ; dafx.h"..#includ
00000090h: 65 20 22 73 70 6C 61 73 68 2E 68 22 0D 0A 0D 0A ; e "splash.h"....
000000a0h: 23 69 6E 63 6C 75 64 65 20 22 4D 61 69 6E 46 72 ; #include "MainFr
000000b0h: 6D 2E 68 22 0D 0A 23 69 6E 63 6C 75 64 65 20 22 ; m.h"..#include "
000000c0h: 53 70 6C 61 73 68 31 2E 68 22 0D 0A 0D 0A 23 69 ; Splash1.h"....#i
000000d0h: 66 64 65 66 20 5F 44 45 42 55 47 0D 0A 23 64 65 ; fdef _DEBUG..#de
000000e0h: 66 69 6E 65 20 6E 65 77 20 44 45 42 55 47 5F 4E ; fine new DEBUG_N
000000f0h: 45 57 0D 0A 23 75 6E 64 65 66 20 54 48 49 53 5F ; EW..#undef THIS_
00000100h: 46 49 4C 45 0D 0A 73 74 61 74 69 63 20 63 68 61 ; FILE..static cha
00000110h: 72 20 54 48 49 53 5F 46 49 4C 45 5B 5D 20 3D 20 ; r THIS_FILE[] = 
00000120h: 5F 5F 46 49 4C 45 5F 5F 3B 0D 0A 23 65 6E 64 69 ; __FILE__;..#endi
00000130h: 66 0D 0A 0D 0A 2F 2F 2F 1C 2F 1D 2F 1C 2F 1D 2F ; f....///././././
00000140h: 1C 2F 1D 2F 1C 2F 1D 2F 1D 2F 1E 2F 1F 2F 1D 2F ; ./././././././././
00000150h: 1C 2F 1D 2F 1E 2F 1C 2F 1D 2F 1C 2F 1D 2F 1D 2F ; ./././././././././
00000160h: 1C 2F 1C 2F 1F 2F 1C 2F 1D 2F 1E 2F 1C 2F 1D 2F ; ./././././././././
00000170h: 1C 2F 1F 2F 1E 2F 1D 2F 1E 2F 1E 2F 1C 2F 1D 2F ; ./././././././././
00000180h: 1E 2F 1D 2F 1F 2F 1D 2F 1C 2F 1D 2F 1E 2F 1C 2F ; ./././././././././
00000190h: 1C 2F 1D 2F 1F 2F 1C 2F 1D 2F 1E 2F 1E 2F 1C 2F ; ./././././././././
000001a0h: 1C 2F 1C 2F 1E 2F 1C 2F 1D 2F 1E 2F 1C 2F 1C 2F ; ./././././././././
```

(c) The stego-program resulting from (b) seen in the window of Microsoft Visual C$^{++}$ editor (upper part) and UltraEditor 32 editor (lower part).

Figure 7.2 Experimental results of sharing a secret program (continued).



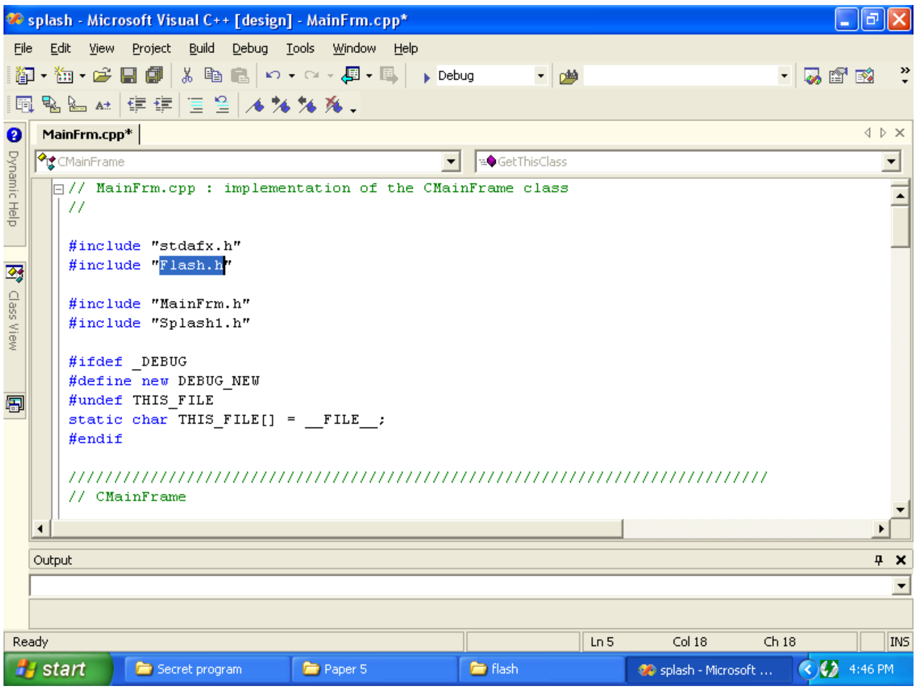
```
private: void BuildOutput( String *inputCharacter )
         {
            String *output,*out1,*out2;
            unsigned int K, K_left, K_right;

            K =Convert::ToInt32(inputCharacter);
            K_left = Convert::ToInt32(S"33597");
            K_right= Convert::ToInt32(S"257");
            K = K * K_left ;
           out1= Convert::ToString(K);
           out2= Convert::ToString(K*K_right);

          output = String::Concat( out1,out2, S"\r\n" );
          outputTextBox->Text = output;
         }
```

(d) Recovered secret program seen in the window of Microsoft Visual C$^{++}$ editor.

Figure 7.2 Experimental results of sharing a secret program (continued).

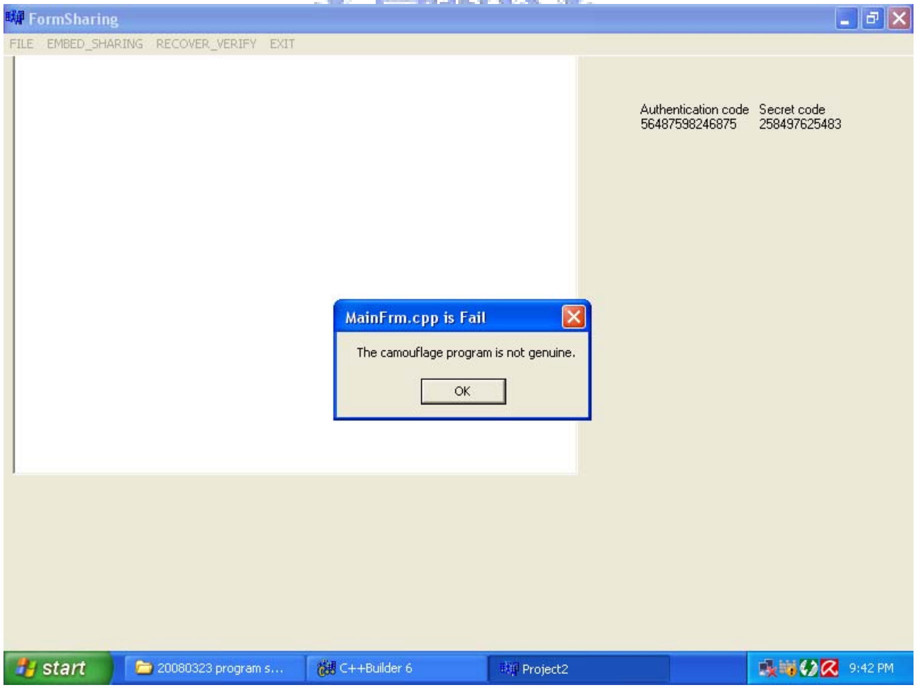
In the secret program recovery process, the reversibility property of the exclusive-OR operation is adopted to recover the secret program using the share data extracted from the stego-programs. To enhance security of keeping the camouflage programs, a secret random key is adopted to verify, during the recovery process, possible incidental or intentional tampering with the hidden share and the camouflage program content in each stego-program. The key is also utilized to prevent unauthorized recovery of the secret program by illegal collection of all the stego-programs and unauthorized execution of part of the proposed algorithms.

Experimental results have shown the feasibility of the proposed method. Future research may be directed to applying the invisible ASCII control codes to other applications, such as watermarking of software programs for copyright protection, secret hiding in software programs for covert communication, authentication of software program correctness, and so on.



```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Flash.h"

#include "MainFrm.h"
#include "Splash1.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////////////////////////////////////////////
// CMainFrame
```

(a) Destructed stego-program of Figure 7.2(b) seen in the window of Microsoft Visual C$^{++}$ editor (the changed characters are highlighted).



Authentication code 56487598246875
Secret code 258497625483

MainFrm.cpp is Fail

The camouflage program is not genuine.

(b) A message showing the content of the original camouflage program has been changed.

Figure 7.3 An experimental result of authenticating a destructed stego-program.

# Chapter 8
# Covert Communication with Authentication via Software Programs Using Invisible ASCII Codes

## 8.1 Idea of Proposed Method

ASCII codes, expressed as hexadecimal numbers, were designed to represent 8-bit characters for information interchange. It is found in this study that some ASCII codes, when embedded in certain locations in $C^{++}$ programs, become *invisible* in the source code editors of Visual C++ and C++ Builder under certain Windows OS environments. This phenomenon may be utilized for data hiding.

In Chapter 7, we have proposed a method for security protection of software programs by information sharing and authentication techniques using some invisible ASCII control codes. The principle of data hiding in source programs is still suitable for covert communication here. But more *invisible codes* have been found in this study, which are categorized into two types, one appearing as *nothing* like being non-existing, and the other as *spaces* just like the ASCII space code 20. We call the former *null code* and the latter *spacing code*. Inserting invisible codes into a program do not change its function.

Such invisibility was found in fours environments formed by Microsoft Visual Studio (MVS) .NET 2003 and Borland $C^{++}$ Builder (BCB), version 6, in Windows XP Service Pack 2 and its Chinese version, which will be called the English and Chinese OS, respectively, subsequently. The details are summarized in Table 8.1.

In type-1 environment with the MVS in the English OS, four null codes, 1C, 1D, 1E, 1F, were found, which are invisible when inserted *between two characters* in a comment in a program. One spacing code, A0, has been found, which appears as a space when inserted *between two words* in a comment. Also found as a spacing code is the tab-control code 09, which *in default* appears as four spaces when inserted *before the end of a program line*, i.e., before the code pair, 0D0A, for *carriage return* and *line feed*. The codes, A0 and 09, will be called *between-word* and *line-end* spacing codes, respectively.

For the other three environment types, invisible codes also exist and are listed in Table 8.1 except that type-2 environment has no null code. Also, 09 appears to be eight spaces in BCB instead of four as in MVS.

Table 8.1. Invisible codes under various environments.

| **Environment** | **Null codes** | **Between-word spacing codes** | **Line-end spacing codes** |
|---|---|---|---|
| Type 1: MVS under English OS | 1C-1F | A0 | 09 |
| Type 2: BCB under English OS | None | A0 | 09 |
| Type 3: MVS under Chinese OS | 1C-1F | 01-08, 0B-0F, 80 | 09, 0B, 0C |
| Type 4: BCB under Chinese OS | 1C-1F, 80 | 01-08, 0B-19, 1B | 09, 0B, 0C |

In the remainder of this chapter, the principle of data hiding for use in covert communication is introduced in Section 8.2. The secret hiding, recovery and authentication processes are described in Section 8.3. The experimental results are shown in Section 8.4. Finally, some concluding remarks are given in Section 8.5.

## 8.2 Data Hiding Using Invisible Codes

We conduct data hiding using invisible codes in three ways as follows.

### 1. *Alternative space coding*

Whenever a space represented by 20 appears between two words in a comment, it may be replaced by a between-word spacing code, like A0 for type-1 environment, without causing visual difference in a source code editor. When there are $2^n-1$ between-word spacing codes $C_1$, ..., $C_{2^n-1}$, by regarding 20 as $C_0$ we may embed $n$ bits $b_1$, $b_2$, ..., $b_n$ as follows:

*if* $b_1b_2....b_n = m$, *replace 20 by* $C_m$

which we call *alternative space coding*.

For the first two environments in Table 8.1, 1-bit alternative space coding is applicable. And for the latter two, there are 14 and 23 spacing codes, respectively and so 3-bit and 4-bit alternative space coding are applicable, respectively.

### 2. *Line-end space coding*

We may place *multiple* line-end spacing codes before each program line end without causing visual difference in a source code editor because such codes appear just like *background spaces* in the window of the editor. Since the code 20 may be used as well to create spaces, when there are $2^n-1$ line-end spacing codes $C_1$, ..., $C_{2^n-1}$, by regarding 20 as $C_0$ we may embed $n$ bits $b_1$, $b_2$, ..., $b_n$ as follows:

*if* $b_0b_1...b_n = m$, *embed* $C_m$ *before the line end*

which we will call *line-end space coding*.

For the first two environments, there is only one line-end spacing code 09, so 1-bit line-end coding is applicable. For the latter two, since there are three such codes

09, 0B, and 0C, 2-bit coding can be implemented.

Line-end space coding may be repeated *unlimited times* before the each line end to increase the data hiding rate. But to avoid creating long lines which reduce the steganographic effect, we require that each processed program line should not appear to be longer than the longest original program line.

### 3. *Null space coding*

Except for type-2 environment, there are four null codes, 1C, 1D, 1E, 1F. Let them be represented by $C_0$ through $C_3$, respectively. We can embed a bit pair $b_0b_1$ as follows:

*if $b_0b_1 = m$, insert $C_m$ between two characters in a comment*

which we call *null space coding*.

Null space coding may be applied repetitively *unlimited times* as well. In practice, we embed message bits *evenly* into all between-character spaces among the comments so that the times will be limited.

## 8.3 Secret Hiding, Recovery and Authentication

The proposed data hiding process essentially is to apply alternative, line-end, and null space coding in order. Since the three schemes are applied to *distinct* locations in a program, the data may be recovered without ambiguity. As an example, we describe in the following an algorithm for type-1 environment. To facilitate data recovery, we prefix to the beginning of the input binary string of the message a binary number specifying the length of the input, resulting in an extended bit string *S*.

1. At each between-word space coded by 20, remove the leading bit *b* from *S*, and replace 20 by A0 if *b* =1.
2. Find the maximum $L_{\max}$ of all program line lengths.

3. For each program line, repeat the operations of removing the leading bit $b$ from $S$ and inserting before the line end the code 0D0A if $b = 1$; or 20 if $b = 0$, until the length of the line, as it appears in the source code editor, reaches $L_{\max}$.
4. Count the number $M$ of all between-character positions in the comments, as well as the number $L$ of the remaining bits in $S$; compute the ceiling value $\lceil L/M \rceil$; add 1 to it to make it even if it is not; and denote the final value as $q$.
5. For each between-character position in the comments, take $q$ leading bits from $S$, and for every two bits $b_0b_1$ of them, insert $C_m$ into the position if $b_0b_1 = m$, where $C_m$ is one of $C_0$ through $C_3$ representing 1C, 1D, 1E, and 1F, respectively.

The proposed data recovery process, after extracting from the input string the leading bits which specify the length of the original message, performs essentially the reverse versions of the three coding schemes involved in the data hiding process. The details are omitted due to the page limit.

In the proposed authentication scheme, we use a 16-bit key $K$ and the input message string to generate an authentication signal $A$ which is then embedded in the stego-program as well using null space coding. The signal is computed as the modulo-$K$ value of the sum of the key value and the 16 bits of every two characters in the input message string. Then, in the data recovery process, the embedded authentication signal $A$ is extracted to match with an authentication signal $A'$ computed similarly from the extracted message content and the key. If the embedded message content has not been tampered with, then $A$ and $A'$ will match. If not, then the message must have been modified. In such a way, even when the data hiding algorithm is known to the public as is usually assumed, without the secret key it is impossible to pass such an authentication process with a modified stego-program.

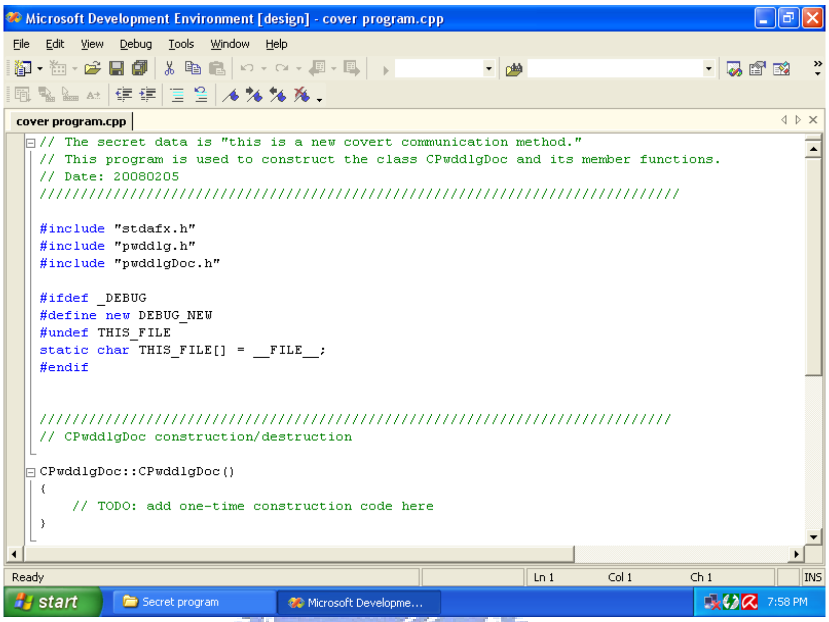
## 8.4 Experimental Results

One of the experiments we conducted for type-1 environment is reported here. A message “This is a new covert communication method” is embedded into a cover program, part of which is shown in Figure 8.1(a). The binary form of the message is obtained from the ASCII characters representing the message. It is 00000001 01100000 01010100 01101000... in which the first 16 bits specify the length of the message string, and the remaining ones represents T, h, and so on. And the encoding result of it is 20 20 20 20 20 20 20 A0 20 A0 A0 20 20 20 20 20 ... The stego-program seen in the source code editor is shown in Figure 8.1(b), which looks no difference from Figure 8.1(a). And the real content of the program seen in the UltraEdit editor is shown in Figure 8.1(c), in which the hidden invisible codes can be seen with those for the first 16 bits being enclosed by rectangles. The recovered message is shown in Figure 8.1(d). As a demonstration of authentication, we show in Figure 8.2(a) a modified version of the stego-program of Figure 8.1(b) in the UltraEdit editor, in which the codes fro the 8th and 9th bits of the message have been modified. The authentication result is shown in Figure 8.2(b) in which a warning message issued by the data recovery process is seen.

## 8.5 Concluding Remarks

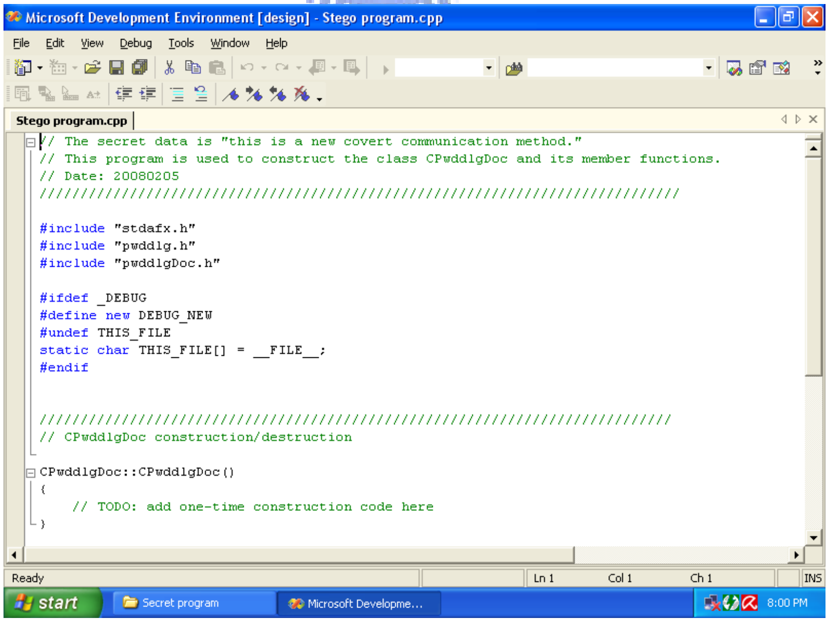
A new method to covert communication via C$^{++}$ source programs using invisible ASCII codes has proposed. A secret message is encoded by some special ASCII codes, which are embedded in a cover program. Such codes are invisible in the source code editors of Visual C++ and C++ Builder under Windows environments, creating a good steganographic effect without changing the original function of the cover program.

To enhance security, tamper-proof authentication of the stego-program content using a secret key has also been proposed. Without the key, false messages cannot pass the authentication process. Experimental results show the feasibility of the proposed method. Future works may be directed to applying the proposed data hiding technique to other applications.


Microsoft Development Environment [design] - cover program.cpp
File Edit View Debug Tools Window Help
cover program.cpp
Ready
Ln 1
Col 1
Ch 1
INS
start
Secret program
Microsoft Developme...
7:58 PM

```
// The secret data is "this is a new covert communication method."
// This program is used to construct the class CPwddlgDoc and its member functions.
// Date: 20080205
/////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "pwddlg.h"
#include "pwddlgDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


/////////////////////////////////////////////////////////////////////////////
// CPwddlgDoc construction/destruction

CPwddlgDoc::CPwddlgDoc()
{
    // TODO: add one-time construction code here
}
```

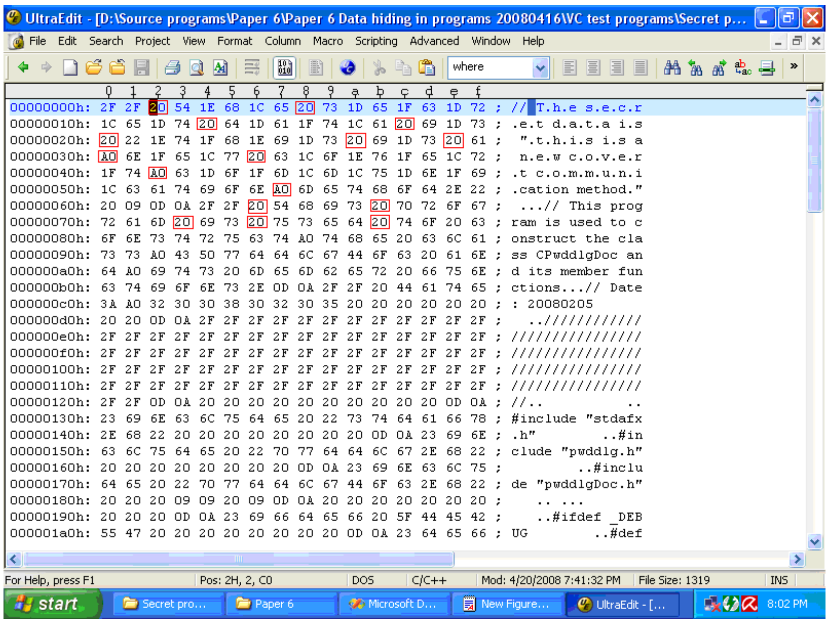
(a) Cover program seen in source code editor.


Microsoft Development Environment [design] - Stego program.cpp
File Edit View Debug Tools Window Help
Stego program.cpp
Ready
Ln 1
Col 1
Ch 1
INS
start
Secret program
Microsoft Developme...
8:00 PM

```
// The secret data is "this is a new covert communication method."
// This program is used to construct the class CPwddlgDoc and its member functions.
// Date: 20080205
/////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "pwddlg.h"
#include "pwddlgDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


/////////////////////////////////////////////////////////////////////////////
// CPwddlgDoc construction/destruction

CPwddlgDoc::CPwddlgDoc()
{
    // TODO: add one-time construction code here
}
```

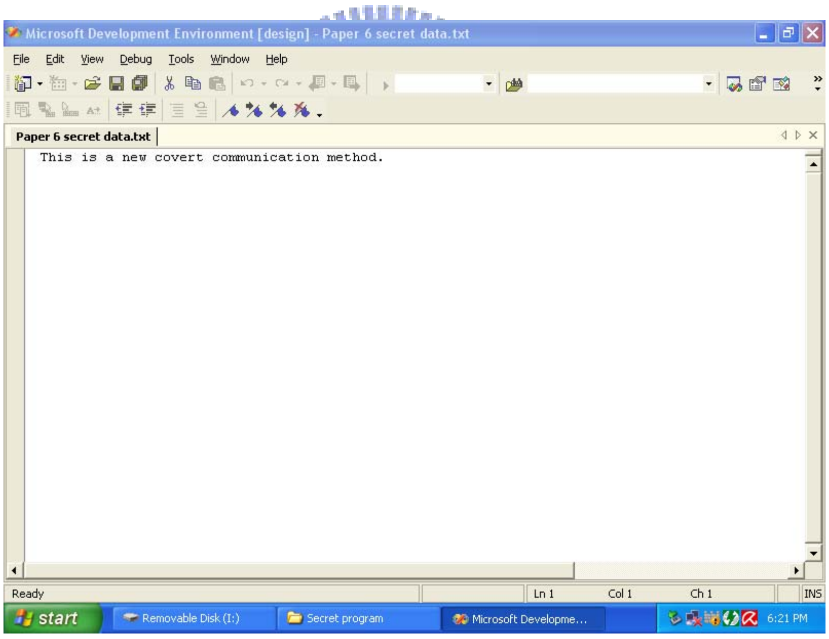
(b) Stego-program seen in source code editor.

Figure 8.1 An experimental result.


UltraEdit - [D:\Source programs\Paper 6\Paper 6 Data hiding in programs 20080416\VC test programs\Secret p...
File Edit Search Project View Format Column Macro Scripting Advanced Window Help
where

```
           0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 2F 2F 20 54 1E 68 1C 65 20 73 1D 65 1F 63 1D 72 ; // T.h.e s.e.c.r
00000010h: 1C 65 1D 74 20 64 1D 61 1F 74 1C 61 20 69 1D 73 ; .e.t d.a.t.a i.s
00000020h: 20 22 1E 74 1F 68 1E 69 1D 73 20 69 1D 73 20 61 ;  ".t.h.i.s i.s a
00000030h: A0 6E 1F 65 1C 77 20 63 1C 6F 1E 76 1F 65 1C 72 ;  n.e.w c.o.v.e.r
00000040h: 1F 74 A0 63 1D 6F 1F 6D 1C 6D 1C 75 1D 6E 1F 69 ; .t c.o.m.m.u.n.i
00000050h: 1C 63 61 74 69 6F 6E A0 6D 65 74 68 6F 64 2E 22 ; .cation method."
00000060h: 20 09 0D 0A 2F 2F 20 54 68 69 73 20 70 72 6F 67 ;  ...// This prog
00000070h: 72 61 6D 20 69 73 20 75 73 65 64 20 74 6F 20 63 ; ram is used to c
00000080h: 6F 6E 73 74 72 75 63 74 A0 74 68 65 20 63 6C 61 ; onstruct the cla
00000090h: 73 73 A0 43 50 77 64 64 6C 67 44 6F 63 20 61 6E ; ss CPwddlgDoc an
000000a0h: 64 A0 69 74 73 20 6D 65 6D 62 65 72 20 66 75 6E ; d its member fun
000000b0h: 63 74 69 6F 6E 73 2E 0D 0A 2F 2F 20 44 61 74 65 ; ctions...// Date
000000c0h: 3A A0 32 30 30 38 30 32 30 35 20 20 20 20 20 20 ; : 20080205
000000d0h: 20 20 0D 0A 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F ;   ..////////////
000000e0h: 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F ; ////////////////
000000f0h: 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F ; ////////////////
00000100h: 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F ; ////////////////
00000110h: 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F ; ////////////////
00000120h: 2F 2F 0D 0A 20 20 20 20 20 20 20 20 20 20 0D 0A ; //..          ..
00000130h: 23 69 6E 63 6C 75 64 65 20 22 73 74 64 61 66 78 ; #include "stdafx
00000140h: 2E 68 22 20 20 20 20 20 20 20 20 0D 0A 23 69 6E ; .h"        ..#in
00000150h: 63 6C 75 64 65 20 22 70 77 64 64 6C 67 2E 68 22 ; clude "pwddlg.h"
00000160h: 20 20 20 20 20 20 20 20 0D 0A 23 69 6E 63 6C 75 ;         ..#inclu
00000170h: 64 65 20 22 70 77 64 64 6C 67 44 6F 63 2E 68 22 ; de "pwddlgDoc.h"
00000180h: 20 20 20 09 09 20 09 0D 0A 20 20 20 20 20 20 20 ;    .. ...
00000190h: 20 20 20 0D 0A 23 69 66 64 65 66 20 5F 44 45 42 ;    ..#ifdef _DEB
000001a0h: 55 47 20 20 20 20 20 20 20 20 0D 0A 23 64 65 66 ; UG        ..#def
```

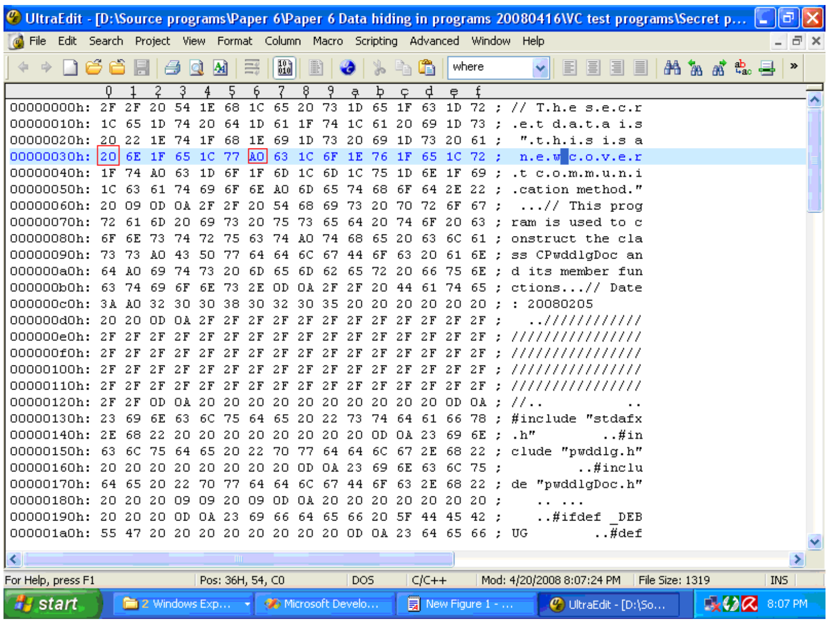
(c) Stego-program seen in UltraEdit.


Microsoft Development Environment [design] - Paper 6 secret data.txt
File Edit View Debug Tools Window Help
Paper 6 secret data.txt

```
This is a new covert communication method.
```

(d) Recovered message.

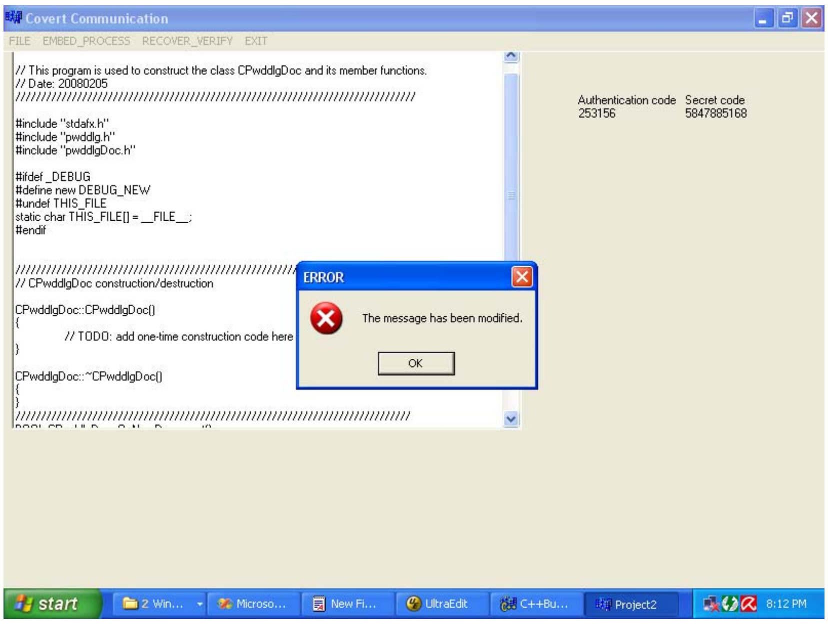
Figure 8.1 An experimental result (continued).


UltraEdit - [D:\Source programs\Paper 6\Paper 6 Data hiding in programs 20080416\VC test programs\Secret p...
File Edit Search Project View Format Column Macro Scripting Advanced Window Help
where

```
            0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 2F 2F 20 54 1E 68 1C 65 20 73 1D 65 1F 63 1D 72 ; // T.h.e s.e.c.r
00000010h: 1C 65 1D 74 20 64 1D 61 1F 74 1C 61 20 69 1D 73 ; .e.t d.a.t.a i.s
00000020h: 20 22 1E 74 1F 68 1E 69 1D 73 20 69 1D 73 20 61 ;  ".t.h.i.s i.s a
00000030h: 20 6E 1F 65 1C 77 A0 63 1C 6F 1E 76 1F 65 1C 72 ;  n.e.w c.o.v.e.r
00000040h: 1F 74 A0 63 1D 6F 1F 6D 1C 6D 1C 75 1D 6E 1F 69 ; .t c.o.m.m.u.n.i
00000050h: 1C 63 61 74 69 6F 6E A0 6D 65 74 68 6F 64 2E 22 ; .cation method."
00000060h: 20 09 0D 0A 2F 2F 20 54 68 69 73 20 70 72 6F 67 ;  ...// This prog
00000070h: 72 61 6D 20 69 73 20 75 73 65 64 20 74 6F 20 63 ; ram is used to c
00000080h: 6F 6E 73 74 72 75 63 74 A0 74 68 65 20 63 6C 61 ; onstruct the cla
00000090h: 73 73 A0 43 50 77 64 64 6C 67 44 6F 63 20 61 6E ; ss CPwddlgDoc an
000000a0h: 64 A0 69 74 73 20 6D 65 6D 62 65 72 20 66 75 6E ; d its member fun
000000b0h: 63 74 69 6F 6E 73 2E 0D 0A 2F 2F 20 44 61 74 65 ; ctions...// Date
000000c0h: 3A A0 32 30 30 38 30 32 30 35 20 20 20 20 20 20 ; : 20080205
000000d0h: 20 20 0D 0A 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F ;   ..////////////
000000e0h: 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F ; ////////////////
000000f0h: 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F ; ////////////////
00000100h: 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F ; ////////////////
00000110h: 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F ; ////////////////
00000120h: 2F 2F 0D 0A 20 20 20 20 20 20 20 20 20 20 0D 0A ; //..          ..
00000130h: 23 69 6E 63 6C 75 64 65 20 22 73 74 64 61 66 78 ; #include "stdafx
00000140h: 2E 68 22 20 20 20 20 20 20 20 20 0D 0A 23 69 6E ; .h"        ..#in
00000150h: 63 6C 75 64 65 20 22 70 77 64 64 6C 67 2E 68 22 ; clude "pwddlg.h"
00000160h: 20 20 20 20 20 20 20 20 0D 0A 23 69 6E 63 6C 75 ;         ..#inclu
00000170h: 64 65 20 22 70 77 64 64 6C 67 44 6F 63 2E 68 22 ; de "pwddlgDoc.h"
00000180h: 20 20 20 09 09 20 09 0D 0A 20 20 20 20 20 20 20 ;    .. ...
00000190h: 20 20 20 0D 0A 23 69 66 64 65 66 20 5F 44 45 42 ;    ..#ifdef _DEB
000001a0h: 55 47 20 20 20 20 20 20 20 20 0D 0A 23 64 65 66 ; UG        ..#def
```

```
For Help, press F1    Pos: 36H, 54, C0    DOS    C/C++    Mod: 4/20/2008 8:07:24 PM    File Size: 1319    INS
```

(a) A modified stego-program of Figure 8. 1(b).



```
// This program is used to construct the class CPwddlgDoc and its member functions.
// Date: 20080205
/////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "pwddlg.h"
#include "pwddlgDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


/////////////////////////////////////////////////////
// CPwddlgDoc construction/destruction

CPwddlgDoc::CPwddlgDoc()
{
        // TODO: add one-time construction code here
}

CPwddlgDoc::~CPwddlgDoc()
{
}
/////////////////////////////////////////////////////////////////////////////
```

Authentication code 253156
Secret code 5847885168

ERROR
The message has been modified.

(b) A warning message issued by authentication process.

Figure 8. 2 An example of authentication results.

Chapter 9

# Covert Communication via PDF Files and PDF File Authentication by Invisible Codes

## 9.1 Idea of Proposed Methods

Portable Document Format (PDF) files, created by Adobe Systems for document exchange [63], are very popular for document exchange nowadays. The format was created by Adobe Systems and is a type of fixed layout for representing documents in a manner independent of the application software, hardware, and operating system. Each PDF file contains a complete description of a 2-D document which includes texts, fonts, images, and vector graphics. Many PDF readers and writers are available for reading and creating PDF files.

Additionally, ASCII codes were designed to represent 8-bit characters for information interchange [65]. There are totally 256 of them among which 95 ones are printable, numbered 32 to 7E (hexadecimal). These 95 codes together with the control code 0A (for *line feeding*) are used for representing secret messages in this study. They are listed in Table 9.1. The width of a text character represented by an ASCII code as seen in a PDF reader may be specified by a value in an array called “widths” in the type-1 font dictionary in a PDF file [63].

It is found in this study that the ASCII code A0 (for *non-breaking space*), when embedded in a string of text characters, become *invisible* in the PDF reader, Adobe Reader 8.1.2, under the Windows OS environment. This phenomenon may be utilized for data hiding, as done in this study.

On the other hand, for security it is necessary to verify the authenticity of a file received from another party or kept for a long time in a certain environment, before the file is used for various purposes. This is the *authentication* problem of the file, which should be solved for protection of the PDF file against unintentional changes and malicious manipulations.

As mentioned previously, we have proposed a method using a data hiding technique for covert communication via PDF files and a method for authentication of PDF files by the invisible codes mentioned above. The principle for the former method will be described in Section 9.2. The detail of it will be described in Section 9.3, and the detail of the latter method will be described in Section 9.4. Some concluding remarks are given in Section 9.5.

Table 9.1 ASCII codes selected for message representations in this study.

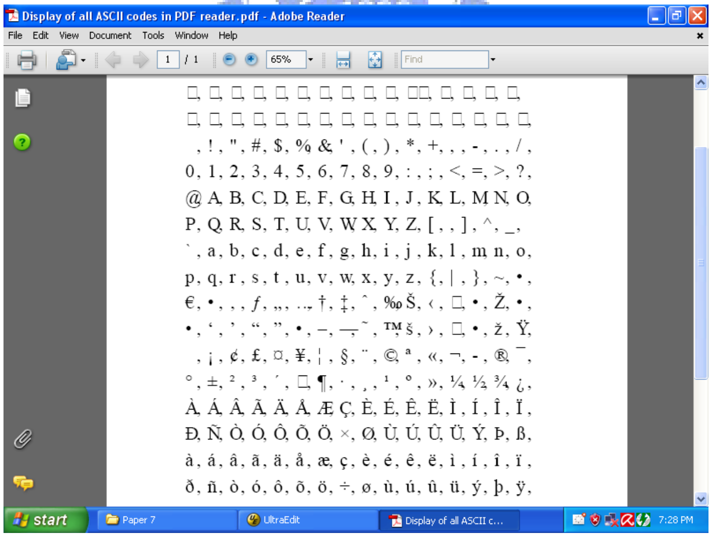
| Index | Chara-cter | Hexadeci-mal code | Index | Chara-cter | Hexadeci-mal code | Index | Chara-cter | Hexadeci-mal code | Index | Chara-cter | Hexadeci-mal code |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | *LF* | 0A | 25 | 7 | 37 | 49 | O | 4F | 73 | g | 67 |
| 2 |  | 20 | 26 | 8 | 38 | 50 | P | 50 | 74 | h | 68 |
| 3 | ! | 21 | 27 | 9 | 39 | 51 | Q | 51 | 75 | i | 69 |
| 4 | " | 22 | 28 | : | 3A | 52 | R | 52 | 76 | j | 6ª |
| 5 | # | 23 | 29 | ; | 3B | 53 | S | 53 | 77 | k | 6B |
| 6 | $ | 24 | 30 | < | 3C | 54 | T | 54 | 78 | l | 6C |
| 7 | % | 25 | 31 | = | 3D | 55 | U | 55 | 79 | m | 6D |
| 8 | & | 26 | 32 | > | 3E | 56 | V | 56 | 80 | n | 6E |
| 9 | ' | 27 | 33 | ? | 3F | 57 | W | 57 | 81 | o | 6F |
| 10 | ( | 28 | 34 | @ | 40 | 58 | X | 58 | 82 | p | 70 |
| 11 | ) | 29 | 35 | A | 41 | 59 | Y | 59 | 83 | q | 71 |
| 12 | * | 2A | 36 | B | 42 | 60 | Z | 5ª | 84 | r | 72 |
| 13 | + | 2B | 37 | C | 43 | 61 | [ | 5B | 85 | s | 73 |
| 14 | , | 2C | 38 | D | 44 | 62 | \ | 5C | 86 | t | 74 |
| 15 | - | 2D | 39 | E | 45 | 63 | ] | 5D | 87 | u | 75 |
| 16 | . | 2E | 40 | F | 46 | 64 | ^ | 5E | 88 | v | 76 |
| 17 | / | 2F | 41 | G | 47 | 65 | _ | 5F | 89 | w | 77 |
| 18 | 0 | 30 | 42 | H | 48 | 66 | ` | 60 | 90 | x | 78 |
| 19 | 1 | 31 | 43 | I | 49 | 67 | a | 61 | 91 | y | 79 |
| 20 | 2 | 32 | 44 | J | 4A | 68 | b | 62 | 92 | z | 7A |
| 21 | 3 | 33 | 45 | K | 4B | 69 | c | 63 | 93 | { | 7B |
| 22 | 4 | 34 | 46 | L | 4C | 70 | d | 64 | 94 | \| | 7C |
| 23 | 5 | 35 | 47 | M | 4D | 71 | e | 65 | 95 | } | 7D |
| 24 | 6 | 36 | 48 | N | 4E | 72 | f | 66 | 96 | ~ | 7E |

## 9.2 Principle of Encoding Message Data

Two types of invisibility may be created from A0. One type is created by specifying the width of A0 appearing in the PDF reader to be the same as that of the *original white space* represented by the ASCII code 20. Then, after being inserted *between two words* in the text of a PDF file, A0 appears to be exactly the same as a white space exhibited by the code 20. Figure 9.1 illustrates this phenomenon. So A0 and 20 may be used alternatively as *between-word spaces* so that we may encode a bit *b* of the secret message and embed it at a between-word location according to the following binary coding technique:

*if b = 1, then replace 20 at the between-word location by A0; else, make no change* (1)

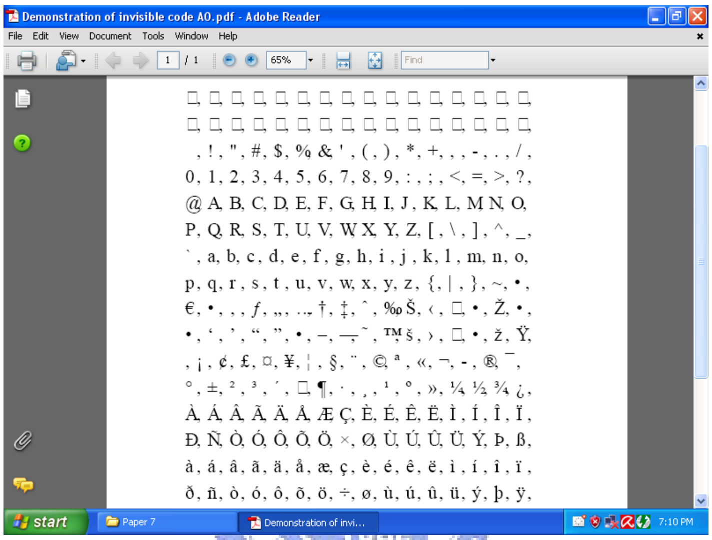
which we will call *alternative space coding*.


Display of all ASCII codes in PDF reader.pdf - Adobe Reader
File Edit View Document Tools Window Help
1 / 1
65%
Find
start
Paper 7
UltraEdit
Display of all ASCII c...
7:28 PM

```
□, □, □, □, □, □, □, □, □, □, □□, □, □, □, □,
□, □, □, □, □, □, □, □, □, □, □, □, □, □, □, □,
 , !, ", #, $, % &, ', (, ), *, +, ,, -, ., /,
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, :, ;, <, =, >, ?,
@ A, B, C, D, E, F, G, H, I, J, K, L, M N, O,
P, Q, R, S, T, U, V, W X, Y, Z, [, , ], ^, _,
`, a, b, c, d, e, f, g, h, i, j, k, l, m n, o,
p, q, r, s, t, u, v, w, x, y, z, {, |, }, ~, •,
€, •, ‚, ƒ, „, …, †, ‡, ˆ, ‰ Š, ‹, □, •, Ž, •,
•, ‘, ’, “, ”, •, –, —, ˜, ™ š, ›, □, •, ž, Ÿ,
 , ¡, ¢, £, ¤, ¥, ¦, §, ¨, © ª, «, ¬, -, ® ¯,
°, ±, ², ³, ´, □ ¶, ·, ¸, ¹, º, », ¼ ½ ¾ ¿,
À Á Â Ã Ä Å Æ Ç, È, É, Ê, Ë, Ì, Í, Î, Ï,
Ð, Ñ, Ò, Ó, Ô, Õ, Ö, ×, Ø, Ù, Ú, Û, Ü, Ý, Þ, ß,
à, á, â, ã, ä, å, æ, ç, è, é, ê, ë, ì, í, î, ï,
ð, ñ, ò, ó, ô, õ, ö, ÷, ø, ù, ú, û, ü, ý, þ, ÿ,
```

Figure 9.1 Display of all ASCII codes in Adobe Reader 8.1.2, in which only 20 and A0 appear to be white spaces (the first spaces in the 3rd and the 11th lines)

The other type of invisibility is created by setting the width of A0 to be *zero* in the PDF file. Then after being inserted *between two characters*, A0 appears to be *nothing* just like nonexistent in a PDF reader, as found in this study. Figure 9.2 illustrates this phenomenon. This invisibility is still true even when multiple A0's are all embedded at a single between-character location. Figure 9.3 illustrates this phenomenon. We say that A0 is used as a *null code* in this way, and contrastively, as a *spacing code* in alternative space coding described by (1) above.



Demonstration of invisible code A0.pdf - Adobe Reader
File Edit View Document Tools Window Help
1 / 1
65%
Find
start
Paper 7
Demonstration of invi...
7:10 PM

```
□, □, □, □, □, □, □, □, □, □, □, □, □, □, □, □,
□, □, □, □, □, □, □, □, □, □, □, □, □, □, □, □,
 , !, ", #, $, %, &, ', (, ), *, +, ,, -, ., /,
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, :, ;, <, =, >, ?,
@, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O,
P, Q, R, S, T, U, V, W, X, Y, Z, [, \, ], ^, _,
`, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o,
p, q, r, s, t, u, v, w, x, y, z, {, |, }, ~, •,
€, •, ‚, ƒ, „, …, †, ‡, ˆ, ‰, Š, ‹, □, •, Ž, •,
•, ‘, ’, “, ”, •, –, —, ˜, ™, š, ›, □, •, ž, Ÿ,
, ¡, ¢, £, ¤, ¥, ¦, §, ¨, ©, ª, «, ¬, -, ®, ¯,
°, ±, ², ³, ´, □, ¶, ·, ¸, ¹, º, », ¼, ½, ¾, ¿,
À, Á, Â, Ã, Ä, Å, Æ, Ç, È, É, Ê, Ë, Ì, Í, Î, Ï,
Ð, Ñ, Ò, Ó, Ô, Õ, Ö, ×, Ø, Ù, Ú, Û, Ü, Ý, Þ, ß,
à, á, â, ã, ä, å, æ, ç, è, é, ê, ë, ì, í, î, ï,
ð, ñ, ò, ó, ô, õ, ö, ÷, ø, ù, ú, û, ü, ý, þ, ÿ,
```

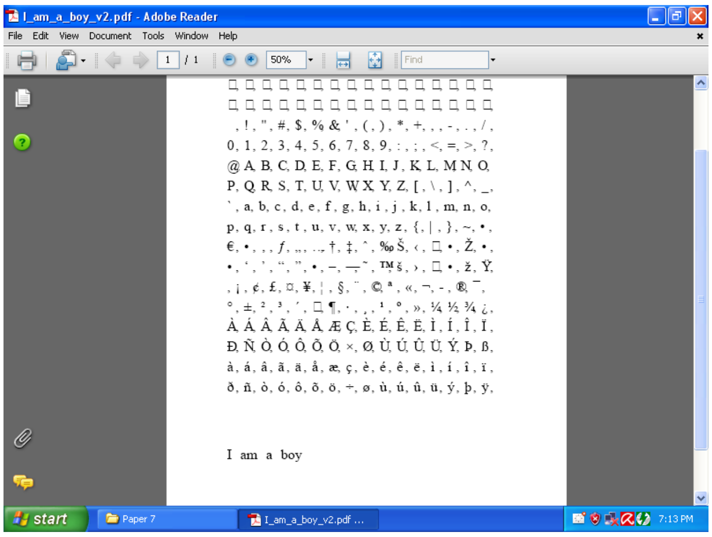
Figure 9.2 Display of all ASCII codes in the window of Adobe Reader 8.1.2, in which the width of A0 was set to be zero so that A0 becomes nonexistent (i.e., there is no space before the first comma in the 11th row, as compared with Figure 9.1).

Note that the width of the original space code 20 cannot be changed to be zero because it is used as a normal space between every two words in a PDF file. So, A0, when used as a null code, has *no symmetrical code* for use to implement *binary coding* like (1). But we may still hide a character *C* of the message by *unitary coding* at a between-character location in the following way:

*if the index of C = m, embed m consecutive A0's at the between-character location* (2)

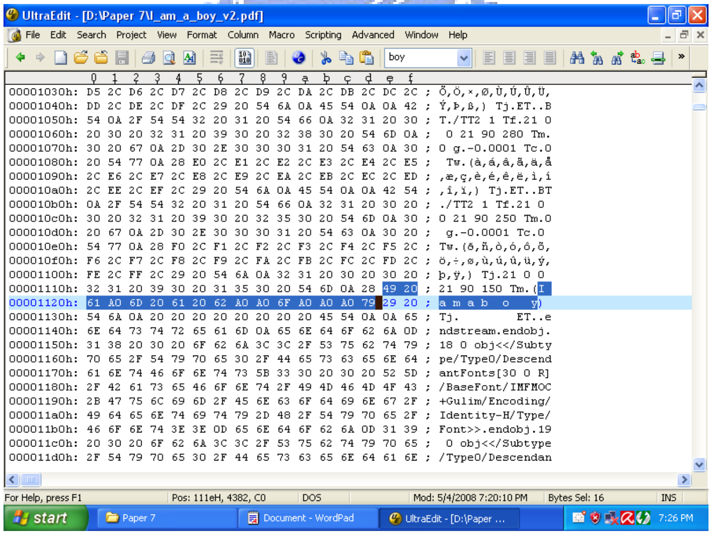
which will be called *null space coding*.

Note that A0 can only be used in one of the two ways of coding and not in both in each *page* of a PDF file because its width can only be specified *once* for each PDF page.


I_am_a_boy_v2.pdf - Adobe Reader
File Edit View Document Tools Window Help
1 / 1
50%
Find
start
Paper 7
I_am_a_boy_v2.pdf ...
7:13 PM

```
, !, ", #, $, % & ' , ( , ) , *, +, , , -, . , / ,
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, : , ; , <, =, >, ?,
@ A, B, C, D, E, F, G, H, I, J , K, L, M, N, O,
P, Q, R, S, T, U, V, W, X, Y, Z, [ , \ , ] , ^, _,
` , a, b, c, d, e, f, g, h, i , j , k, l , m, n, o,
p, q, r, s, t, u, v, w, x, y, z, {, | , }, ~, •,
€, •, , , ƒ, „, …, †, ‡, ˆ, ‰ Š, ‹, □ •, Ž, •,
•, ‘, ’, “, ”, •, –, —, ˜, ™ š, ›, □ •, ž, Ÿ,
, ¡, ¢, £, ¤, ¥, ¦, §, ¨, ©, ª, «, ¬, -, ®, ¯,
°, ±, ², ³, ´, □ ¶, ·, ¸, ¹, º, », ¼ ½ ¾ ¿,
À, Á, Â, Ã, Ä, Å, Æ Ç, È, É, Ê, Ë, Ì, Í, Î, Ï,
Ð, Ñ, Ò, Ó, Ô, Õ, Ö, ×, Ø, Ù, Ú, Û, Ü, Ý, Þ, ß,
à, á, â, ã, ä, å, æ, ç, è, é, ê, ë, ì, í, î, ï,
ð, ñ, ò, ó, ô, õ, ö, ÷, ø, ù, ú, û, ü, ý, þ, ÿ,

I  am  a  boy
```

(a) Appearance in Adobe Reader 8.1.2 of a sentence "I am a boy" with one, two, and three A0's inserted at locations between the characters a and m, b and o, and o and y.


UltraEdit - [D:\Paper 7\I_am_a_boy_v2.pdf]
File Edit Search Project View Format Column Macro Scripting Advanced Window Help
boy
start
Paper 7
Document - WordPad
UltraEdit - [D:\Paper ...
7:26 PM

```
            0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00001030h: D5 2C D6 2C D7 2C D8 2C D9 2C DA 2C DB 2C DC 2C ; Õ,Ö,×,Ø,Ù,Ú,Û,Ü,
00001040h: DD 2C DE 2C DF 2C 29 20 54 6A 0A 45 54 0A 0A 42 ; Ý,Þ,ß,) Tj.ET..B
00001050h: 54 0A 2F 54 54 32 20 31 20 54 66 0A 32 31 20 30 ; T./TT2 1 Tf.21 0
00001060h: 20 30 20 32 31 20 39 30 20 32 38 30 20 54 6D 0A ;  0 21 90 280 Tm.
00001070h: 30 20 67 0A 2D 30 2E 30 30 30 31 20 54 63 0A 30 ; 0 g.-0.0001 Tc.0
00001080h: 20 54 77 0A 28 E0 2C E1 2C E2 2C E3 2C E4 2C E5 ;  Tw.(à,á,â,ã,ä,å
00001090h: 2C E6 2C E7 2C E8 2C E9 2C EA 2C EB 2C EC 2C ED ; ,æ,ç,è,é,ê,ë,ì,í
000010a0h: 2C EE 2C EF 2C 29 20 54 6A 0A 45 54 0A 0A 42 54 ; ,î,ï,) Tj.ET..BT
000010b0h: 0A 2F 54 54 32 20 31 20 54 66 0A 32 31 20 30 20 ; ./TT2 1 Tf.21 0
000010c0h: 30 20 32 31 20 39 30 20 32 35 30 20 54 6D 0A 30 ; 0 21 90 250 Tm.0
000010d0h: 20 67 0A 2D 30 2E 30 30 30 31 20 54 63 0A 30 20 ;  g.-0.0001 Tc.0
000010e0h: 54 77 0A 28 F0 2C F1 2C F2 2C F3 2C F4 2C F5 2C ; Tw.(ð,ñ,ò,ó,ô,õ,
000010f0h: F6 2C F7 2C F8 2C F9 2C FA 2C FB 2C FC 2C FD 2C ; ö,÷,ø,ù,ú,û,ü,ý,
00001100h: FE 2C FF 2C 29 20 54 6A 0A 32 31 20 30 20 30 20 ; þ,ÿ,) Tj.21 0 0
00001110h: 32 31 20 39 30 20 31 35 30 20 54 6D 0A 28 49 20 ; 21 90 150 Tm.(I
00001120h: 61 A0 6D 20 61 20 62 A0 A0 6F A0 A0 A0 79 29 20 ; a m a b  o   y)
00001130h: 54 6A 0A 20 20 20 20 20 20 20 20 45 54 0A 0A 65 ; Tj.        ET..e
00001140h: 6E 64 73 74 72 65 61 6D 0A 65 6E 64 6F 62 6A 0D ; ndstream.endobj.
00001150h: 31 38 20 30 20 6F 62 6A 3C 3C 2F 53 75 62 74 79 ; 18 0 obj<</Subty
00001160h: 70 65 2F 54 79 70 65 30 2F 44 65 73 63 65 6E 64 ; pe/Type0/Descend
00001170h: 61 6E 74 46 6F 6E 74 73 5B 33 30 20 30 20 52 5D ; antFonts[30 0 R]
00001180h: 2F 42 61 73 65 46 6F 6E 74 2F 49 4D 46 4D 4F 43 ; /BaseFont/IMFMOC
00001190h: 2B 47 75 6C 69 6D 2F 45 6E 63 6F 64 69 6E 67 2F ; +Gulim/Encoding/
000011a0h: 49 64 65 6E 74 69 74 79 2D 48 2F 54 79 70 65 2F ; Identity-H/Type/
000011b0h: 46 6F 6E 74 3E 3E 0D 65 6E 64 6F 62 6A 0D 31 39 ; Font>>.endobj.19
000011c0h: 20 30 20 6F 62 6A 3C 3C 2F 53 75 62 74 79 70 65 ;  0 obj<</Subtype
000011d0h: 2F 54 79 70 65 30 2F 44 65 73 63 65 6E 64 61 6E ; /Type0/Descendan
For Help, press F1   Pos: 111eH, 4382, C0   DOS   Mod: 5/4/2008 7:20:10 PM   Bytes Sel: 16   INS
```

(b) Appearances of the A0's in the window of UltraEdit (in the highlighted portion).

Figure 9.3 Invisibility of multiple A0's at between-character locations.

Alternative space coding has the advantage of incurring no increase of the PDF file size because it just replaces the space exhibited by 20 by another exhibited by A0.

However, if the between-word locations in a PDF file are few, then only a small number of bits may be embedded. On the contrary, since theoretically an unlimited number of A0's as null codes may be inserted at a between-character location, and since there are much more between-character locations than between-word locations, encoding efficiency of null space coding is much higher. But an obvious disadvantage is that the resulting PDF file size will be increased.

Therefore, three ways of coding for use in different application conditions, namely, pure alternative space coding, pure null space coding, and a mixture of them. In the third way, we may use alternative space coding first to embed as many bits in the secret message as possible, and then apply null space coding to embed the remaining portion of the message (in unit of character) using the last pages of the PDF files.

## 9.3 Message Hiding and Recovery for Covert Communication and Experimental Results

Normal text messages may be represented by the 96 characters with their corresponding ASCII codes listed in Table 9.1. For alternative space coding, each secret message should be transformed first into a bit string. For this, we concatenate the binary ASCII codes (each consisting of 8 bits) of the characters in the message as the desired string. The bit string then is embedded, bit by bit sequentially, into the between-word locations in the cover PDF file according to Rule (1) above.

To implement null space coding, the secret message is regarded as a string of characters represented by the 96 ASCII codes listed in Table 9.1. However, to reduce the total number of inserted A0's and so the resulting stego-file size, instead of applying Rule (2) above directly in which the number of A0's used to encode a character *C* is the index value *m* of *C*, we use less A0's to encode characters with higher occurrence frequencies in the secret message, following the principle of Huffman coding. That is, we assign a single A0 to encode the character with the largest frequency, two A0's to encode the character with the second largest frequency, and so on. And those characters among the 96 ones which do not appear in the secret message are encoded by *zero* A0 (because these characters will not be processed in the message decoding procedure). The encoding result is summarized as a table, called the *null space coding table*, with 96 entries filled with the corresponding numbers of A0's so obtained. For example, given the secret message "This is a covert communication method," after counting the frequencies of the 16 distinct characters in it, we have the corresponding null space coding table as shown in Table 9.2, in which all 0's in the entries have been removed to make the table more readable.

Every message will have a distinct null space coding table. For the purpose of message decoding using this table, it should be embedded as well in the cover PDF file as part of the hidden data. In practice, we do not embed all the content of the table but the numbers of A0's only into the first 96 consecutive between-character locations in the text of the cover PDF.

Finally, it is mentioned that the data recovery process is essentially a reverse of the data hiding process, with retrieval of the null space coding table conducted first, followed by extraction and decoding of the hidden message.

We report one of the experiments we conducted for null space coding here. The input secret message is "This is a covert communication method" to which the corresponding null space coding table has been shown in Table 9.2. After the table followed by the message was embedded into the cover PDF file shown in Figure 9.4(a), the initial part of the stego-file appearing in the UltraEdit window is shown in Figure 9.4(b), from which we can see the repeating A0's encoding each of the 96 commonly-used characters.

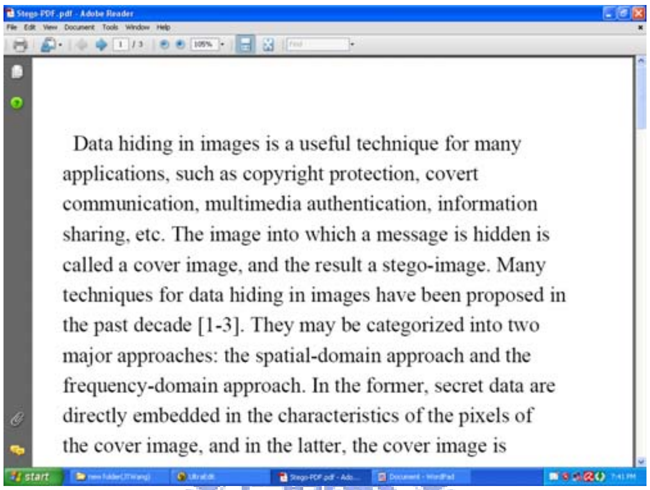
Table 9.2 Null space coding table for message “This is a covert communication method.”

| Index | Chara-cter | #A0’s embe-dded | Frequ-ency | Index | Chara-cter | #A0’s embe-dded | Frequ-ency | Index | Chara-cter | #A0’s embe-dded | Frequ-ency | Index | Chara-cter | #A0’s embe-dded | Frequ ency |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | *LF* | 12 | 1 | 25 | 7 | | | 49 | O | | | 73 | g | | |
| 2 | | 1 | 5 | 26 | 8 | | | 50 | P | | | 74 | h | 9 | 2 |
| 3 | ! | | | 27 | 9 | | | 51 | Q | | | 75 | i | 2 | 4 |
| 4 | " | | | 28 | : | | | 52 | R | | | 76 | j | | |
| 5 | # | | | 29 | ; | | | 53 | S | | | 77 | k | | |
| 6 | $ | | | 30 | < | | | 54 | T | 13 | 1 | 78 | l | | |
| 7 | % | | | 31 | = | | | 55 | U | | | 79 | m | 5 | 3 |
| 8 | & | | | 32 | > | | | 56 | V | | | 80 | n | 10 | 2 |
| 9 | ' | | | 33 | ? | | | 57 | W | | | 81 | o | 3 | 4 |
| 10 | ( | | | 34 | @ | | | 58 | X | | | 82 | p | | |
| 11 | ) | | | 35 | A | | | 59 | Y | | | 83 | q | | |
| 12 | * | | | 36 | B | | | 60 | Z | | | 84 | r | 15 | 1 |
| 13 | + | | | 37 | C | | | 61 | [ | | | 85 | s | 11 | 2 |
| 14 | , | | | 38 | D | | | 62 | \ | | | 86 | t | 6 | 3 |
| 15 | - | | | 39 | E | | | 63 | ] | | | 87 | u | 16 | 1 |
| 16 | . | | | 40 | F | | | 64 | ^ | | | 88 | v | 17 | 1 |
| 17 | / | | | 41 | G | | | 65 | _ | | | 89 | w | 77 | |
| 18 | 0 | | | 42 | H | | | 66 | ` | | | 90 | x | 78 | |
| 19 | 1 | | | 43 | I | | | 67 | a | 7 | 2 | 91 | y | 79 | |
| 20 | 2 | | | 44 | J | | | 68 | b | | | 92 | z | 7A | |
| 21 | 3 | | | 45 | K | | | 69 | c | 4 | 3 | 93 | { | 7B | |
| 22 | 4 | | | 46 | L | | | 70 | d | 14 | 1 | 94 | \| | 7C | |
| 23 | 5 | | | 47 | M | | | 71 | e | 8 | 2 | 95 | } | 7D | |
| 24 | 6 | | | 48 | N | | | 72 | f | | | 96 | ~ | 7E | |

In our experiments, to generate a stego-file, the process goes in the following way. First, we embed the text of the secret message into the PDF text in the “.txt.” format. We then tranform the result into a PDF file by a special PDF writer, which was implemented in this study, as the desired stego-file.

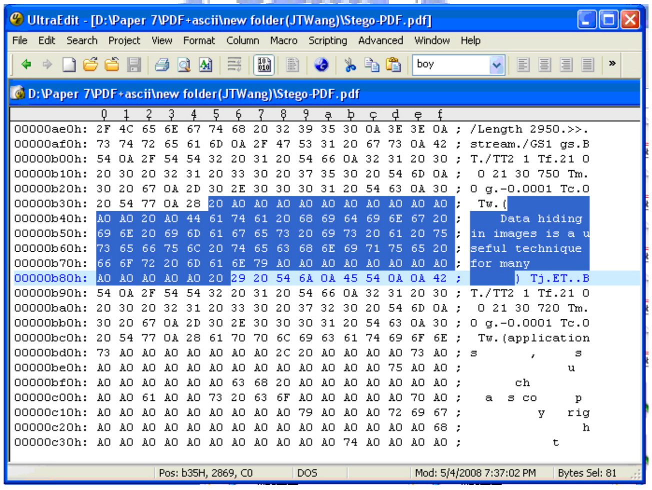
In particular, we can see in the highlight portion the 12 A0’s representing the

ASCII code 0A (line feed), the single A0 representing the space code 20, the 13 A0's representing the character T (the first character in the secret message), and so on. The stego-file appears as Figure 9.4(c) which is identical to Figure 9.4(a). The recovered message is shown in Figure 9.4(d).



Data hiding in images is a useful technique for many applications, such as copyright protection, covert communication, multimedia authentication, information sharing, etc. The image into which a message is hidden is called a cover image, and the result a stego-image. Many techniques for data hiding in images have been proposed in the past decade [1-3]. They may be categorized into two major approaches: the spatial-domain approach and the frequency-domain approach. In the former, secret data are directly embedded in the characteristics of the pixels of the cover image, and in the latter, the cover image is

(a) Cover file seen in Adobe Reader 8.1.2 window.



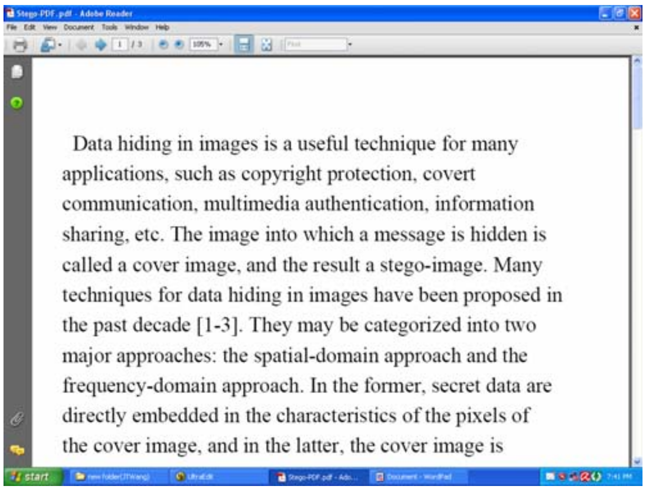
UltraEdit - [D:\Paper 7\PDF+ascii\new folder(JTWang)\Stego-PDF.pdf]
File Edit Search Project View Format Column Macro Scripting Advanced Window Help
boy
D:\Paper 7\PDF+ascii\new folder(JTWang)\Stego-PDF.pdf

```
            0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000ae0h: 2F 4C 65 6E 67 74 68 20 32 39 35 30 0A 3E 3E 0A ; /Length 2950.>>.
00000af0h: 73 74 72 65 61 6D 0A 2F 47 53 31 20 67 73 0A 42 ; stream./GS1 gs.B
00000b00h: 54 0A 2F 54 54 32 20 31 20 54 66 0A 32 31 20 30 ; T./TT2 1 Tf.21 0
00000b10h: 20 30 20 32 31 20 33 30 20 37 35 30 20 54 6D 0A ;  0 21 30 750 Tm.
00000b20h: 30 20 67 0A 2D 30 2E 30 30 30 31 20 54 63 0A 30 ; 0 g.-0.0001 Tc.0
00000b30h: 20 54 77 0A 28 20 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 ;  Tw.(
00000b40h: A0 A0 20 A0 44 61 74 61 20 68 69 64 69 6E 67 20 ;     Data hiding
00000b50h: 69 6E 20 69 6D 61 67 65 73 20 69 73 20 61 20 75 ; in images is a u
00000b60h: 73 65 66 75 6C 20 74 65 63 68 6E 69 71 75 65 20 ; seful technique
00000b70h: 66 6F 72 20 6D 61 6E 79 A0 A0 A0 A0 A0 A0 A0 A0 ; for many
00000b80h: A0 A0 A0 A0 A0 20 29 20 54 6A 0A 45 54 0A 0A 42 ;       ) Tj.ET..B
00000b90h: 54 0A 2F 54 54 32 20 31 20 54 66 0A 32 31 20 30 ; T./TT2 1 Tf.21 0
00000ba0h: 20 30 20 32 31 20 33 30 20 37 32 30 20 54 6D 0A ;  0 21 30 720 Tm.
00000bb0h: 30 20 67 0A 2D 30 2E 30 30 30 31 20 54 63 0A 30 ; 0 g.-0.0001 Tc.0
00000bc0h: 20 54 77 0A 28 61 70 70 6C 69 63 61 74 69 6F 6E ;  Tw.(application
00000bd0h: 73 A0 A0 A0 A0 A0 A0 A0 2C 20 A0 A0 A0 A0 73 A0 ; s       ,     s
00000be0h: A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 75 A0 A0 ;              u
00000bf0h: A0 A0 A0 A0 A0 A0 63 68 20 A0 A0 A0 A0 A0 A0 A0 ;       ch
00000c00h: A0 A0 61 A0 A0 73 20 63 6F A0 A0 A0 A0 A0 70 A0 ;   a  s co     p
00000c10h: A0 A0 A0 A0 A0 A0 A0 A0 A0 79 A0 A0 A0 72 69 67 ;          y   rig
00000c20h: A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 68 ;                h
00000c30h: A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 74 A0 A0 A0 A0 ;            t
```

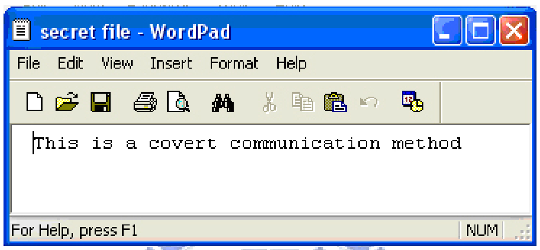
Pos: b35H, 2869, C0 | DOS | Mod: 5/4/2008 7:37:02 PM | Bytes Sel: 81

(b) Stego-file seen in UltraEdit window.

Figure 9.4 An experimental result of null space coding.


Data hiding in images is a useful technique for many applications, such as copyright protection, covert communication, multimedia authentication, information sharing, etc. The image into which a message is hidden is called a cover image, and the result a stego-image. Many techniques for data hiding in images have been proposed in the past decade [1-3]. They may be categorized into two major approaches: the spatial-domain approach and the frequency-domain approach. In the former, secret data are directly embedded in the characteristics of the pixels of the cover image, and in the latter, the cover image is

(c) Stego-file seen in Adobe Reader 8.1.2 window.



```
This is a covert communication method
```

(d) Extracted message.

Figure 9.4 An experimental result of null space coding (continued).

## 9.4 PDF Authentication Process and Experimental Results

As mentioned previously, the ASCII code A0, when embedded between two characters in PDF texts with its width set to zero (i.e., with *no width*), appears to be nothing like *nonexistent* in the PDF reader, Adobe Reader 8.1.2, under the Windows OS environment. This *invisibility* is still true when multiple A0's are embedded at a single between-character location. Embedding of such *invisible* codes in PDF files as authentication signals will enhance the security of the signals. Note that A0 is not used in common text contents.

The proposed PDF file authentication method generates an 8-bit number for each word in the text of the PDF file to be protected, with the help of a secret key in order to enhance security of the generated value. The value for each word then is transformed into a number of repeating A0's as the desired authentication signal, which is then embedded to the right of the word for use in future authentication.

Before describing the details of the proposed authentication signal generating and embedding process, we define some notations. We know that 1-bit exclusive-ORing of two bits $a$ and $b$, $a \oplus b$, results in 0 if $a = b$ and 1 if $a \neq b$. We define *8-bit* exclusive-OR operation on two ASCII codes $A = a_1a_2...a_8$ and $B = b_1b_2...b_8$ as $A \oplus B = c_1c_2...c_8$ where each $c_i = a_i \oplus b_i$ for $i = 1, 2, ..., 8$.

Let a word $W$ in the text $T$ of a PDF file $F$ be expressed as a string of characters represented by their corresponding 8-bit ASCII codes $A_{1,}$ $A$ , ..., $A_n$, that is, let $W = A_1A_2...A_n$. And let $k$ be a secret key used as the seed for an 8-bit random number generating function $f$. The proposed authentication signal generating and embedding process is as follows.

1. Scan the text $T$ in the input PDF file $F$, and for each word $W = A_1A_2...A_n$ in $T$, use the key $k$ and the function $f$ to generate in order a random number $K$.
2. Compute an 8-bit number $S$ for $W$ as $S = A_1 \oplus A_2 \oplus ... \oplus A_n \oplus K$.
3. Map $S$ to an integer $N$ which is the modulo-8 value of $S$, i.e., compute $N = S$ mod 8.
4. Embed $N$ repeating A0's as the authentication signal for $W$ to the right of $W$, i.e., embed them at the location between the character $A_n$ and the white space next to $A_n$.
5. Repeat the above steps until all words in $T$ are processed.

For example, let the word being processed is $W$ = "an" whose hexadecimal ASCII codes are 61 and 6E, and binary codes are 01100001 and 01101110, respectively. Suppose that the random number generated by $f$ is $K$ = 01010101. The

8-bit value $S$ then is [(01100001) ⊕ (01101110)] ⊕ (01010101) = (00001111) ⊕ (01010101) = 01011010, which is 90 in decimal form. And so $N$ is 90 mod 8 = 2. Consequently, we embed two A0's to the right of $W$.

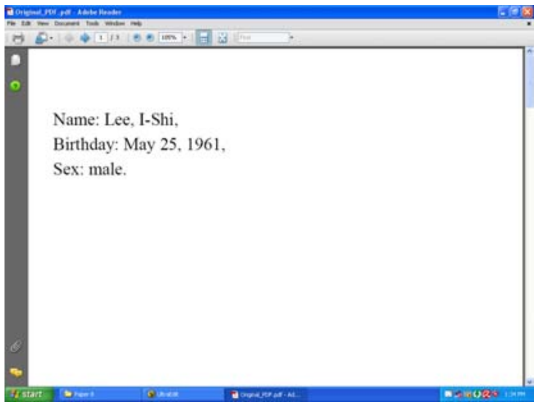
The detail of the authentication process with input PDF file $F'$ is as follows, where the key $k$ and the random number generator $f$ are the same as those used in the authentication signal generation scheme. The purpose of the process is to check the integrity of the text $T'$ in $F'$.

1. Scan the text $T'$ in the input file $F'$; and for each word $W' = A_1'A_2'\ldots A_m'$ in $T'$, count the number $N$ of A0's embedded to the right of $W'$, and use the key $k$ and the function $f$ to generate in order a random number $K$.
2. Compute an 8-bit value $S'$ for $W'$ as $S' = A_1' \oplus A_2' \oplus \ldots \oplus A_n' \oplus K$.
3. Map $S'$ to an integer $N'$ which is the modulo-8 value of $S'$, i.e., compute $N' = S'$ mod 8.
4. Compare $N'$ and $N$, and if $N' \neq N$, regard $W'$ as having been modified and mark it by changing all its characters into squares.
5. Repeat the above process until all words in $T'$ are processed.

Continuing the last example, suppose that the word "an" has been modified to be simply "a" because of the noun after it has been changed. Then, $W'$ = "a" has a single binary code 01100001. Also, assume that no word in the file has been deleted so that the random number $K$ generated for it is the same as that used before, i.e., $K$ = 01010101. So $S'$ = 01100001⊕01010101 = 00110100 which is 52 in decimal form. And thus $N'$ = 52 mod 8 = 4. But we know from the last example that the number $N$ of embedded A0's for the current word is $N$ = 2 which is not equal to 4. So we can decide that the word "a" is a result of tampering.

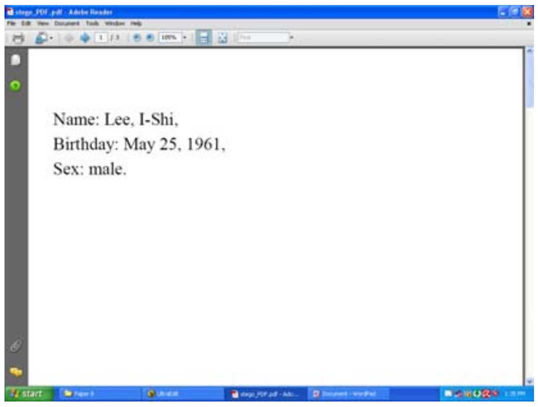
We take the modulo-8 value of the generated 8-bit number $S$ or $S'$ as the number of the invisible code A0 to be embedded. So, there are 8 possible cases, 0 through 7 A0's being embedded. This means that the probability for an attacker to guess the number of embedded A0's correctly to create a fake word is 1/8. To increase the security, we may, for example, take the modulo-12 value or even the modulo-16 value instead. Then, the probability will be decreased to 1/12 or 1/16 at the expense of inserting more A0's for each word. If this is still not satisfiable to the application need, one further enhancement is to compute the authentication signal using not just the data of the *current* word and the generated random number, but also the computed 8-bit number for the *previous* word, so that the formula $S = A_1 \oplus A_2 \oplus \ldots \oplus A_n \oplus K$ used in Step 2 of the authentication signal generating and embedding process described above is changed to $S = A_1 \oplus A_2 \oplus \ldots \oplus A_n \oplus K \oplus S'$ with $S'$ being the 8-bit number computed for the previous word. In this way, even if the number of A0's is guessed correctly for the current word (with a probability of 1/8), that for each of all the subsequent words must also be guessed, in contrast to the original case of independent guessing for each single word. The probability for correct guessing for the current word and all the subsequent ones will decrease exponentially because if there are $k$ words after the current one, this probability is $(1/8)^{k+1} = 1/8^{k+1}$.

We report a simple one of the experiments we have conducted. The text in a PDF file to be protected includes three lines of words: "Name: Lee, I-Shi," "Birthday: May 25, 1961," and "Sex: male." To have a complete authentication of all the characters in the text, the punctuation next to a word is also considered as part of the word in the authentication signal generation. So, totally there are nine words in the above three text lines. The appearance of these lines in the window of Adobe Reader 8.1.2 is shown in Figure 9.5(a). After the corresponding numbers of A0's for the nine words are computed and embedded to the right of the words, the resulting PDF file looks like Figure 9.5(b) which is identical to Figure 9.5(a), meaning that the inserted A0's are invisible indeed. The appearance of the resulting file in the UltraEdit window is shown in Figure 9.5(c), in which we can see the embedded A0's to the right of the words. We then simulated the case that the text in the file was tampered with intentionally, so that the last name "Lee" and the year "1961" were changed to be "Lin" and "1951," respectively. After the proposed authentication process was applied to the modified PDF file, the resulting PDF file is shown in Figure 9.5(d) in which the modified words have been marked as squares. Note that in the above results, for simpler demonstration using the figures, we did not adopt random numbers in the computations of the authentication signals.



Name: Lee, I-Shi,
Birthday: May 25, 1961,
Sex: male.

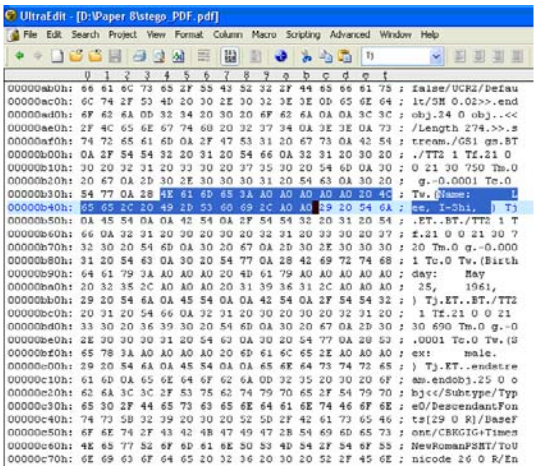
(a) Appearance of three text lines of original PDF file in window of Adobe Reader 8.1.2.



Name: Lee, I-Shi,
Birthday: May 25, 1961,
Sex: male.

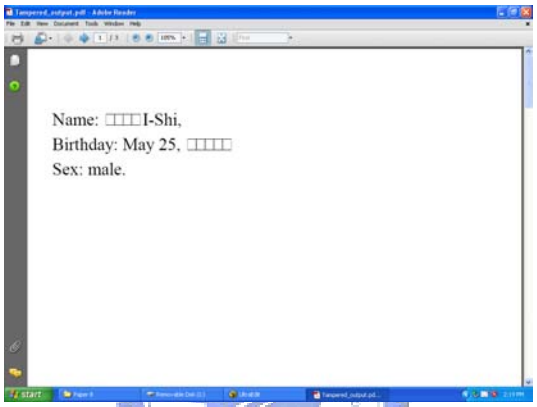
(b) Appearance of resulting PDF file with authentication signals (A0's) embedded.

Figure 9.5 An experimental result for authenticating a PDF file.


UltraEdit - [D:\Paper 8\stego_PDF.pdf]
File Edit Search Project View Format Column Macro Scripting Advanced Window Help
Tj

```
           0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000ab0h: 66 61 6C 73 65 2F 55 43 52 32 2F 44 65 66 61 75 ; false/UCR2/Defau
00000ac0h: 6C 74 2F 53 4D 20 30 2E 30 32 3E 3E 0D 65 6E 64 ; lt/SM 0.02>>.end
00000ad0h: 6F 62 6A 0D 32 34 20 30 20 6F 62 6A 0A 0A 3C 3C ; obj.24 0 obj..<<
00000ae0h: 2F 4C 65 6E 67 74 68 20 32 37 34 0A 3E 3E 0A 73 ; /Length 274.>>.s
00000af0h: 74 72 65 61 6D 0A 2F 47 53 31 20 67 73 0A 42 54 ; tream./GS1 gs.BT
00000b00h: 0A 2F 54 54 32 20 31 20 54 66 0A 32 31 20 30 20 ; ./TT2 1 Tf.21 0
00000b10h: 30 20 32 31 20 33 30 20 37 35 30 20 54 6D 0A 30 ; 0 21 30 750 Tm.0
00000b20h: 20 67 0A 2D 30 2E 30 30 30 31 20 54 63 0A 30 20 ;  g.-0.0001 Tc.0
00000b30h: 54 77 0A 28 4E 61 6D 65 3A A0 A0 A0 A0 A0 20 4C ; Tw.(Name:      L
00000b40h: 65 65 2C 20 49 2D 53 68 69 2C A0 A0 29 20 54 6A ; ee, I-Shi,  ) Tj
00000b50h: 0A 45 54 0A 0A 42 54 0A 2F 54 54 32 20 31 20 54 ; .ET..BT./TT2 1 T
00000b60h: 66 0A 32 31 20 30 20 30 20 32 31 20 33 30 20 37 ; f.21 0 0 21 30 7
00000b70h: 32 30 20 54 6D 0A 30 20 67 0A 2D 30 2E 30 30 30 ; 20 Tm.0 g.-0.000
00000b80h: 31 20 54 63 0A 30 20 54 77 0A 28 42 69 72 74 68 ; 1 Tc.0 Tw.(Birth
00000b90h: 64 61 79 3A A0 A0 A0 20 4D 61 79 A0 A0 A0 A0 A0 ; day:    May
00000ba0h: 20 32 35 2C A0 A0 A0 20 31 39 36 31 2C A0 A0 A0 ;  25,    1961,
00000bb0h: 29 20 54 6A 0A 45 54 0A 0A 42 54 0A 2F 54 54 32 ; ) Tj.ET..BT./TT2
00000bc0h: 20 31 20 54 66 0A 32 31 20 30 20 30 20 32 31 20 ;  1 Tf.21 0 0 21
00000bd0h: 33 30 20 36 39 30 20 54 6D 0A 30 20 67 0A 2D 30 ; 30 690 Tm.0 g.-0
00000be0h: 2E 30 30 30 31 20 54 63 0A 30 20 54 77 0A 28 53 ; .0001 Tc.0 Tw.(S
00000bf0h: 65 78 3A A0 A0 A0 A0 20 6D 61 6C 65 2E A0 A0 A0 ; ex:     male.
00000c00h: 29 20 54 6A 0A 45 54 0A 0A 65 6E 64 73 74 72 65 ; ) Tj.ET..endstre
00000c10h: 61 6D 0A 65 6E 64 6F 62 6A 0D 32 35 20 30 20 6F ; am.endobj.25 0 o
00000c20h: 62 6A 3C 3C 2F 53 75 62 74 79 70 65 2F 54 79 70 ; bj<</Subtype/Typ
00000c30h: 65 30 2F 44 65 73 63 65 6E 64 61 6E 74 46 6F 6E ; e0/DescendantFon
00000c40h: 74 73 5B 32 39 20 30 20 52 5D 2F 42 61 73 65 46 ; ts[29 0 R]/BaseF
00000c50h: 6F 6E 74 2F 43 42 4B 47 49 47 2B 54 69 6D 65 73 ; ont/CBKGIG+Times
00000c60h: 4E 65 77 52 6F 6D 61 6E 50 53 4D 54 2F 54 6F 55 ; NewRomanPSMT/ToU
00000c70h: 6E 69 63 6F 64 65 20 32 36 20 30 20 52 2F 45 6E ; nicode 26 0 R/En
```

(c) Appearance of resulting PDF file in window of UltraEdit (the A0's can be seen).


Tampered_output.pdf - Adobe Reader
File Edit View Document Tools Window Help
start

Name: □□□□I-Shi,
Birthday: May 25, □□□□□
Sex: male.

(d) Appearances of authentication result with squares indicating detected changes.

Figure 9.5 An experimental result for authenticating a PDF file (continued).

## 9.5 Concluding Remarks

A new covert communication method via PDF files is proposed. A secret message may be hidden steganographically into PDF files by alternative space coding and null space coding using the special ASCII code A0 which is invisible between words and between characters in the windows of common PDF readers if its width is set to be the same as that of the space code 20 and to be zero, respectively. Experimental results show the feasibility of the proposed method.

Also, a method for authenticating PDF files using a special ASCII code A0 has been proposed. For each word in the text of a PDF file to be protected, an authentication signal composed of repeating A0's is generated from the 8-bit ASCII codes of the characters composing the word as well as a random number. The signal is then embedded to the right of the word. These A0's are invisible in the window of common PDF readers, enhancing the security of the embedded authentication signals.

A corresponding authentication process to check the integrity of a processed PDF file has also been proposed. Each modified word in the file will be detected. Without the original secret key for use in generating the random numbers, malicious creation of a fake file is nearly impossible. Experimental results show the feasibility of the proposed authentication method.

Future researches may be directed to applying the proposed methods to other applications like watermarking of PDF files for copyright protection, enhancing the security of the proposed method, etc.

# Chapter 10
# Secret Communication through Web Pages and Automatic Authentication of Web Pages Using Special Space Codes in HTML Files

## 10.1 Idea of Proposed Method

Due to high accessibility on the Internet, it is convenient to use the web page as a communication channel by hiding secret messages in the HTML file of a cover web page. A merit here is that the secret message cannot be destructed illegally unless the website publishing the web page is intruded and the HTML file modified.

The proposed new secret communication method by embedding special space codes in the HTML files of web pages is described here. These codes appear as white spaces in the web page, and so may be used to encode secret message bits with steganographic effects. The codes are the result of a thorough investigation of all possible coding systems which can be applied in the HTML file. There are many of such codes, and each of them may be used to encode at least three message bits, increasing the data hiding capability and eliminating the weakness of certain methods [37] of using more than two space codes to encode one bit and creating undesirable double spacing at originally single-spaced between-word locations.

The proposed method carries out the communication work between two sites, a sender and a receiver, through the Internet via web page publishing and downloading in the following way.

1. At the sender site:

1.1 Create a web page containing mainly a piece of text.

1.2 Hide the secret message to be transmitted in the HTML file of the page by the proposed method.

1.3 Publish the web page on the Internet to make it accessible.

2. At the receiver site:

2.1 Browse the web page on the Internet.

2.2 Download its HTML codes by a code editor like UltraEdit or by a special program (*not* directly by the web browser using the "save as new file" command).

2.3 Extract the secret message hidden in the codes by the proposed method.

On the other hand, with rapid network technology developments, web pages published on the Internet often suffer from attacks. It is desired to have an *automatic* authentication scheme to check the fidelity and integrity of concerned web pages periodically without invoking human visual inspection. Specifically, it is wished to verify the text content of each web page more precisely *at the word level*. A new method based on the data hiding method for this purpose is proposed in this study.

A new automatic authentication method for checking the integrity of web page text contents is proposed. The method, aiming to check the authenticity of each single word, is based on a data hiding technique which uses some special space codes as authentication signals. Such codes, which are found in this study to be multiple and appear identical to normal white spaces in web pages, are used to encode certain binary mapping results from the word contents. These codes are then taken to replace the between-word spaces in the HTML codes, resulting in good steganographic effects. Security enhancement has also been considered, and related problems are solved by the use of secret keys and a multiple word encoding scheme.

In the sequel, in Section 10.2 we describe how secret messages are encoded. In Section 10.3, we describe the detail of the proposed covert communication method and some experimental result. In Section 10.4, we describe the proposed scheme to generate authentication signals using the special space codes, followed by the authentication signal embedding as well as authentication processes. The techniques proposed for security enhancement are described in Section 10.5, followed by some experimental results. Finally, some concluding remarks in Section 10.6.

## 10.2 Secret Message Coding Using Space Characters in HTML

The HTML, Hypertext Markup Language, was created for describing the structure of a web page, including its appearance and semantics. Many coding systems are applicable in the HTML to specify characters used in the web pages. It is found in this study that there exist many codes in the HTML, all of which appear to be a *white space* in the window of the web page browser of the Internet Explore (IE). These codes come from two distinct types of space characters, named (*normal*) *space and non-breaking space*, and are specified in the following ways.

1. *Direct character entry of the (normal) space* ---

   A white space will appear in a line of HTML if the space bar on the keyboard is pushed during character typing, and the hexadecimal ASCII code 20 will be inserted in the program codes of the HTML file.

2. *Numeric character reference of the (normal) space* ---

   We can also represent a (normal) space character in the HTML using a so-called *numeric character reference*, by the form **&#xhhhh;**, where hhhh = 0020 is the hexadecimal value representing the character's Unicode scalar value;

or by the form **&#dddd;**, where dddd = 0032 is the decimal value equivalent to the hexadecimal value. That is, we may represent the white space as **&#x20;** or **&#32;**. It is found in this study that the code **&#32** with the semicolon ";" missing is displayed as a space as well in the IE browser, while the code **&#x20** without the semicolon will *not* but as the code &#x20 *itself*, a peculiar phenomenon! A constraint to use **&#32** is that the character following it should not be a digit number; otherwise, it will become another code. We assume this constraint is satisfied in the HTML text in which this code is embedded.

3. *Numeric character reference of the non-breaking space* ---

   The non-breaking space with the hexadecimal ASCII code A0 is displayed in a web page browser like IE as a white space, too. Therefore, we may similarly represent it in the HTML using a numeric character reference, by one of the three forms **&#xA0;**, **&#160**;, and **&#160** (without a semicolon).

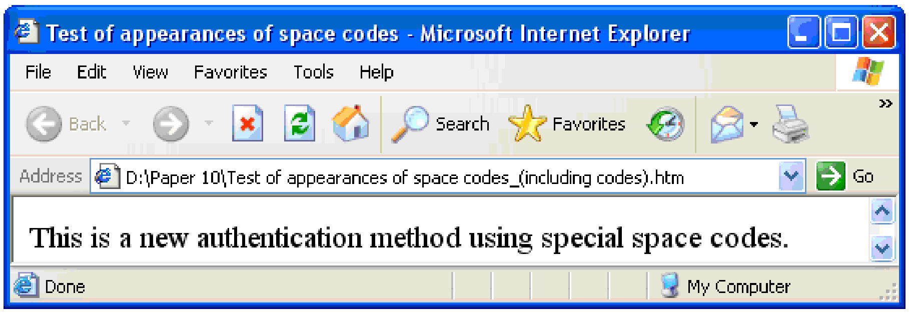
4. *Character entity reference of the non-breaking space* ---

   The HTML accepts a third way of character specification, called *character entity reference*, which is a short-length text name used to identify a character. For the non-breaking space, it is **&nbsp;**. It is found that without the semicolon, the code **&nbsp** still appears to be a white space, so two codes are available for representing the white space.

Totally, nine distinct codes may be used to specify a character which appears to be a white space in the web page browser of the IE, as summarized in Table 10.1. They are called *space codes* subsequently. An illustration of the appearances of all the space codes is shown in Figure 10.1. The first eight space codes of the nine ones are used to encode three message bits in this study as shown in the last table column, although all nine of them may be used to encode a digit of a novenary number as well.

Table 10.1 Character representations in HTML.

| No. | name | Reference type | Code type | Code inserted in HTML | Bits encoded |
|---|---|---|---|---|---|
| 1 | (normal) space | direct character entry | ASCII | typed space (with **20h** inserted) | 000 |
| 2 | (normal) space | numeric character reference | Unicode | **&#x20;** | 001 |
| 3 | (normal) space | numeric character reference | Unicode | **&#32;** | 010 |
| 4 | (normal) space | numeric character reference | Unicode | **&#32** | 011 |
| 5 | non-breaking space | numeric character reference | Unicode | **&#xA0;** | 100 |
| 6 | non-breaking space | numeric character reference | Unicode | **&#160**; | 101 |
| 7 | non-breaking space | numeric character reference | Unicode | **&#160** | 110 |
| 8 | non-breaking space | character entity reference | HTML name | **&nbsp;** | 111 |
| 9 | non-breaking space | character entity reference | HTML name | **&nbsp** | unused |

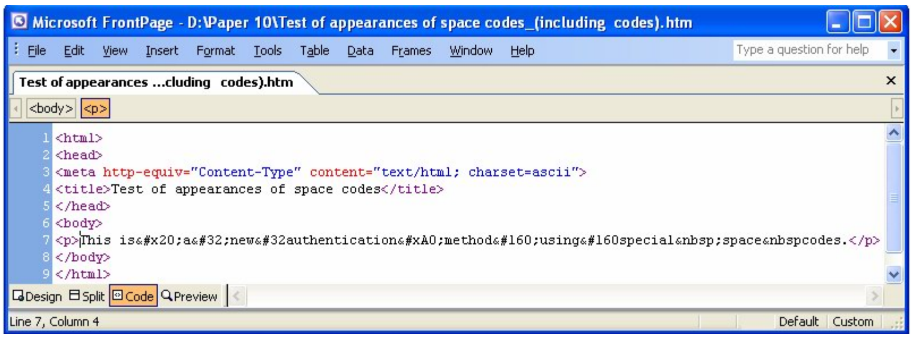

Test of appearances of space codes - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
Go
Done
My Computer

D:\Paper 10\Test of appearances of space codes_(including codes).htm

This is a new authentication method using special space codes.

(a) The space codes seen in the window of the IE.

Figure 10.1 Appearances of nine space codes as white spaces in the window of the IE.


Microsoft FrontPage - D:\Paper 10\Test of appearances of space codes_(including  codes).htm
File Edit View Insert Format Tools Table Data Frames Window Help
Type a question for help
Test of appearances ...cluding  codes).htm
<body> <p>
Design Split Code Preview
Line 7, Column 4
Default Custom

```html
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ascii">
<title>Test of appearances of space codes</title>
</head>
<body>
<p>This is&#x20;a&#32;new&#32authentication&#xA0;method&#160;using&#160special&nbsp;space&nbspcodes.</p>
</body>
</html>
```

(b) The codes inserted at between-word locations seen in the window of the FrontPage.

Figure 10.1 Appearances of nine space codes as white spaces in the window of the IE (continued).

## 10.3 Message Hiding and Experimental Results

During message hiding, we regard a given message as a sequence of characters, including letters, punctuations, white spaces, symbols, etc. Each character is represented as an 8-bit ASCII code, resulting in a string of bits which we encode three by three into the first eight space codes shown in Table 10.1. Each space code is then embedded at a between-word location in the *cover text* in the HTML file, replacing the original code 20h there, resulting in a *stego-text*. The embedded codes, after being extracted during message recovery, can be decoded uniquely by table lookup using Table 10.1.

To increase the security of the embedded message, we use a random number generator to randomize the order of the characters in the message string before they are encoded sequentially. A secret key is provided as the seed for the generator. The key is used again in message recovery to re-arrange the order of the extracted characters. Without the key, if the hidden characters cannot be properly re-ordered to get the correct message.

The detailed algorithms for embedding a gven message is as follows.

**Algorithm 10.1** ***Embedding of a secret message.***

***Input:*** a secret message $S$ in the form of a character string, a cover HTML text $T$, a secret key $K$, and a random number generator $f$.

***Output:*** a stego-HTML text $T'$ with $S$ embedded.

***Steps:***

1. Create a randomized version $S' = C_1'C_2'...C_n'$ of $S = C_1C_2...C_n$ in the following way, where $C_i$ and $C_i'$ represent characters of $S$ and $S'$, respectively, and $n$ is the number of characters in $S$.
   - 1.1 Generate $n$ distinct random numbers $k_1$, $k_2$, ..., $k_n$, within the range of 1 through $n$ using the generator $f$ with the secret key $K$ as the seed.
   - 1.2 For $i = 1, 2, .., n$, take $C_i'$ in $S'$ to be $C_{k_i}$ in $S$.
2. Convert the length $n$ of $S$ in the unit of character into a binary number and add leading 0's to it to form a $3m$-bit binary string $B$, where $m$ is a pre-selected integer such that $3m$ is no smaller than the length of any possible message to be hidden.
3. Transform each character in $S'$ into its 8-bit binary ASCII code and concatenate them to form a binary string $S_1$.
4. Concatenate $B$ and $S_1$ to form a binary string $S''$.
5. Embed $S''$ in $T$ in the following way.
   - 5.1 Append zero, one, or two 0's to $S_1$ to form another binary string $S_2$ with its length $n_2$ being a multiple of 3.
   - 5.2 Encode every three bits of $S_2$ into a space code $D$ according to the last column of Table 10.1.
   - 5.3 Embed $D$ in $T$ by replacing the (normal) space code 20h at a between-word location, starting from the top leftmost one in $T$ in a raster scanning order.
6. Take the resulting HTML text $T'$ as the output.

In the above algorithm we assume that the text $T$ is long enough to embed the message $S$. Also, the length of the message is also embedded in the leading between-word locations in $T$. This is necessary for the later work of message recovery to extract a correct numbers of characters from the stego-text. The detailed algorithm for extracting the embedded message is as follows.

**Algorithm 10.2.** ***Extraction of a secret message*.**

***Input:*** a stego-HTML text $T'$ with a message $S$ embedded, and a secret key $K$ and a random number generator $f$ as those used in Algorithm 10.1.

***Output:*** the embedded message $S$.

***Steps:***

1. Extract the length $n$ of the embedded message $S$ in $T'$ in the following way.
   1.1 For each of the $m$ leading between-word locations in $T'$ where $m$ is a pre-selected integer mentioned in Algorithm 10.1, acquire the space code embedded there and decode it into three bits according to the last column of Table 10.1, resulting in a $3m$-bit binary string $B$.
   1.2 While ignoring the leading 0's in $B$, convert it into an integer $n$ which presumably is the length of the embedded message $S$.
2. Compute the value $n_1 = \lceil n\times8/3\rceil$ which is the number of between-word locations in $T'$ where $S$ is embedded.
3. For each of the $n_1$ between-word locations after the $m$ leading ones in $T'$, acquire the space code there and decode it into three bits according to the last column of Table 10.1, resulting in $3n_1$-bit binary string $S_2$.
4. Take the leading $n\times8$ bits of $S_2$ to form a string $S'$ and transform every 8 bits of $S'$ into an ASCII character.
5. Create a randomized version $S = C_1C_2...C_n$ of $S' = C_1'C_2'...C_n'$ in the following way,

where $C_i$ and $C_i'$ represent characters of $S$ and $S'$, respectively, and $n$ is the number of characters in $S'$.

5.1 Generate $n$ distinct random numbers $k_1$, $k_2$, ..., $k_n$, within the range of 1 through $n$ using the generator $f$ with the same secret key $K$ as the seed.

5.2 For $i = 1, 2, .., n$, take $C_i$ in $S$ to be $C_{k_i}'$ in $S'$, resulting in a string of characters $S = C_1C_2...C_n$ as the desired output.

For security consideration, the length of secret data should be long enough, e. g., more than 256 characters, to reduce the probability for a hacker to guess the message correctly. Otherwise, another way of security protection may be adopted, that is, to conduct the reordering operation in Step 1 of Algorithm 10.1 and Step 5 of Algorithm 10.2 in unit of bits instead of in unit of characters. Since there are normally so many bits, it is almost impossible to get a correct guess. If these measures of security enhancement are taken, it can be figured out from the above algorithm that without a correct key, the embedded message, even when the stego-text is intercepted, is almost impossible to be recovered by a hacker.

In order to have a clear illustration of the proposed method and to see clearly the embedded codes in web page and HTML editor windows, we report first a simple example of the experiments we conducted without embedding the length of the message and without using a secret key. Let the message to be embedded be “sky” whose three characters “s,” “k,” and “y” have 8-bit ASCII codes 01110011, 01101011, and 01111001, respectively. So the message in binary string form is 011 100 110 110 101 101 111 001 which includes eight 3-bit segments, and can be encoded into eight space codes &#32 &#xA0; &#160 &#160 &#160; &#160; &nbsp; &#x20;. We embedded these codes at eight consecutive between-word locations in the following HTML text:

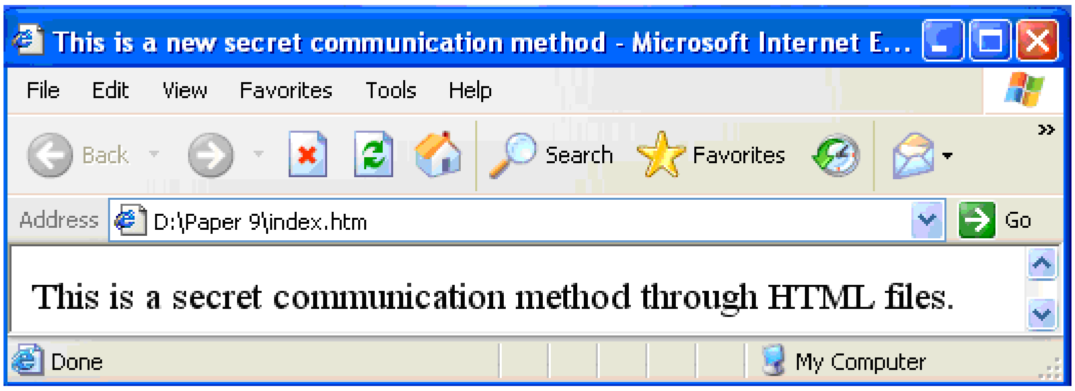
This is a secret communication method through HTML files.

Then the result is:

This&#32is&#xA0;a&#160new&#160communication&#160;method&#160;through&nbsp;HTML&#x20;files.

This stego-text, when observed in the web page browser of the IE, appears to be identical to that of the cover text, as shown in Figure 10.2.

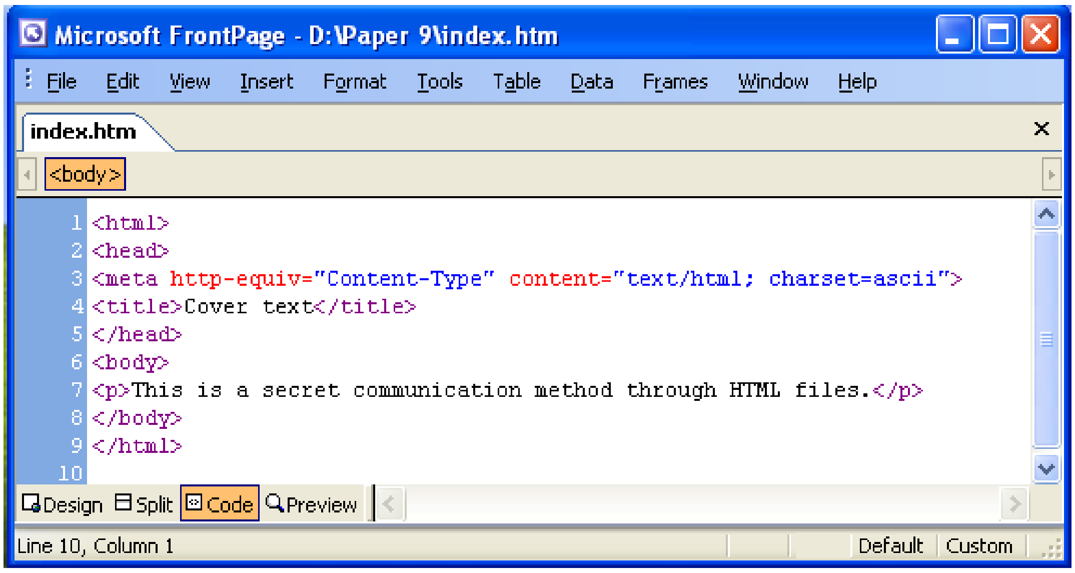
Another example of our experimental results is shown in Figure 10.3, in which we show a cover text in the IE and the Frontpage windows in Figures 10.3(a) and 10.3(b), respectively; and a secret message in the Notepad window in Figure 10.3(c). The length of the message is 96 characters which are embedded first into the cover text as a 15-bit number. The stego-text appearing in the IE and the Frontpage windows is shown in Figures 10.3(d) and 10.3(e), repsectively. From the identicalness of Figures 10.3(a) and 10.3(d), the steganographic effect of the space codes is confirmed.



This is a secret communication method through HTML files.

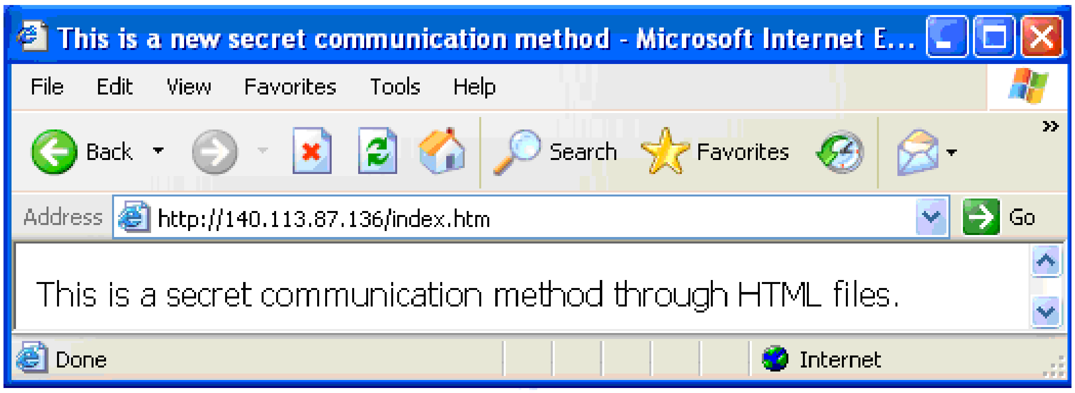
(a) Cover text seen in the window of the IE.

Figure 10.2. Invisibility of space codes for the message “sky” in an HTML text.


Microsoft FrontPage - D:\Paper 9\index.htm
File Edit View Insert Format Tools Table Data Frames Window Help
index.htm
<body>
Design Split Code Preview
Line 10, Column 1
Default Custom

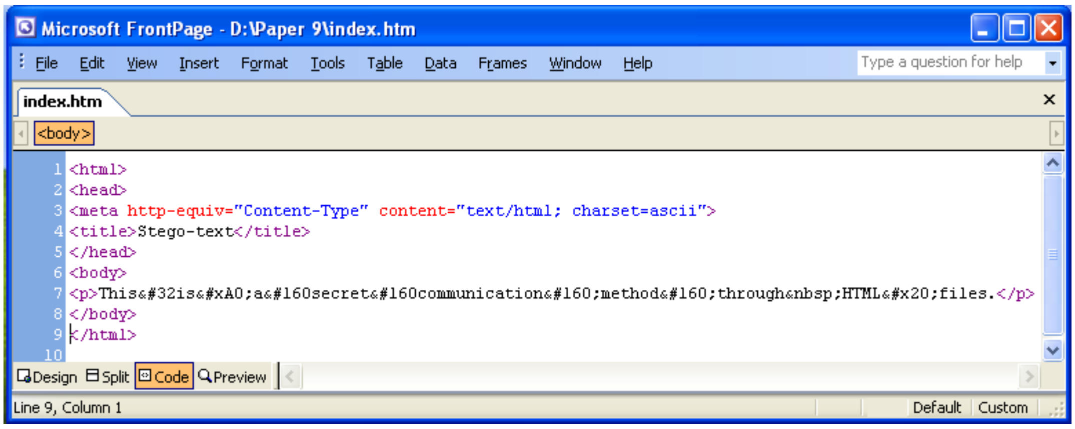
```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ascii">
<title>Cover text</title>
</head>
<body>
<p>This is a secret communication method through HTML files.</p>
</body>
</html>
```

(b) Cover text seen in the window of the FrontPage editor.


This is a new secret communication method - Microsoft Internet E...
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://140.113.87.136/index.htm
Go
Done
Internet

This is a secret communication method through HTML files.

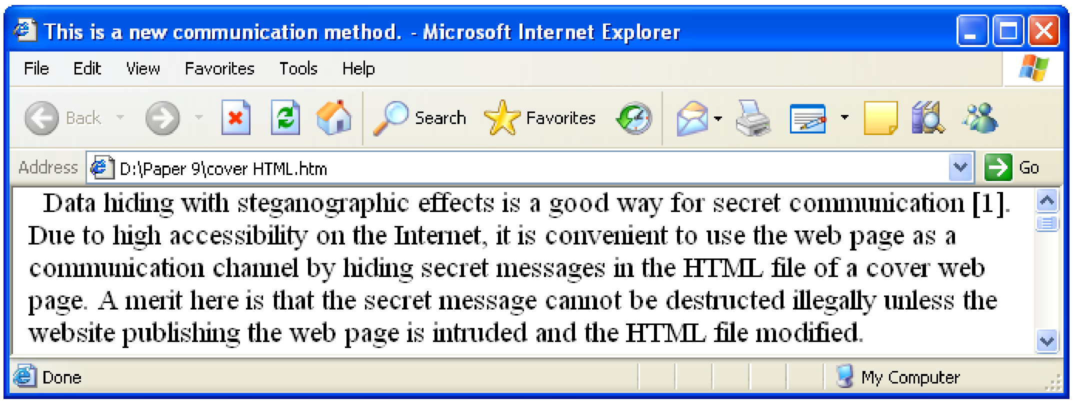
(c) Stego-text seen in the window of the IE.


Microsoft FrontPage - D:\Paper 9\index.htm
File Edit View Insert Format Tools Table Data Frames Window Help
Type a question for help
index.htm
<body>
Design Split Code Preview
Line 9, Column 1
Default Custom

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ascii">
<title>Stego-text</title>
</head>
<body>
<p>This&#32is&#xA0;a&#160secret&#160communication&#160;method&#160;through&nbsp;HTML&#x20;files.</p>
</body>
</html>
```

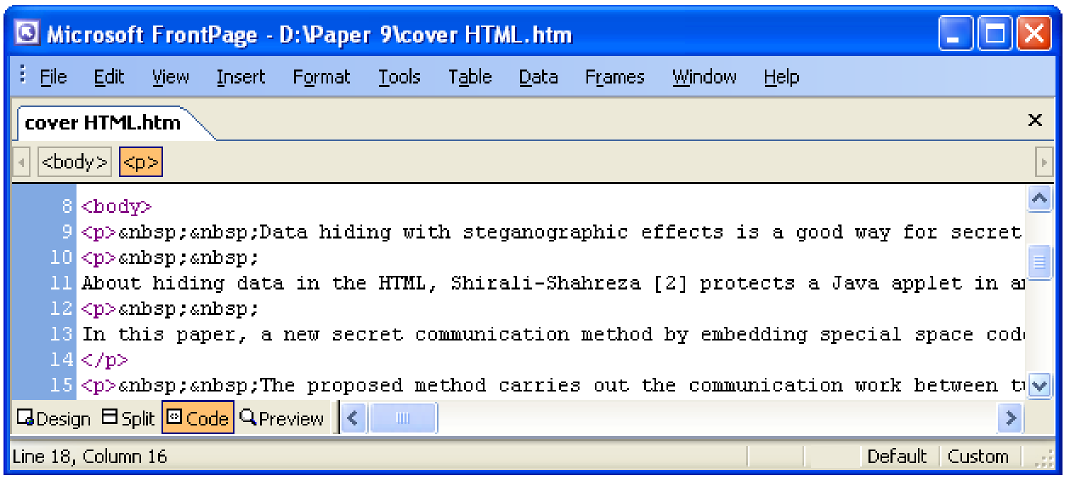
(d) Stego-text seen in the window of the FrontPage editor.

Figure 10.2. Invisibility of space codes for the message “sky” in an HTML text (continued).


This is a new communication method. - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address D:\Paper 9\cover HTML.htm Go
Done My Computer

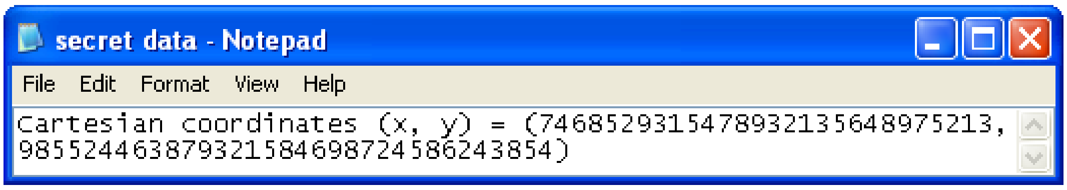
Data hiding with steganographic effects is a good way for secret communication [1]. Due to high accessibility on the Internet, it is convenient to use the web page as a communication channel by hiding secret messages in the HTML file of a cover web page. A merit here is that the secret message cannot be destructed illegally unless the website publishing the web page is intruded and the HTML file modified.

(a) Cover text seen in the window of the IE.


Microsoft FrontPage - D:\Paper 9\cover HTML.htm
File Edit View Insert Format Tools Table Data Frames Window Help
cover HTML.htm
<body> <p>
Design Split Code Preview
Line 18, Column 16 Default Custom

```
<body>
<p>&nbsp;&nbsp;Data hiding with steganographic effects is a good way for secret
<p>&nbsp;&nbsp;
About hiding data in the HTML, Shirali-Shahreza [2] protects a Java applet in a
<p>&nbsp;&nbsp;
In this paper, a new secret communication method by embedding special space code
</p>
<p>&nbsp;&nbsp;The proposed method carries out the communication work between tw
```

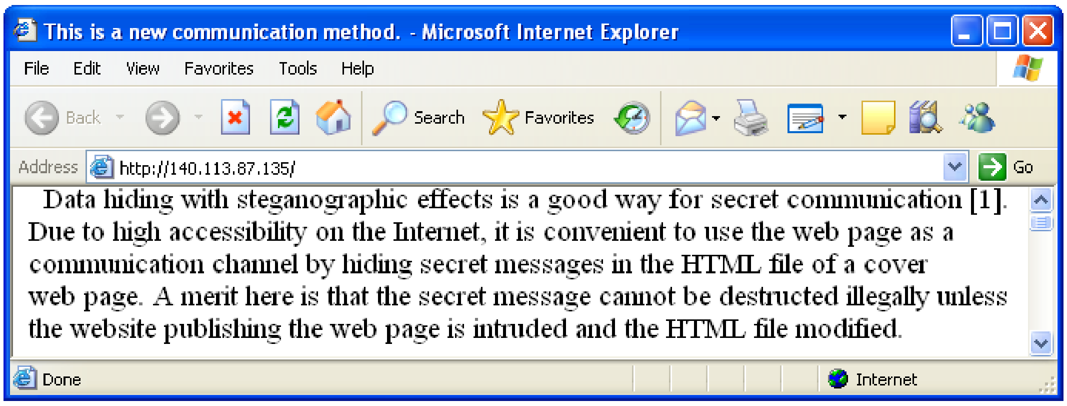
(b) Cover text seen in the window of the FrontPage editor.



```
Cartesian coordinates (x, y) = (746852931547893213564897521З,
985524463879321584698724586243854)
```

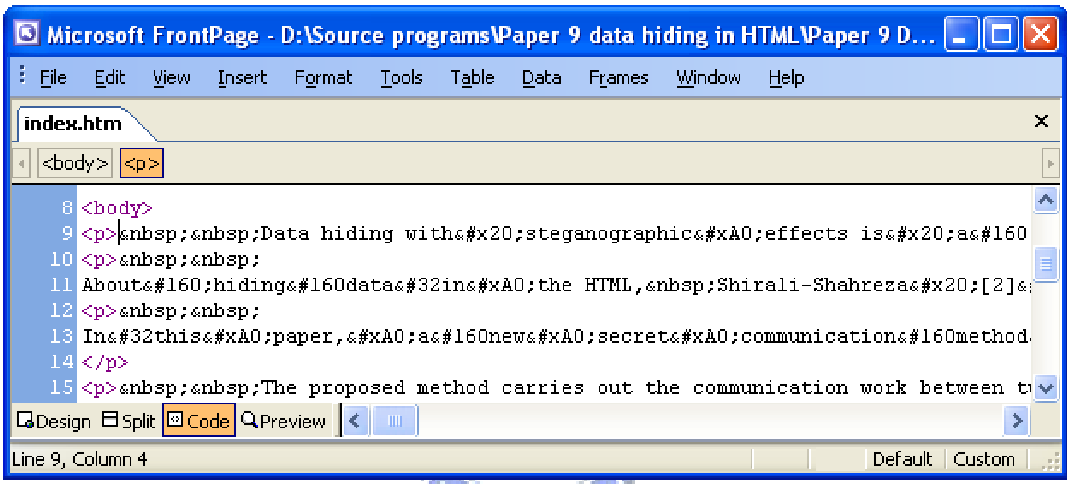
(c) A secret message seen in the Notepad window.

Figure 10.3. The embedded secret data.


This is a new communication method. - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://140.113.87.135/
Go
Done
Internet

Data hiding with steganographic effects is a good way for secret communication [1]. Due to high accessibility on the Internet, it is convenient to use the web page as a communication channel by hiding secret messages in the HTML file of a cover web page. A merit here is that the secret message cannot be destructed illegally unless the website publishing the web page is intruded and the HTML file modified.

(d) Stego-text seen in the window of the IE.


Microsoft FrontPage - D:\Source programs\Paper 9 data hiding in HTML\Paper 9 D...
File Edit View Insert Format Tools Table Data Frames Window Help
index.htm
<body> <p>
Design Split Code Preview
Line 9, Column 4
Default Custom

```
<body>
<p>&nbsp;&nbsp;Data hiding with&#x20;steganographic&#xA0;effects is&#x20;a&#160
<p>&nbsp;&nbsp;
About&#160;hiding&#160data&#32in&#xA0;the HTML,&nbsp;Shirali-Shahreza&#x20;[2]&
<p>&nbsp;&nbsp;
In&#32this&#xA0;paper,&#xA0;a&#160new&#xA0;secret&#xA0;communication&#160method
</p>
<p>&nbsp;&nbsp;The proposed method carries out the communication work between t
```

(e) Stego-text seen in the window of the FrontPage editor.

Figure 10.3. The embedded secret data (cont'd).

## 10.4 Automatic Authentication of Web Page Text Contents

To accomplish the goal of authenticating automatically the text of a web page at the word level, an authentication signal should be created for each word in the text, and embedded in the HTML codes of the text for periodical verification by a program implementing the authentication process. Warning should be issued if any word in the text is authenticated to have been modified, deleted, or inserted. An authentication signal generation and embedding process utilizing the space codes discussed previously as authentication signals is proposed as follows.

1. Map each word *w* in the HTML codes of the text of the web page to be protected by a function *h* into a binary integer *s*, called the *numerical authentication signal* of *w*, i.e., compute *s* = *h*(*w*).
2. Encode *s* by a space code *c* of the first eight ones listed in Table 10.1, called the *symbolic authentication signal* of *w*.
3. Replace the original space code 20 located at the right-hand side of *w* by the code *c*.

Since each space code also appears to be a white space in a web page browser, the resulting *stego-HTML* codes will appear in the browser to be a web page totally identical to the original one, arousing no suspicion from the observer. On the other hand, the proposed automatic authentication process is just a process of matching the previously-embedded authentication signal *s* for each word *w′* with the one *s′* computed from the current content of *w′* using the same mapping function *h*.

As a simple example, let the word *w* to be protected be "no" whose two characters have the decimal ASCII codes 110 and 111, respectively. Assume that the mapping function *h* takes the modulo-8 value of the sum of the decimal ASCII code values of the characters in the word. Then, the computed numerical authentication signal *s* for *w* is $s = h(w) = (110 + 111) \bmod 8 = 5$ whose 3-bit equivalent binary number is 101. According to Table 10.1, the space code encoding 101 is **&#160;** which is then taken to replace the hexadecimal code 20 to the right of the word. Now, suppose that the word "*no*" has been modified to be "ok" with an opposite meaning. The decimal ASCII code values for the two characters in it are 111 and 107. So the numerical authentication signal for this word *w′* is $s' = h(w') = (111 + 107) \bmod 8 = 2$ whose 3-bit equivalent is 010 and is encoded by the space code **&#32;**. This space code is different from the embedded one **&#160;**. So it is decided that the word “ok” is a result of tampering.

## 10.5 Security Consideration and Experimental Results

The above-mentioned simple processes, however, have several weaknesses in security from the viewpoint of automatic authentication without human involvement, as discussed in the following, in which solutions for removing these weaknesses are also proposed.

(1) Word position disordering and replacement of entire web page contents --- A hacker, who knows the above processes (including the used function *h*) as is usually assumed in information hiding studies, may destroy the web page content by just exchanging the orders of the words (each word assumed to include the embedded space code next to it). It can be figured out that this false web page can pass the authentication process. Even worse is the case that the hacker replaces the entire text content of a web page with all authentication signals for the new words recomputed and embedded. Such a fake web page obviously will also pass the above authentication. We propose to solve these problems by first putting the words into a certain order and then generating a series of corresponding random numbers, one for a word, to compute the authentication signals by the mapping $s_i = h(w_i, k_i)$ where $k_i$ is the random number generated for $w_i$. The random numbers are generated by a function controlled by a secret key as the seed. In this way, a web page with changed word orders cannot pass the authentication process, as can be figured out, because a word $w$ with its position changed will now be given a different random number so that the computed numerical authentication signal becomes different from the previously-embedded one. Also, it is easy to see that a hacker’s replacement of the entire text content of a web page with embedded authentication signals computed

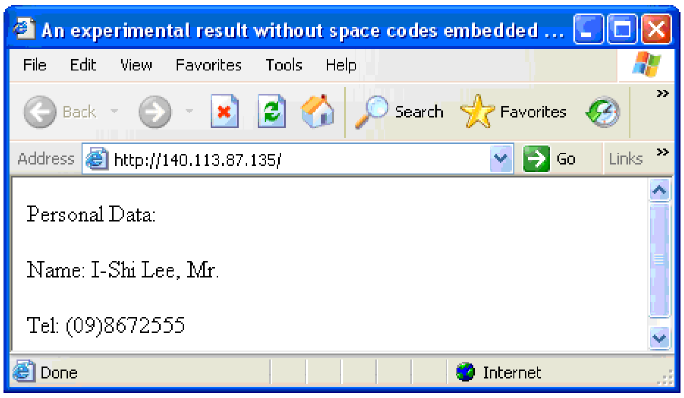
without a key will not pass the authentication process now.

(2) Guessing of authentication signals without a key --- The above modified process of authentication signal generation still has a weakness, i.e., the generated authentication signal for each word is a 3-bit number, which is encoded into one of the eight space codes so that the probability to guess it correctly is 1/8. That is, after inserting a replacing word, the hacker only has to guess the authentication signal for the word eight times before he/she can pass the authentication of the word. This is not secure enough. As a remedy, we propose to allow the mapping function $h(w, k)$ to yield a numerical authentication signal which, when transformed into binary, has more bits than three. For example, if we allow $h$ to yield 12 bits which may be encoded, three by three, into four space codes, then we may use four words to provide the four white spaces at their right-hand sides to embed the four space codes. This way of *multiple word encoding* is equivalent to regard four words as a single one by concatenating them together. More generally, if we want to yield $3n$ bits as the numerical authentication signal, we regard every $n$ words as a single one in computing the authentication signal $s = h(w, k)$. The signal $s$ is encoded into $n$ space codes which are then embedded at the right-hand sides of the $n$ words. Additionally, the mapping $h$ may be taken to be any reasonable function, such as one of the various existing hashing algorithms. We may even adopt the famous secure SHA-1 algorithm as $h$ with 54 words as input, and use a secret key as the seed to generate random numbers as its initial values. The algorithm yields 160 bits as output, to which we may affix two bits of 0's. We then encode the resulting 162 bits into 54 space codes (54 = 162/3) and embed the codes at the right-hand sides of the 54 words. The security of the protected 54 words will then be very high.

For a clearer illustration, we report a simple one of the experiments we have conducted, without using random numbers in computing the authentication signals.

The text in an HTML file to be protected includes three text lines: "Personal Data:" "Name: I-Shi Lee, Mr." and "Tel: (09)8672555." The corresponding web page seen in the IE window is shown in Figure 10.4(a). We regard a punctuation following a word as part of the word, and adopted a simple mapping function $h$ which considers two words as a single one, adds up the decimal values of the ASCII codes of all the characters in them to obtain a sum $S$, takes the modulo-64 value $M$ of $S$ as a 6-bit numerical authentication signal $s$, and encodes $M$ as two 3-bit numbers into two space codes by Table 10.1 as the symbolic authentication signal. These two space codes are finally taken to replace the two normal space codes 20 located to the right of the two words. That is, if the two words are $w_1 = c_{11}c_{12}...c_{1n_1}$ and $w_2 = c_{21}c_{22}...c_{2n_2}$ with $c_{ij}$'s being their ASCII codes and $d_{ij}$ the corresponding decimal values, then we compute $s$ as $s = h(w_1, w_2) = (d_{11}+d_{12} +...+d_{1n_1}+d_{21}+d_{22}+...+d_{2n_2}) \bmod 64 = b_1b_2...b_6$, with $b_1b_2b_3$ encoded into a space code and $b_4b_5b_6$ into another. After all the symbolic authentication signals for the word pairs were computed in this way and embedded appropriately, the resulting web page, as viewed in the IE window, appears to be as Figure 10.4(b), which looks no different from that shown in Figure 10.4(a). Figure 10.4(c) shows the corresponding stego-HTML codes in the FrontPage window, which can be seen to include all the space codes. To simulate web page intrusion and modification, the last name "Lee" in the second line was replaced by another, "Lin." After the authentication process was performed, the word pair "Lin, Mr." was authenticated to have been tampered with, and so was marked as bold italic, as shown in Figure 10.4(d). More of our experimental results show the feasibility of the proposed method.

A problem mentioned previously which need be solved is that the two space codes **&#32** and **&#160**, after being inserted, should not be followed by digits; otherwise, they will be regarded as codes with more digits instead of 32 and 160. One way out is to append to either of **&#32** and **&#160** one additional space code other than these two to stop this ambiguity, and decode the resulting *code pair* as just the first one only, which may still be done uniquely.

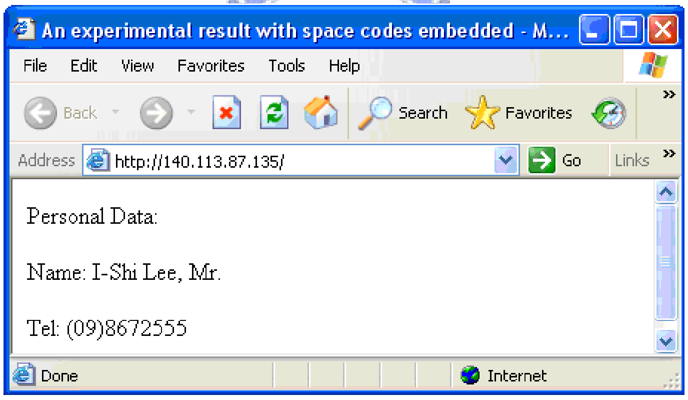

An experimental result without space codes embedded ...
File Edit View Favorites Tools Help
Back Search Favorites
Address http://140.113.87.135/ Go Links
Done Internet

Personal Data:

Name: I-Shi Lee, Mr.

Tel: (09)8672555

(a) Original web page seen in IE.


An experimental result with space codes embedded - M...
File Edit View Favorites Tools Help
Back Search Favorites
Address http://140.113.87.135/ Go Links
Done Internet

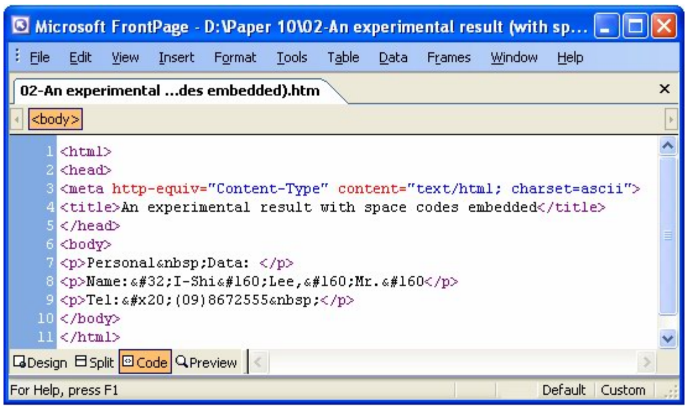
Personal Data:

Name: I-Shi Lee, Mr.

Tel: (09)8672555

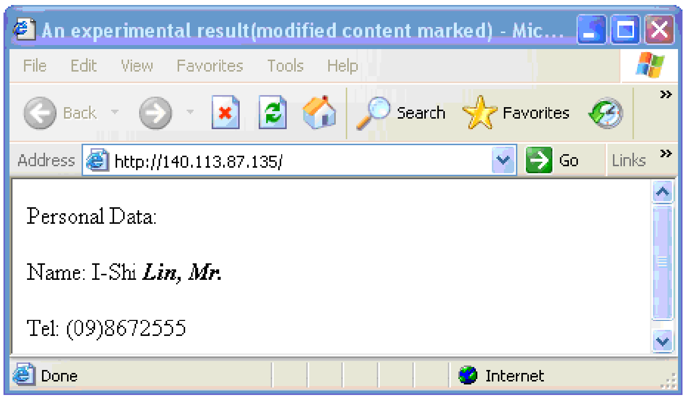
(b) Web page with embedded authentication signals (space codes) seen in IE.

Figure 10.4 An experimental result of authentication of a modified web page.


Microsoft FrontPage - D:\Paper 10\02-An experimental result (with sp...
File Edit View Insert Format Tools Table Data Frames Window Help
02-An experimental ...des embedded).htm
<body>
Design Split Code Preview
For Help, press F1 Default Custom

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ascii">
<title>An experimental result with space codes embedded</title>
</head>
<body>
<p>Personal&nbsp;Data: </p>
<p>Name:&#32;I-Shi&#160;Lee,&#160;Mr.&#160</p>
<p>Tel:&#x20;(09)8672555&nbsp;</p>
</body>
</html>
```

(c) Content of (b) with embedded space codes seen in FrontPage.


An experimental result(modified content marked) - Mic...
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://140.113.87.135/
Go
Links
Done
Internet

Personal Data:

Name: I-Shi ***Lin, Mr.***

Tel: (09)8672555

(d) Web page with detected modified word pair "Lee, Mr." marked as bold italic.

Figure 10.4 An experimental result of authentication of a modified web page (continued).

## 10.6 Concluding Remarks

A new secret communication method via web pages using special space codes in HTML files has been proposed. These codes appear as white spaces in the web page, and so may be used to encode secret message bits with steganographic effects. The codes are the result of a thorough investigation of all possible coding systems which can be applied in the HTML file. The character string of each message, before being embedded, is randomized with a secret key to enhance the security against illegal intercept and extraction. The original message embedded in the HTML text is non-destructible unless the web page server is intruded. Our experimental results show that the proposed method is feasible.

Also, an automatic authentication method for verifying a web page against illegal modifications of the words in the text of the web page has been proposed. The special space codes are used to encode binary mapping results from the word contents as authentication signals, and are embedded at between-word spaces in the HTML codes. Security enhancement techniques to prevent illicit word tampering and guessing of authentication signals have also been proposed, including the use of secret keys and the scheme of multiple word encoding. Experimental results show the feasibility of the proposed method.

Future researches may be directed to utilizing the space codes in other data hiding applications, further promotion of the security of the proposed method, and applying the space codes to other purposes, like copy protection.

# Chapter 11
# Conclusions and Suggestions for Future Research

## 11.1 Conclusions

In this dissertation, we have proposed ten techniques for data hiding in various types of images and text documents. Discussions and concluding remarks for each method have been given at the end of each chapter before. A brief summary of them are as follows:

(1) data hiding in binary images with distortion-minimizing capabilities by optimal block pattern coding and dynamic programming techniques;

(2) data hiding in grayscale images by dynamic programming based on a human visual model;

(3) data hiding in emails and applications by unused ASCII control codes;

(4) data hiding in color images by color replacements with reduction of image distortion and change noticeability;

(5) security protection of software programs by information sharing and authentication techniques using invisible ASCII control codes;

(6) covert communication with authentication via software programs using invisible ASCII codes;

(7) covert communication via PDF files by a data hiding technique;

(8) authentication of PDF files by invisible ASCII codes;

(9) secret communication via web pages using special space codes in HTML files;

(10) automatic authentication of web pages by data hiding using multiple space codes

in HTML files.

Experimental results have also been shown to prove the feasibility and practicality of the proposed methods.

## 11.2 Suggestions for Future Research

In the subsequent study, the following topics will be investigated.

(1) Data hiding in binary images ---

The proposed method is based on the use of 2×2 blocks. It may be extended by processing larger blocks because then, the number of block patterns which can be selected to encode messages will become larger as well, resulting in greater reductions of image distortions. However, there is a tradeoff here, i.e., the resulting data embedding capacity will decrease. Other future works may be directed to designing a better cost function from the perspective of the human visual system, imposing more constraints on the cost function to yield better image quality, and finding a better way to design encoding tables to reduce stego-image distortion further.

(2) Data hiding in grayscale images ---

The methods proposed previously are for data hiding in binary images. But binary images are few in real applications. Therefore, it is desired to extend the methods for data hiding in grayscale images. One possible way is to extent to embed multiple message data in a grayscale image for protecting the intellectual property right and authenticating multimedia data. It is also hoped that the human vision model be considered in the extension so that the resulting stego-image will cause less noticeability from observers. Other future works may be directed to design better encoding tables to reduce image distortion further.

(3) Data hiding in color images ---

Needless to say, data hiding in color images is even more useful for real applications. Although the methods for grayscale images may be extended directly to color images by considering each color channel as a grayscale image, we want to design a more genuine method by dealing the color image itself. Future researches may be directed to minimizing image distortion by uses of variable-sized color cubes, uses of a perspective HVS, random distributions of groups' colors in color cubes, etc. It is also hoped that the proposed method can be extended for various applications.

(4) Data hiding in text documents ---

It is desired to design data hiding methods for embedding data in e-mails in the future study. Possible applications of such methods include covert communication through e-mails and authentication of e-mail fidelity and integrity. It is also hoped that data hiding in software programs can be developed in this study, so that intellectual properties of various programs can be protected. Any illegal duplication or stealing of protected programs with embedded owner information can be disclosed. Finally, more investigations on hiding data in PDF and HTML documents utilize the rich data structures in such document formats.

# References

[1] S. Katzenbeisser and F. A. P. Petitolas, *Information Hiding Techniques for Steganography and Digital Watermarking*, Artech House, Boston, U. S. A., 2000.

[2] E. Koch and J. Zhao, "Embedding robust labels into images for copyright protection," *Proceedings of the International Congress on Intellectual Property Rights for Specialized Information, Knowledge and New Techniques*, pp. 242-251, Munich, Germany, 1995.

[3] D. Kundur, "Energy allocation principles for high capacity data hiding," *Proceedings of the IEEE International Conference on Image Processing*, Vancouver, Canada, Vol. 1, pp. 423-426, September 2000.

[4] L. M. Marvel, J. C. G. Boncelet, and C. T. Retter, "Spread spectrum image steganography," *IEEE Transactions on Image Processing*, Vol. 8, No. 8, pp. 1075-1083, August 1999.

[5] K. Matsui and K. Tanaka, "Video-steganography: how to secretly embed a signature in a picture," *Proceedings of the IMA Intellectual Property Project,* Vol.1, No. 1, 1994.

[6] H. K. Pan, Y. Y. Chen, and Y. C. Tseng, "A secure data hiding scheme for two-color images," *Proceedings of the IEEE Fifth Symposium on Computers and Communications(ISCC2000)*, pp. 750-755, Antibes, France, July 2000.

[7] M. Swanson, M. Kobayashi, and A. Tewfik, "Multimedia data-embedding and watermarking technologies," *Proceedings of the IEEE*, Vol. 86, pp. 1064-1088, 1998.

[8] Y. C. Tseng and H. K. Pan, "Secure and invisible data hiding in 2-color images," *Proceedings of the IEEE INFOCOM 2001 The Conference on Computer Communications*, No. 1, pp. 887-896, Anchorage, Alaska, U. S. A. 2001.

[9] C. H. Tzeng and W. H. Tsai, “A new technique for authentication of image/video for multimedia applications,” *Proceedings of the ACM Multimedia 2001 Workshops --- Multimedia and Security*: *New Challenges*, pp. 23-26, Ottawa, Ontario, Canada, Oct. 2001.

[10] C. H. Tzeng and W. H. Tsai, “A new approach to authentication of binary images for multimedia communication with distortion reduction and security enhancement,” *IEEE Commun. Lett.*, Vol. 7, No. 9, pp. 443–445, Sep. 2003.

[11] H. C. Wang, "Data hiding techniques for printed binary images," *Proceedings of the IEEE International Conference on Information Technology: Coding and Computing*, pp. 55-59, Las Vegas, NV, U. S. A., April 2001.

[12] M. Wu, E. Tang, and B. Liu, “Data hiding in digital binary image,” *Proceedings of the IEEE International Conference on Multimedia & Exposition 2000(ICME’00)*, Vol. 1, pp. 393-396, New York, New York, 2000.

[13] D. C. Wu and W. H. Tsai, “Spatial-domain image hiding using an image differencing,” *Processings of the IEE Proceedings-Vision, Image, and Signal*, Vol. 147, No. 1, pp. 29-37, Feb. 2000.

[14] D. C. Wu, M. K. Hsu, and J. H. Jheng, “Data hiding and authentication techniques for 2-color digital documents based on adjusting lengths of runs,” *Proceedings of the 16th IPPR Conference on Computer Vision, Graphics and Image Processing (CVGIP 2003)*, pp. 818-822, Kinmen, Taiwan, R. O. C., Aug. 2003.

[15] R. Z. Wang, C. F. Lin, and J. C. Lin, “Hiding data in Images by optimal moderately-significant-bit replacement,” *IEEE Electronics Letters*, Vol. 36, No. 25, pp. 2069-2070, December 2000.

[16] C. C. Chang, J. Y. Hsiao, and C. S. Chan, “Finding optimal least-significant-bit substitution in image hiding by dynamic programming strategy,” *Pattern*

*Recognition*, Vol. 36, pp. 1583-1595, 2003.

[17] C. K. Chan, and L. M. Cheng, “Improved hiding data in images by optimal moderately-significant-bit replacement,” *IEEE Electronics Letters*, Vol. 37, No. 16, pp. 1017-1018, August 2001.

[18] C. K. Chan, and L. M. Cheng, “Hiding data in images by simple LSB substitution,” *Pattern Recognition*, Vol. 37, pp. 469-474, 2004.

[19] C. C. Thien and J. C. Lin, “A simple and high-hiding capacity method for hiding digit-by-digit data in images based on modulus function,” *Pattern Recognition*, Vol. 36, pp. 2875-2881, 2003.

[20] Y. K. Lee and L. H. Chen, “High capacity image steganographic model,” *IEE Proceedings on Vision, Image Signal Process*, Vol. 147 No. 3, June 2000.

[21] S. H. Liu, T. H. Chen, H. X. Yao, and W. Gao, “A variable depth LSB data hiding technique in images,” *Proceedings of the 3rd International Conference on Machine Learning and Cybernetics*, pp. 3990-3994, Shanghai, P. R. China, August 2004.

[22] W. N. Lie and L. C. Chang, “Data hiding in images with adaptive numbers of least significant bits based on the human visual system,” *Proceedings of the IEEE International Conference on Image Processing*, Vol. 1, pp. 286-290, Taipei, Taiwan, R. O. C., October 1999.

[23] S. Lyu and H. Farid, “ Detecting hidden messages using higher-order statistics and support vector machines,” *Lecture Notes in Computer Science*, Vol. 2578, pp. 340-354, 2003.

[24] J. Fridrich, M. Goljan, and R. Du, “Detecting LSB steganography in color and gray-scale images,” *IEEE Multimedia*, Vol. 8, No. 4, pp. 22-28, 2001.

[25] I. S. Lee and W. H. Tsai, “Data hiding in binary images with distortion-minimizing capabilities by optimal block pattern coding and dynamic

programming techniques," *IEICE Transactions on Information and Systems,* Vol. E90-D, No. 8, pp. 1142-1150, 2007.

[26] A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, Singapore, 1989.

[27] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding," *IBM System Journal*, Vol. 35, No. 3 & 4, Feb. 1996.

[28] Y. Y. Tsai and C. M. Wang, "A novel data hiding scheme for color images using a BSP tree," *Journal of Systems and Software*, Vol. 80, pp. 429–437, 2007.

[29] D. C. Lou and J. L. Liu, "Steganographic Method for Secure Communications," *Computers and Security*, Vol. 21, No. 5, pp. 449-460, Oct. 2002.

[30] Low, S. H., N. F. Maxemchuk, and A. M. Lapone, "Document Identification for Copyright Protection Using Centroid Detection," *IEEE Transactions on Communication*, Vol. 46, No. 3, pp. 372-383, 1998.

[31] J., Brassil, N. F. Maxemchuk, and L. O'Gorman, "Electronic Marking and Identification Techniques to Discourage Document Copying," *Proceedings of the 13th Annual IEEE Conference on Computer Communications(INFOCOM'94)*, pp.1278-1287, Toronto, Ontario, Canada,1994.

[32] P. Wayner, "Strong Theoretical Steganography," *Cryptologia*, Vol. XIX/3, pp. 285-299, 1995.

[33] G. Cantrell and D. D. Dampier, "Experiments in Hiding Data Inside the File Structure of Common Office Documents: A Stegonography Application," *Proceedings of the 2004 International Symposium on Information and Communication Technologies*, pp. 146-151, Las Vegas, Nevada, U. S. A., 2004.

[34] N. F. Johnson, Z. Duric, and S. Jajodia, *Information Hiding: Steganography and Watermarking-Attacks and Countermeasures*, Kluwer Academic Publishers, Boston, MA, U. S. A., 2001.

[35] R., Anderson, R. Needham, and A. Shamir, “The Steganographic file System,” *Proceedings of the Second International Workshop on Information Hiding, Lecture Notes in Computer Science*, Vol. 1525, pp. 73-82, Springer, Berlin, Germany, 1998.

[36] T. G., Handel, and M. T. Stanford, “Hiding Data in the OSI Network Model,” *Proceedings of the First International Workshop on Information Hiding*, pp. 23-38, Cambridge, UK, 1996.

[37] Y. H. Chang and W. H. Tsai, “A steganographic method for copyright protection of HTML documents,” *Proceedings of the 2003 National Computer Symposium,* Taichung, Taiwan, R. O. C., Dec. 2003.

[38] J. B. Postel, "Simple Mail Transfer Protocol," *STD 10, RFC 821, IETF*, August 1982.

[39] D. Crocker, "Standard for the Format of ARPA Internet Text Messages," *STD 11, RFC 822, IETF*, August 1982.

[40] P. Resnick, "Internet Message Format," *RFC 2822, IETF*, April 2001.

[41] J. G. Myers and M. T. Rose, “Standard Post Office Protocol - Version 3,” *STD 53, RFC 1939, IETF*, May 1996.

[42] N. Freed and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," *RFC 2045, IETF*, Nov. 1996.

[43] N. Freed and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types," *RFC 2046, IETF*, Nov. 1996.

[44] N, Freed. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples," *RFC 2049, IETF*, Nov. 1996.

[45] M. Crispin, " Internet Message Access Protocol - Version 4rev1," *RFC 3501, IETF*, March 2003

[46] A. Shamir, “How to share a secret,” *Communications of the Association for*

*Computing Machinery*, Vol. 22, No. 11, pp. 612-613, 1979.

[47] M. Naor and A. Shamir, "Visual cryptography," *Advances in Cryptology --- EUROCRYPT'94*, *Lecture Notes in Computer Science*, Vol. 950, pp. 1-12, 1995.

[48] G. Ateniese, C. Blundo, A. De Santis, and D. R. Stinson, "Visual cryptography for general access structures," *Information and Computation*, Vol. 129, pp. 86-106, 1996.

[49] M. Naor and B. Pinkas, "Visual authentication and identification," *Advances in Cryptology --- CRYPTO'97*, *Lecture Notes in Computer Science*, Vol. 1294, pp. 322-336, 1997.

[50] C. Blundo and A. De Santis, "Visual cryptography schemes with perfect reconstruction of black pixels," *Computers & Graphics*, Vol. 22, No. 4, pp. 449-455, 1998.

[51] E. R. Verheul and H. C. A. van Tilborg, "Construction and properties of k out of n visual secret sharing schemes," *Designs, Codes, and Cryptography*, Vol. 11, pp. 179-196, 1997.

[52] C. C. Lin and W. H. Tsai, "Secret image sharing with steganography and authentication," *Journal of Systems & Software*, Vol. 73, No. 3, pp. 405-414, 2004.

[53] I. S. Lee and W. H. Tsai "Data hiding in emails and applications by unused ASCII control codes," *Proceedings of the 2007 National Computer Symposium*, Vol. 4, pp. 414-422, Taichung, Taiwan, R. O. C., Dec. 2007.

[54] W. Zhu, C. Thomborson, and F. Y. Wang, "A survey of software watermarking," *Proceedings of the IEEE International Conference on Intelligence and Security Informatics*, *Lecture Notes in Computer Science*, Vol. 3495, pp. 454-458, May 2005.

[55] R. Venkatesan, V. Vazirani, and S. Sinha. "A graph theoretic approach to

software watermarking," *Proceedings of the 4th International Information Hiding Workshop*, *Lecture Notes in Computer Science*, Vol. 2137, pp. 157-168, Apr. 2001.

[56] C. Collberg and C. Thomborson. "Watermarking, tamper-proofing, and obfuscation — tools for software protection," *IEEE Transactions on Software Engineering*, Vol. 28, pp. 735-746, 2002.

[57] H. M. Meral, E. Sevinc, E. Unkar, B. Sankur, A. S. Ozsoy, and T. Gungor, "Syntactic tools for text watermarking," *Proceedings of the SPIE International Conference on Security, Steganography, and Watermarking of Multimedia Contents*, San Jose, CA, Jan.-Feb. 2007.

[58] M. Topkara, U. Topkara, and M. J. Atallah, "Information hiding through errors: a confusing approach," *Proceedings of the SPIE International Conference on Security, Steganography, and Watermarking of Multimedia Contents*, San Jose, CA, Jan.-Feb. 2007.

[59] F. A. P. Petitcolas, R. J. Anderson and M. G. Kuhn, "Information hiding - A survey," *Proceedings of the IEEE*, Special Issue on Protection of Multimedia Content, Vol. 87, No. 7, pp. 1062-1078, July 1999.

[60] D. C. Wu and P. H. Lai, "Novel Techniques of Data Hiding in HTML Documents," *Proceedings of the 2005 Conference on Digital Contents Managements & Applications*, pp. 21-30, Kaohsiung, Taiwan, R. O. C., June, 2005.

[61] T. Y. Liu and W. H. 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, No. 1, pp. 24-30, March 2007.

[62] T. Y. Liu and W. H. Tsai, "Robust Watermarking in Slides of Presentations by

Blank Space Coloring: A New Approach," accepted and to appear in *Transactions on Data Hiding and Multimedia Security, Lecture Notes in Computer Science*.

[63] Adobe Systems Incorporated, Portable document format reference manual, Version 1.7, http://www.adobe.com, November, 2006.

[64] S. Zhong, X. Cheng and T. Chen, "Data hiding in a kind of PDF texts for secret communication," *Int'l Journal of Network Security*, Vol.4, No.1, pp.17–26, Jan. 2007.

[65] ASCII Code - The extended ASCII table. Retrieved April 30, 2008, from http://www.ascii-code.com/

[66] M. U. Celik, G. Sharma, E. Saber, and A. M. Tekal, "Hierarchical watermarking for secure image authentication with localization," *IEEE Trans. Image Processing*, Vol. 11, pp. 585–505, June 2002.

[67] D. Coppersmith, F. Mintzer, C. Tresser, C. W. Wu, and M. M. Yeung, "Fragile imperceptible digital watermark with privacy control," *Proceedings of the SPIE, Security and Watermarking of Multimedia Contents*, pp. 79–84, San Jose, CA, Jan. 2000.

[68] E. Izquierdo and V. Guerra, "An Ill-Posed Operator for Secure Image Authentication," *IEEE Trans. Circuits & Systems for Video Technology*, Vol. 13, No. 8, pp. 842-852, August 2003.

[69] S. Walton, "Information authentication for a slippery new age," *Dr. Dobbs Journal*, Vol. 20, No. 4, pp. 18–26, Apr. 1995.

[70] P. W. Wong, "A public key watermark for image verification and authentication," *Proceedings of the International Conference on Image Processing (ICIP),* Vol. 1, pp. 425-429, Chicago, Illinois, October. 1998.

[71] M. M. Yeung and F. Mintzer, "An invisible watermarking technique for image

verification," *Proceedings of the International Conference on Image Processing (ICIP)*, Vol. 2, pp.680-683, Santa Barbara, CA., 1997.

[72] M. Shirali-Shahreza, "Java applets copy protection by steganography," *Proceedings of the 2006 Int'l Conference on Intelligent Information Hiding & Multimedia Signal Processing*, pp. 388-391, Pasadena, California, U. S. A., Dec. 2006.

[73] C. C. Wu, C. C. Chang and S. R. Yang, "An efficient fragile watermarking for web pages tamper-proof," *Advances in Web and Network Technologies, and Information Management, Lecture Notes in Computer Science*, Vol. 4537, pp. 654-663, Springer, Berlin, Germany, 2007.

[74] Invisible Secrets. Retrieved May 15, 2008, from http://www.invisiblesecrets.com.

[75] Q. Zhao and H. Lu, "PCA-based web page watermarking," *Pattern Recognition*, Vol. 40, pp. 1334 – 1341, 2007.

[76] S. Voloshynovskiy, T. Pun, J. Fridrich, F. Pérez-González, and N. Memon (Guest Editors), Special Issue on Security Of Data Hiding Technologies, *Signal Processing*, Vol. 82, Iss. 10, pp. 1511-1512, Oct. 2002.

# Publication List

**Journal Papers**:

(1) I. S. Lee and W. H. Tsai, "Data Hiding in Binary Images with Distortion-Minimizing Capabilities by Optimal Block Pattern Coding and Dynamic Programming Techniques," *IEICE Transactions on Information and Systems*, Vol. E90-D, No. 8, pp. 1142-1150, August 2007 (SCI) .

(2) I. S. Lee and W. H. Tsai, "Data Hiding in Grayscale Images by Dynamic Programming Based on A Human Visual Model," *Pattern Recognition*. (to be published after minor revision) (SCI)

(3) I. S. Lee and W. H. Tsai, "Security Protection of Software Programs by Information Sharing and Authentication Techniques Using Invisible ASCII Control Codes," accepted for *International Journal of Network Security*.

(4) I. S. Lee and W. H. Tsai, "Data Hiding in Emails and Applications by Unused ASCII Control Codes," accepted for *Journal of Information Technology and Applications*.

(5) I. S. Lee and W. H. Tsai, "Data Hiding in Color Images by Color Replacements with Reduction of Image Distortion and Change Noticeability," submitted to *IET Image Processing*.

(6) I. S. Lee and W. H. Tsai, "Covert Communication with Authentication via Software Programs Using Invisible ASCII Codes --- A New Approach," submitted to *IEEE Signal Processing Letters*.

(7) I. S. Lee and W. H. Tsai, "A New Method for Covert Communication via PDF Files by A Data Hiding Technique," submitted to *Signal Processing*.

(8) I. S. Lee and W. H. Tsai, "PDF File Authentication by Invisible Codes," submitted to *Tamkang Journal of Science and Engineering*(EI).

(9) I. S. Lee and W. H. Tsai, "Secret Communication via Web Pages Using Special Space Codes in HTML Files," submitted to *International Journal of Applied Science and Engineering*.

(10) I. S. Lee and W. H. Tsai, "Automatic Authentication of Web Pages by Data Hiding Using Multiple Space Codes in HTML Files," submitted to *Signal Processing*.

## Conference Papers

[1] I. S. Lee and W. H. Tsai, "A Dynamic-Programming Approach to Data Hiding in Binary Image Using Block Pattern Coding with Distortion Minimization," *Proceedings of 2003 National Computer Symposium (NCS 2003)*, Taichung,, Taiwan, pp. 1406-1413, December 18-19, 2003.

[2] I. S. Lee and W. H. Tsai, "Data Hiding in Grayscale Images by Dynamic Programming Based on A Human Visual," *20th Compuer Vision, Graphics, and Image Processing (CVGIP) 2007 conference proceeding*, Miaoli, Taiwan, pp. 204-209, August 19-21, 2007. (Excellent paper award of CVGIP 2007)

[3] I. S. Lee and W. H. Tsai, "Data Hiding in Emails and Applications by Unused ASCII Control Codes," in *Proceedings of 2007 National Computer Symposium,* Taichung, Taiwan, Taichung, R. O. C , pp. 414-422, December 20-21, 2007. (Best paper award of Information Security and Networks Workshop of NCS 2007)

[4] I. S. Lee and W. H. Tsai, " Security Protection of Software Programs by Information Sharing and Authentication Techniques Using Invisible ASCII Control Codes," accepted for *21th Compuer Vision, Graphics, and Image Processing (CVGIP) 2008 conference proceeding*, Ilan, Taiwan, August 24-26, 2008.

# Vita

**I-Shi Lee** was born in Taipei, Taiwan, R.O.C., in 1961. He received the B. S. degree in electronic engineering from National Taiwan University of Science and Technology, Taipei, Taiwan, Republic of China in 1987, the M. S. degree in the Department of Computer Science and Information Science at National Chiao Tung University in 1989, and the Ph.D. degree in the Institute of Computer Science and Engineering, College of Computer Science from National Chiao Tung University in 2008.

In 1992, he joined the Department of Management Information at Northern Taiwan Institute of Science and Technology and acted as a lecturer from 1992 to now. His recent research interests include pattern recognition, watermarking, and image hiding.