

國立交通大學

多媒體工程研究所

碩士論文

利用特殊字元編碼的新資訊隱藏技術與其於網際網路
上之應用

A Study on New Data Hiding Techniques Using Special
Character Codes and Their Applications on the Internet

研究生：王以安

指導教授：蔡文祥 教授

中華民國一百年六月

利用特殊字元編碼的新資訊隱藏技術
與其於網際網路上之應用
A Study on New Data Hiding Techniques Using Special Character Codes
and Their Applications on the Internet

研究生：王以安

Student: Yi-An Wang

指導教授：蔡文祥

Advisor: Prof. Wen-Hsiang Tsai

國立交通大學
多媒體工程研究所
碩士論文

A Thesis
Submitted to Institute of Multimedia Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

June 2011

Hsinchu, Taiwan, Republic of China

中華民國一百年六月

利用特殊字元編碼的新資訊隱藏技術與其於網際網路上之應用

研究生：王以安

指導教授：蔡文祥 博士

國立交通大學多媒體工程研究所

摘要

隨著電腦和網路科技的發展，越來越多人透過網路來進行通訊，所以有必要保護網際網路上通訊訊息的安全性。以此，本論文提出了四種資訊隱藏的方法，用來對部落格、電子佈告欄系統(BBS)及電子郵件三種受歡迎的網路應用進行秘密通訊或文章驗證。

對於部落格，本研究利用不可視的特殊美國標準資訊交換碼(ASCII)控制碼組成的驗證訊號來識別部落格文章是否有被更改，達到文章驗證的功能。在電子公佈欄方面，本研究提出了兩種資訊隱藏的方法，分別是利用不可視的大五碼(Big-5)字元碼以及特殊的大五碼的空白碼來組成秘密資訊，此兩種方法透過電子公佈欄做媒介，都可用以進行秘密通訊以及文章驗證。最後，對於電子郵件，本研究提出了一個利用特殊八位元萬國碼(UTF-8)的空白碼的資訊隱藏技術，可用來驗證電子郵件，以偵測任何對受保護之郵件的惡意竄改。

上述所提應用都是利用這些特殊字元碼在各個應用平台上的不可視性，來隱藏秘密資訊而不被察覺，並將特殊字元編碼所構成的秘密訊息藏入原始文章當中，達到資訊隱藏的目的，而且所隱藏的秘密資訊皆可在之後正確地還原回來。

最後本論文也提供了相關的實驗結果，來證明所提方法的可行性。

A Study on New Data Hiding Techniques Using Special Character Codes and Their Applications on the Internet

Student: Yi-An Wang

Advisor: Wen-Hsiang Tsai

Institute of Multimedia Engineering, College of Computer Science
National Chiao Tung University

ABSTRACT

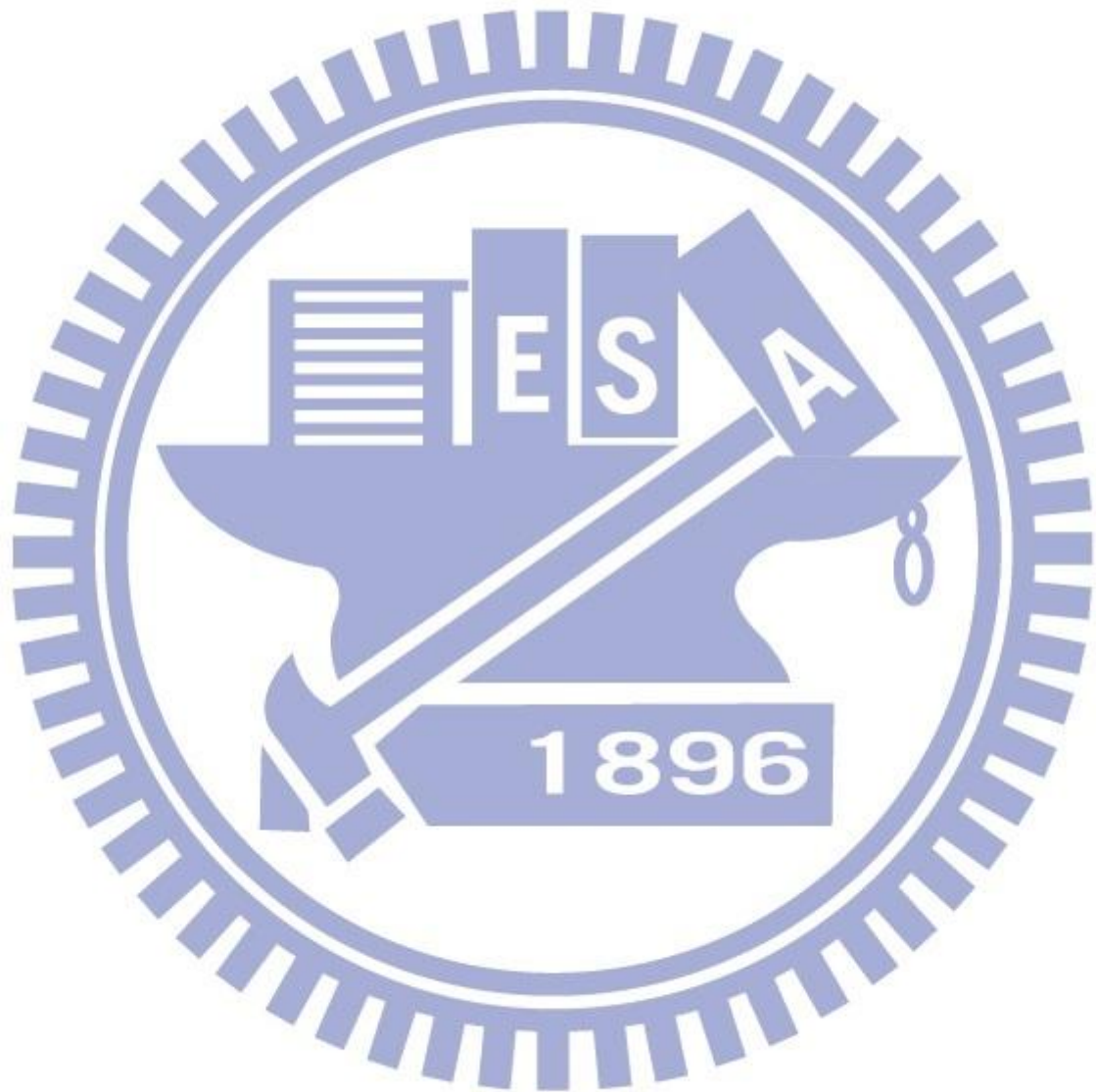
With the progress of computer and networking technologies, communication via the Internet has become more and more popular nowadays, and so security protection of communication messages on the Internet has become a necessity. In this study, four data hiding methods for covert communication or article authentication are proposed for use on three Internet applications, namely, the blog, BBS, and email.

For the blog, a new article authentication method based on one of the proposed data hiding techniques is proposed, which uses invisible ASCII control codes to construct authentication signals to verify whether a blog article is tampered with or not. For the BBS, two data hiding techniques are proposed. One uses invisible Big-5 codes, and the other uses special Big-5 space codes, for embedding message data imperceptibly. Each of the two techniques may be used to accomplish covert communication as well as BBS article authentication. Finally, for email, a data hiding technique via the use of special UTF-8 codes is proposed for webmail authentication, so that malicious tampering with a protected email may be found out.

In all the applications mentioned above, the invisibility of the proposed special codes on appropriate platforms of Internet applications is utilized to achieve the aim

of hiding data imperceptibly. A stego-article is generated by embedding the secret message information composed of the adopted special codes into the ends of the text lines of a cover article. The hidden secret information can be recovered correctly later.

Experimental results showing the feasibility of the proposed methods are also included.

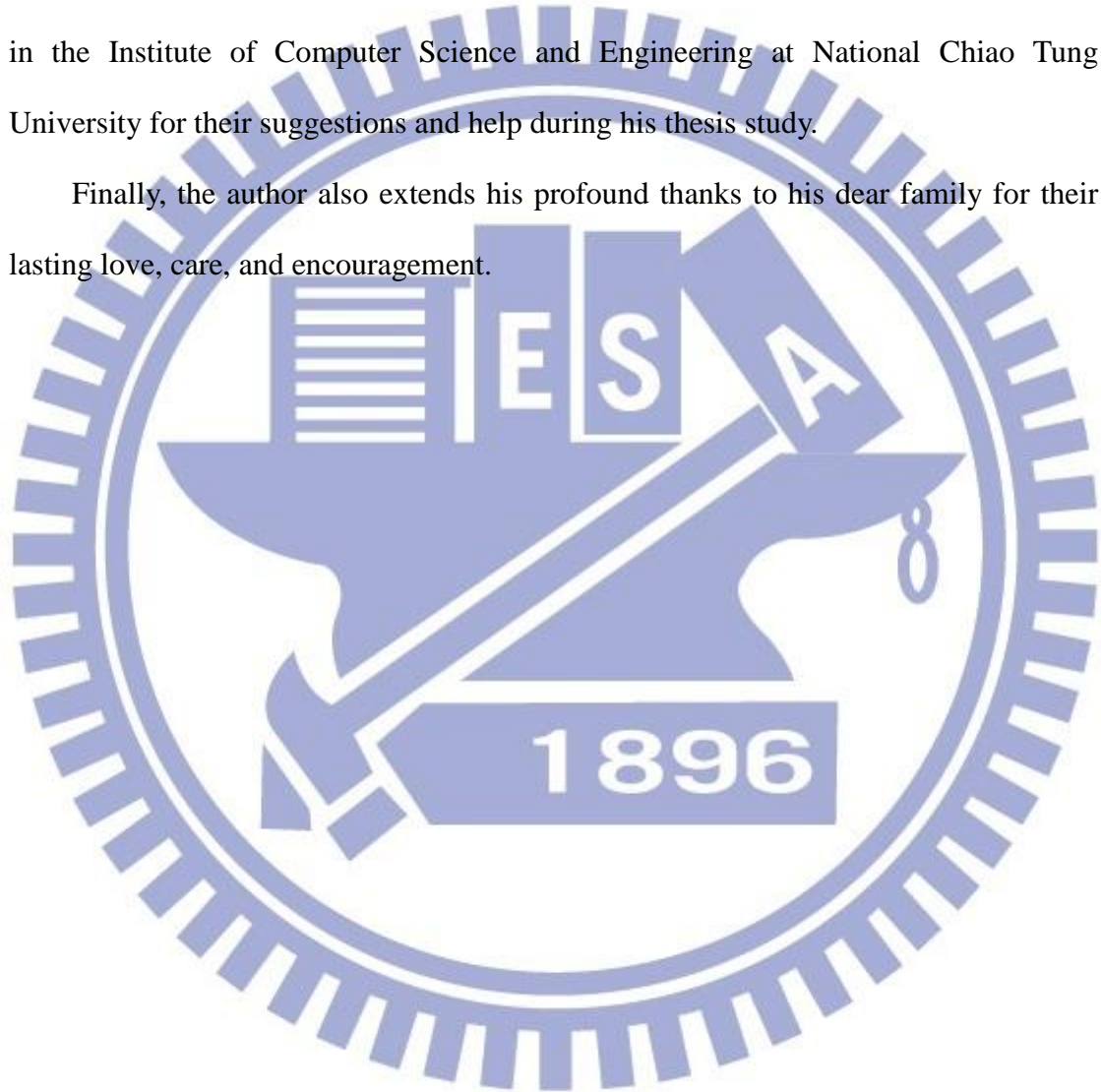


ACKNOWLEDGEMENTS

The author is in hearty appreciation of the continuous guidance, discussions, and support from his advisor, Dr. Wen-Hsiang Tsai, not only in the development of this thesis, but also in every aspect of his personal growth.

Appreciation is also given to the colleagues of the Computer Vision Laboratory in the Institute of Computer Science and Engineering at National Chiao Tung University for their suggestions and help during his thesis study.

Finally, the author also extends his profound thanks to his dear family for their lasting love, care, and encouragement.



CONTENTS

| | |
|----------------------------|-----|
| ABSTRACT (IN CHINESE)..... | i |
| ABSTRACT (in English)..... | ii |
| ACKNOWLEDGEMENTS | iv |
| CONTENTS | v |
| LIST OF FIGURES | vii |
| LIST OF TABLES..... | x |

Chapter 1 Introduction 1

| | |
|--|---|
| 1.1 Motivation and Background..... | 1 |
| 1.1.1 Motivation of Study | 1 |
| 1.1.2 Introduction to Used Media | 2 |
| 1.2 Overview of Related Works | 4 |
| 1.3 Overview of Proposed Methods..... | 5 |
| 1.3.1 Definitions of Terms..... | 5 |
| 1.3.2 Brief Description of Proposed Methods..... | 6 |
| 1.4 Contributions..... | 9 |
| 1.5 Thesis Organization..... | 9 |

Chapter 2 Review of Related Works and Character Coding Formats 10

| | |
|---|----|
| 2.1 Previous Studies on Data Hiding Techniques Using Special Character Codes..... | 10 |
| 2.1.1 Review of Data Hiding Techniques via Text Documents | 11 |
| 2.1.2 Review of Data Hiding Techniques for Internet Applications | 12 |
| 2.1.3 Review of Other Techniques and Summary..... | 14 |
| 2.2 Review of Related Character Coding Formats..... | 14 |
| 2.2.1 Review of ASCII Format | 14 |
| 2.2.2 Review of Big-5 Format..... | 15 |
| 2.2.3 Review of UTF-8 Format..... | 16 |

Chapter 3 Authentication of Blog Articles by Invisible ASCII Control Codes. 19

| | |
|---|----|
| 3.1 Introduction and Problem Definition | 19 |
| 3.2 Major Idea of Proposed Method by Use of Invisible ASCII Control Codes | 20 |
| 3.2.1 Use of Special Character Codes | 20 |
| 3.2.2 Necessity of Distributing Embedded Codes Evenly at Line Ends to | |

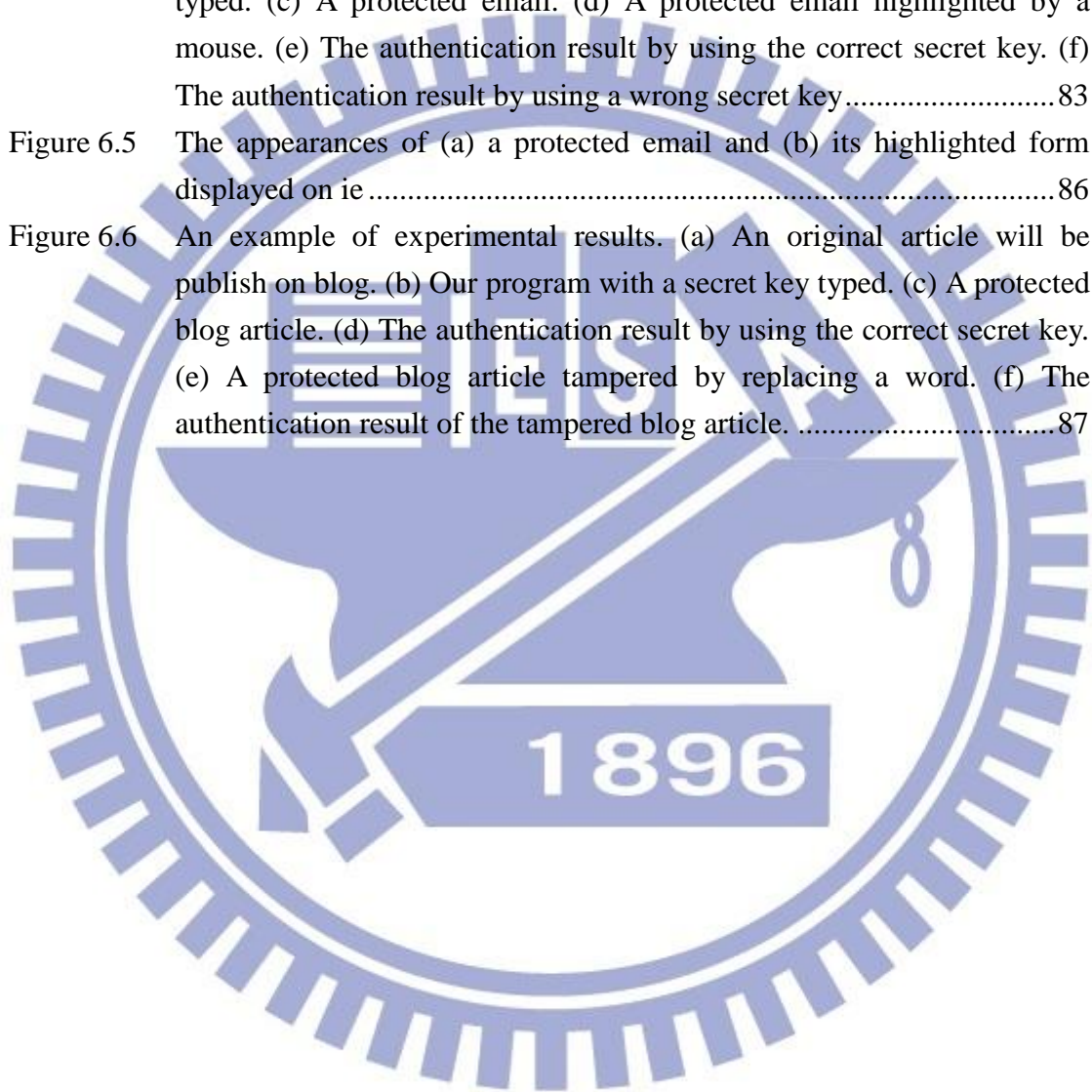
| | |
|---|-----------|
| Reduce Suspicion | 23 |
| 3.2.3 Construction of End Signals..... | 25 |
| 3.3 Authentication Signal Generation and Embedding Process..... | 25 |
| 3.4 Authentication Signal Extraction and Blog Verification Process..... | 28 |
| 3.5 Experimental Results..... | 30 |
| 3.6 Summary | 31 |
| Chapter 4 Covert Communication via the BBS Using Special BIG-5 Codes.... | 37 |
| 4.1 Introduction and Problem Definition | 37 |
| 4.2 Major Ideas of Proposed Methods by Use of Special Big-5 Codes | 38 |
| 4.2.1 Data Hiding by Invisible Big-5 Codes | 38 |
| 4.2.2 Data Hiding by Special Big-5 Space Codes..... | 42 |
| 4.3 Proposed Algorithm for Data Embedding..... | 46 |
| 4.4 Proposed Algorithm for Data Extraction..... | 49 |
| 4.5 Experimental Results..... | 51 |
| 4.6 Summary | 52 |
| Chapter 5 BBS Article Authentication by Special BIG-5 Codes | 58 |
| 5.1 Introduction and Problem Definition | 58 |
| 5.2 Major Idea of Proposed Method by Use of Special Big-5 Codes | 59 |
| 5.3 Authentication Signal Generation and Embedding Process..... | 59 |
| 5.4 Authentication Signal Extraction and Verification Process | 61 |
| 5.5 Experimental Results..... | 63 |
| 5.6 Summary | 64 |
| Chapter 6 Email Authentication by Special UTF-8 Space Codes | 71 |
| 6.1 Introduction and Problem Definition | 71 |
| 6.2 Major Idea of Proposed Method by Use of Special UTF-8 Codes | 72 |
| 6.3 Authentication Signal Generation and Embedding Process..... | 76 |
| 6.4 Authentication Signal Extraction and Verification Process | 79 |
| 6.5 Experimental Results..... | 81 |
| 6.6 Adaptability of Proposed Method for Authentication of Blog Articles ... | 81 |
| 6.7 Summary | 82 |
| Chapter 7 Conclusions and Suggestions for Future Works | 90 |
| 7.1 Conclusions | 90 |
| 7.2 Suggestions for Future Works | 91 |
| References | 93 |

LIST OF FIGURES

| | | |
|------------|---|----|
| Figure 1.1 | The login screen and a normal article on a bbs. | 3 |
| Figure 1.2 | An instance of blogs. | 4 |
| Figure 1.3 | Two popular email systems. (a) G-mail. (b) Hotmail. | 4 |
| Figure 2.1 | Example of data hidden using white space [3]. (a) Normal text. (b) White space encoded text. | 11 |
| Figure 2.2 | An experimental result found in lee and tsai [10]. (a) Cover file seen in adobe reader 8.1.2 window. (b) Stego-file seen in adobe reader 8.1.2 window with message “this is a covert communication method” embedded. | 12 |
| Figure 2.3 | An experimental result found in lee and tsai [12]. (a) Cover text seen in the window of the ie. (b) Stego-text (with message about “cartesian coordinates” embedded) seen in the window of the ie. | 13 |
| Figure 2.4 | Big-5 coding format. | 16 |
| Figure 2.5 | An example of utf-8 coding [17]. | 17 |
| Figure 3.1 | Some examples of highlighted blog articles with secret codes embedded in them on (a) mozilla firefox and (b) google chrome. | 23 |
| Figure 3.2 | Some examples of highlighted blog articles with secret codes embedded in them in ie. (a) The stego-article with secret codes embedded in order. (b) The stego-article with secret codes embedded evenly using the proposed method. | 25 |
| Figure 3.3 | Flowchart of proposed authentication signal generation and embedding process. | 28 |
| Figure 3.4 | Flowchart of the proposed authentication signal extraction and blog article verification process. | 29 |
| Figure 3.5 | An example of experimental results. (a) An original blog article. (b) A user interface used to generate authentication signal and protected blog article. (c) The protected blog article with an authentication signal embedded. (d) The authentication report with the message “authentication is successful.” (e) A protected blog article with a tempered word. (f) The verification result of the tempered blog article. | 33 |
| Figure 3.6 | Another example of experimental results. (a) An original blog article. (b) The user interface with a secret key “nctu”. (c) The protected blog article with an authentication signal embedded. (d) The authentication result by using the correct secret key. (e) The authentication result by using a wrong secret key. | 35 |

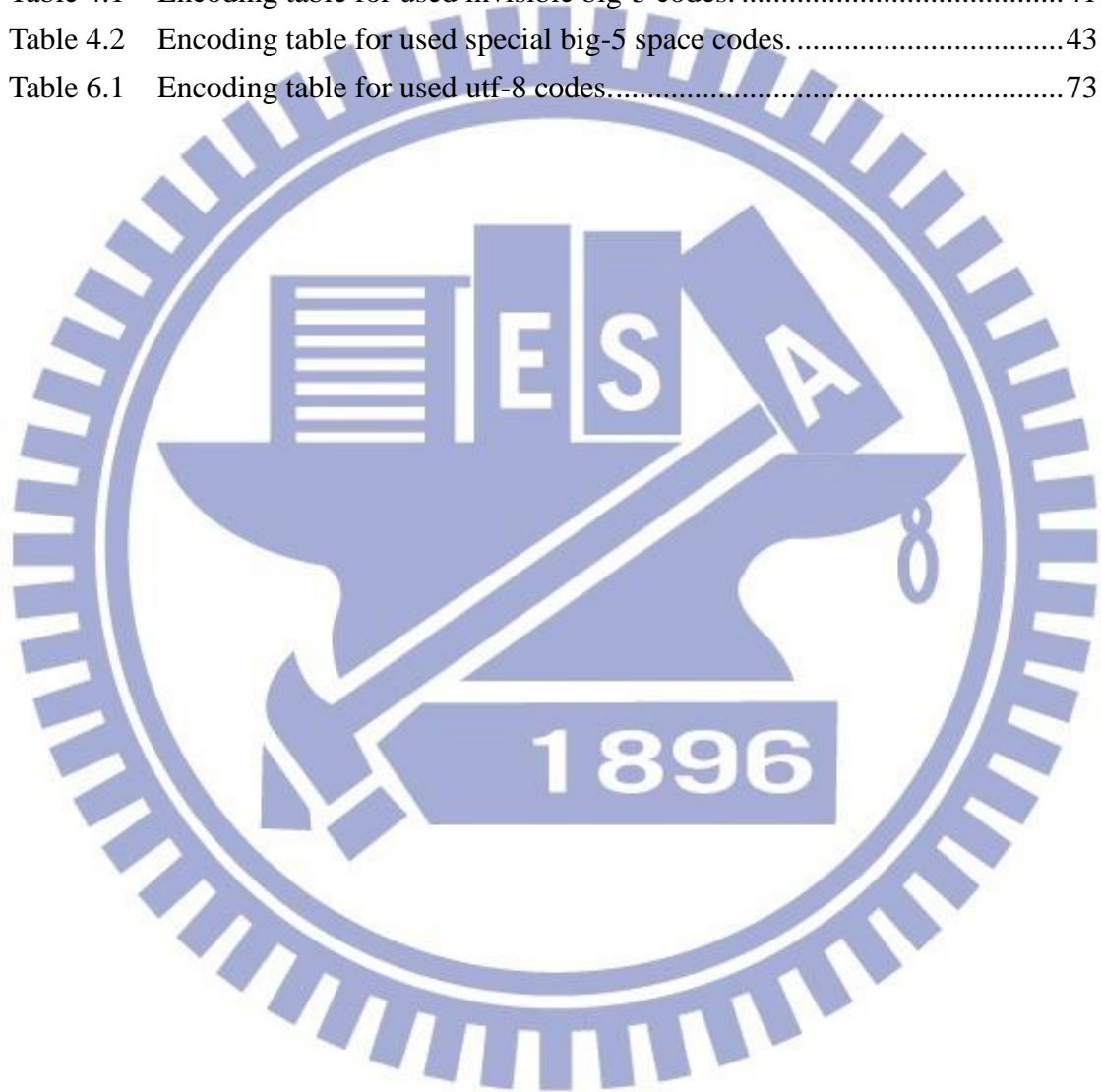
| | | |
|------------|--|----|
| Figure 4.1 | Stego-articles with some embedded invisible big-5 symbols displayed on (a) pman, (b) kkman, (c) piety, and (d) the telnet connection program, respectively. | 40 |
| Figure 4.2 | Stego-articles with secret messages embedded displayed on some well-known bbs's. (a) on pman. (b) on kkman. (c) on piety. (d) on the telnet connection program. | 45 |
| Figure 4.3 | Flow chart of proposed process of embedding secret messages. | 47 |
| Figure 4.4 | Flow chart of proposed data extraction process | 50 |
| Figure 4.5 | An example of experimental results. (a) A normal article displayed on the pman with our program in the upper right. (b) Data embedding process: type a secret key and a secret message, select a hiding method, highlight a cover article, and press the <i>hiding-button</i> to generate a stego article with the secret message embedded. (c) The displayed stego-article with the secret message embedded on the pman. (d) Data extraction process: extract the secret message by the use of using the correct secret key, select the same method, and press the <i>extraction-button</i> . (e) Result of using a wrong key to extract the secret message. (f) An extracted wrong message. | 53 |
| Figure 4.6 | Another example of experimental results. (a) Another normal article. (b) Embedding a secret message by method 2, using special big-5 space codes. (c) Stego-article with the secret message embedded. (d) Extracted correct secret message. | 56 |
| Figure 5.1 | Flowchart of proposed authentication signal generation process. | 60 |
| Figure 5.2 | Flow chart of proposed bbs verification process. | 62 |
| Figure 5.3 | An example of experimental results. (a) A generation and sending process of a protected bbs mail. (b) A protected bbs mail displayed on the pman. (c) A protected bbs article authenticated with a correct secret key. (d) A protected bbs article authenticated with a wrong key. | 65 |
| Figure 5.4 | Another example of experimental results. (a) A generation and sending process of a protected bbs article. (b) A protected bbs article displayed on the pman. (c) An authentication result of a protected bbs article with a correct secret key. (d) An authentication result of a protected bbs article tampered by replacing a word. | 67 |
| Figure 5.5 | An experimental result displayed on the kkman. (a) A normal bbs mail. (b) A protected bbs mail. | 69 |
| Figure 5.6 | An experimental result displayed on the telnet connection program. (a) A normal bbs mail. (b) A protected bbs mail. | 70 |
| Figure 6.1 | A highlighted stego-email with secret symbols embedded (a) just in | |

| | | |
|------------|---|----|
| | order or (b) evenly using the proposed method. | 75 |
| Figure 6.2 | Flowchart of proposed authentication signal generation and embedding process. | 77 |
| Figure 6.3 | Flowchart of the proposed authentication signal extraction and email verification process. | 80 |
| Figure 6.4 | An example of experimental results. (a) An original email will be send through the g-mail webmail platform. (b) Our program with a secret key typed. (c) A protected email. (d) A protected email highlighted by a mouse. (e) The authentication result by using the correct secret key. (f) The authentication result by using a wrong secret key..... | 83 |
| Figure 6.5 | The appearances of (a) a protected email and (b) its highlighted form displayed on ie | 86 |
| Figure 6.6 | An example of experimental results. (a) An original article will be publish on blog. (b) Our program with a secret key typed. (c) A protected blog article. (d) The authentication result by using the correct secret key. (e) A protected blog article tampered by replacing a word. (f) The authentication result of the tampered blog article. | 87 |



LIST OF TABLES

| | | |
|-----------|---|----|
| Table 2.1 | ASCII code chart..... | 15 |
| Table 2.2 | UTF-8 encoding format. | 17 |
| Table 3.1 | ASCII control codes and description [1]..... | 21 |
| Table 3.2 | Encoding table for used invisible ascii control codes. | 23 |
| Table 4.1 | Encoding table for used invisible big-5 codes. | 41 |
| Table 4.2 | Encoding table for used special big-5 space codes. | 43 |
| Table 6.1 | Encoding table for used utf-8 codes..... | 73 |



Chapter 1

Introduction

1.1 Motivation and Background

1.1.1 Motivation of Study

Good media for information communication to promote social developments are indispensable nowadays. Ever since paper was invented, knowledge dissemination and science advancement have progressed every day. Today, at the time of breakthroughs in information communication, the Internet has become an extremely important medium.

However, the Internet is just like a double-edged sword. When we communicate information on the Internet, we may be pleased to see the surprising propagation velocity and the great capability of the network. However, we are also putting the information in danger in the meantime. To exchange information on the Internet *safely*, the use of the data hiding technique is a solution, which was intensively studied in the past decade.

One of the applications of data hiding is *steganography*, which is one form of *covert communication*. Unlike cryptography, the imperceptibility of steganography conceals the behavior of secret transmission so that the risk for the secret to be detected by malicious users decreases. Besides, authentication is also an application of data hiding. It aims to verify the integrity and fidelity of data. As it is hard to prevent malicious users from intercepting and tampering with information on the Internet, an authentication process is necessary for many communication applications.

In the previous studies, the proposed data hiding techniques mostly were applied to document files, such as pictures, videos, and text files. These methods hide information in controllable items like data structures, special file syntax, and even file headers. However, studies on data hiding for some popular Internet applications, such as the BBS, blog, and email, are few and even not found yet. Because these Internet applications use non-conventional media, the above-mentioned controllable items are not found in them. These applications just accept typed words or uploaded pictures given by users. For these reasons, it is desired to design new data hiding techniques for them in this study. Specifically, we want to develop covert communication or authentication techniques for these Internet applications.

1.1.2 Introduction to Used Media

In this study, it is desired to propose data hiding techniques for three kinds of Internet applications, including the BBS, blog, and email. We briefly introduce them subsequently.

1.1.2.1 Introduction to the BBS

The BBS (bulletin board system) is a kind of Internet forum; it is popularly used in Taiwan, Hong Kong, and China. An instance of the BBS is like Figure 1.1.

In Taiwan, the number of users on the most popular BBS site, PTT [2], can reach 60 to 150 thousand at any time. The BBS is a text-type internet forum and its screen view is presented simply by monochrome or chromatic text. Users can ask various questions, discuss any matter, interact with one another, and even send BBS mails mutually. Furthermore, journalists also write reports by adopting some conspicuous articles from the BBS.

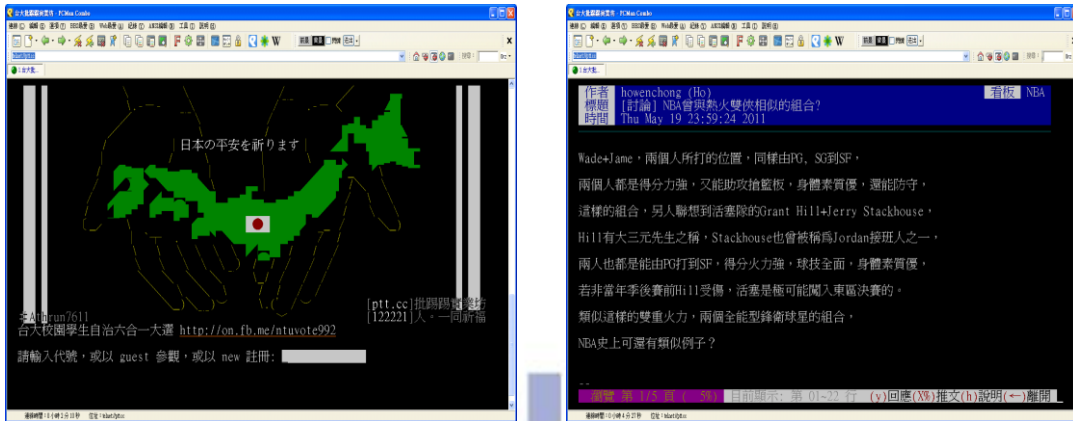


Figure 1.1 The login Screen and a normal article on a BBS.

1.1.2.2 Introduction to Blog

Weblog is a term coined by combining two words, *web* and *log*. Later, because someone jokingly broke the word *weblog* into the phrase *we blog*, the term “blog” is coined. Blog is a type of website or part of a website, and is usually maintained by an individual or a management team with regular entries of commentary, descriptions of events, or other material such as graphics or video. Visitors can interact on a blog by leaving comments and even messaging each other via widgets, and it is this interactivity that distinguishes them from other static websites. A typical blog combines text, images, and links to other blogs, web pages, and other media related to its topic. A typical blog is shown in Figure 1.2.



Figure 1.2 An instance of blogs.

1.1.2.3 Introduction to Email

Electronic mail, commonly called email or e-mail, is a method of exchanging digital messages from an author to one or more recipients. Today, almost everyone has not only a real address to send and receive mails but also a virtual address for emails. Existing email systems are based on a store-and-forward model. Hence, email servers are responsible to accept, forward, store, and deliver messages, so that users can send or receive emails at any time. An email can include a message subject, a message body, and some attached small files like pictures and text documents. There are many popular email websites such as g-mail, hotmail, and yahoo mail on the Internet, and two instances are shown in Figure 1.3.

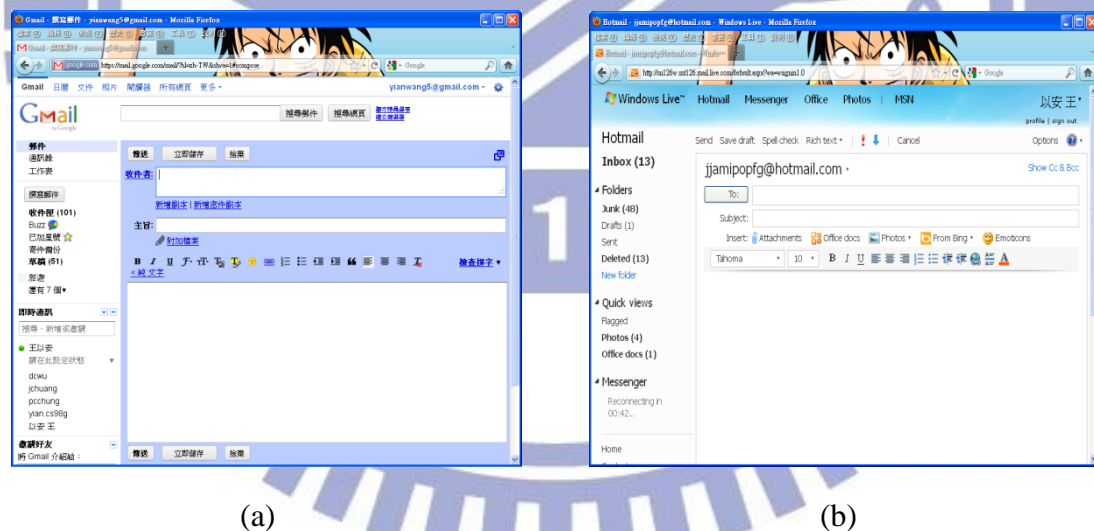


Figure 1.3 Two popular email systems. (a) G-mail. (b) Hotmail.

1.2 Overview of Related Works

Data hiding is a technique proposed to embed concerned data in a cover medium with little or no change on the appearance of the medium, so people in most cases will

not notice the existence of the hidden data. Many techniques have been proposed in recent years for hiding data. However, in this study, we think that hiding data by special character codes is a more appropriate method for internet applications such as the BBS, blog, and email. In Chapter 2, we will review some techniques of data hiding using special character codes for text documents and internet applications. In addition, we will also review the related character coding formats there, including the ASCII code, Big-5 code, and Unicode formats.

1.3 Overview of Proposed Methods

1.3.1 Definitions of Terms

The definitions of some related terminologies used in this study are described as follows.

1. *Cover media*: cover media, such as images, documents, or videos, are files into which data can be embedded.
2. *BBS*: the BBS is a popular internet forum for people.
3. *Cover article*: a cover article is an article into which data can be embedded.
4. *Stego-article*: a stego-article is an article with some data embedded in it.
5. *Protected article*: a protected article is an article into which an authentication signal is embedded.
6. *Cover email*: a cover email is an email into which data can be embedded.
7. *Stego-email*: a stego-email is an email with some data embedded in it.
8. *Protected email*: a protected email is an email into which an authentication signal is embedded.

1.3.2 Brief Description of Proposed Methods

1.3.2.1 Proposed Method for Authentication of Blog Articles

An authentication method for verifying the integrity and fidelity of blog articles via the use of invisible ASCII control codes is proposed in this study. The idea of embedding data in emails is introduced first by Lee and Tsai [1] via Outlook Express and IE under the operating system of the traditional Chinese version of Microsoft Windows XP, service pack 2, 2002. Their method is based on the use of unused ASCII codes. Secret data are encoded by these special ASCII control codes and embedded into cover emails by inserting the data into the text line ends of emails. We use this idea to hide data in blog articles under the same operating system except that the service pack is version 3 instead of 2. It was discovered that the special ASCII codes, when displayed in many kinds of blog articles by the most popular browsers such as Google Chrome, Mozilla Firefox, and IE, are invisible or look just like spaces, achieving an 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 Internet browsers and blogs.

To apply the idea to authentication of blog articles, we use a given cover blog article to produce authentication a signal, and hide the signal in the original cover blog article, resulting in a stego-article. Through the authentication process, we can verify whether the protected blog article has been tampered with or not by comparing the authentication signal extracted from the stego-article with those computed from the original cover blog article in the stego-article. The detailed authentication process and the related data embedding and extraction algorithms will be described in Chapter 3.

1.3.2.2 Proposed Method for Covert Communication via The BBS

Two new methods for data hiding using special Big-5 codes via the BBS with Big-5 servers are proposed for covert communication in this study. One is implemented under the operating system of the traditional Chinese version of Microsoft Windows XP, service pack 3, 2002. And the other can be applied under most general operating systems.

Because the ASCII control codes are utilized to implement some system functions, we cannot hide data by the same method used for blog articles mentioned above. Through continually testing and observation in our experiments, we discovered that by the transcoding of different text coding systems with Big-5 and Unicode formats, some special Big-5 codes are invisible when they are displayed on popular BBS browsers such as PCMan, KKMan, and Pietty. So we use these special Big-5 codes for data hiding in the BBS in the first method. More specifically, we insert the invisible Big-5 codes into the text line ends, which do not change the meanings of the sentences in the cover article, neither cause any noticeable difference to the reader.

Furthermore, we develop a second new method to hide data by the use of some special Big-5 *space* codes. These codes are defined in the Big-5 standard format, so that the proposed method can be used generally in most operating systems.

By the two methods, the embedded secret data are all hard to be discerned. Hence, the proposed methods can be used for covert communication. The detailed embedding and extraction processes of the proposed methods will be described in Chapter 4.

1.3.2.3 Proposed Method for BBS Authentication

We propose also a technique for BBS authentication in this study by the idea used in the above-mentioned two methods of data hiding in the BBS. There is much important information, like goods orders, meeting places, business transactions, etc., on the BBS or in BBS mails, so it is necessary sometimes to conduct authentication of such BBS articles or mails. This activity is called BBS authentication in this study. The proposed method for this purpose will be described in Chapter 5.

1.3.2.4 Proposed Method for Email Authentication

A new method for email authentication by the use of special UTF-8 space codes is proposed in this study. To reach the goal of email authentication, firstly we also tried to implement the method proposed by Lee and Tsai [1], i.e., data hiding in emails by using the unused ASCII control codes. However, in using webmails like the g-mail, hotmail, and yahoo mail, when sending an email, the unused ASCII control codes will be removed or changed into general white spaces. Because this obviously is not appropriate for data hiding in the webmail, we propose the use of special UTF-8 space codes to achieve the aim in this study. The idea is inspired from one of the above-mentioned methods: the data hiding technique for the BBS using special Big-5 space codes. Today, almost all webmail servers are built using the Unicode UTF-8 format as their text coding systems. It is a computing industry standard for consistent encoding, representation, and handling of text expressed in most of the world's writing systems, and has codes corresponding to the special Big-5 space codes which are used in the proposed method for data hiding in the BBS mentioned previously. Thus, we can find the special UTF-8 space codes to hide data in the webmail and use the result for email authentication. The detailed processes will be described in Chapter 6.

1.4 Contributions

Some contributions made in this study are listed in the following.

1. For the first time blog articles are used as cover media for data hiding applications.
2. For the first time the BBS is used as a cover medium for data hiding applications.
3. An authentication method for verification of integrity and fidelity of blog articles by the use of invisible ACSII control codes is proposed.
4. A new data hiding technique using invisible Big-5 codes is proposed for the two applications of covert communication via the BBS as well as BBS authentication.
5. A new data hiding technique using special Big-5 space codes is proposed for the two applications of covert communication via the BBS as well as BBS authentication.
6. A new data hiding technique using special UTF-8 space codes is proposed for email authentication.

1.5 Thesis Organization

In the remainder of this thesis, related works about data hiding using special character codes and the used character coding formats are reviewed in Chapter 2. In Chapter 3, the proposed authentication method for verification of integrity and fidelity of blog articles is described. And the proposed methods for covert communication via the BBS as well as BBS authentication are described in Chapters 4 and 5, respectively. In Chapter 6, the proposed method for email authentication by the use of special UTF-8 space codes is described. Finally, conclusions and some suggestions for future works are given in Chapter 7.

Chapter 2

Review of Related Works and Character Coding Formats

2.1 Previous Studies on Data Hiding Techniques Using Special Character Codes

With prosperity of the computer network, the Internet has become a very popular medium for information communication. A lot of important information is interchanged on the Internet all the time. Especially, on text-typed Internet applications like the BBS, blog, and email, people communicate messages, discuss private matters, publish articles, and even do business. But article created or sent in these activities might be tampered with illegally by hackers on the line. Therefore, it is necessary to protect these articles. In this study, we design data hiding techniques to achieve this purpose by covert communication and authentication on the Internet applications.

Articles on the Internet applications belong to soft-copy texts [3]. In recent years, some methods of data hiding via text documents have been proposed, like using file headers [4], file structures, and file features [5]. However, the Internet applications are not conventional media, so these controllable items are not found in them. Hence, we implement data hiding techniques on them by embedding special character codes. In the past, some data hiding methods about using character codes have been proposed. Wayner [6] 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. Bender et al. [3] proposed the use of infrequent additional spaces to form secret data and transmitted 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.” An illustration of the method is shown in Figure 2.1.

Data hiding techniques via special character codes are also used for some popular text documents and Internet applications. A survey of them is conducted in this chapter.

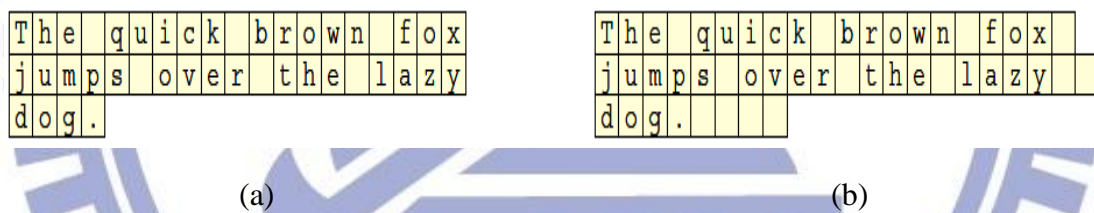


Figure 2.1 Example of data hidden using white space [3]. (a) Normal text. (b) White space encoded text.

2.1.1 Review of Data Hiding Techniques via Text Documents

Every day, numerous text documents are interchanged on the Internet. It is hard to prevent malicious users from intercepting and tampering with them, so developing data hiding techniques to protect important information on them is necessary.

For the XML which is a set of rules for encoding documents in machine-readable form, Inoue et al. [7] proposed a technique to embed secret data by inserting white

spaces in tags. Representation of a tag is accomplished by including either some white spaces before the close bracket, or no white space [8]. By inserting or deleting spaces, they can embed the data preserving all meanings of original documents. For the PDF which is a popular file format with independency of different computer platforms, data hiding techniques can also be attained by using equivalent white space codes or invisible ASCII codes, as proposed by Lai and Tsai [9] and Lee and Tsai [10]. An experimental result found in Lee and Tsai [10] is shown in Figure 2.2. And even for software programs like the Visual C++ and C++ Builders, three ways to hide data using invisible ASCII control codes were proposed by Lee and Tsai [11], including 1) alternative space coding, 2) line-end space coding, and 3) null space coding.

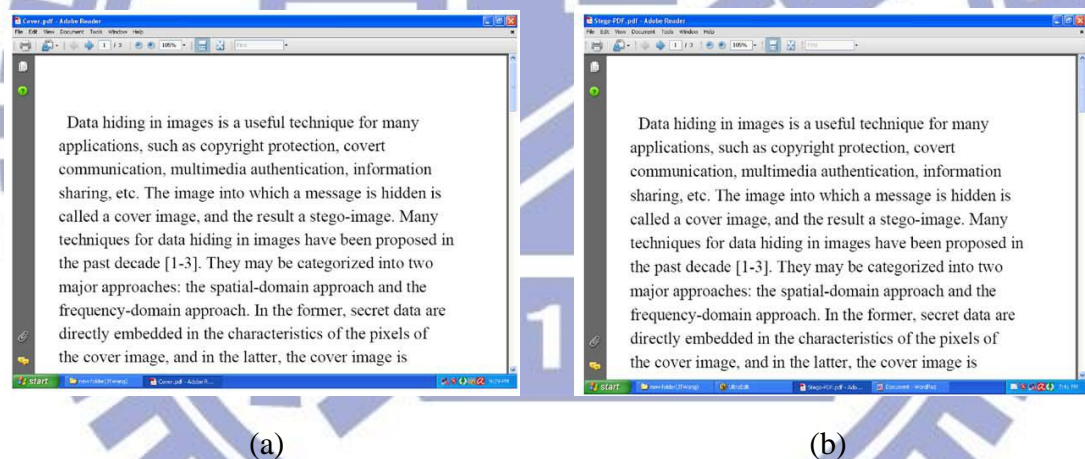
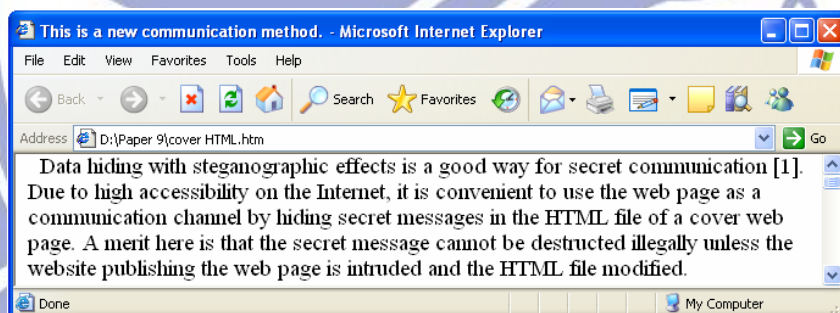


Figure 2.2 An experimental result found in Lee and Tsai [10]. (a) Cover file seen in Adobe Reader 8.1.2 window. (b) Stego-file seen in Adobe Reader 8.1.2 window with message “This is a covert communication method” embedded.

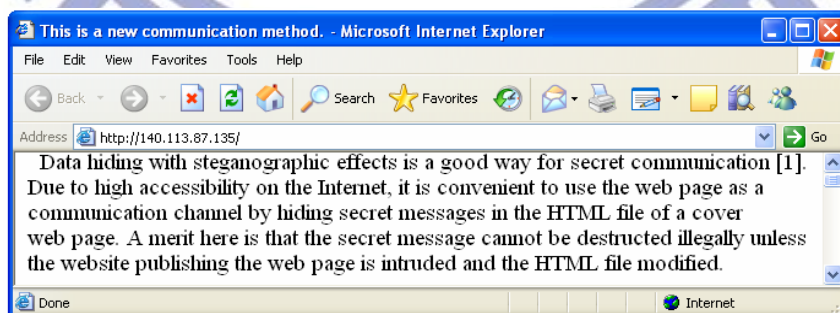
2.1.2 Review of Data Hiding Techniques for Internet Applications

Although many information hiding techniques have been proposed in recent

decades, methods about hiding data in Internet applications directly are very few. Lee and Tsai [12] and Huang and Tsai [13] proposed some techniques for data hiding by embedding special codes in HTML files to substitute for the original white spaces in the files, and an experimental result found in Lee and Tsai [12] is shown in Figure 2.3. In these cases, message data were hidden in HTML files so that these files became stego-media for secret communication or secret sharing when the HTML files are displayed on the Internet. However, these methods are *indirect* data hiding techniques for Internet applications. In another paper published by the same authors, Lee and Tsai [1], a *direct* data hiding technique was proposed to embed secret data into email text line ends using special ASCII control codes. These special ASCII control codes are invisible when displayed on the screen and so will not affect a user's reading of the resulting email.



(a)



(b)

Figure 2.3 An experimental result found in Lee and Tsai [12]. (a) Cover text seen in the window of the IE. (b) Stego-text (with message about “Cartesian coordinates” embedded) seen in the window of the IE.

2.1.3 Review of Other Techniques and Summary

For some text documents, data hiding methods using not only special character codes but also special file syntax or file features have been proposed. Chang and Tsai [14] used pseudo-spaces, the specific string “ ,” to encode copyright data into the text of an HTML file; duplicated the copyright data to enhance the robustness against HTML manipulations; and combined the blank character code and the HTML special syntax to hide data. Zhong, et al. [15] proposed a data hiding method for PDF documents by adjusting the positions of the text characters slightly to embed the secret data. They also hid data by combining character codes and certain special file features.

In conclusion, the text-typed Internet applications and text documents are good choice as a covert channel for data hiding because they are commonly used for information exchanges in daily works and for communication on the Internet. Some data hiding techniques applied on different kinds of text document formats have been proposed over the past decade. However, studies on data hiding via the Internet applications like the BBS, blog, and email are very few and even not found yet, so we will propose new data hiding techniques and related applications for them in this study.

2.2 Review of Related Character Coding Formats

2.2.1 Review of ASCII Format

The American Standard Code for Information Interchange, ASCII, is a

character-encoding scheme based on the ordering of the English alphabet. ASCII codes are used to represent text in computers, communications equipment, and other devices that use text. A standard ASCII code is composed of seven bits and usually expressed as two hexadecimal numbers. The first edition of the standard was published in 1963, a major revision in 1967, and the most recent update in 1986 [16]. The ASCII codes include 128 characters: 33 are non-printing control characters (now mostly obsolete), 94 are printable characters, and the space which is considered as an invisible graphic. All the ASCII codes are listed in Table 2.1.

As computer technology spreads throughout the world, many coding standards have been developed to facilitate the expression of non-English alphabets. However, these character coding standards, such as the Unicode and Big-5, all include the ASCII codes as the kernel set. Now, almost all web servers are built with the Unicode. Therefore, in many current Internet applications, the properties of the ASCII codes are still preserved.

Table 2.1 ASCII code chart.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|-----|-----|----|----|----|-----|
| 0 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | SO | SI |
| 1 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| 2 | | ! | " | # | \$ | % | & | ' | (|) | * | + | , | - | . | / |
| 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 4 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 5 | P | Q | R | S | T | U | V | W | X | Y | Z | [| \ |] | ^ | _ |
| 6 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 7 | p | q | r | s | t | u | v | w | x | y | z | { | | } | ~ | DEL |

2.2.2 Review of Big-5 Format

The Big-5 or Big5 is a character encoding method used in Taiwan, Hong Kong,

and Macau for Traditional Chinese characters. When the Unicode has not been developed yet, many different language coding standards existed in various countries under old operating systems like MS-DOS. Now, under the current operating systems such as Windows XP and Window 7, all internal messages are interchanged using the Unicode and all the different language coding standards such as Big-5 and ASCII can be supported. In the recent years, the most often used Big-5 version is defined in Microsoft Windows Codepage 950 (CP950) [17]. A standard Big-5 code is a double-byte character set and the structure is shown in Figure 2.4.

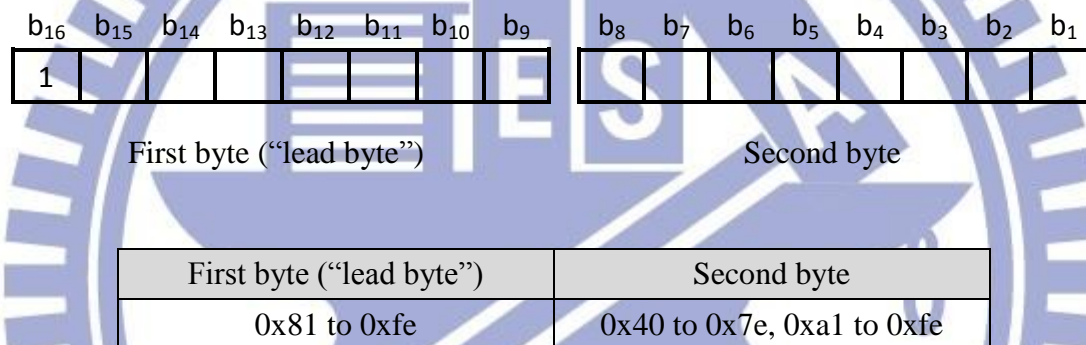


Figure 2.4 Big-5 coding format.

Though most of the current Internet applications do not support Big-5 coding, for the BBS which is a very popular kind of forum in Taiwan, China, and Hong Kong, the Big-5 is still the major character coding system for BBS servers.

2.2.3 Review of UTF-8 Format

UTF-8 (8-bit Unicode Transformation Format) is a multibyte character encoding style for the Unicode. And the Unicode is a computing industry standard for consistent encoding, representation, and handling of text expressed in most of the world's writing systems. It is developed in conjunction with the Universal Character

Set standard and published in book form as *The Unicode Standard* [18]. The success of the use of the Unicode to unify character set coding has led to its widespread and predominant use in the internationalization and localization of computer software. The standard has been implemented in many recent technologies, including the XML, the Java programming language, the Microsoft .NET Framework, and many modern operating systems. The Unicode can be implemented by different character encodings, and the UTF-8 is the most commonly used one. Unlike the original Unicode which is a double-byte character set, the UTF-8 is a variable-length encoding scheme, with each character represented by one to four bytes. The UTF-8 encoding format is shown in Table 2.2 and an example is shown in Figure 2.5 [17].

Table 2.2 UTF-8 encoding format.

| Code point range | Binary code point | UTF-8 bytes | Annotations |
|-------------------------|----------------------------------|--|---|
| U+0000 to U+007F | 0xxxxxxx | 0xxxxxxx | the range of ASCII the highest bit of the byte is 0 |
| U+0080 to U+07FF | 00000yyy yyxxxxxx | 110yyyyy 10xxxxxx | the first byte starts with 110 the following byte starts with 10 |
| U+0800 to U+FFFF | zzzzyyyy yyxxxxxx | 1110zzzz 10yyyyyy 10xxxxxx | the first byte starts with 1110 the following bytes start with 10 |
| U+010000 to U+10FFFF | 000wwwzz zzzzyyyy yyxxxxxx | 11110www 10zzzzzz 10yyyyyy 10xxxxxx | the first byte starts with 11110 the following bytes start with 10 |



Figure 2.5 An example of UTF-8 coding [17].

The UTF-8 has many advantages. For example, every valid ASCII character is also a valid UTF-8 encoded Unicode character with the same binary value, so old systems and software with the ASCII encoding format can make no change or just a few slight modifications to be used further. Therefore, it is gradually becoming a top-priority character coding system for e-mail, website, and other Internet applications.



Chapter 3

Authentication of Blog Articles by Invisible ASCII Control Codes

3.1 Introduction and Problem Definition

3.1.1 Introduction

In this chapter, we will specifically introduce the proposed data hiding method for authentication of blog articles. In Section 3.1.2, the problem definition is described, and the major idea of the proposed data hiding method for authentication is described in Section 3.2. In Section 3.3, we present the technique we propose to generate an authentication signal and the process to embed it. In Section 3.4, a process for extraction and verification of the authentication signal is proposed. Experimental results showing the feasibility of the proposed method are shown in Section 3.5. Finally, a brief summary is given in Section 3.6.

3.1.2 Problem Definition

Blog is a virtual channel, which allows people to express feelings, and many public figures like politicians and entertainers also advertise political philosophies or raise awareness by interacting with their fans on the blog. Some blog articles are important messages worth long-time recording. Hence, to check the integrity and fidelity of articles on the blog to see whether they have been attacked or not is

important. For this purpose, we propose a data hiding technique for authentication of blog articles in this study.

For public figures, the proposed method offers significant functions. For example, their blogs are often managed by a management team, and everyone in the team has the authority to publish or modify the contents of the blogs. It results that if a malicious person tampers with the content of an article or even posts a fake message on a celebrity's blog, the members of the management team cannot take action as soon as possible, because they may think that the change is made by another member of the team. By the proposed method, they can rapidly authenticate the integrity and fidelity of the blog articles, preventing the negative advertising effect brought by malicious tampering. And for people who manage their blogs by themselves, the method not only can save their time spent on literally tedious checking of the authenticity of their articles published before, but also can let other people, who do not have the authority to modify the blog articles, help the work of authenticating them if they are given the keys to enter the system. The major idea of the proposed method is specifically introduced in the next section.

3.2 Major Idea of Proposed Method by Use of Invisible ASCII Control Codes

3.2.1 Use of Special Character Codes

On the blog, as mentioned in Chapter 2, users can only type simple words and upload some small files like pictures and short videos. Many controllable items

facilitating data hiding, like data structures, special file syntax, and file headers, cannot be found on the blog, so we use some *special character codes* to achieve the goal of data hiding in this study.

Specifically, we hide data in the blog using *invisible ASCII control codes*. Part of the ASCII codes from 00 through 1F, namely, the ASCII *control codes*, were originally designed to control some computer peripheral devices such as teletypes, tape drivers, printers, etc. All the ASCII control codes are listed in Table 3.1 [1]. Now, because of the rapid development of new peripheral hardware technologies, however, the ASCII control codes are rarely used for their original purposes, except the codes for text display control, like 0A and 08 with the meanings of *line feed* and *backspace*, respectively.

Table 3.1 ASCII control codes and description [1].

| 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 |

Furthermore, through continuous tests and observations in our experiments, we discovered that some of the control codes are *invisible* or just shown as *white spaces* when they appear in a blog article on many popular web browsers under certain software environments. In addition, almost all of the current web servers today use the UTF-8 standard as the system text coding format, and the properties and features of the ASCII codes are completely included as the kernel set; the UTF-8 codes equate exactly to the ASCII codes for code values smaller than 128.

In this study, it is desired to use the invisible ASCII control codes in the UTF-8 to embed data in blog articles without causing noticeable artifacts under the popular software environments of Google Chrome, Mozilla Firefox, IE, and the operating system of the traditional Chinese version of Microsoft Windows XP, service pack 3, 2002. To achieve this aim, we divide a secret message into 2-bit segments, then map them to the corresponding special ASCII control codes which are discovered through continuous testing mentioned previously, and embed them as well as some *end signals* into a cover article. Table 3.2 shows the used special ASCII codes and the mapping relationship devised in this study for this purpose.

In more detail, we use the four ASCII codes of 1C, 1D, 1E, and 1F, which, when displayed, look just as *nothing* on the web browsers of Mozilla Firefox and Google Chrome. Two examples are shown in Figure 3.1, where Figure 3.1(a) shows a result displayed in Firefox in which the embedded codes at the line ends *cannot be highlighted*, while Figure 3.1(b) shows a result displayed in Chrome in which the spaces between the ends of all text lines and the right end of the blog article window are *all highlighted* when the entire article is highlighted, no matter whether there are embedded codes at line ends or not. Both cases have *no* worry about leaking embedded characters at text line ends. However, in IE, these codes are displayed as *white spaces*. This causes the codes to be *visible* to the user, and so decreases the

security of the embedded message using the codes. A solution is proposed in this study, which is discussed next.

Table 3.2 Encoding table for used invisible ASCII control codes.

| Bit stream (binary) | 00 | 01 | 10 | 11 | End signal |
|--------------------------|----|----|----|----|------------|
| ASCII code (Hexadecimal) | 1C | 1D | 1E | 1F | 1F0A |

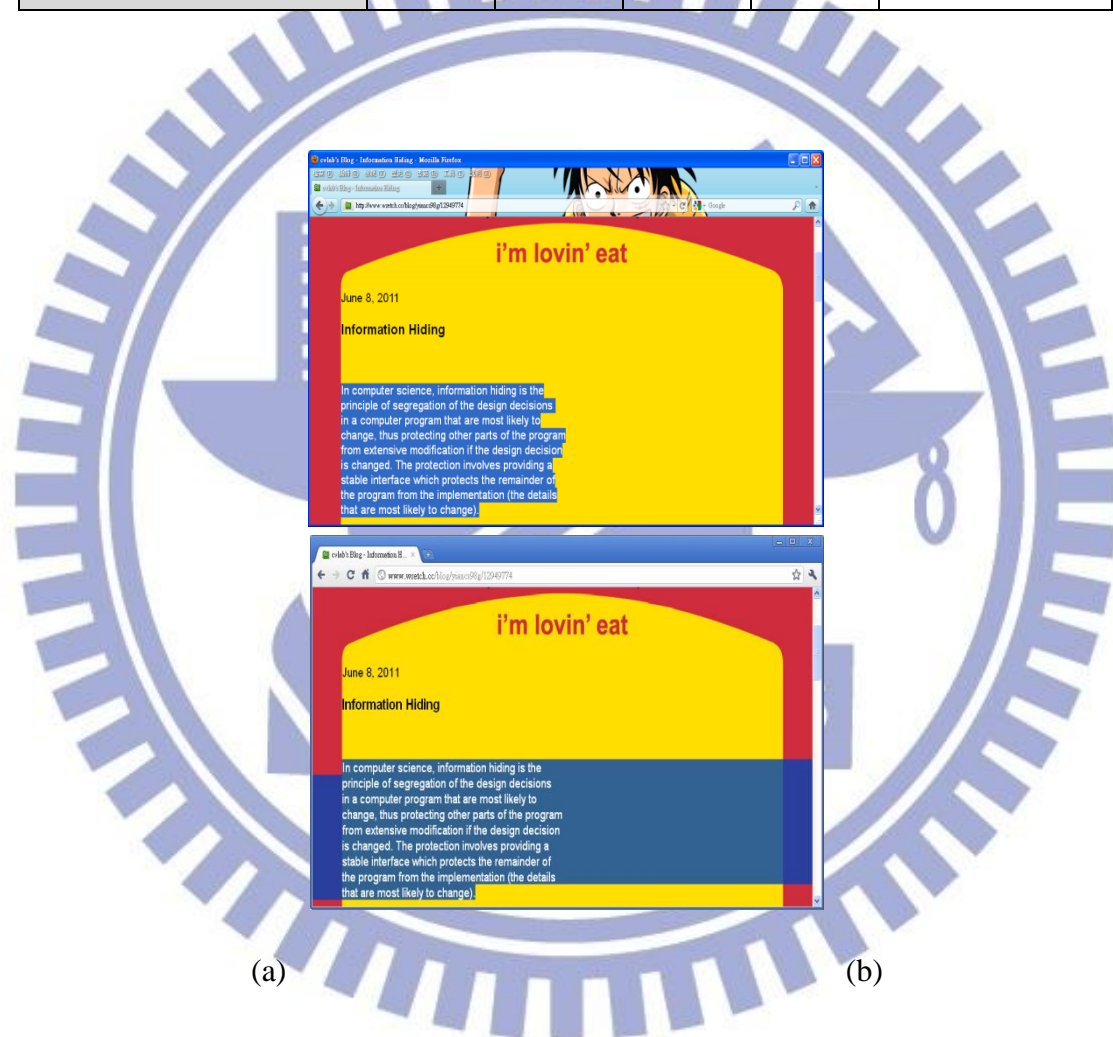


Figure 3.1 Some examples of highlighted blog articles with secret codes embedded in them on (a) Mozilla Firefox and (b) Google Chrome.

3.2.2 Necessity of Distributing Embedded Codes

Evenly at Line Ends to Reduce Suspicion

As mentioned previously, in IE, when all the text of a blog is highlighted, the embedded special character codes will appear to be white spaces, thus leaking the secret hidden in the blog article. This secret leakage phenomenon is even clearer when all the special codes are embedded at the line ends in sequential orders, starting from the first line, because then long blank spaces will appear at the ends of the beginning lines, like the example shown in Figure 3.2(a). This phenomenon will tend to arouse the attacker's suspicion.

One way to solve this problem, as proposed in this study, is to distribute the embedded codes *evenly* into all the line ends in order not to create long blank spaces at the ends of the beginning lines. More specifically, we embed different numbers of special ASCII codes in accordance with the variable lengths of the text lines in the blog article. That is, if a text line is longer, then we embed less codes, and vice versa. To implement this idea, we estimate the size of the *data embedding slot* at each text line end, and compute accordingly the required number of special codes embeddable at the end of each line. Then, we sequentially embed the codes into the blog article according to the computed numbers. A stego-article with special codes embedded in this way is shown in Figure 3.2(b), in which the special codes are seen to have been embedded evenly at the line ends. The detailed hiding method will be described in Section 3.3.



(a)

(b)

Figure 3.2 Some examples of highlighted blog articles with secret codes embedded in them in IE. (a) The stego-article with secret codes embedded in order. (b) The stego-article with secret codes embedded evenly using the proposed method.

3.2.3 Construction of End Signals

The previously-mentioned end signal used in this study is composed of two ASCII control codes, 1F and 0A, which together specify a unique signal for unambiguous identification of the line end. The code 0A appears at each text line end, originally purely for the use as a *line feed* signal. However, because in the proposed method end signals are only embedded at text line ends, we can just create them by merging the two existing codes 1F and 0A instead of finding a new one.

In summary, by the use of a secret key and the proposed data hiding technique, we can produce a protected blog article with a barely imperceptible authentication signal embedded in it for detection of any falsification incurred by malicious users.

3.3 Authentication Signal Generation and Embedding Process

In this section, we describe the proposed process to generate an authentication

signal and embed it into a blog article. An illustration of the process is shown in Figure 3.3. When the stego-article is displayed on the blog, it is desired that the article body can fit the width of the blog article window. For this, first we fold longer article lines into shorter ones, leaving at least eight characters at each line end as a *data embedding slot*. Next, we remove from the folded blog article all the *line feed* signals so that the verification process described in the next section will not be interfered by redundant line feed signals. The modified blog article and a secret key then are used to generate an authentication signal using a hash function and the exclusive-OR operation. Subsequently, the authentication signal is divided into 2-bit segments. Finally, we map them into corresponding special ASCII control codes, and embed them into the text line ends accompanied with end signals to obtain a protected blog article. The detail is described in the following.

Algorithm 3.1 Authentication signal generation and embedding process.

Input: a secret key K , a hash function f (such as MD5), and a blog article B to be protected.

Output: a protected blog article B' .

Steps.

1. Fold sequentially each text line l_i with a length larger than 60 units (with a unit meaning the length of an ASCII code displayed on the blog) in blog article B into a 60-unit line by inserting a line feed, denoted as LF, occupying *zero unit*, and represented by ASCII code 0A, *after the original 60th character in l_i* to generate a *folded article*, denoted as F .
2. Compute the size L_i of the data embedding slot at the end of each text line l_i in F by:

$$L_i = 68 - \text{the length of } l_i,$$

which means that the maximum number of characters that can be inserted at the end of l_i .

3. Remove all the line feed signals in F , use the result and the secret key K as inputs to the hash function f to generate two 128-bit digests F' and K' , respectively, and return all the removed LF signals back into their original positions in F .
4. Compute the exclusive-OR value $F' \oplus K'$ to obtain a 128-bit authentication signal S .
5. Separate the bits of S into 64 two-bit segments t_1, t_2, \dots, t_{64} .
6. Map t_1 through t_{64} into invisible ASCII control codes p_1 through p_{64} , respectively, according to Table 3.2 and let $N_1 = N_2 = 64$ for use as parameters in subsequent steps.
7. Scan F from the first line to find the line, say the i -th, with the longest slot and calculate the number n_i of invisible ASCII control codes *embeddable* in the i -th line in the following way:
 - (1) if $L_i > 1$, then increment n_i by 1, decrement L_i by 1, decrement N_1 by 1, and perform Step 7 again;
 - (2) if $L_i = 1$ or $N_1 = 0$, then perform Step 8.
8. Embed the symbols p_1, p_2, \dots, p_{64} sequentially into F , starting from the first line in the following way:
 - (1) Scan l_i to find the *line feed* LF, remove it, sequentially embed n_i symbols in l_i at the end, decrement N_2 by n_i , and append an end signal, in which an LF is included, to the end of the embedded symbols.
 - (2) If the last line is processed, and if $N_2 > 0$, then embed the remaining symbol/symbols below F as one or more blank lines in the following way.
 - 8.1 Embed as many symbols as possible into a new line sequentially before the length of the line (in units) becomes larger than 67, and append an

end signal to the line end.

8.2 If all symbols are embedded, then continue; otherwise, repeat Step 8.1 again.

9. Take the final version of F as the desired protected blog article B' .

It is possible that the symbols of the authentication signal cannot be embedded in the text line ends *completely*. This might happen when the blog article is too short. In this case, more lines are appended to the end of the article, all being empty, and the remaining symbols then are all embedded into them, as done in Step 8 of the above algorithm.

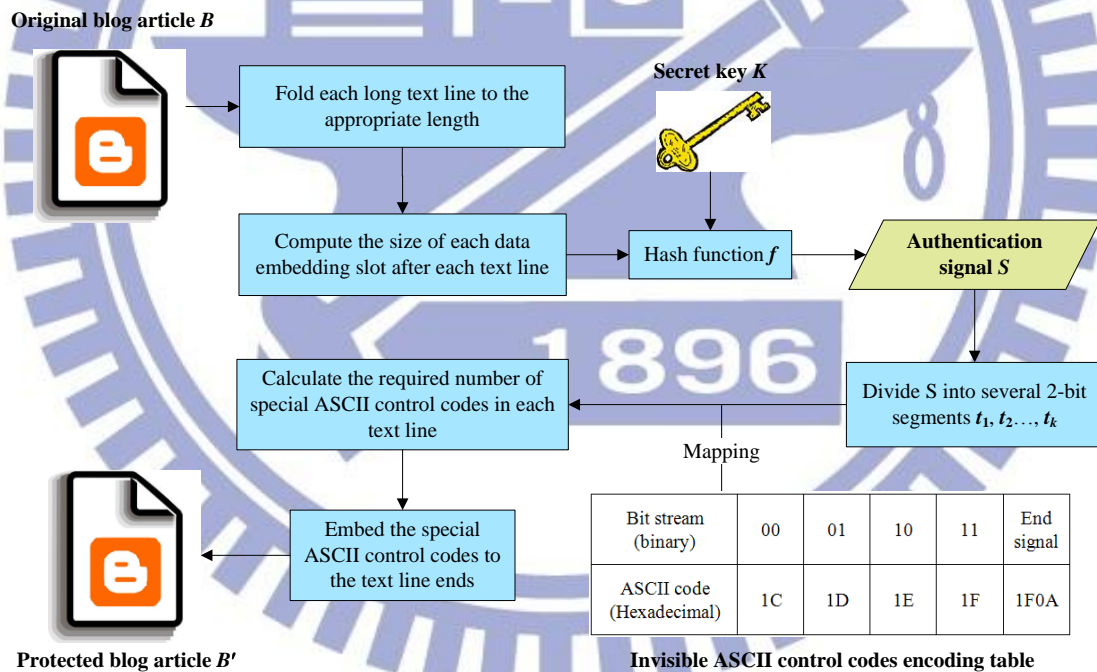


Figure 3.3 Flowchart of proposed authentication signal generation and embedding process.

3.4 Authentication Signal Extraction

and Blog Verification Process

The proposed authentication signal verification scheme can be used to verify the integrity and fidelity of a protected blog article, and the detailed secret data extraction process for blog authentication is illustrated in Figure 3.4. First, we extract the special ASCII codes embedded in the protected blog article and transform them to be an authentication signal S . Then, we use the same secret key and hash function as those used in Algorithm 3.1 to transform the blog article, in which all the secret data and the line feed signals are removed, into a verification signal T . Finally, by comparing the two signals S and T , we can decide whether the protected blog article has been modified or not. The detailed algorithm is described in the following.

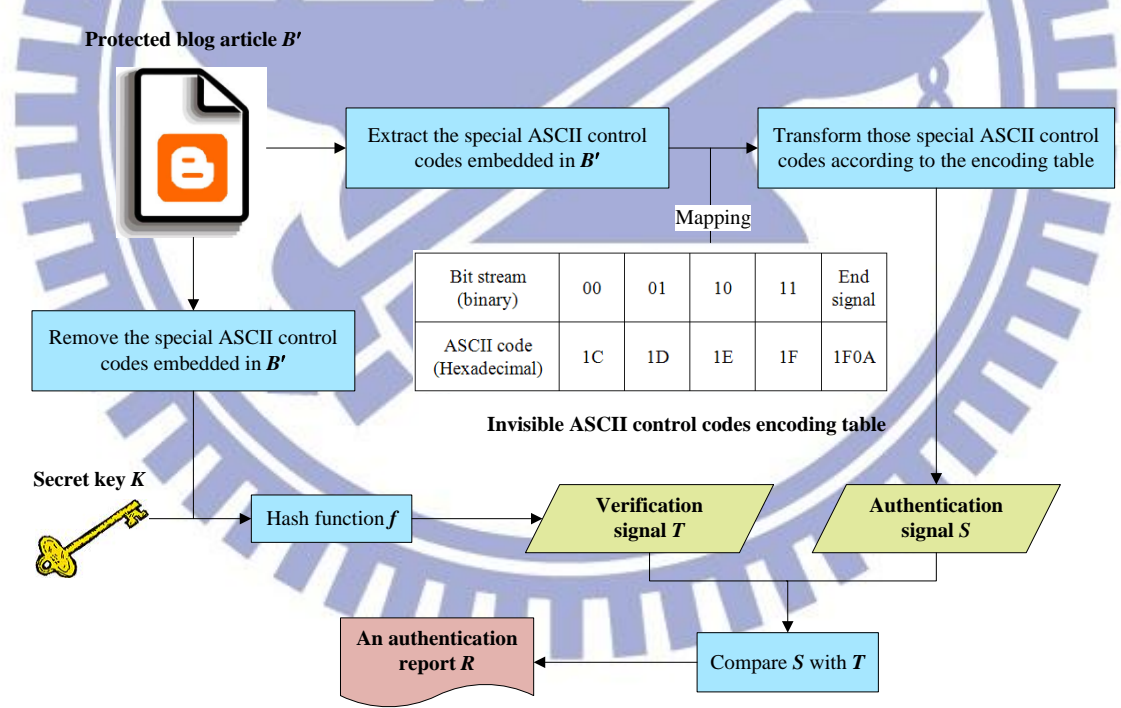


Figure 3.4 Flowchart of the proposed authentication signal extraction and blog article verification process.

Algorithm 3.2 Authentication signal extraction and blog article verification.

Input: a secret key K and a hash function f both being the same as those used in Algorithm 3.1; and a protected blog article B' .

Output: an authentication report R .

Steps.

1. Check each line l_i in the protected blog B' sequentially, starting from the first line; and extract the special ASCII control codes embedded in front of the end signal in l_i .
2. Concatenate all the extracted special ASCII codes sequentially into a set of 64 codes, p_1, p_2, \dots, p_{64} .
3. Map p_1 through p_{64} to corresponding two-bit segments t_1, t_2, \dots, t_{64} according to Table 3.2.
4. Concatenate t_1 through t_{64} into a 128-bit authentication signal S .
5. Use the secret key K as an input to the hash function f to generate a 128-bit digest K' .
6. Remove all the secret data and line feed signals from the blog article, and use the result as an input to the hash function f to generate a 128-bit digest B'' .
7. Compute the exclusive-OR value $B'' \oplus K'$ to get a 128-bit verification signal T .
8. Compare S and T , resulting in the following two cases.
 - (1) If $S = T$, then regard the input B' as *unmodified* and mark it so in the authentication report R .
 - (2) If $S \neq T$, then regard B' as *modified* and mark it so in R .
9. Output the authentication report R .

3.5 Experimental Results

Several experiments using the proposed algorithms about authentication of blog

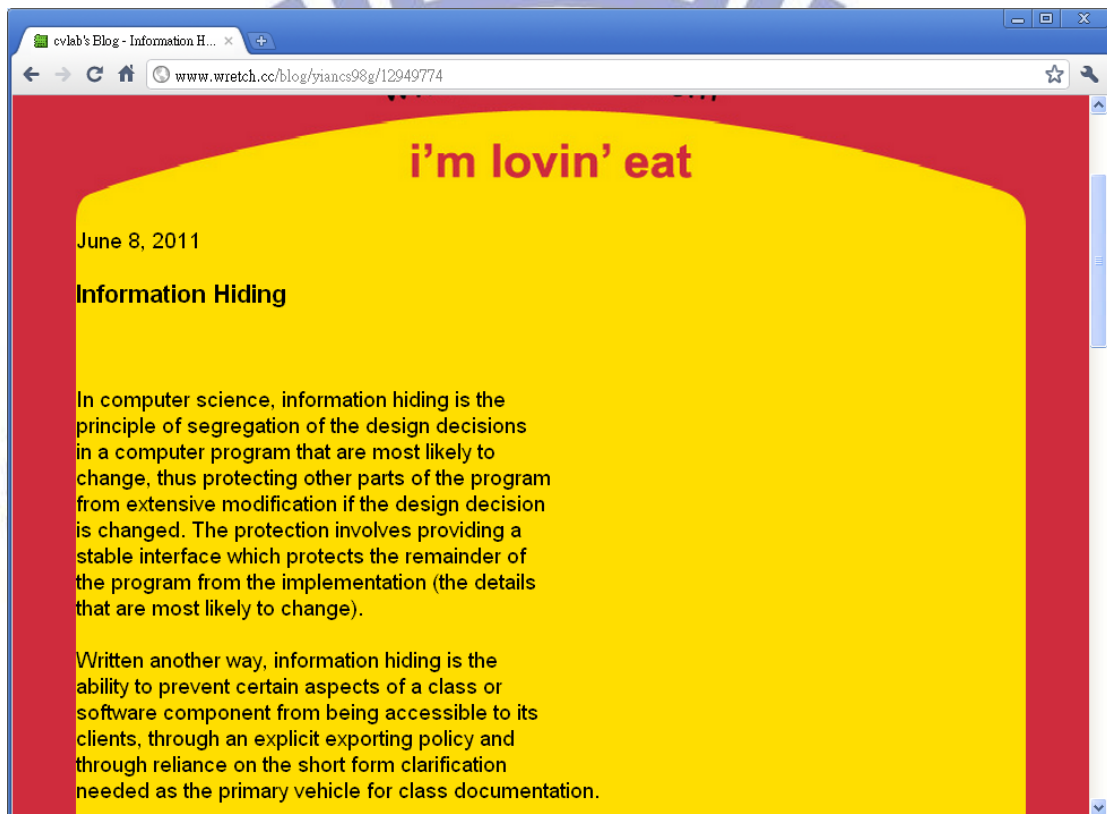
articles have been conducted. The algorithms were implemented using the language of Microsoft Visual C#. We wrote blog articles on a public blog system, and these articles can be displayed on popular web browsers such as Google Chrome, Mozilla Firefox, and IE. In this section, we show some experimental results displayed on Google Chrome.

In Figure 3.5(a), an original blog article displayed on the web browser of Google Chrome is shown. And the user interface is shown in Figure 3.5(b). By entering a secret key, we generated an authentication signal and embedded it in the original article, resulting in a protected blog article as shown in Figure 3.5(c). If the protected article is not modified, the mark “Authentication is successful.” will be shown on the authentication report like Figure 3.5(d). However, if a malicious user tries to tamper with the protected article, yielding a modified blog article, people who have the same secret key can verify the integrity and fidelity of it. A tampered article and the corresponding verification result are shown in Figures 3.5(e) and 3.5(f), respectively. And Figure 3.6 shows another example of our experimental results displayed on the web browser of Mozilla Firefox. In this figure, we obtain an authentication report by using the correct secret key and a wrong secret key, respectively, and get the different results. These experimental results show that, using the proposed method, we can verify whether a blog article has been tampered with or not successfully.

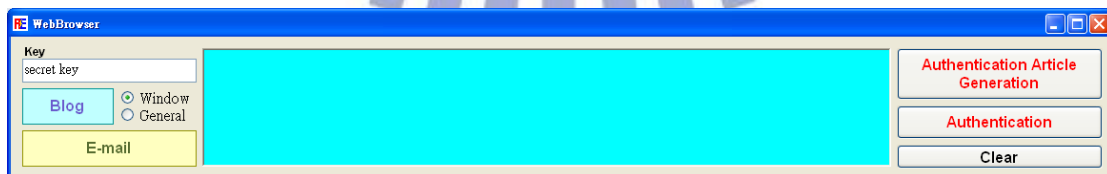
3.6 Summary

In this chapter, a method for authentication of the integrity and fidelity of blog articles using a new data hiding technique has been proposed. Authentication signals of the form of invisible ASCII control codes are generated using a folded version of a given blog article. They are embedded sequentially in the folded article according to

pre-computed numbers of secret symbols in the lines. Even in the most unfavorable web browser IE, the embedding result is good to arouse no suspicion. A secret key was used also to randomize the content of the authentication signal so that malicious users cannot forge easily the text content and the corresponding authentication signal. The proposed method is reliable to protect blog article from being tampered with, as proved by the experimental results.

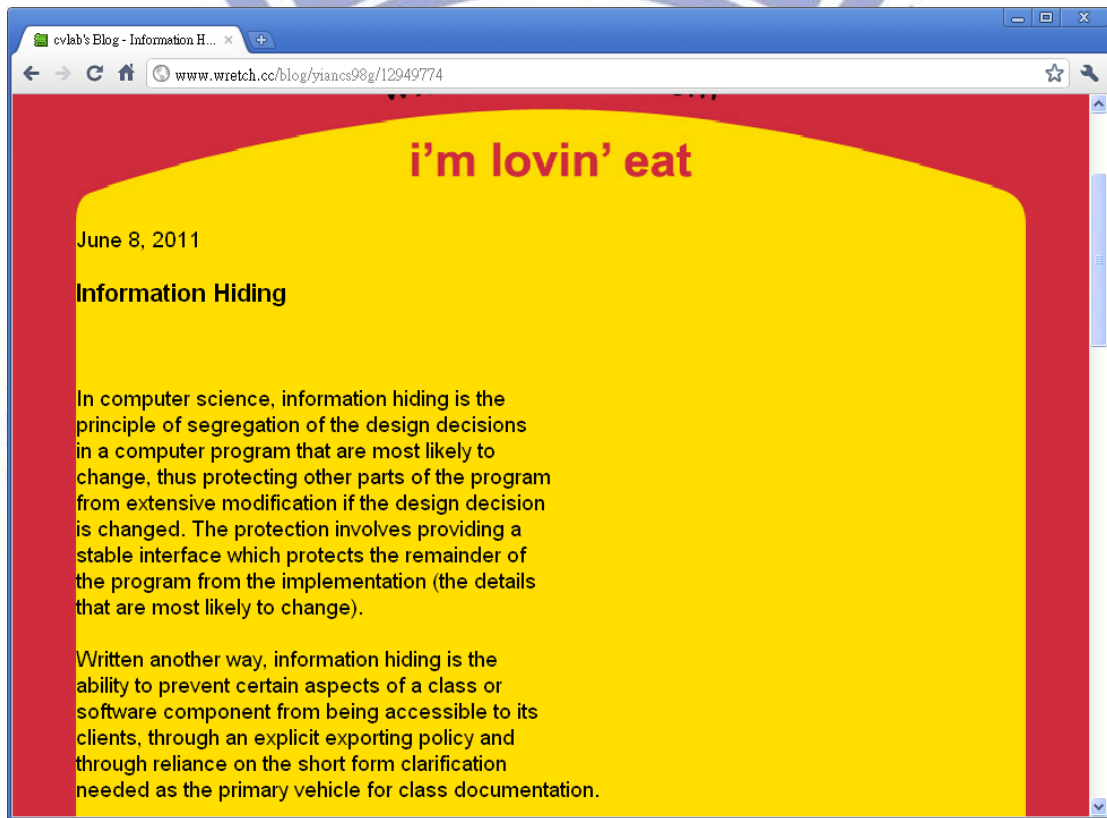


(a)

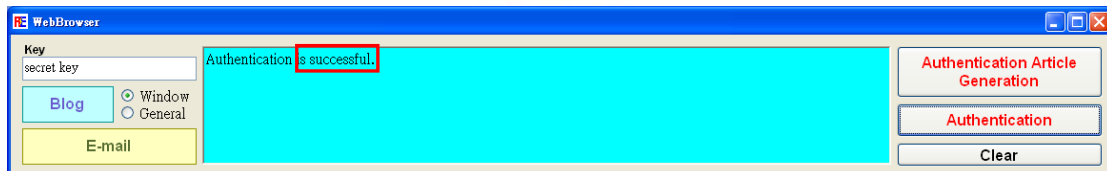


(b)

Figure 3.5 An example of experimental results. (a) An original blog article. (b) A user interface used to generate authentication signal and protected blog article. (c) The protected blog article with an authentication signal embedded. (d) The authentication report with the message “Authentication is successful.” (e) A protected blog article with a tempered word. (f) The verification result of the tempered blog article.

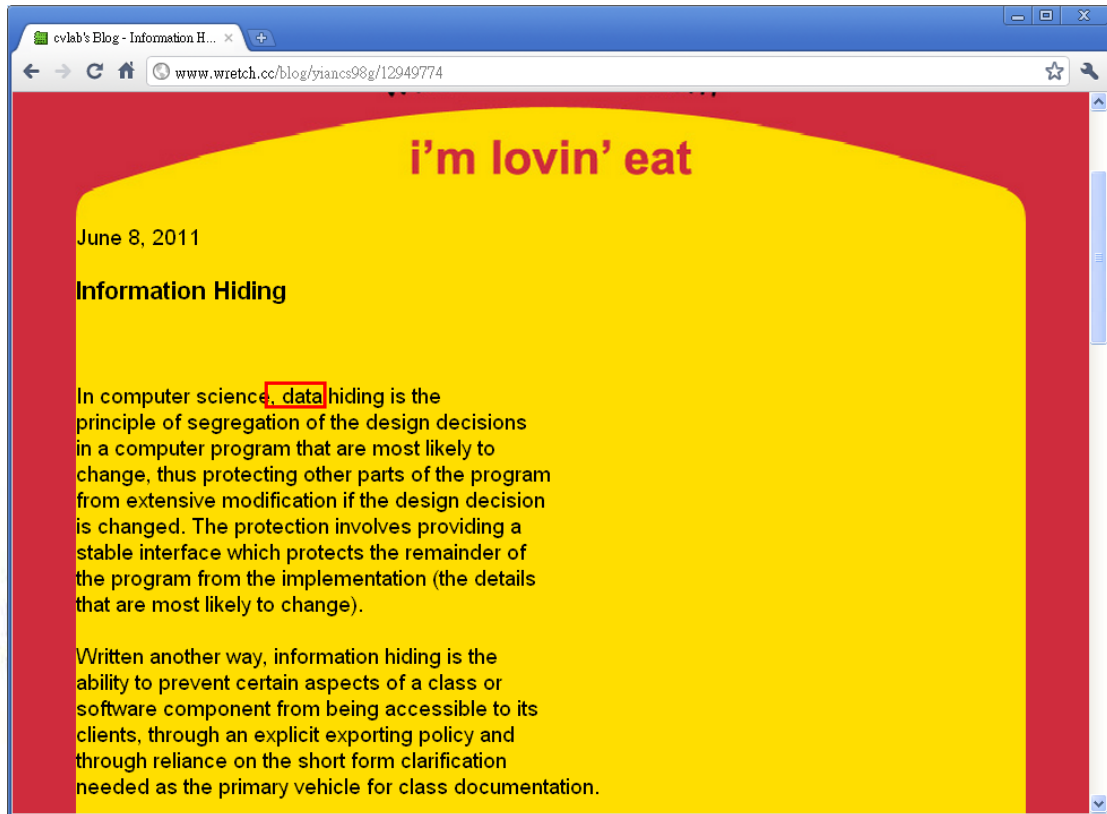


(c)

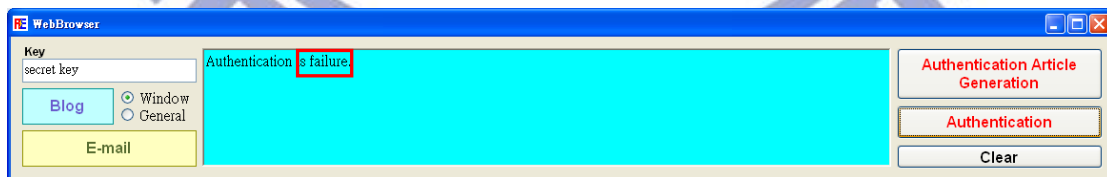


(d)

Figure 3.5 An example of experimental results (continued). (a) An original blog article. (b) A user interface used to generate authentication signal and protected blog article. (c) The protected blog article with an authentication signal embedded. (d) The authentication report with the message “Authentication is successful.” (e) A protected blog article with a tempered word. (f) The verification result of the tempered blog article.



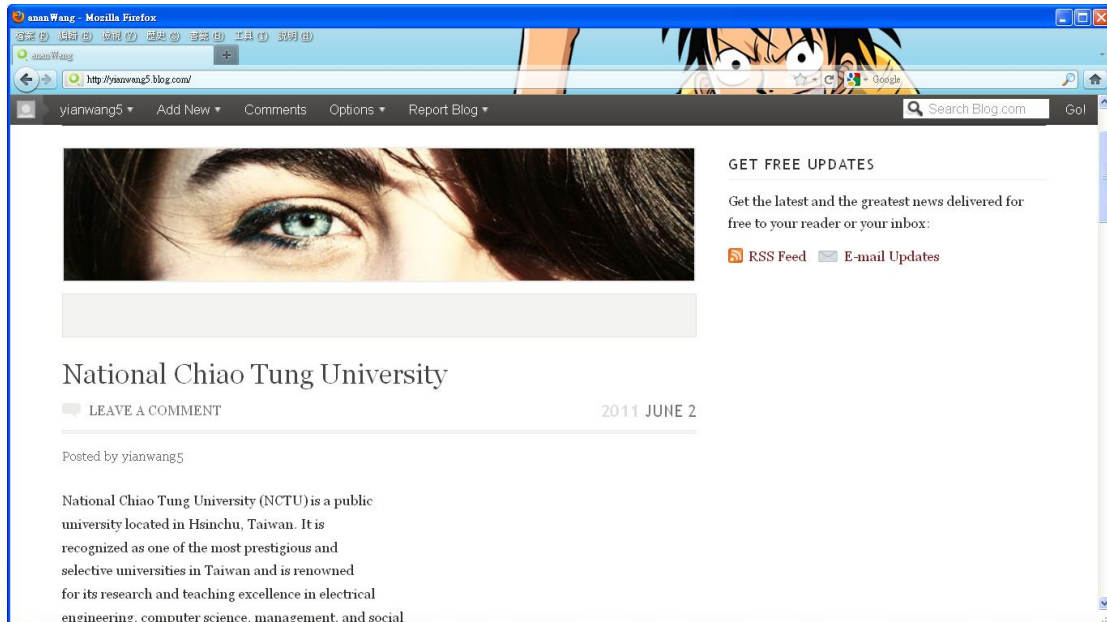
(e)



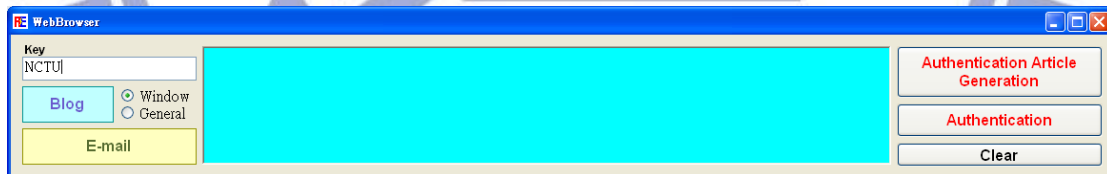
(f)

Figure 3.5 An example of experimental results (continued). (a) An original blog article. (b) A user interface used to generate authentication signal and protected blog article. (c) The protected blog article with an authentication signal embedded. (d) The authentication report with the message “Authentication is successful.” (e) A protected blog article with a tempered word. (f) The verification result of the tempered blog article.

“Authentication is successful.” (e) A protected blog article with a tempered word. (f) The verification result of the tempered blog article.

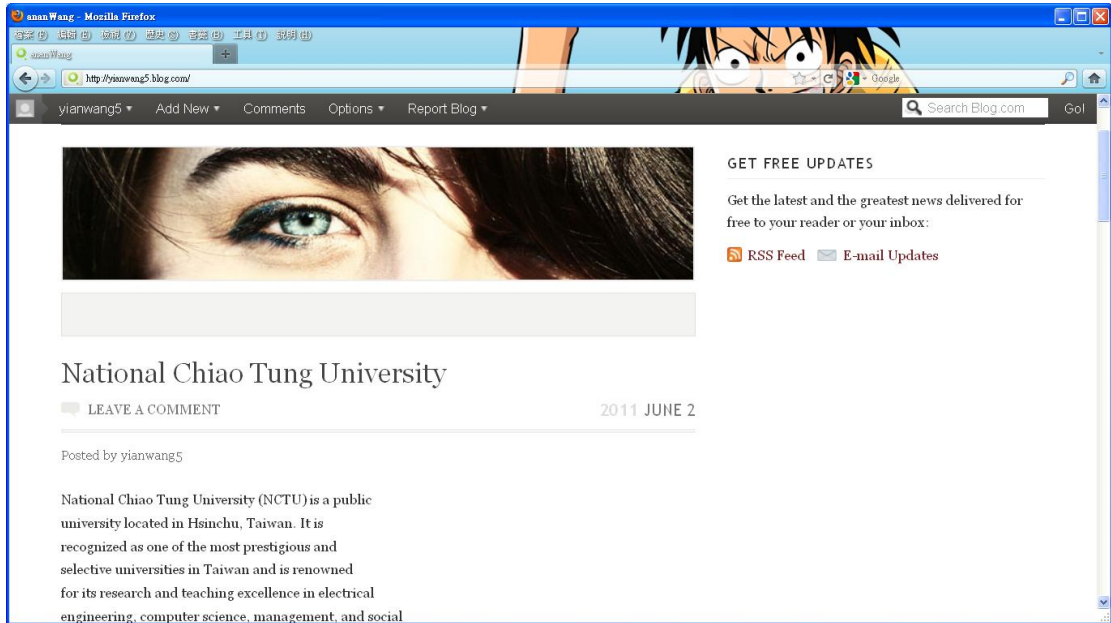


(a)

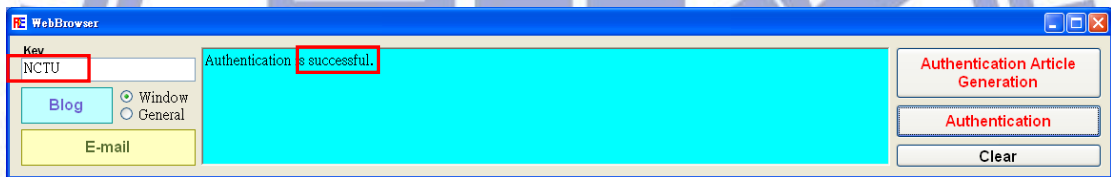


(b)

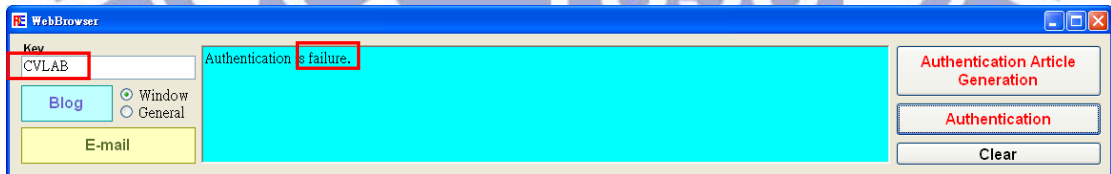
Figure 3.6 Another example of experimental results. (a) An original blog article. (b) The user interface with a secret key “NCTU”. (c) The protected blog article with an authentication signal embedded. (d) The authentication result by using the correct secret key. (e) The authentication result by using a wrong secret key.



(c)



(d)



(e)

Figure 3.6 Another example of experimental results (continued). (a) An original blog article. (b) The user interface with a secret key “NCTU”. (c) The protected blog article with an authentication signal embedded. (d) The authentication result by using the correct secret key. (e) The authentication result by using a wrong secret key.

Chapter 4

Covert Communication via the BBS

Using Special Big-5 Codes

4.1 Introduction and Problem Definition

4.1.1 Introduction

In this chapter, we will specifically introduce the proposed data hiding methods for covert communication via the BBS. The problem definition is described in the Section 4.1.2. And in Section 4.2, the basic ideas of the proposed methods are described. Detailed data embedding and extraction algorithms are presented in Sections 4.3 and 4.4, respectively. In Section 4.5, experimental results showing the feasibility of the methods are given. Lastly, we briefly summarize the work we have done in Section 4.6.

4.1.2 Problem Definition

The BBS (bulletin board system) is a popular interaction platform for discussions, entertainments, shopping, etc. Every day, numerous articles are published on BBS's. Thus, it is an appropriate channel for covert communication. Furthermore, BBS administrators have the supreme authority to read or delete any article and even read private mails or messages on the BBS, so covert communication via the BBS is not only appropriate but also necessary. The aim of this kind of covert communication is

to send secret messages through the articles published on the BBS without arousing suspicions of hackers. Accordingly, we develop two techniques for covert communication via the BBS by the use of special Big-5 codes in this study. They are introduced in the following sections.

4.2 Major Ideas of Proposed Methods by Use of Special Big-5 Codes

In this study, we propose two data hiding methods for covert communication via the BBS. One is to use *invisible* Big-5 codes; the other is to use special Big-5 *space* codes, and we generally call the two kinds of codes we use *special Big-5 codes*.

4.2.1 Data Hiding by Invisible Big-5 Codes

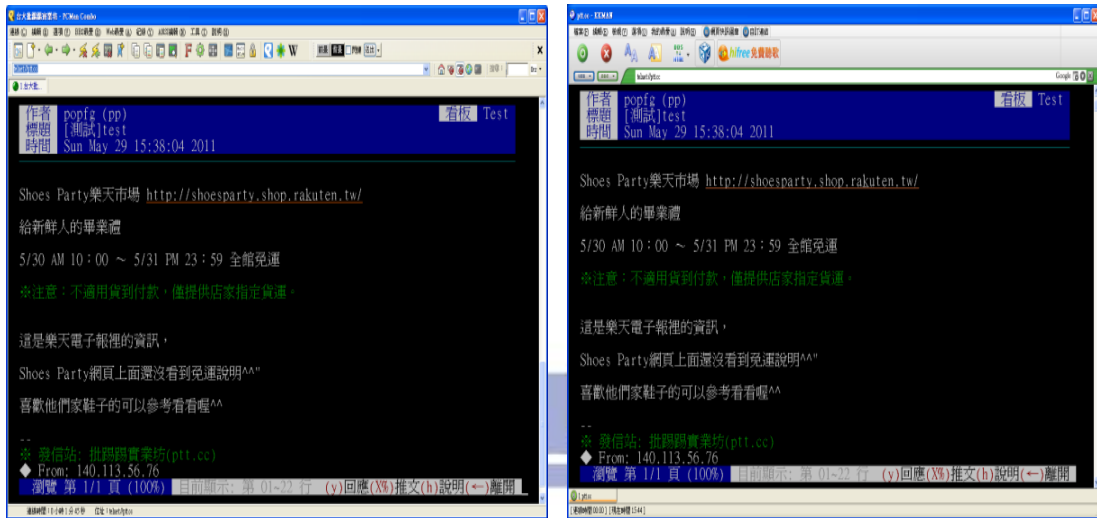
To achieve the goal of data hiding in BBS articles, at the beginning of this study we have tried the technique of using invisible ASCII codes mentioned previously in Chapter 3, because the ASCII codes are compatible with the Big-5 codes as the kernel set. However, invisible ASCII control codes are utilized to implement some system functions on the BBS, so we have to develop new data hiding technique.

The first proposed new data hiding method via the BBS is to use invisible Big-5 codes. In Taiwan, many BBS's like the PTT and the school BBS sites are built on the servers with the Big-5 coding format, so the proposed first technique is appropriate for them. Nowadays, most of the popular operating systems such as Windows XP and Windows 7 use the Unicode format as their text coding systems, because the Unicode is a universal and complete standard format. No matter what coding formats are used for text, they will be transformed into the appearance of the Unicode format by these

operating systems when they are displayed on the screen. Taking Windows XP as an example. In this operating system which contains many different conversion tables for transcoding between various text coding formats and the Unicode, all text with the Big-5 format on the BBS will be displayed on the screen with the Unicode format by referring to the *CodePage 950* which is a transcoding table between the Big-5 and the Unicode [17].

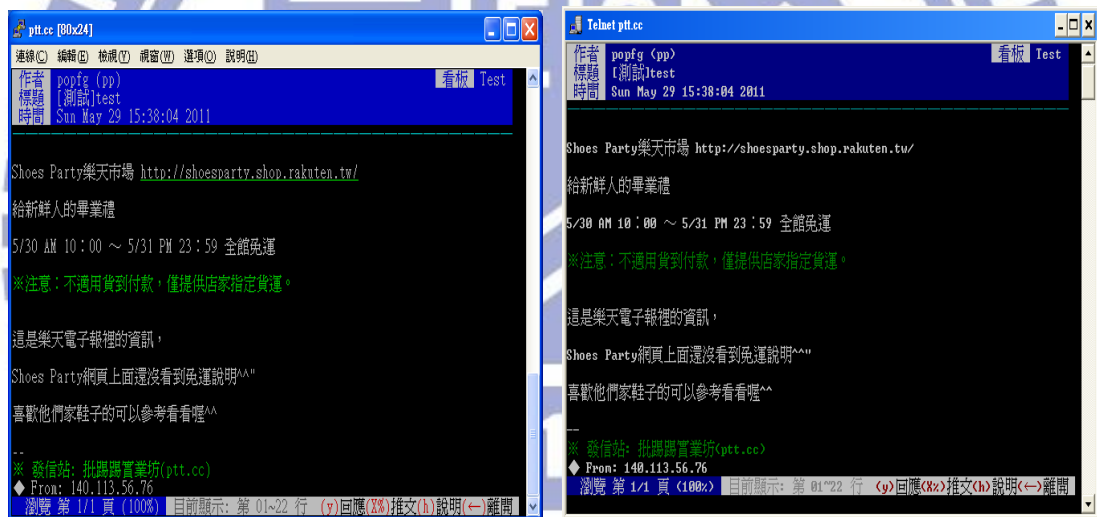
For this reason, we tried to find the mapping relationship between all Big-5 codes and Unicode codes, and discovered that some special Big-5 codes, which originally represent certain rarely-used Chinese characters or Japanese characters, are invisible, and look just like white spaces when these codes are transcoded into the Unicode format and displayed on the BBS. This phenomenon resulted from the fact that these corresponding Unicode codes are located in the Unicode *Private Use Area*, which ranges from code E000 to code E8FF and does not contain any character assignment so that no character code chart is provided for this area.

However, on some popular BBS browsers such as PCMan and Pietty, to facilitate users to read and type some special characters, certain above-mentioned special Big-5 codes are presented as their original appearances through the simulated *Unicode compensation plan* implemented by the BBS browser software. So, through continuous tests and observations in our experiments on popular BBS browsers including PCMan, KKMan, Pietty, and the basic telnet connection program provided by Windows XP, we have found 185 special Big-5 codes useful for our study, and we supplemented the 185 codes to a total of 256 symbols by padding a white space after each of the first 71 ones of them. The appearances of embedding some of these symbols in BBS articles on the above-mentioned browsers are shown in Figure 4.1. And the codes are listed in Table 4.1. Note that we have created an *end signal* which is composed of a special Big-5 code, FEAE, and the original white space.



(a)

(b)



(c)

(d)

Figure 4.1 Stego-articles with some embedded invisible Big-5 symbols displayed on (a) PCMan, (b) KKMan, (c) Pietty, and (d) the telnet connection program, respectively.

Table 4.1 Encoding table for used invisible Big-5 codes.

| | | | | | | | | | | | |
|----|-------|----|-------|----|-------|-----|-------|-----|-------|-----|-------|
| 0 | 8F 53 | 31 | 9C D9 | 62 | 9D BE | 93 | C8 A6 | 124 | FB 47 | 155 | FC AA |
| 1 | 90 F6 | 32 | 9C E4 | 63 | 9D BF | 94 | C8 A7 | 125 | FB 48 | 156 | FC AB |
| 2 | 90 F7 | 33 | 9C E7 | 64 | 9D C1 | 95 | C8 A8 | 126 | FB 4B | 157 | FC B8 |
| 3 | 90 F9 | 34 | 9D 5F | 65 | 9D C2 | 96 | C8 A9 | 127 | FB 4F | 158 | FC C8 |
| 4 | 90 FA | 35 | 9D 60 | 66 | 9D C3 | 97 | C8 AA | 128 | FB 50 | 159 | FC CD |
| 5 | 91 C7 | 36 | 9D 61 | 67 | 9D C4 | 98 | C8 AB | 129 | FB 54 | 160 | FC E0 |
| 6 | 91 C8 | 37 | 9D 62 | 68 | 9D C5 | 99 | C8 AC | 130 | FB 59 | 161 | FC EC |
| 7 | 91 CF | 38 | 9D 63 | 69 | 9D C6 | 100 | C8 AD | 131 | FB 61 | 162 | FC EE |
| 8 | 91 D0 | 39 | 9D 6A | 70 | 9D C7 | 101 | C8 AE | 132 | FB 62 | 163 | FD 44 |
| 9 | 91 D8 | 40 | 9D 6D | 71 | 9D CC | 102 | C8 AF | 133 | FB 68 | 164 | FD 4B |
| 10 | 91 D9 | 41 | 9D 6E | 72 | 9D D3 | 103 | C8 B0 | 134 | FB 71 | 165 | FD 78 |
| 11 | 91 DA | 42 | 9D 6F | 73 | 9D D4 | 104 | FA 45 | 135 | FB 74 | 166 | FD A8 |
| 12 | 91 DF | 43 | 9D 70 | 74 | 9D D6 | 105 | FA 46 | 136 | FB 75 | 167 | FD C2 |
| 13 | 91 E1 | 44 | 9D 78 | 75 | 9D DF | 106 | FA 47 | 137 | FB 77 | 168 | FD EA |
| 14 | 91 E2 | 45 | 9D 79 | 76 | 9D E0 | 107 | FA 48 | 138 | FB 79 | 169 | FE 46 |
| 15 | 91 E3 | 46 | 9D A1 | 77 | 9D E6 | 108 | FA 49 | 139 | FB 7B | 170 | FE 47 |
| 16 | 91 E4 | 47 | 9D A2 | 78 | 9D E8 | 109 | FA 4A | 140 | FB A4 | 171 | FE 55 |
| 17 | 91 E5 | 48 | 9D A3 | 79 | 9D E9 | 110 | FA 4C | 141 | FB AC | 172 | FE 58 |
| 18 | 91 E7 | 49 | 9D A4 | 80 | 9D EA | 111 | FA 4D | 142 | FB B1 | 173 | FE 59 |
| 19 | 91 EF | 50 | 9D A5 | 81 | 9D EB | 112 | FA 4E | 143 | FB B9 | 174 | FE 65 |
| 20 | 91 FD | 51 | 9D A6 | 82 | 9D EC | 113 | FA 50 | 144 | FB BB | 175 | FE 69 |
| 21 | 91 FE | 52 | 9D A7 | 83 | 9F 53 | 114 | FA 51 | 145 | FB C0 | 176 | FE 6B |
| 22 | 92 65 | 53 | 9D A8 | 84 | 9F 54 | 115 | FA 52 | 146 | FB DE | 177 | FE 6C |
| 23 | 98 FB | 54 | 9D A9 | 85 | A0 47 | 116 | FA 53 | 147 | FB E3 | 178 | FE 70 |
| 24 | 98 FD | 55 | 9D AA | 86 | A0 7D | 117 | FA 56 | 148 | FB EE | 179 | FE 72 |
| 25 | 98 FE | 56 | 9D AB | 87 | A0 A1 | 118 | FA 58 | 149 | FB F4 | 180 | FE 73 |
| 26 | 99 FB | 57 | 9D AC | 88 | A0 A6 | 119 | FA 59 | 150 | FB F9 | 181 | FE A5 |
| 27 | 9B DE | 58 | 9D AD | 89 | A0 FD | 120 | FA 5A | 151 | FC 4A | 182 | FE A9 |
| 28 | 9B DF | 59 | 9D AE | 90 | C8 7A | 121 | FA 5B | 152 | FC 50 | 183 | FE AB |
| 29 | 9B E3 | 60 | 9D AF | 91 | C8 A4 | 122 | FA 60 | 153 | FC 70 | 184 | FE AE |
| 30 | 9C D7 | 61 | 9D BD | 92 | C8 A5 | 123 | FA 63 | 154 | FC A3 | | |

(Note: codes No. 185 through 255 are generated by combining the first 71 special Big-5 codes with the original white space. And the *end signal* is composed of the special Big-5 code FE AE and the original white space.)





4.2.2 Data Hiding by Special Big-5 Space Codes

We have also proposed another data hiding technique for the BBS by the use of some special Big-5 *space* codes. In this method, two kinds of Big-5 codes are used, one being the original white space code and the other a Big-5 space code. Because the two codes are both included in the Big-5 standard, and appear to be invisible when they are displayed on BBS browsers, we can use them to achieve the aim of data hiding in the BBS under most general operating systems by assembling them in a proper order.

On the BBS, many users are accustomed to publishing articles with alternate blank lines and this habit facilitates us to hide secret messages after the *line feed* code of each line end. So, we tried to devise an appropriate scheme to efficiently utilize the two mentioned invisible codes for the largest utilization ratio of the blank spaces in each line.

More specifically, there are two kinds of character lengths on the BBS. One occupies a unit which is defined to be the same as the unit mentioned in Algorithm 3.1, like the original white space; and the other is two-unit long, like the special Big-5 space code. For the *variability* and *efficiency* of using the Big-5 symbols for data hiding, we allow them to be a special Big-5 code, a combination of a special Big-5 code and a white space, or a combination of a special Big-5 code and several white spaces, as shown in Table 4.2, which we mention as an encoding table for the used special Big-5 space codes. Here, *efficiency* is judged by the average required number of units for hiding one bit.

In our study, we only use four types (except the *end signal*) of symbols, including:

type 1: , type2: , type3: , and type4: .







Though we can create more types of symbols following the same symbol-creating logic by padding more white space codes after the symbol , yet we can prove that the 4-symbol codes created in Table 4.2 have the *largest* efficiency of symbols, as discussed next.

Table 4.2 Encoding table for used special Big-5 space codes.

| Bit stream (binary) | 00 | 01 | 10 | 11 | End signal |
|---------------------------------------|---|---|---|--|---|
| Special Big-5 codes (embedded symbol) |  |  |  |  |  |
| Occupied units | two | three | four | five | two |

(Note: : special Big-5 space code. : original white space code. **LF**: line feed code.)

Assume that the probability for each symbol to appear is identical, and that a symbol can represent n embedded bits. And let u_i and p_i specify the occupied units and the appearance probability of the i -th type of symbols like those defined in Table 4.2, respectively. Then, the *efficiency* E of the symbols is defined by the following equation:

$$E = \left(\sum_{i=1}^{2^n} u_i \times p_i \right) / n. \quad (1)$$

Also, under the assumption that all the symbols have equal appearance probabilities, we may substitute u_i and p_i in (1) above with their real values to obtain

$$E = f(n) = \left[2 \times \frac{1}{2^n} + 3 \times \frac{1}{2^n} + \dots + (2^n + 1) \times \frac{1}{2^n} \right] / n \quad (2)$$

which can be reduced easily to be

$$f(n) = (2^n + 3) / 2n \quad (3)$$

and differentiated to get

$$\frac{df(n)}{dn} = g(n) = [(2^{n+1}) \times n \times \ln 2 - 2^{n+1} - 6] / 4n^2. \quad (4)$$

Setting $g(n) = 0$ in (4) above results in the following equation:

$$(2^{n+1}) \times n \times \ln 2 - 2^{n+1} - 6 = 0 \quad (5)$$

which can be solved to get the extreme value for n . However, because the number of bits must be an integer and such an integer satisfying Eq. (5) is inexistent, we must take n to be 2 for $g(n)$ to be closest to zero. Alternatively, from Eq. (3) we have $f(n) = 5/2, 7/4$, and $11/6$ for $n = 1, 2$, and 3 , respectively. Since $5/2 \geq 7/4 \leq 11/6$, we see that $n = 2$ indeed is an optimal value to make $f(n)$ minimum, i.e., to yield the smallest average required number of units, $7/4$, for hiding one bit. This completes the proof that the 4-symbol codes (except the end signal) listed in Table 4.2 are optimal, yield the largest efficiency of coding.

Some examples of BBS articles in which secret messages are embedded by the proposed method are given in Figure 4.2. By the way, we created the end signal composed of a special Big-5 space code and a *line feed* code rather than a white space code and a *line feed* code, since all white spaces between any other code and the *line feed* will be removed when an article is published on the BBS.

Because secret messages embedded by the two proposed methods are almost imperceptible on the BBS even when a user highlights BBS articles by a mouse, we can use the methods to achieve the goal of covert communication on the BBS. The

detailed algorithms about embedding and extraction of the secret message are described specifically in the subsequent sections.

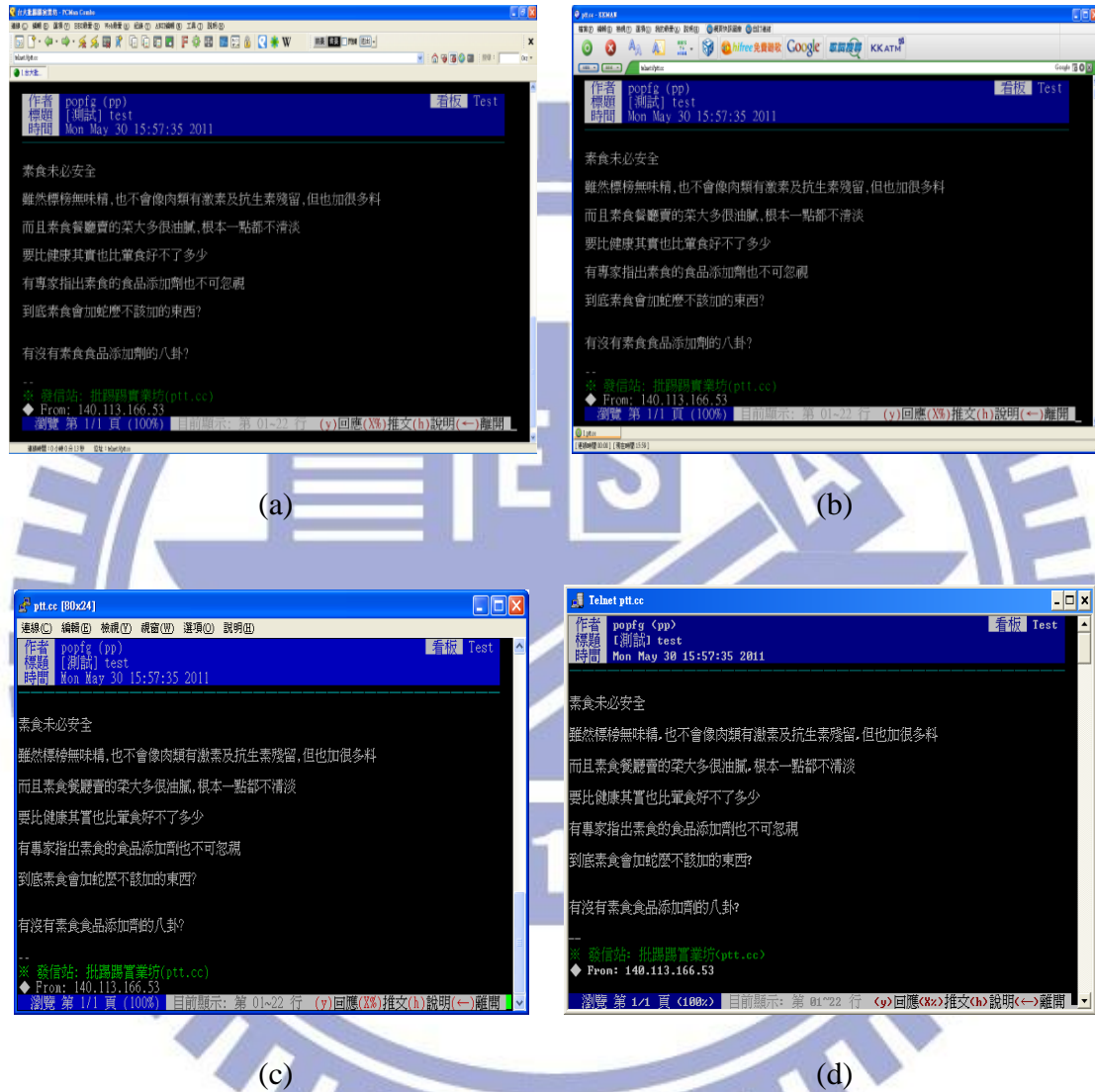


Figure 4.2 Stego-articles with secret messages embedded displayed on some well-known BBS's. (a) On PCMan. (b) On KKMan. (c) On Pietty. (d) On the telnet connection program.

4.3 Proposed Algorithm for Data Embedding

In this section, first the process of covert communication via the BBS by using the two proposed methods is illustrated by a flow chart shown in Figure 4.3. In the process, first we fold longer article lines into shorter ones, leaving at least eight characters at each line end as a *data embedding slot*. Next, we use a secret key as a seed to randomize the content of a secret message which we want to embed in a cover article for covert communication. Then, we map the randomized message to corresponding invisible symbols according to the user's choice. If method 1 as mentioned previously is chosen, we conduct the mapping by referring to Table 4.1; otherwise, when method 2 is chosen, we replace each special Big-5 space code in the cover article with two original white space codes so that the process of data extraction in the next section can be performed correctly, and conduct the mapping by referring to Table 4.2. Finally, we sequentially embed the symbols obtained from the mapping into the folded article to obtain a stego-article with the randomized secret message hidden imperceptibly.

The algorithm for conducting this process is described in the following, in which *a line* in a BBS article means a number of characters in a row with an LF appended to the end of the line.

Algorithm 4.1 *Data embedding for covert communication.*

Input: a secret message S , a secret key K , and a cover BBS article B .

Output: a stego-article B' .

Steps.

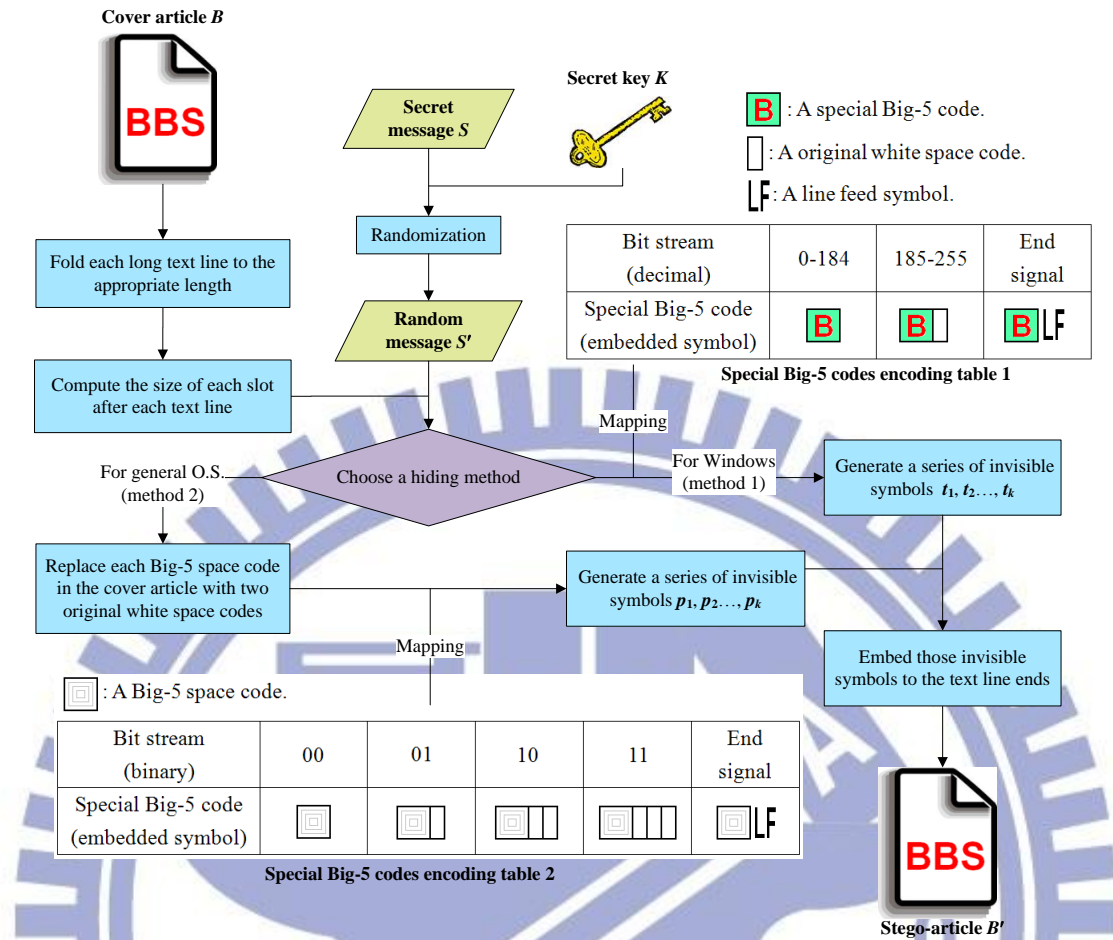


Figure 4.3 Flow chart of proposed process of embedding secret messages.

1. Fold sequentially each text line l_i with a length larger than 70 units (with a unit meaning the length of an ASCII code displayed on the BBS) in BBS article B into a 70-unit line by inserting a line feed, denoted as LF and occupying zero unit, after the original 70th character in l_i to generate a folded article, denoted as F .
2. Compute the size L_i of the data embedding slot at the end of each text line l_i in F by:

$$L_i = 78 - \text{the length of } l_i,$$

which means that the maximum number of characters that can be inserted at the end of l_i .

3. Use the secret key K as a seed to generate a sequence Q of random numbers.
4. Randomize the input secret message S with Q to get a *randomized message* S' .
5. Choose a method to hide S' :
 - (1) If method 1 is chosen, then perform Step 6.
 - (2) If method 2 is chosen, then go to Step 9.
6. (*Method 1*) Separate the bits of S' into 8-bit segments and map them to invisible symbols t_1, t_2, \dots, t_k , respectively, according to Table 4.1.
7. Let $|l|$ be the total number of lines, $|T|$ be the total number of t_1 through t_k , and Ut_1, Ut_2, \dots, Ut_k be the numbers of units occupied by t_1 through t_k , respectively.
8. Embed the invisible symbols obtained in Step 6 sequentially into the folded article F from the first line (that is, take the index number i of l_i and the index number k of t_k both to be 1 initially), and then conduct the following steps.
 - 8.1 If $i \leq |l|$, then perform one of the following three operations at the end of l_i ; otherwise, perform Step 8.2.
 - (1) If $k \leq |T|$ and $L_i - Ut_k > 2$, then embed t_k in the data embedding slot of l_i , decrement L_i by Ut_k , increment k by 1, and repeat Step 8.1 again.
 - (2) If $k \leq |T|$ and $L_i - Ut_k \leq 2$, then scan l_i to find the line feed LF, remove it, embed an end signal in the data embedding slot of l_i , increment i by 1, and repeat Step 8.1 again.
 - (3) If $k > |T|$, then embed an end signal in the data embedding slot of l_i , and go to Step 13.
 - 8.2 Embed the remaining symbol/symbols below F as one or more blank lines with an end signal appended at each line end, and go to Step 13.
9. (*Method 2*) Replace each special Big-5 code in the folded article F with two white space codes.
10. Separate the bits of S' into 2-bit segments and map them to invisible symbols $p_1,$

p_2, \dots, p_k , respectively, according to Table 4.2.

11. Let $|l|$ be the total number of lines, $|P|$ be the total number of p_1 through p_k , and Up_1, Up_2, \dots, Up_k be the numbers of units occupied by p_1 through p_k , respectively.
12. Embed the invisible symbols sequentially into the folded article F from the first line (that is, take the index number i of l_i and the index number k of p_k both to be 1 initially), and then conduct the following steps.
 - 12.1 If $i \leq |l|$, then perform one of the following three operations at the end of l_i ; otherwise perform Step 12.2.
 - (1) If $k \leq |P|$ and $L_i - Up_k > 2$, then embed p_k in the data embedding slot of l_i , decrement L_i by Up_k , increment k by 1, and repeat Step 12.1 again.
 - (2) If $k \leq |P|$ and $L_i - Up_k \leq 2$, then scan l_i to find the line feed LF, remove it, embed an end signal in the data embedding slot of l_i , increment i by 1, and repeat Step 12.1 again.
 - (3) If $k > |P|$, then embed an end signal in the data embedding slot of l_i , and perform Step 13.
 - 12.2 Embed the remaining symbol/symbols below F as one or more blank lines with an end signal at each line end, and continue.
13. Take the final version of F as the desired stego-BBS article B' .

4.4 Proposed Algorithm for Data Extraction

In this section, we will specifically introduce the process for extraction of secret data. A flow chart of the process is shown in Figure 4.4. First, we extract the invisible symbols embedded in a stego-article. Next, according to the adopted different

methods of embedding the invisible symbols, we conduct different processes. If method 1 is used, we map the symbols into 8-bit segments by referring to Table 4.1; otherwise, when method 2 is used, we map the symbols into 2-bit segments by referring to Table 4.2. Then, we concatenate the segments into a random message. Finally, by using the same key which is used for embedding the message, we can recover the correct secret message. The detailed algorithm for extraction of the secret message is described in the following.

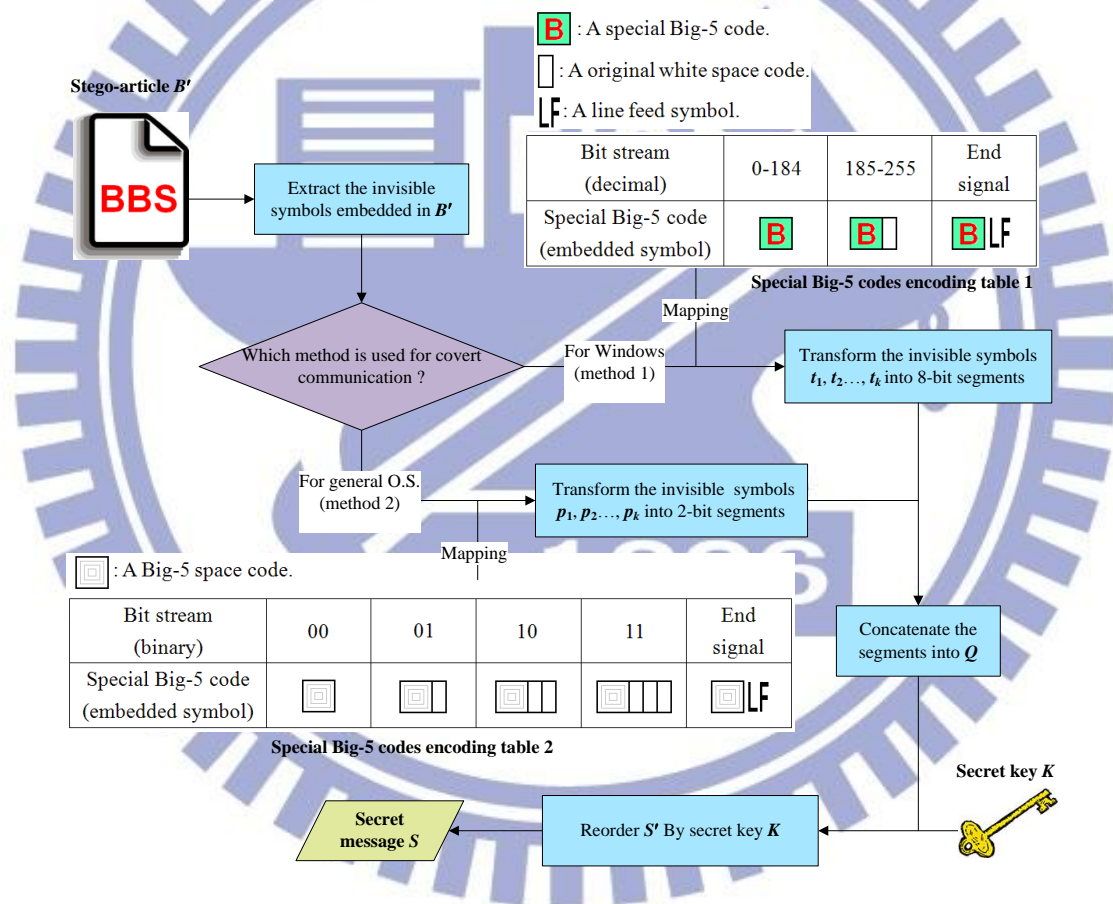


Figure 4.4 Flow chart of proposed data extraction process

Algorithm 3.2 Data extraction for covert communication.

Input: a stego-BBS article B' and the secret key K used in Algorithm 3.1.

Output: a secret message S .

Steps.

1. Check each line l_i in the stego-BBS article B' sequentially, starting from the first line; and extract the invisible symbols embedded in front of the end signal in l_i .
2. Transform the extracted symbols according to the different method used for embedding the secret message to be extracted.
 - (1) If method 1 is used, map them into 8-bit segments t_1, t_2, \dots, t_k , respectively, by referring to Table 4.1.
 - (2) If method 2 is used, map them into 2-bit segments p_1, p_2, \dots, p_k , respectively, by referring to Table 4.2.
3. Concatenate the extracted segments into a random message Q .
4. Use the secret key K to reorder Q to obtain a result as the desired secret message S .

4.5 Experimental Results

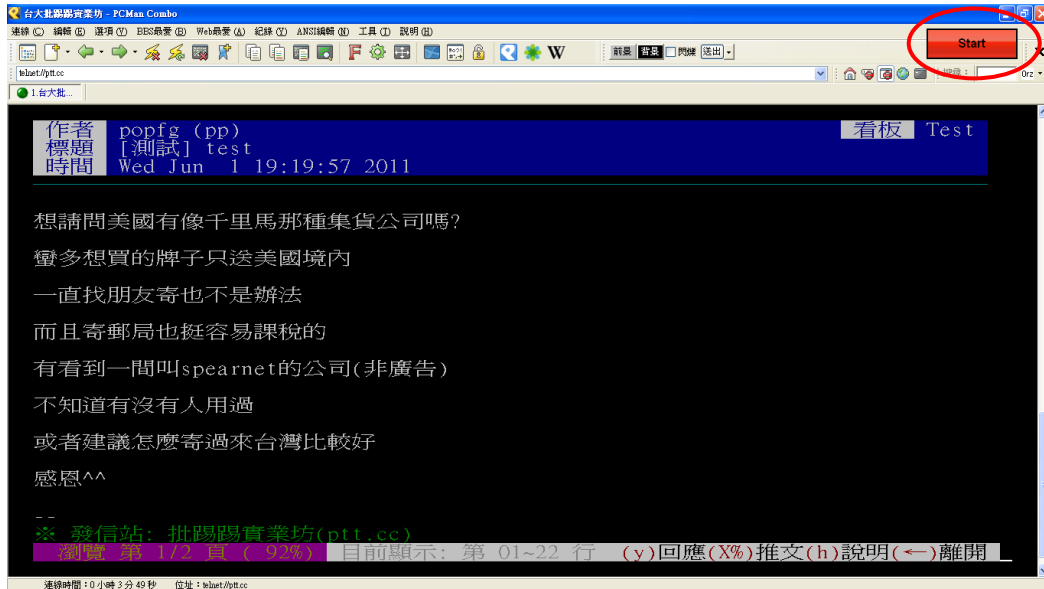
A series of experiments have been conducted to test the proposed algorithms for covert communication via the BBS under the popular software environments of PCMan, KKMan, Piety, and the operating system of the traditional Chinese version of Microsoft Windows XP, service pack 3, 2002. In the following, we show some experimental results.

A normal BBS article displayed on the PCMan is shown in Figure 4.5(a), and shown at the top right of this figure is a *start-button* generated by our program designed for covert communication. When we generally surf the BBS articles, the program generates the small start-button first. However, when we want to publish a stego-article for covert communication, it can be expanded to be a bar for writing a secret key and a secret message, as shown in Figure 4.5(b). Specifically, we just have

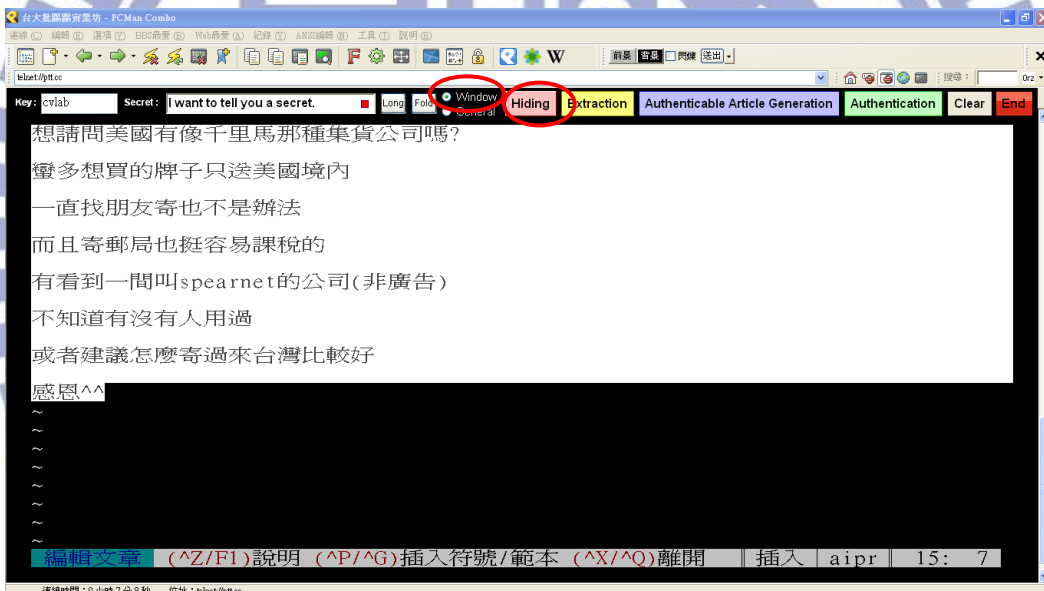
to select a hiding method, highlight a cover article by a mouse, and press the *hiding-button* also shown in Figure 4.5(c). In this way, we obtained a stego-BBS article with a secret message embedded, and the appearance of the stego-article is shown in Figure 4.5(d). Later, we extracted the secret message by typing the correct secret key, selecting the same method, highlighting the stego-article, and pressing the *extraction-button* as shown in Figure 4.5(e). On the other hand, as shown in Figure 4.5(f), when the typed secret key was wrong, the correct secret message was obtained. Another example of our experimental results is shown in Figure 4.6.

4.6 Summary

In this chapter, two new methods of data hiding using special Big-5 codes in BBS articles have been proposed for covert communication. One is appropriate for the operating systems with the Unicode standard as the kernel set and using the *CodePage 950* as their transcoding table between the Big-5 and the Unicode; and the other is appropriate for most general operating systems. And both were implemented completely in this study. The secret message hidden in a BBS article is not easy to be observed from the appearance. Even through a malicious user knows the proposed algorithms and tries to extract the secret from a stego-BBS article, the secret message can still be protected by a secret key. Furthermore, all experimental results show the feasibility of the proposed methods.

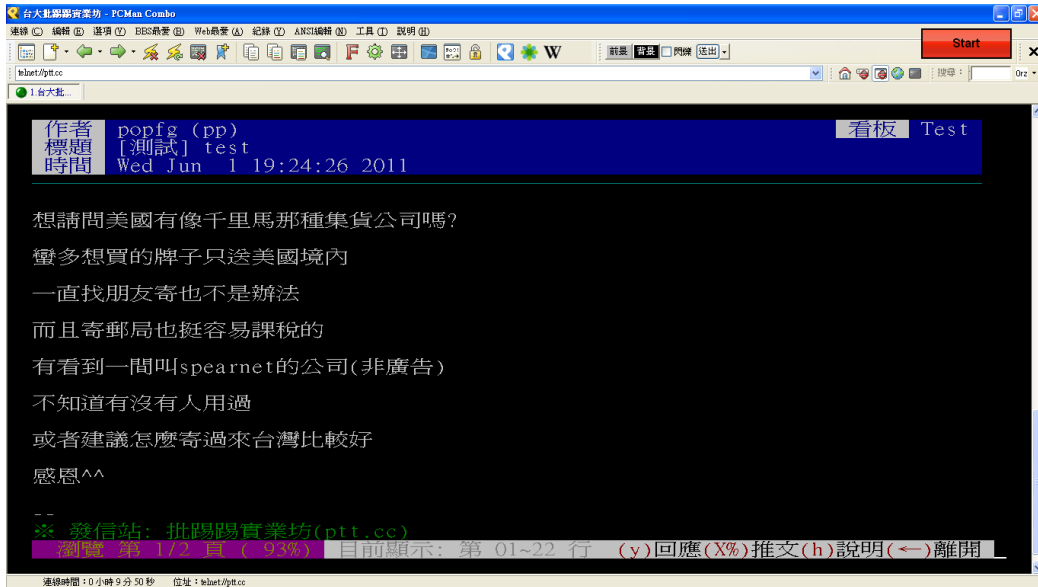


(a)

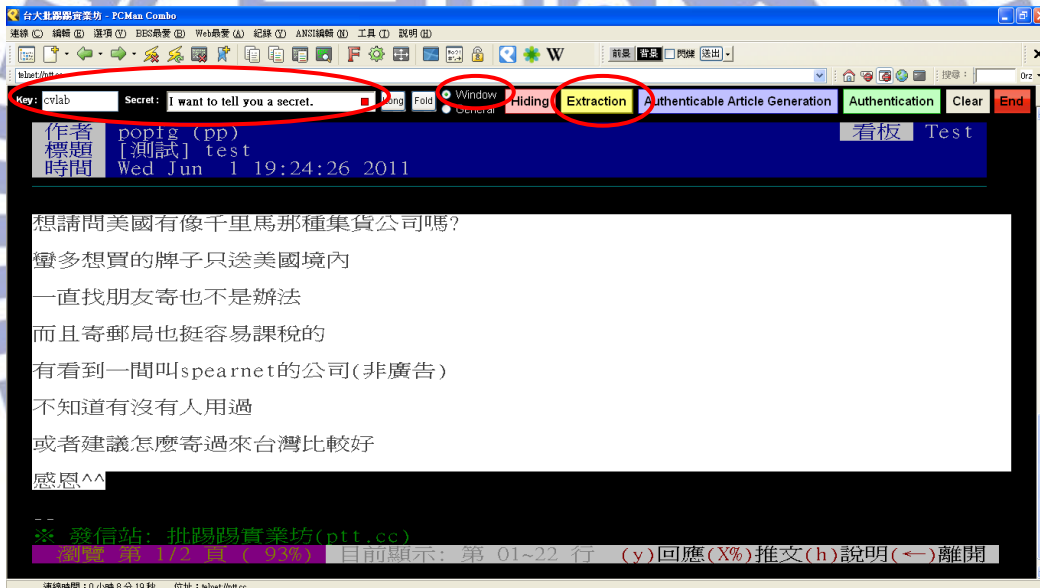


(b)

Figure 4.5 An example of experimental results. (a) A normal article displayed on the PCMan with our program in the upper right. (b) Data embedding process: type a secret key and a secret message, select a hiding method, highlight a cover article, and press the *hiding-button* to generate a stego article with the secret message embedded. (c) The displayed stego-article with the secret message embedded on the PCMan. (d) Data extraction process: extract the secret message by the use of using the correct secret key, select the same method, and press the *extraction-button*. (e) Result of using a wrong key to extract the secret message. (f) An extracted wrong message.

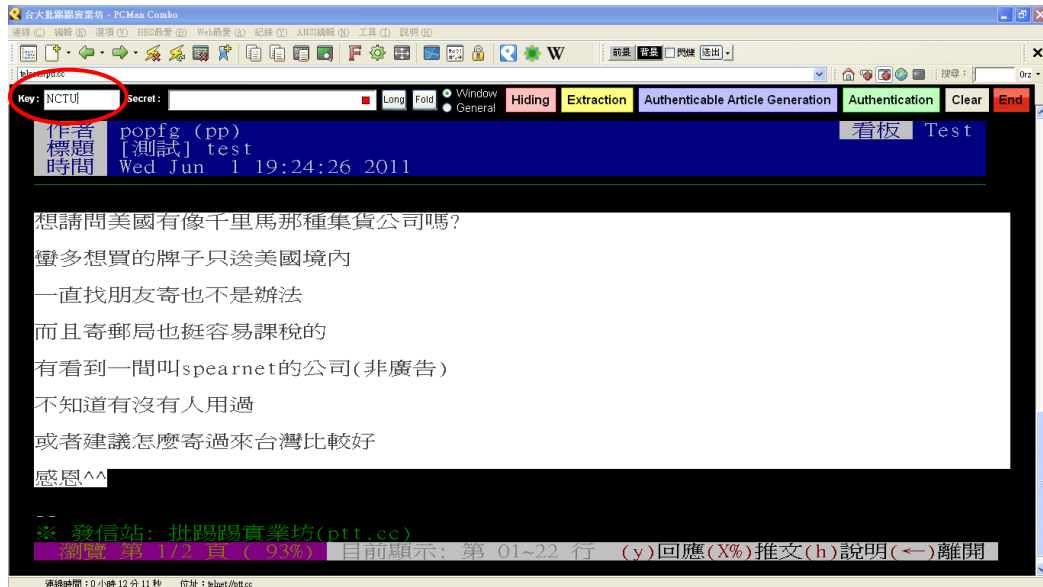


(c)

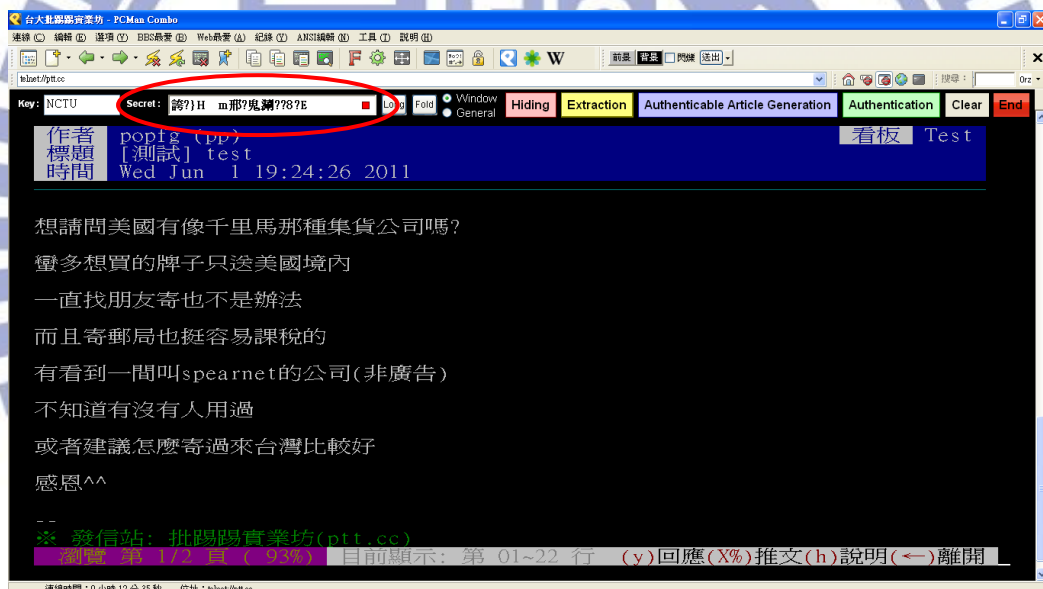


(d)

Figure 4.5 An example of experimental results (continued). (a) A normal article displayed on the PCMan with our program in the upper right. (b) Data embedding process: type a secret key and a secret message, select a hiding method, highlight a cover article, and press the *hiding-button* to generate a stego article with the secret message embedded. (c) The displayed stego-article with the secret message embedded on the PCMan. (d) Data extraction process: extract the secret message by the use of using the correct secret key, select the same method, and press the *extraction-button*. (e) Result of using a wrong key to extract the secret message. (f) An extracted wrong message.

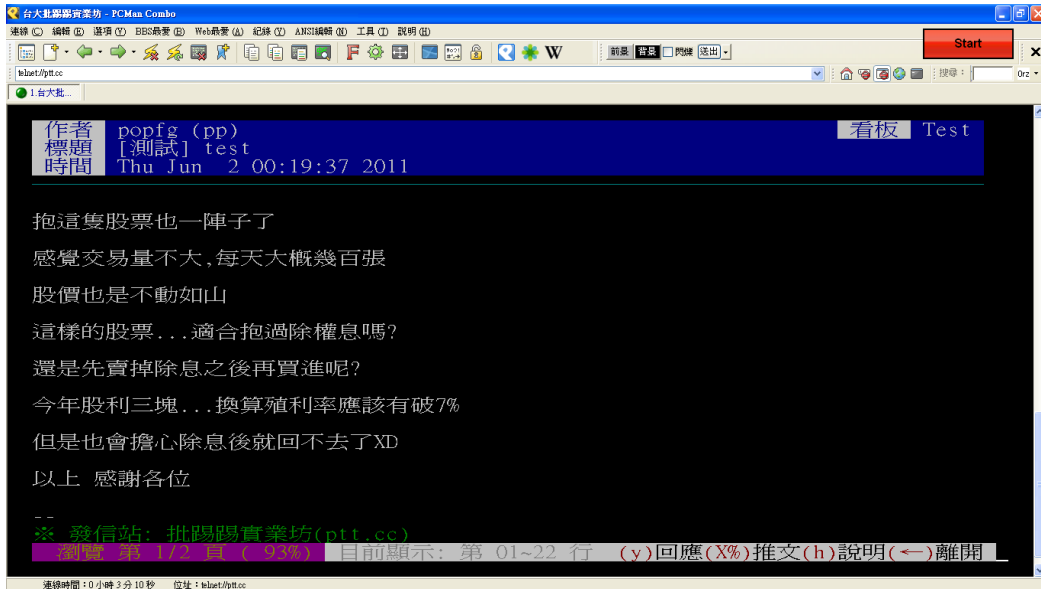


(e)

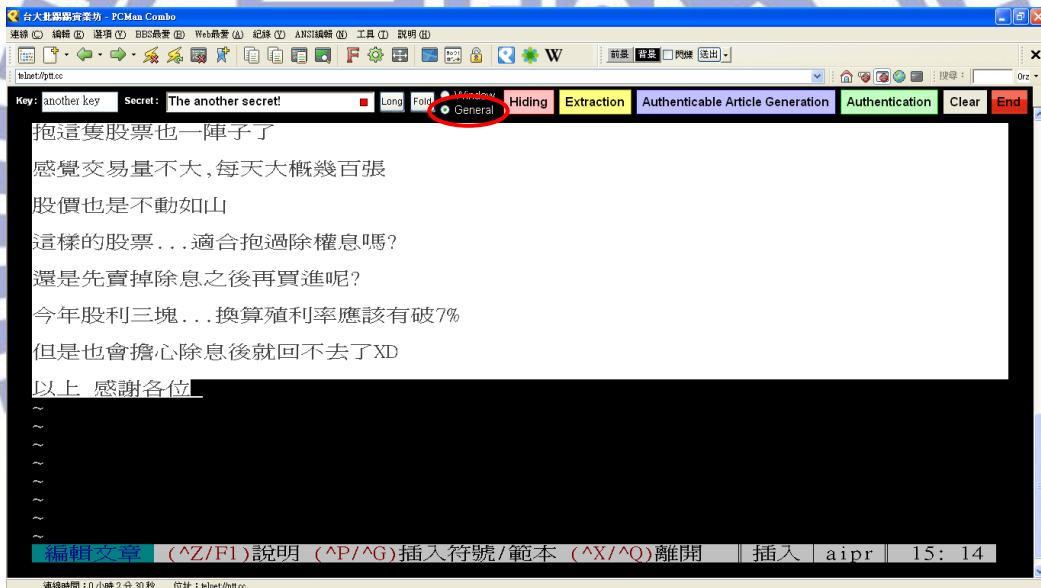


(f)

Figure 4.5 An example of experimental results (continued). (a) A normal article displayed on the PCMan with our program in the upper right. (b) Data embedding process: type a secret key and a secret message, select a hiding method, highlight a cover article, and press the *hiding-button* to generate a stego article with the secret message embedded. (c) The displayed stego-article with the secret message embedded on the PCMan. (d) Data extraction process: extract the secret message by the use of using the correct secret key, select the same method, and press the *extraction-button*. (e) Result of using a wrong key to extract the secret message. (f) An extracted wrong message.

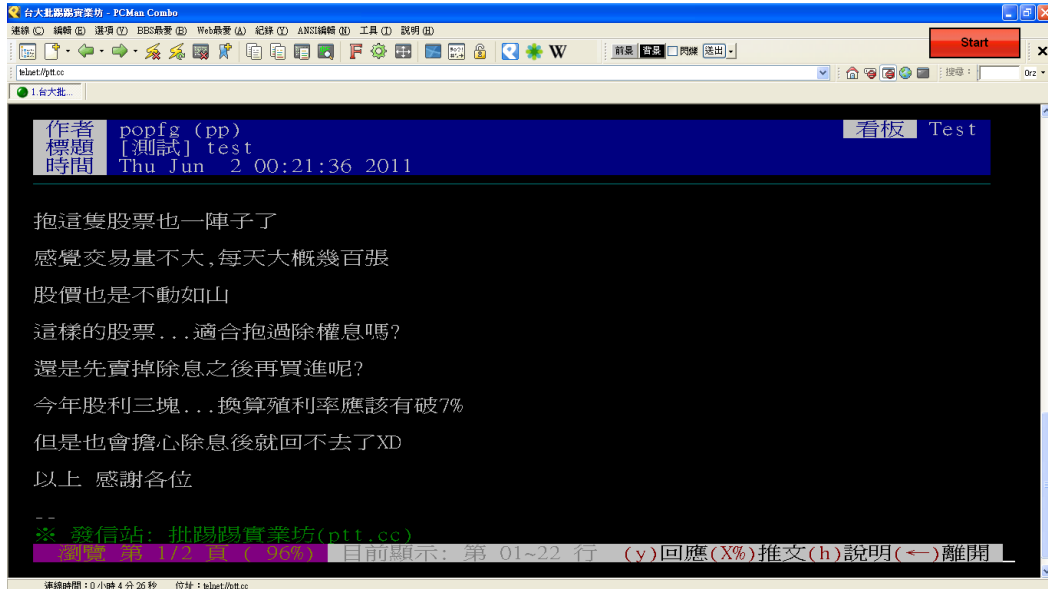


(a)

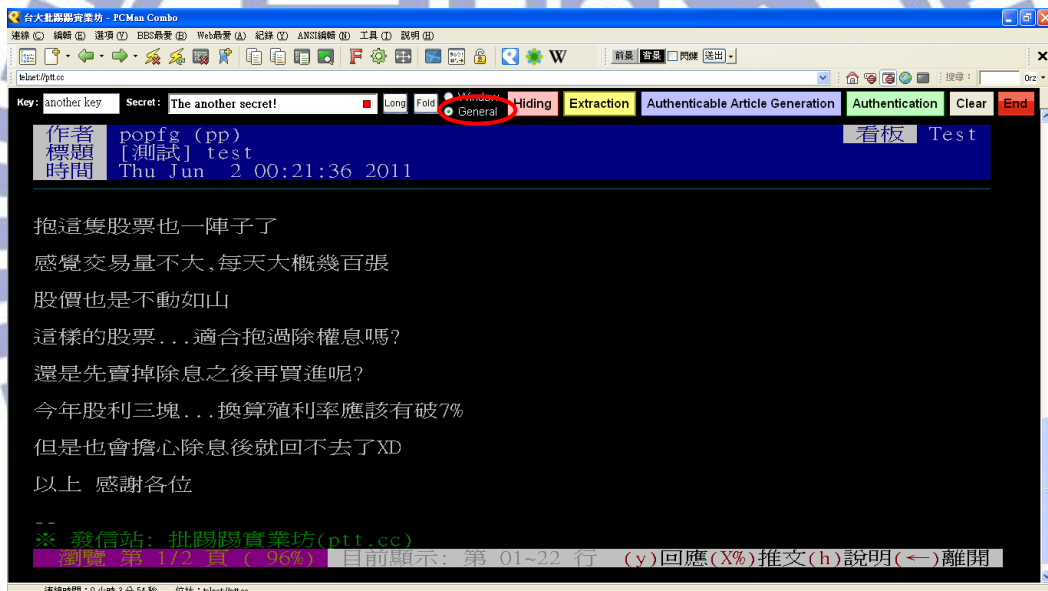


(b)

Figure 4.6 Another example of experimental results. (a) Another normal article. (b) Embedding a secret message by method 2, using special Big-5 space codes. (c) Stego-article with the secret message embedded. (d) Extracted correct secret message.



(c)



(d)

Figure 4.6 Another example of experimental results (continued). (a) Another normal article. (b) Embedding a secret message by method 2, using special Big-5 space codes. (c) Stego-article with the secret message embedded. (d) Extracted correct secret message.

Chapter 5

BBS Article Authentication by Special Big-5 Codes

5.1 Introduction and Problem Definition

5.1.1 Introduction

In this study, we also studied the topic of BBS article authentication, and the detail is described in this chapter. The problem definition is given in Section 5.1.2, and the major idea of the proposed method for BBS article authentication is given in Section 5.2. In Sections 5.3 and 5.4, the processes for generating a protected BBS article and for verifying the integrity of it are described, respectively. In Section 5.5, we show some experimental results to prove the feasibility of the proposed methods. At last, we give a brief summary for this chapter in Section 5.6.

5.1.2 Problem Definition

Through the technique of covert communication proposed in the previous chapter, we are able to prevent private messages from being browsed by illicit users on the BBS. However, because the user ID and password are unsafe on the Internet, hackers or malicious users may crack passwords to tamper with the contents of BBS articles, resulting in unpredictable consequences. Moreover, for BBS administrators, in their jurisdictions, they can read or delete all the articles and mails, and even

tamper with the articles or send fake mails, on the BBS. Thus, authentication of BBS articles is also a necessary technique for BBS applications.

5.2 Major Idea of Proposed Method by Use of Special Big-5 Codes

In this study, we reach the aim of BBS authentication by the previously-mentioned data hiding techniques which use special Big-5 space codes. In the previous chapter, we proposed two methods for covert communication via BBS articles. Covert communication is a technique which needs higher imperceptibility, and the two methods proposed previously both can accomplish this requirement. However, for article authentication, the required imperceptibility of an authenticated message is relatively low; even if the message is discovered illegally, any tampering with the message content will still result in a wrong verification result. Thus, we also can use the two methods to achieve the goal of BBS authentication.

5.3 Authentication Signal Generation and Embedding Process

In this section, we specifically describe the proposed process of generating a protected BBS article. After an authentication signal of a BBS article is obtained, we can regard it as a secret message and embed it in the original article by Algorithm 4.1 described previously in Chapter 4. A flow chart of the proposed authentication signal generation process is given in Figure 5.1. First, we fold the longer text lines in a BBS article, which we want to protect, into shorter ones, leaving at least eight characters at the end of each line as a *data embedding slot*. Then, if method 2 mentioned in the last

chapter, which uses special Big-5 space codes, is selected to hide data, we replace each Big-5 space code in the cover article with two white space codes. Next, we remove from the folded BBS article all the *line feed* signals so that the verification process described in the next section will not be interfered by redundant line feed signals. The modified BBS article and a secret key then are used to generate an authentication signal using a hash function and the exclusive-OR operation. Finally, we can regard the signal as a secret message and hide it in the folded article using the proposed data embedding process in Algorithm 4.1 with the same secret key to obtain a protected BBS article. The detailed algorithm is described in the following.

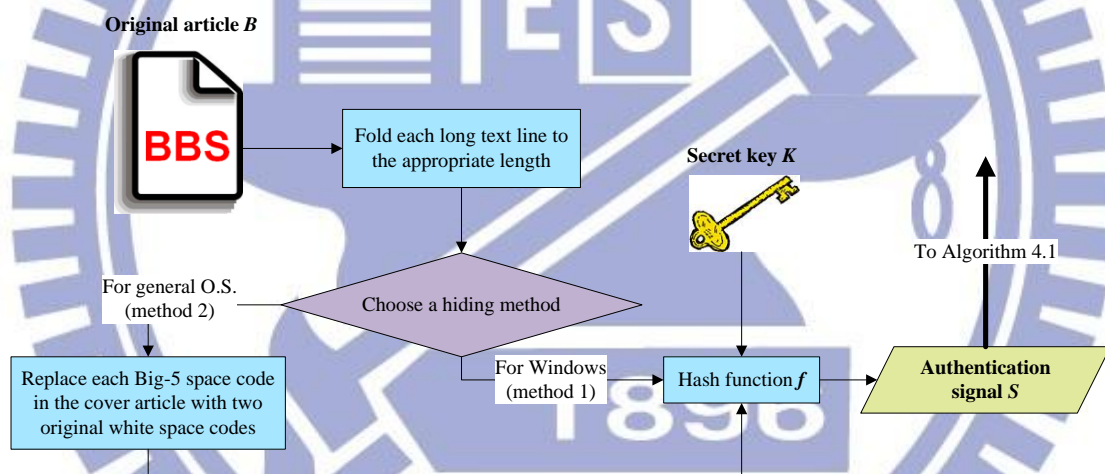


Figure 5.1 Flowchart of proposed authentication signal generation process.

Algorithm 5.1 Authentication signal generation and embedding process.

Input: a secret key K , a hash function f (such as MD5), and a cover BBS article B .

Output: a protected BBS article B' .

Steps.

1. Fold sequentially each text line l_i with a length larger than 70 units (with a unit meaning the length of an ASCII code displayed on the BBS) in BBS article B into

a 70-unit line by inserting a line feed, denoted as LF and occupying *zero unit*, after the original 70th character in l_i to generate a *folded article*, denoted as F .

2. Choose one of the two previously-described hiding methods to embed secret data in the folded article by one of the following ways:
 - (1) if method 1 is selected, then perform Step 4;
 - (2) if method 2 is selected, then perform Step 3.
3. Replace each special Big-5 code in the folded article F with two white space codes.
4. Remove all the line feed signals in F , use the result and the secret key K as inputs to the hash function f to generate two 128-bit digests F' and K' , respectively, and return all the removed LF signals back into their original positions in F .
5. Compute the exclusive-OR value $F' \oplus K'$ to obtain a 128-bit authentication signal S .
6. Regard S as a secret message and embed it in the cover article using the proposed data hiding process in Algorithm 4.1 to obtain a protected BBS article as output.

5.4 Authentication Signal Extraction and Verification Process

The detail of the proposed authentication signal verification scheme is described in this section. First, we use a protected BBS article as input to the proposed data extraction process in Algorithm 4.2 with the secret key used in Algorithm 5.1 to obtain a secret message, and regard it as the authentication signal S of the article. Next, we can use the same key and hash function to transform the BBS article, after all the secret symbols and the line feed signals in it are removed, into a verification signal T . Finally, we decide whether the protected blog article has been modified or not by

comparing the two signals S and T . A flow chart for the process is illustrated in Figure 5.2, and the detailed algorithm is given in the following.

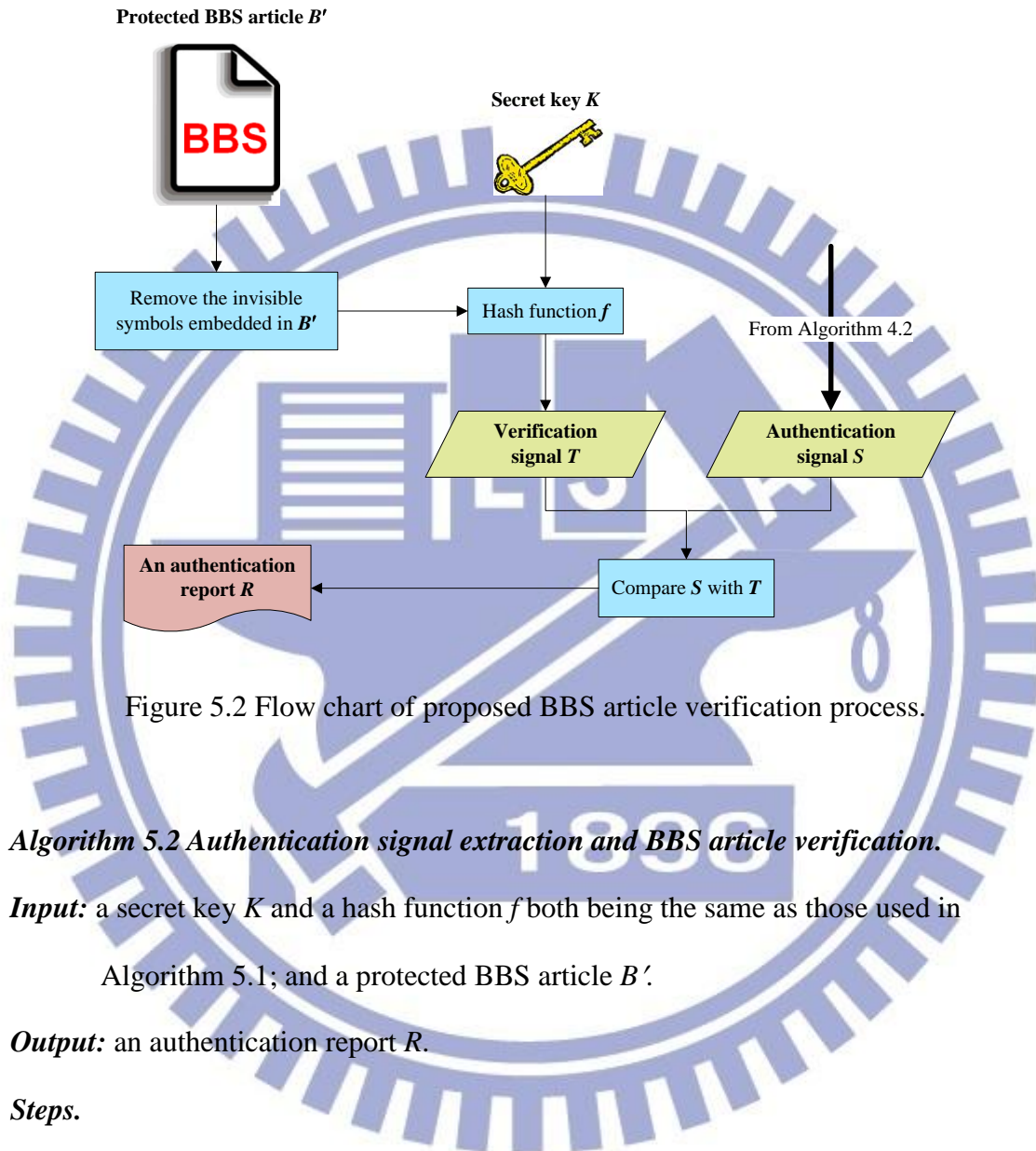


Figure 5.2 Flow chart of proposed BBS article verification process.

Algorithm 5.2 Authentication signal extraction and BBS article verification.

Input: a secret key K and a hash function f both being the same as those used in Algorithm 5.1; and a protected BBS article B' .

Output: an authentication report R .

Steps.

1. Use the protected BBS article B' as input to the proposed data extraction process in Algorithm 4.2 with the secret key K to obtain a secret message, and regard it as the authentication signal S of the article.
2. Remove all the secret symbols and line feed signals from the BBS article, and use the result, as an input to the hash function f to generate a 128-bit digest B'' .

3. Use the secret key K as an input to the hash function f to generate a 128-bit digest K' .
4. Compute the exclusive-OR value $B' \oplus K'$ to get a 128-bit verification signal T .
5. Compare the authentication signal S and T , resulting in the following two cases.
 - (3) If $S = T$, then regard the input B' as *unmodified* and mark it so in the authentication report R .
 - (4) If $S \neq T$, then regard B' as *modified* and mark it so in R .
6. Output the authentication report R .

5.5 Experimental Results

We have conducted experiments to prove the feasibility of the proposed BBS authentication scheme under the same software environments used in Chapter 4. In the following, we will illustrate some examples of our experimental results.

As shown in Figure 5.3(a), we generated a protected BBS mail by the use of a secret key, and its appearance displayed on the PCMan is shown in Figure 5.3(b). Later, a receiver of the mail verified whether a BBS mail is a fake or not by using the same secret key, as shown in Figure 5.3(c). However, if the key is wrong, then the authentication result will fail as shown in Figure 5.3(d).

In another example, we generated and published a protected BBS article like Figure 5.4(a) and its appearance is shown in Figure 5.5(b). The correct verification result by using the right secret key is shown in Figure 5.5(c). However, as shown in Figure 5.5(d), we tampered with the protected article by replacing a number “500” of the content with another number “900,” so that we obtained a wrong authentication result. Other examples of our experimental results are given in Figures 5.6 and 5.7, in which normal articles and protected articles are displayed on the KKMan and the

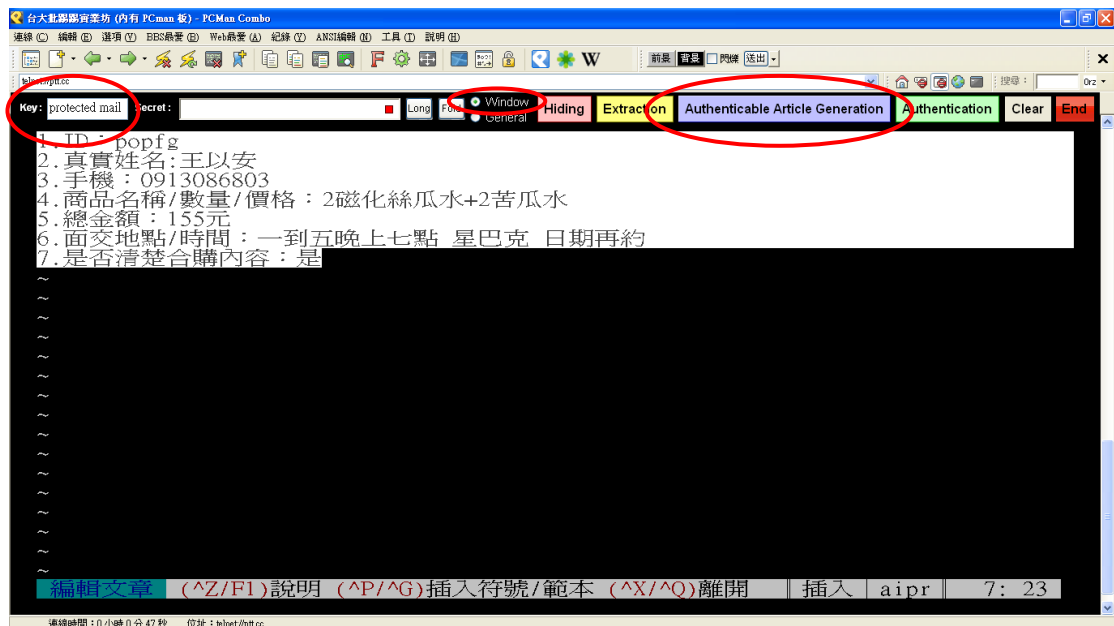
telnet connection program, respectively.

From these experimental results, it can be seen that the proposed method indeed can be used to authenticate BBS articles.

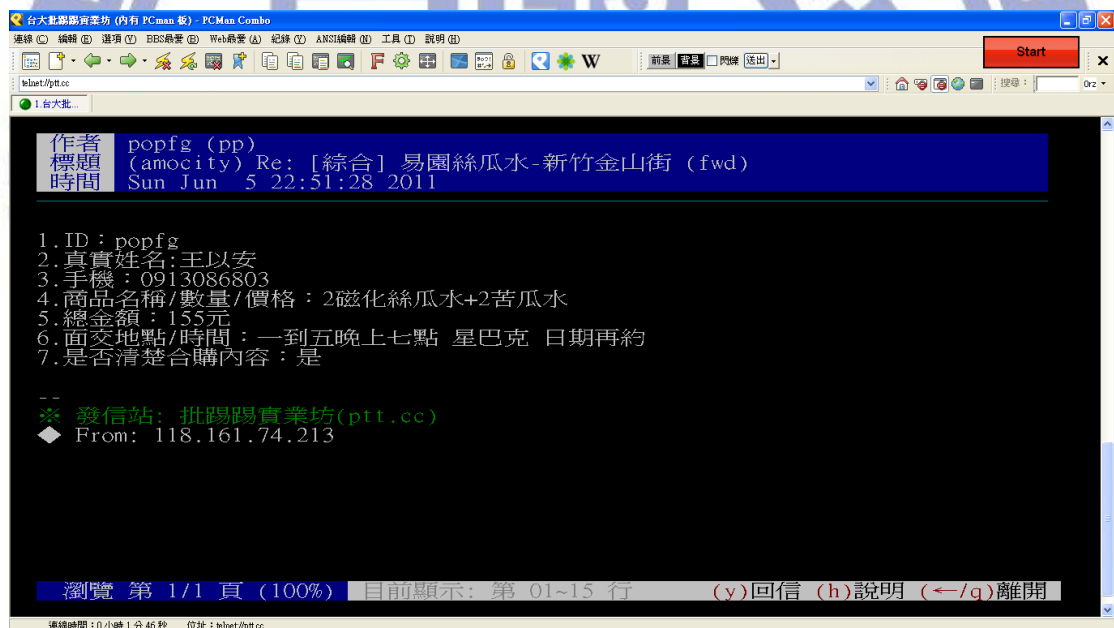
5.6 Summary

In this chapter, a scheme for authentication of BBS articles by using the two proposed data hiding methods for covert communication described previously in Chapter 4 is proposed. We regard an authentication signal generated from a folded cover article as a secret message and hide it through the embedding process previously-described in Algorithm 4.1 into the folded BBS article to obtain a protected article. Later, the protected BBS article is authenticated by comparing the authentication signal obtained from the extraction process described in Algorithm 4.2 with the verification signal computed from the protected BBS article directly. And through the various experiments conducted in this study, we can prove the feasibility of the proposed method.

By the way, it is mentioned that we implement the two data hiding methods for covert communication and authentication, respectively, in a single program. By using the program, we can easily use the two different techniques to achieve the purpose of protecting BBS articles, conducting secret communication, or both of them.

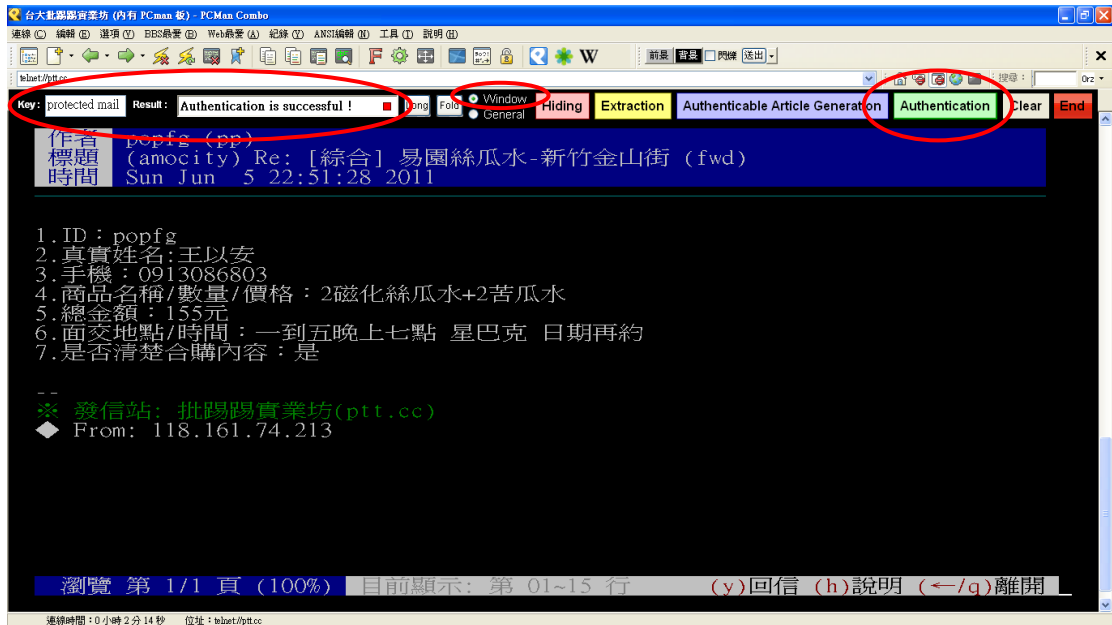


(a)

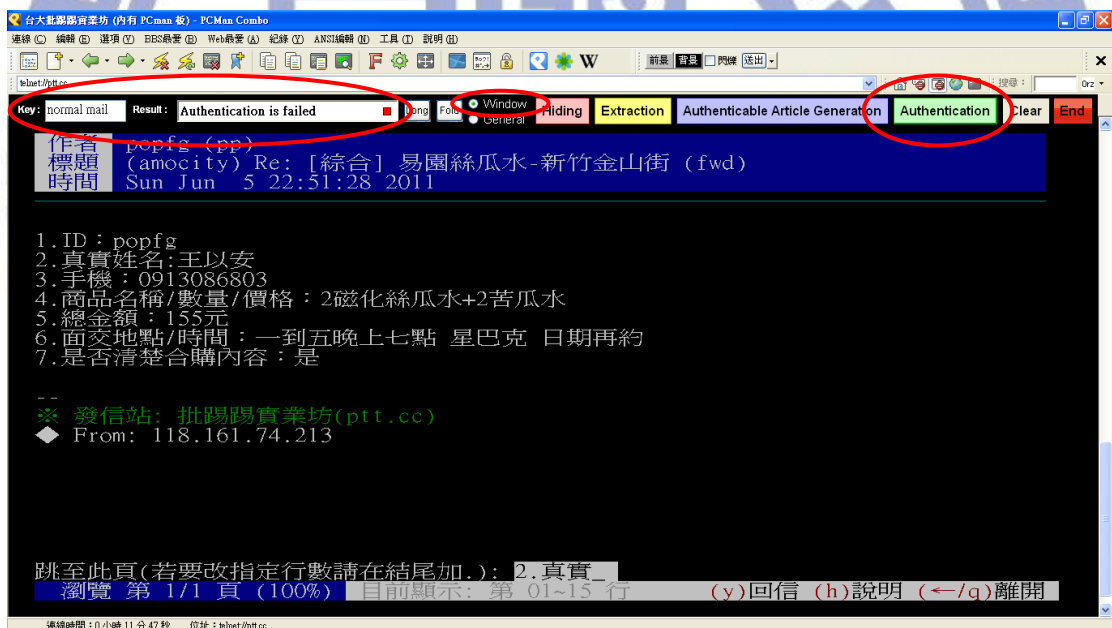


(b)

Figure 5.3 An example of experimental results. (a) A generation and sending process of a protected BBS mail. (b) A protected BBS mail displayed on the PCMan. (c) A protected BBS article authenticated with a correct secret key. (d) A protected BBS article authenticated with a wrong key.

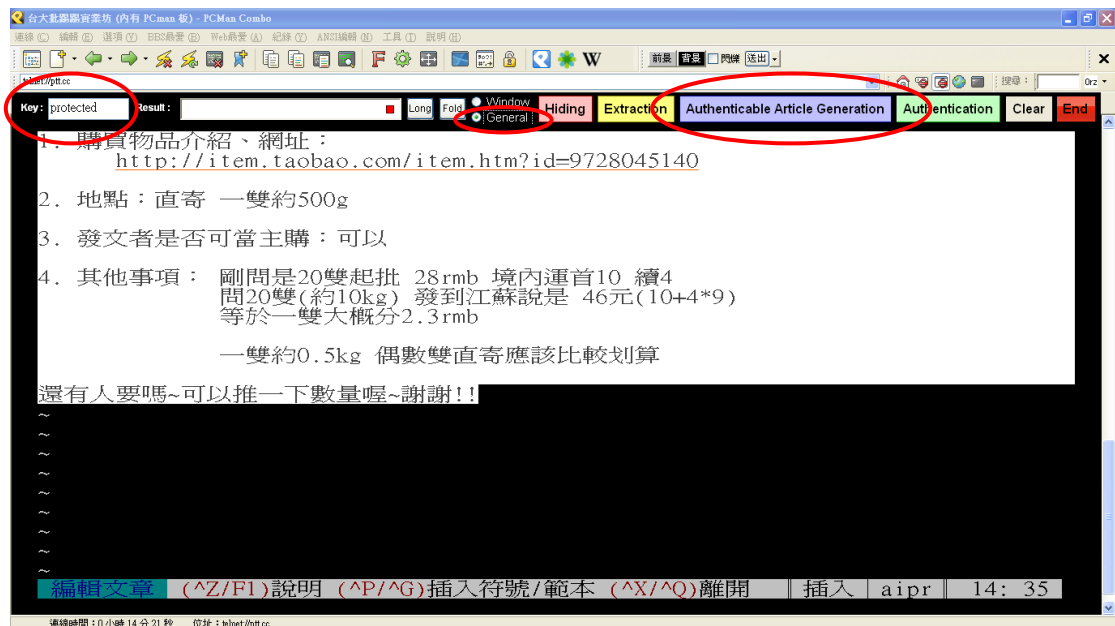


(c)

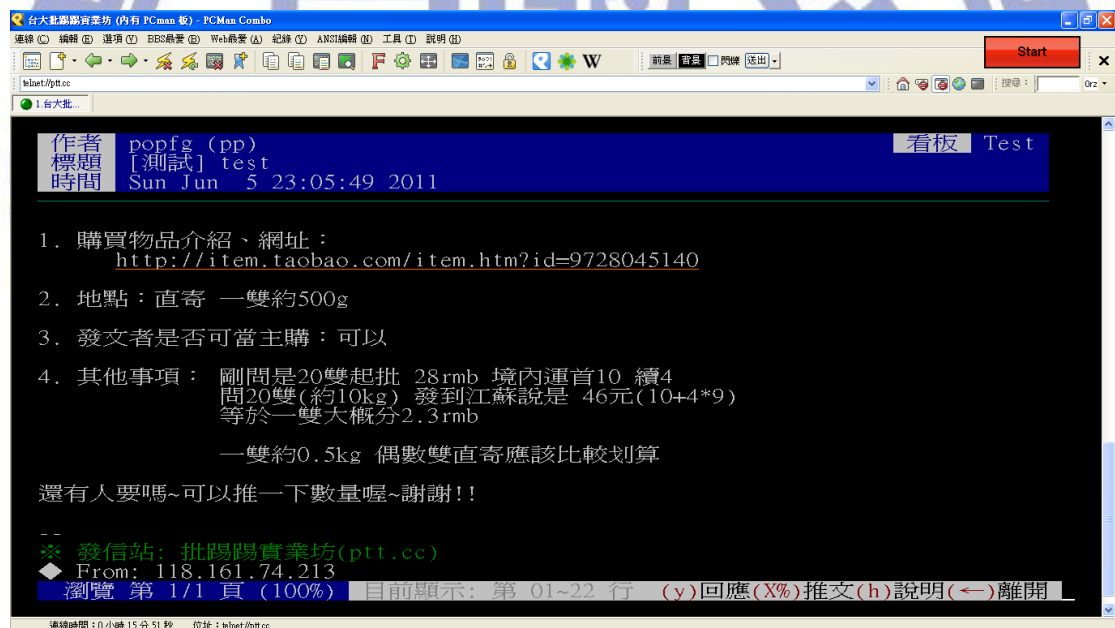


(d)

Figure 5.3 An example of experimental results (continued). (a) A generation and sending process of a protected BBS mail. (b) A protected BBS mail displayed on the PCMan. (c) A protected BBS article authenticated with a correct secret key. (d) A protected BBS article authenticated with a wrong key.

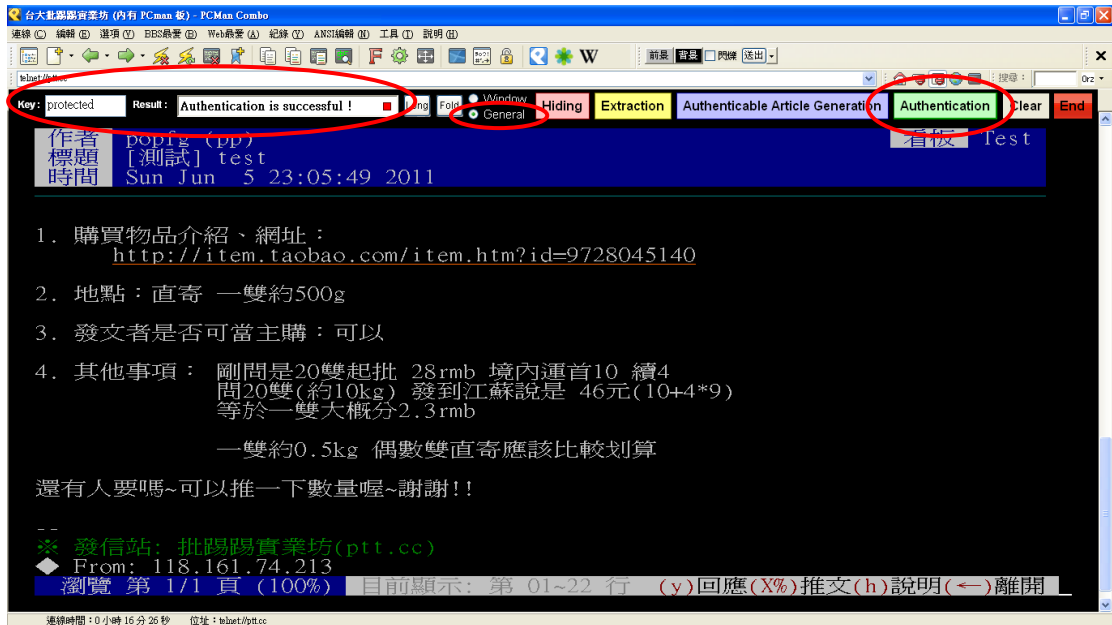


(a)

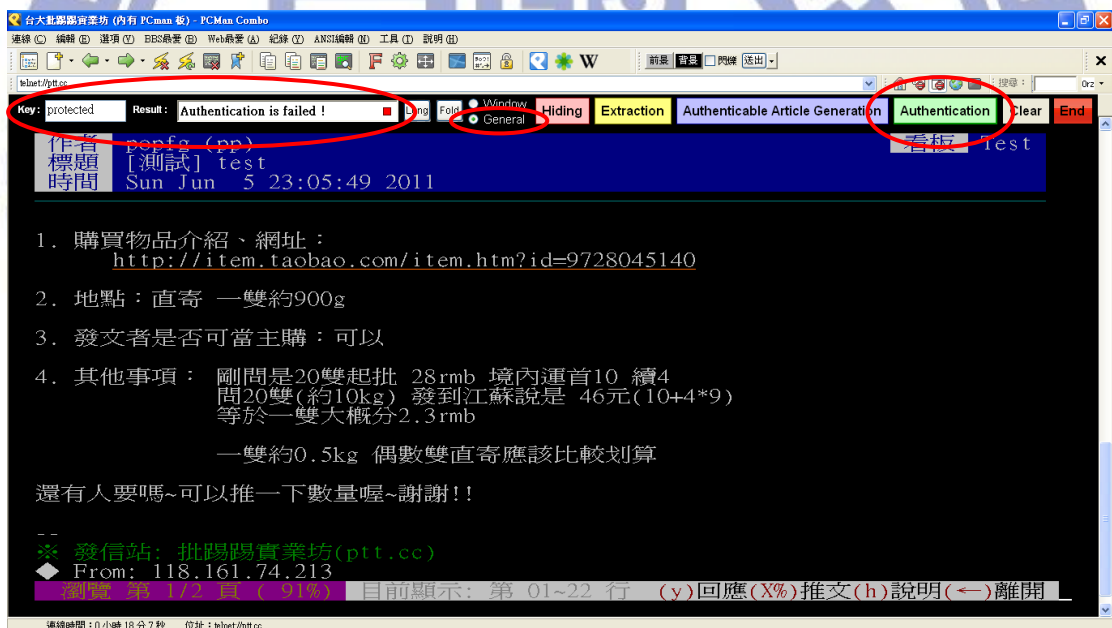


(b)

Figure 5.4 Another example of experimental results. (a) A generation and sending process of a protected BBS article. (b) A protected BBS article displayed on the PCMan. (c) An authentication result of a protected BBS article with a correct secret key. (d) An authentication result of a protected BBS article tampered by replacing a word.



(c)

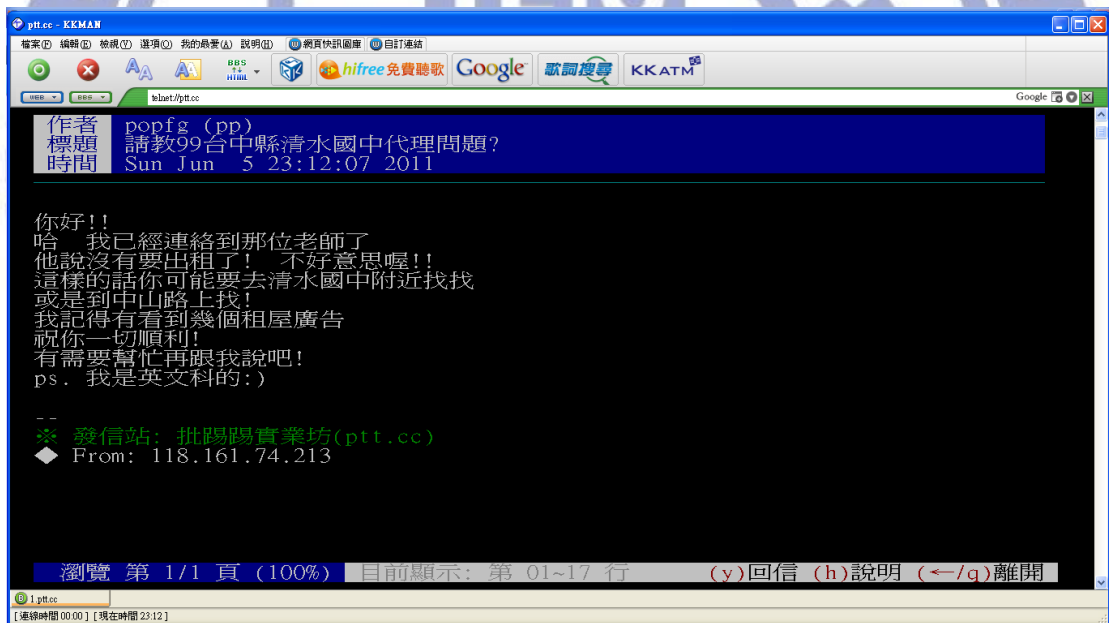


(d)

Figure 5.4 Another example of experimental results (continued). (a) A generation and sending process of a protected BBS article. (b) A protected BBS article displayed on the PCMan. (c) An authentication result of a protected BBS article with a correct secret key. (d) An authentication result of a protected BBS article tampered by replacing a word.

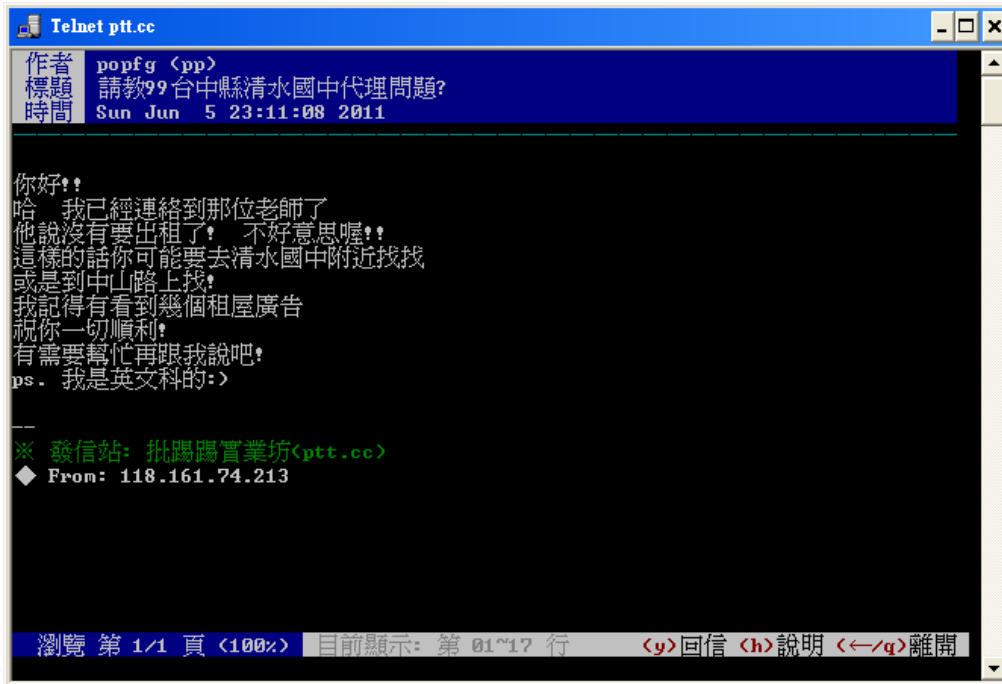


(a)



(b)

Figure 5.5 An experimental result displayed on the KKMan. (a) A normal BBS mail. (b) A protected BBS mail.



(a)



(b)

Figure 5.6 An experimental result displayed on the telnet connection program. (a) A normal BBS mail. (b) A protected BBS mail.

Chapter 6

Email Authentication by Special UTF-8 Space Codes

6.1 Introduction and Problem Definition

6.1.1 Introduction

In this chapter, the detail of the proposed email authentication method and the corresponding data hiding technique will be introduced. In Section 6.1.2, the problem definition is described, and the major idea of the proposed method is described in Section 6.2. In Section 6.3, we specifically describe the process of generating a protected email with an authentication signal embedded, and in Section 6.4, the proposed email verification process is described. In Section 6.5, some experimental results are given to show the feasibility of the proposed method. In Section 6.6, an example for proving the adaptability of the proposed method for authentication of blog articles is given. Finally, a brief summary is given in Section 6.7.

6.1.2 Problem Definition



Nowadays, with the rapid development of the Internet, people often communicate with others through emails. Specifically, a newly-developed type of email, the webmail, can be sent, read, and received at any place as long as an internet-browser, which can connect to the network without the need of other specific

client software, is available. Furthermore, emails can even be read directly on webmail platforms rather than be loaded to client servers. This trend will be more prominent in the coming era of cloud computing. Due to the convenience of webmail, people are accustomed to use it as the most popular type of email.

On the other hand, it is hard to prevent malicious users from intercepting and tampering with the content of emails or cracking user passwords to send fake emails. Therefore, an authentication scheme for verifying the fidelity and integrity of emails is important. To this aim, we propose a data hiding technique for email authentication in this study, and the detail is described in the subsequent sections.





6.2 Major Idea of Proposed Method by Use of Special UTF-8 Codes



In Chapter 3, we described how we fulfill authentication of blog articles on popular web browsers by using *special ASCII control codes* which become invisible after being embedded into blog articles. With this success, naturally we tried to implement the same data hiding technique on webmails for email authentication. However, for some popular webmail platforms such as G-mail, when we send a stego-email with some special ASCII control codes embedded, all the codes will be removed so that nothing can be extracted later for verification of the mail. Nevertheless, we tried further to hide data by the method proposed in Chapters 4 and 5. It was used there to achieve the goals of covert communication via BBS articles and authentication of BBS articles, using *special invisible Big-5 code*. This time these special codes can be preserved after undergoing the mail sending and receiving processes conducted on webmail platforms, but unfortunately for some popular web browsers like Mozilla Firefox and Google Chrome, these codes are revealed and

appear as special patterns provided by them. For example, the special Big-5 code “FDEA” is transcoded to the corresponding Unicode code “E25F” and displayed graphically on the Firefox and Chrome as  and , respectively. Thus the above two data hiding methods are not appropriate for webmail authentication, either.


Finally, we tried the use of some *special UTF-8 space codes* to achieve the aim of email authentication. This time we succeeded. The idea is inspired from the data hiding technique for the BBS using *special Big-5 space codes*. Specifically, we found the UTF-8 code “E38080” useful for our purpose here, which is transcoded from a Big-5 space code and is a standard Unicode code. Because it is located in the normal character area with a space chart, and is invisible when it is displayed on browsers, we can combine it with white spaces to become special symbols for use in data hiding in emails. The used UTF-8 codes and the devised code mapping relationship are listed in Table 6.1.

Table 6.1 Encoding table for used UTF-8 codes.

| Bit stream (binary) | 0 | 1 | Start signal | End signal |
|---------------------------------------|---|---|--|---|
| Special UTF-8 codes (embedded symbol) |  |  |  |  |

(Note: : Special UTF-8 space code. : Original white space code.

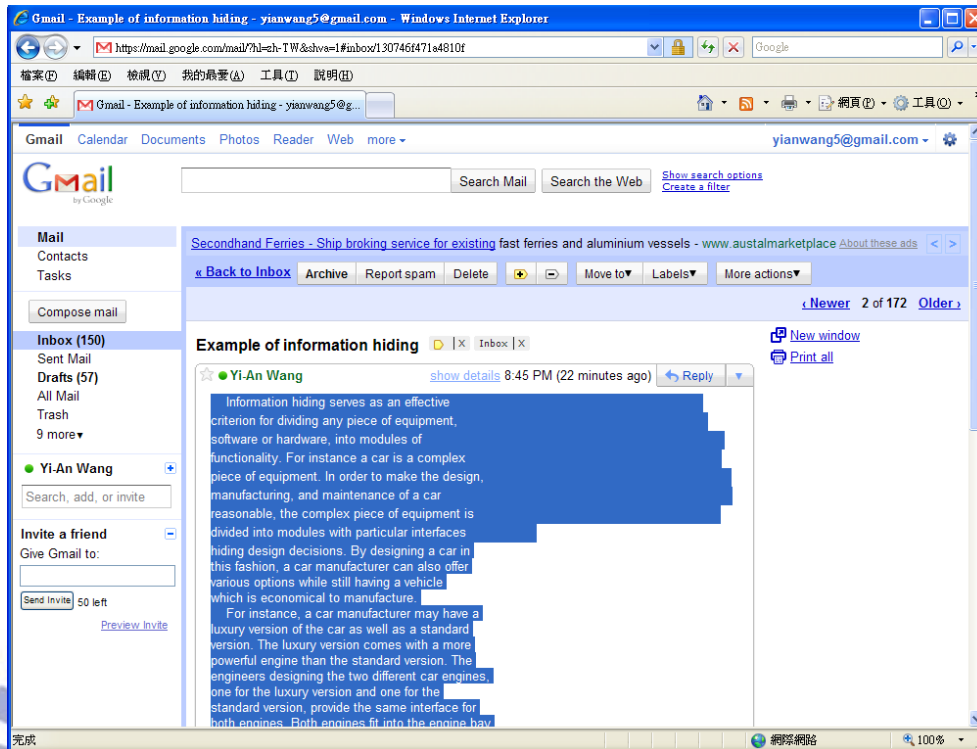
: Line feed code)

The process of embedding these secret symbols is similar to the hiding procedure used for blog authentication presented in Chapter 3, except that we embed the initial  (special space) before all symbols in each *data embedding slot* to be a *start signal*. It can assist us in finding the starting position of the secret symbols embedded

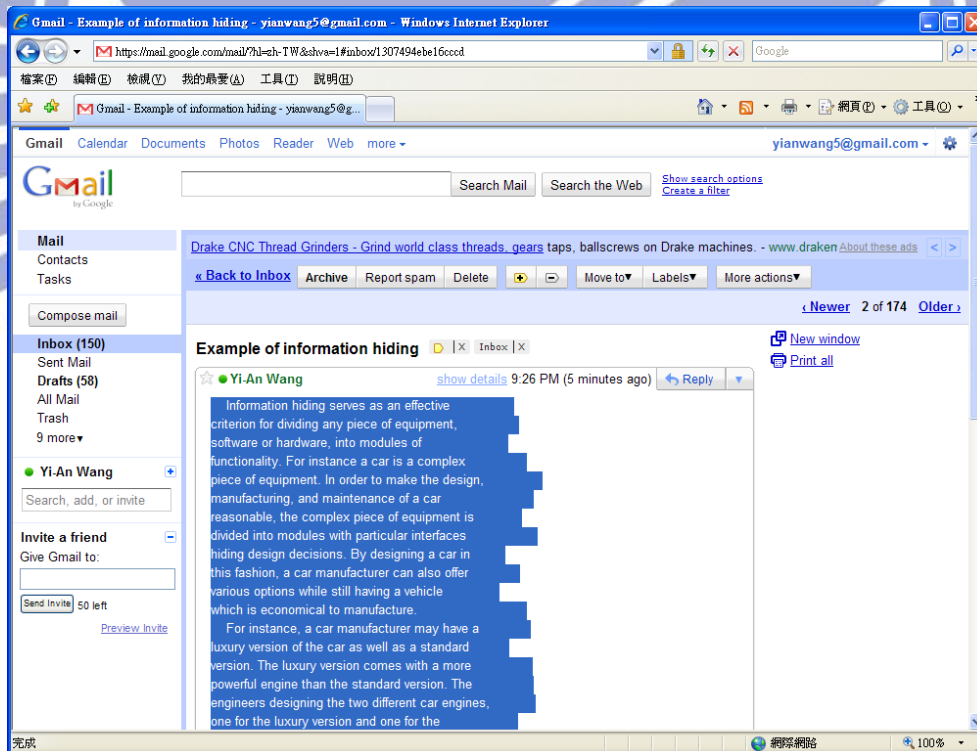
in each line when we conduct the verification process.

We also have to use the *distributional* embedding method mentioned in Chapter 3 to disperse all secret symbols at the ends of the text lines. The reason is that, when we read a protected email on a web browser, if we highlight the article content of the email by a mouse, then the embedded secret symbols will appear at the text line ends as some white spaces. This phenomenon will become an undesirable leakage of the embedded authentication signal. Two examples for highlighting a stego-email on the IE are shown in Figure 6.1, where Figure 6.1(a) shows the case of hiding the secret symbols at the line ends *just in normal order* and Figure 6.1(b) shows the case of embedding the symbols *evenly* at the line ends.

The imperceptibility of these symbols composed of UTF-8 space codes is relatively lower than the secret symbols used in the previous chapters. However, these symbols are still idoneous to be used for email authentication. They do not disturb users to read emails at all, and even if malicious users work out our hiding method, they still cannot arbitrarily tamper with a protected email and escape from our verification. Furthermore, this method is appropriate for most operating systems and even compatible with other text-styled Internet applications, because all the used secret symbols are composed of UTF-8 codes. These codes are defined in the standard Unicode format with normal charts, and nowadays almost all websites use the UTF-8 standard as their text encoding format. Thus, in the proposed method special UTF-8 space codes are used to embed data for email authentication. The detailed processes implementing the method are described in the next sections.



(a)



(b)

Figure 6.1 A highlighted stego-email with secret symbols embedded (a) just in order or (b) evenly using the proposed method.

6.3 Authentication Signal Generation and Embedding Process

The proposed process for generating an authentication signal and embedding it into an email is described in this section. In Figure 6.2, an illustration of the process is shown, and this process is appropriate for most popular webmail platforms. First, we fold longer email article lines into shorter ones, leaving some character spaces at each line end as a *data embedding slot*, and replace each UTF-8 space code (if it exists) in the article with an approximately equal-length combination of four white space codes. (The length of a UTF-8 space is approximately equal to the length of four white spaces displayed in webmails.) Next, we remove from the folded email article all the *line feed* signals so that the verification process described in the next section will not be interfered by redundant line feed signals. The modified email article and a secret key then are used to generate an authentication signal using a hash function and the exclusive-OR operation. Subsequently, we map each bit of the authentication signal into our devised secret symbols one by one according to a table (Table 6.1). Finally, these secret symbols are embedded into the text line ends accompanied with end signals to obtain a protected email. The detailed procedure is described in Algorithm 6.1 below.

Algorithm 6.1 *Authentication signal generation and embedding process.*

Input: a secret key K , a hash function f (such as MD5), and an email E to be protected.

Output: a protected email E' .

Steps.

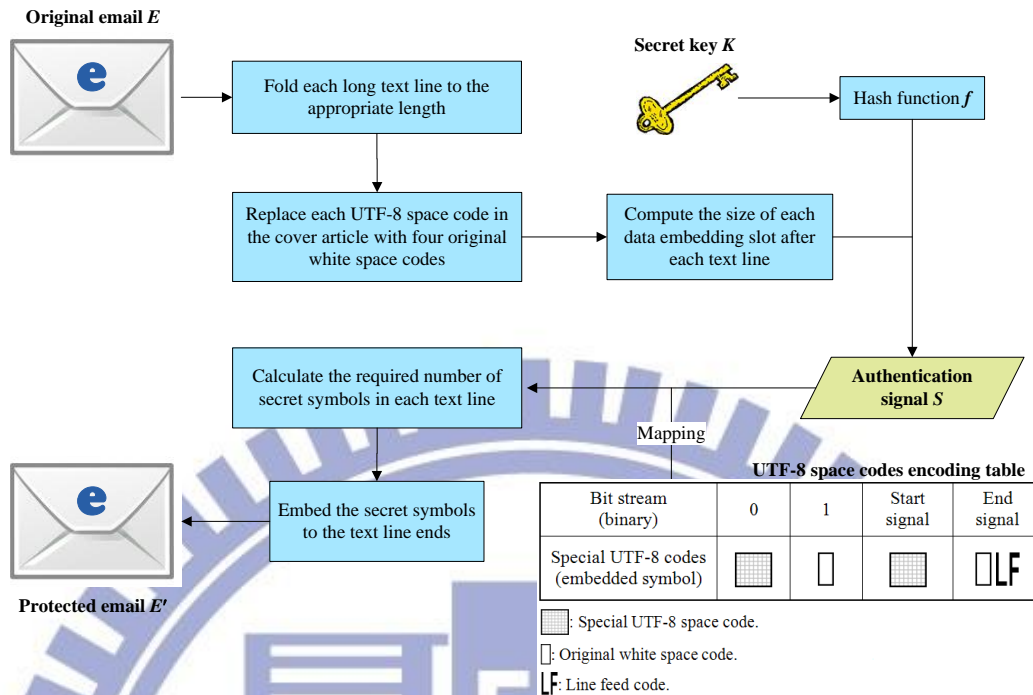


Figure 6.2 Flowchart of proposed authentication signal generation and embedding process.

1. Fold sequentially each text line l_i with a length larger than 60 units (with a unit meaning the length of an ASCII code displayed on web browsers) in email E into a 60-unit line by inserting a line feed, denoted as LF and occupying *zero unit*, after the original 60th character in l_i to generate a *folded article*, denoted as F .
2. Replace each UTF-8 space code (if it exists) in F with an approximately equal-length combination of four white space codes.
3. Compute the size L_i of the data embedding slot at the end of each text line l_i in F by:

$$L_i = 90 - \text{the length of } l_i,$$

which means the maximum number of characters that can be inserted at the end of l_i .

4. Remove all the line feed signals in F , use the result and the secret key K as inputs to the hash function f to generate two 128-bit digests F' and K' , respectively, and

return all the removed *LF* signals back into their original positions in *F*.

5. Compute the exclusive-OR value $F' \oplus K'$ to obtain a 128-bit authentication signal *S*, and let N_1 and N_2 both denote the total number of bits of *S*.
6. Map sequentially each bit of the authentication signal into secret symbols p_1, p_2, \dots, p_{128} , according to Table 6.1.
7. Scan *F* from the first line to find the line, say the *i*-th, with the longest slot and calculate the number n_i of secret symbols *embeddable* in this *i*-th line in the following way:
 - (1) if $L_i > 4$, then increment n_i by 1, decrement L_i by 2, decrement N_1 by 1, and perform Step 7 again;
 - (2) if $L_i \leq 4$ or $N_1 = 0$, then perform Step 8.
8. Embed the symbols p_1 through p_{128} sequentially into *F*, starting from the first line, in the following way.
 - (1) Scan l_i to find the *line feed* LF, replace it with a *start signal*, sequentially embed n_i symbols in l_i at the end, decrement N_2 by n_i , and append an end signal, in which an LF is included, to the end of the embedded symbols.
 - (2) If the last line is processed, and if $N_2 > 0$, then embed the remaining symbol/symbols below *F* as one or more blank lines in the following way.
 - 8.1 Embed as many symbols as possible into a new line sequentially before the length of the line (in units) becomes larger than 87, insert a start signal at the line start, and append an end signal to the line end.
 - 8.2 If all symbols are embedded, then continue; otherwise, repeat Step 8.1 again.
9. Take the final version of *F* as the desired protected email *E'*.

In Step 2 of the above algorithm, because it is possible that the UTF-8 space

appears in an original cover email, we need to replace it with a combination of four white space codes, which is approximately equal-length to a UTF-8 space, so that the verification process described in the next section can be conducted correctly.

6.4 Authentication Signal Extraction and Verification Process

The proposed email authentication signal extraction and verification process is described in this section. First, we extract the secret symbols embedded in the protected email and transform them into an authentication signal S . Then, we use the same secret key and hash function as those used in Algorithm 6.1 to transform the email, in which all the secret data and the line feed signals are removed, into a verification signal T . Finally, we can verify the integrity and fidelity of the email by comparing the two signals S and T . The main process is similar to the authentication process of blog articles mentioned in Algorithm 3.2, except that when we conduct the extraction step for secret symbols, first we must find the start signals before other symbols in the text lines so that we can differentiate each white space code appearing in the email, which is typed in the original cover email or appended to be a secret symbol. A flow chart of the proposed process is shown in Figure 6.3, and the detailed algorithm is given in Algorithm 6.2.

Algorithm 6.2 *Authentication signal extraction and blog article verification.*

Input: a secret key K and a hash function f both being the same as those used in Algorithm 6.1; and a protected email E' .

Output: an authentication report R .

Steps.

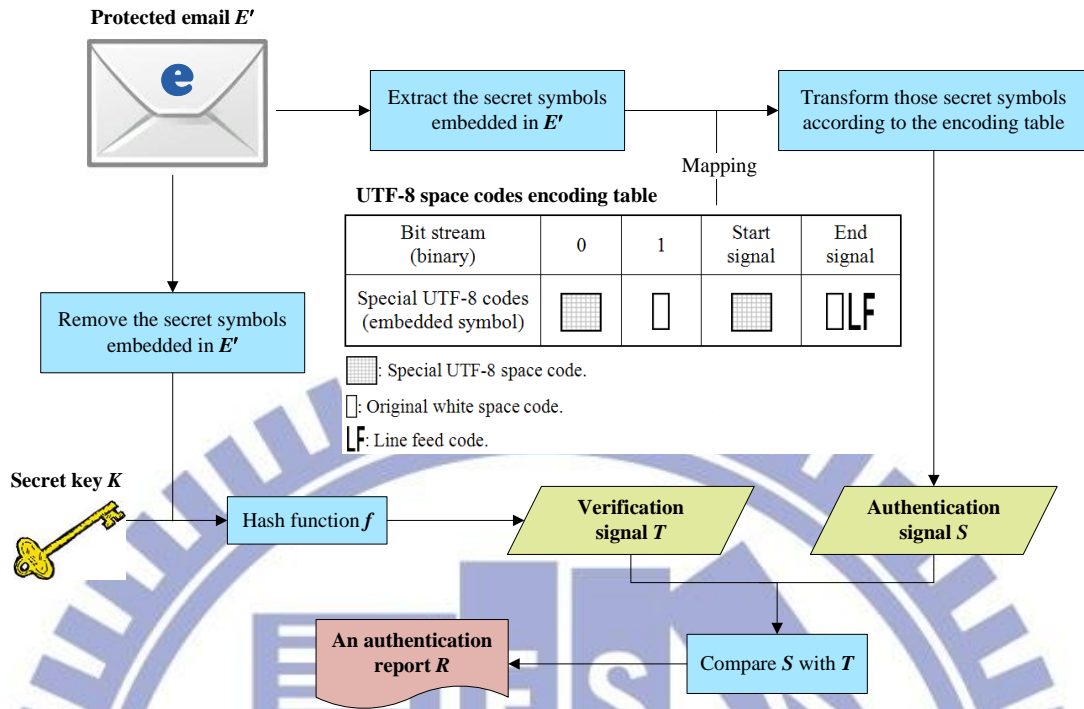


Figure 6.3 Flowchart of the proposed authentication signal extraction and email verification process.

1. Check each line l_i in the protected email E' sequentially, starting from the first line; find the start signal; and extract the subsequent secret symbols embedded in front of the end signal in l_i .
2. Concatenate all the extracted secret symbols sequentially into a set of 128 codes, p_1, p_2, \dots, p_{64} .
3. Map each of p_1 through p_{64} to a corresponding bit 0 or 1, according to Table 6.1.
4. Concatenate these bits into a 128-bit authentication signal S .
5. Use the secret key K as an input to the hash function f to generate a 128-bit digest K' .
6. Remove all the secret data and line feed signals from the email, and use the result as an input to the hash function f to generate a 128-bit digest E'' .
7. Compute the exclusive-OR value $E'' \oplus K'$ to get a 128-bit verification signal T .

8. Compare S and T , resulting in the following two cases.
 - (5) If $S = T$, then regard the input E' as *unmodified* and mark it so in the authentication report R .
 - (6) If $S \neq T$, then regard E' as *modified* and mark it so in R .
9. Output the authentication report R .

6.5 Experimental Results

Some examples of our experimental results are given in this section. We tried to generate protected emails using the proposed method through many popular webmail platforms such the G-mail, hotmail, and yahoo mail; and all the experimental results prove that the proposed email authentication scheme is feasible.

In Figure 6.4(a), we tried to send an email through the webmail platform G-mail on Chrome, and we use Algorithm 6.1 with a secret key as input to transform the email into a protected email. The appearances of the interface of the algorithm and a protected email are shown in Figures 6.4(b) and 6.4(c), and the highlighted protected email is shown in Figure 6.4(d). Later, when the protected email was received, we verified the integrity and fidelity of it by the use of a correct secret key, as shown in Figure 6.4(e). However, if the wrong key is typed as shown Figure 6.4(f), then the verification will fail and be marked to be so in the authentication report. And the appearances of the protected email and a highlighted form of it displayed on the IE are shown in Figure 6.5.

6.6 Adaptability of Proposed Method for Authentication of Blog Articles

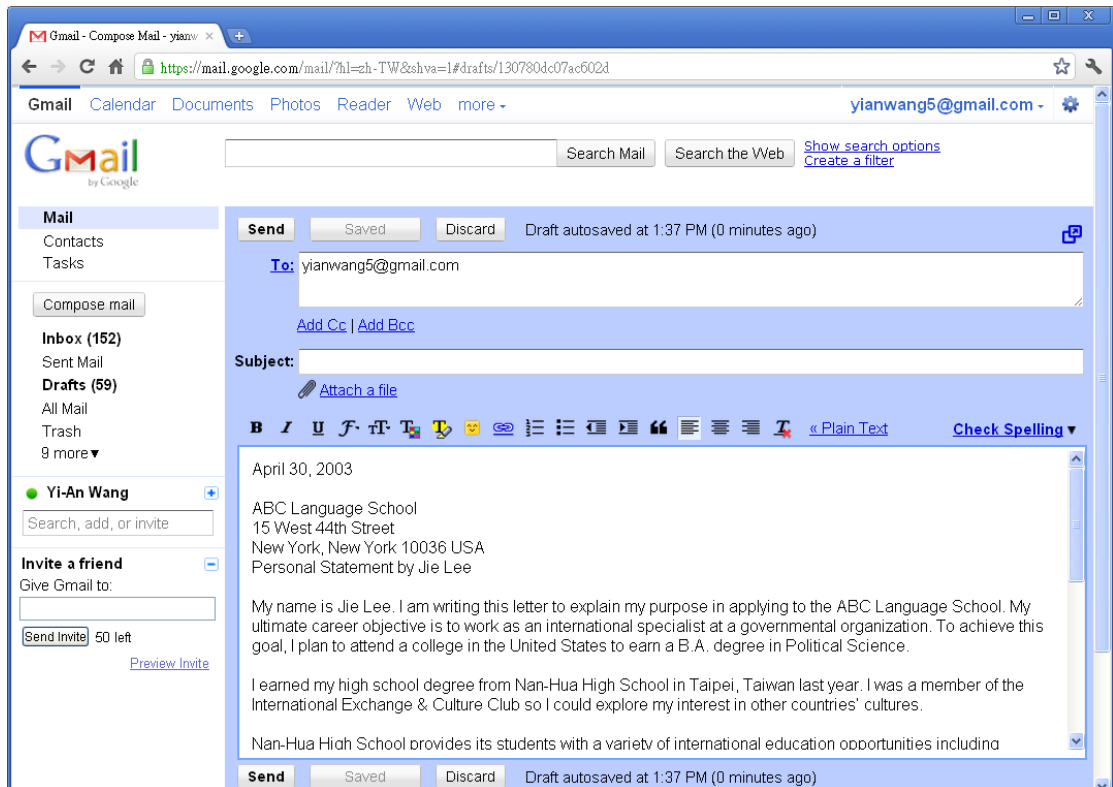
In Section 6.2, we mentioned that the proposed hiding method by the use of

UTF-8 space codes is not only appropriate for email authentication but also feasible for other text-styled Internet applications. Thus, in this section, we provide an example of our experimental results by implementing the proposed method to accomplish authentication of blog articles.

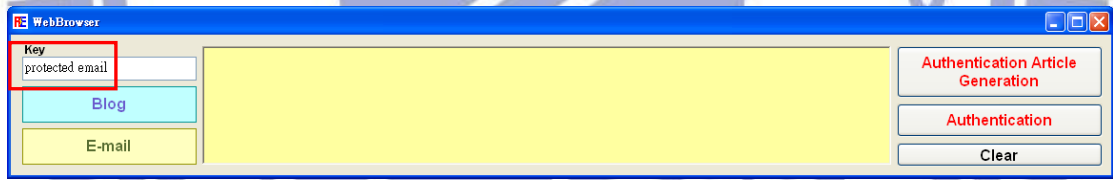
In Figure 6.6(a), we tried to publish a protected blog article by our method with a secret key as shown in Figure 6.6(b). And the appearance of the protected blog article is shown in Figure 6.6(c). We can verify whether the protected blog article is modified by malicious users by the proposed method with a correct secret key, as shown in Figure 6.6(d). And as shown in Figure 6.6(e), if the protected article is modified, we will obtain a wrong authentication result even if a correct secret key is typed.

6.7 Summary

In this chapter, we propose an authentication method for emails by a new data hiding technique using special UTF-8 space codes, which is inspired from the proposed method by the use of special Big-5 space codes described previously in Chapter 4. The method is appropriate for many operating systems, because all the used secret symbols are composed of UTF-8 codes which are defined in the standard Unicode format with normal charts. And through the experimental results, we can prove the feasibility of the proposed method for authentication of emails on many popular webmail platforms and other Internet applications such as blog article authentication.

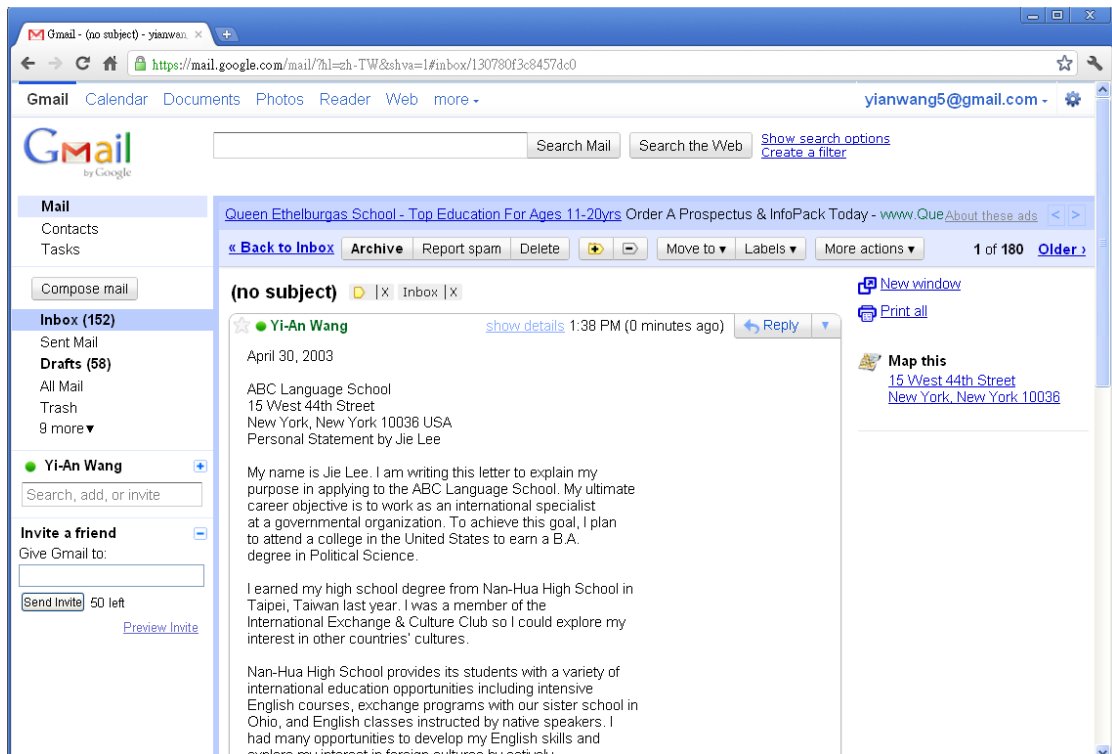


(a)



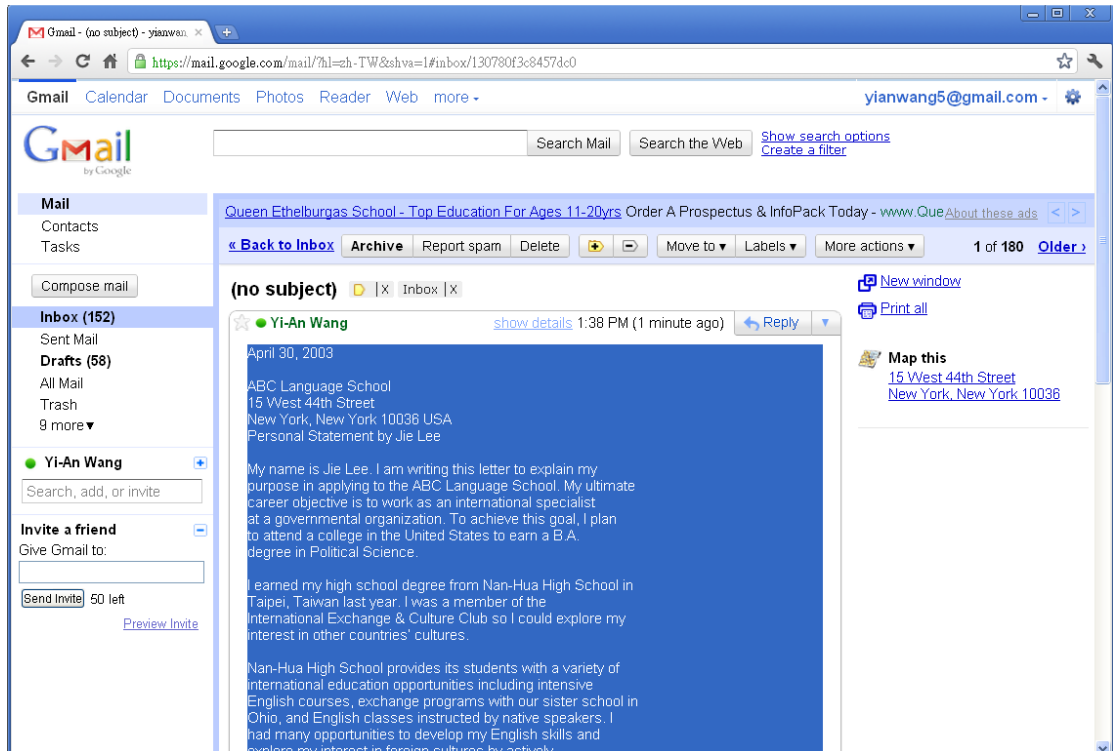
(b)

Figure 6.4 An example of experimental results. (a) An original email will be send through the G-mail webmail platform. (b) Our program with a secret key typed. (c) A protected email. (d) A protected email highlighted by a mouse. (e) The authentication result by using the correct secret key. (f) The authentication result by using a wrong secret key

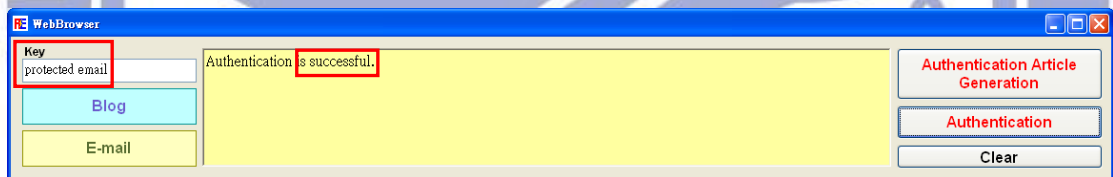


(c)

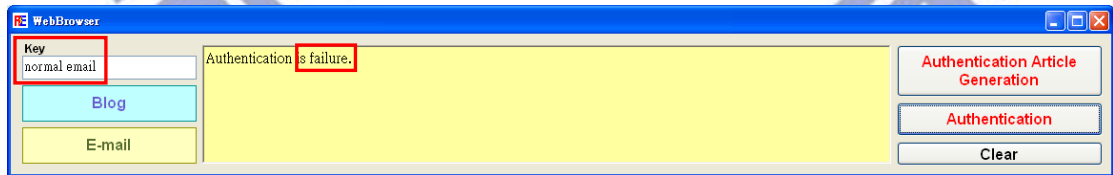
Figure 6.4 An example of experimental results (continued). (a) An original email will be send through the G-mail webmail platform. (b) Our program with a secret key typed. (c) A protected email. (d) A protected email highlighted by a mouse. (e) The authentication result by using the correct secret key. (f) The authentication result by using a wrong secret key.



(d)

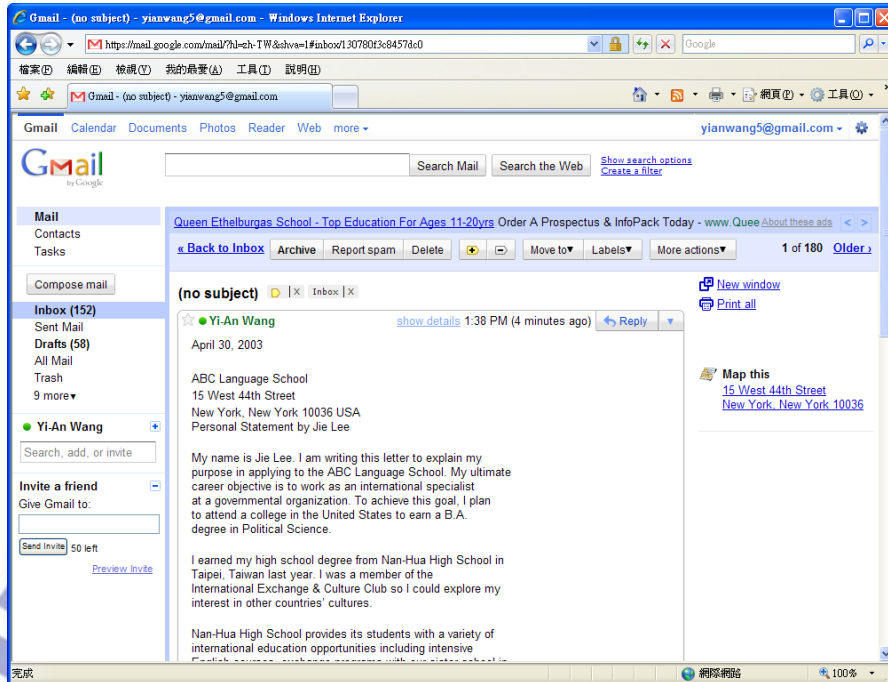


(e)

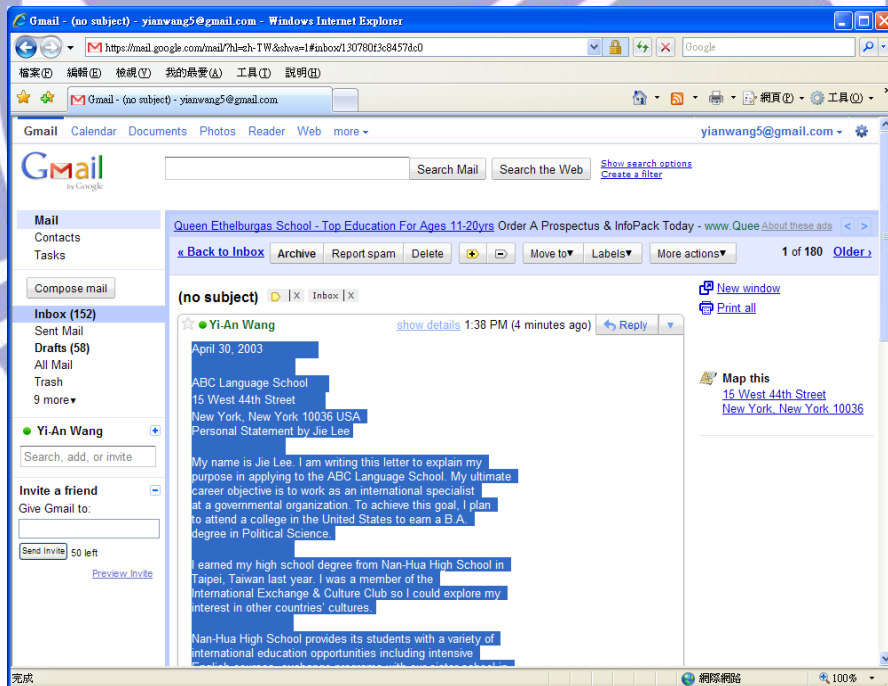


(f)

Figure 6.4 An example of experimental results (continued). (a) An original email will be send through the G-mail webmail platform. (b) Our program with a secret key typed. (c) A protected email. (d) A protected email highlighted by a mouse. (e) The authentication result by using the correct secret key. (f) The authentication result by using a wrong secret key.

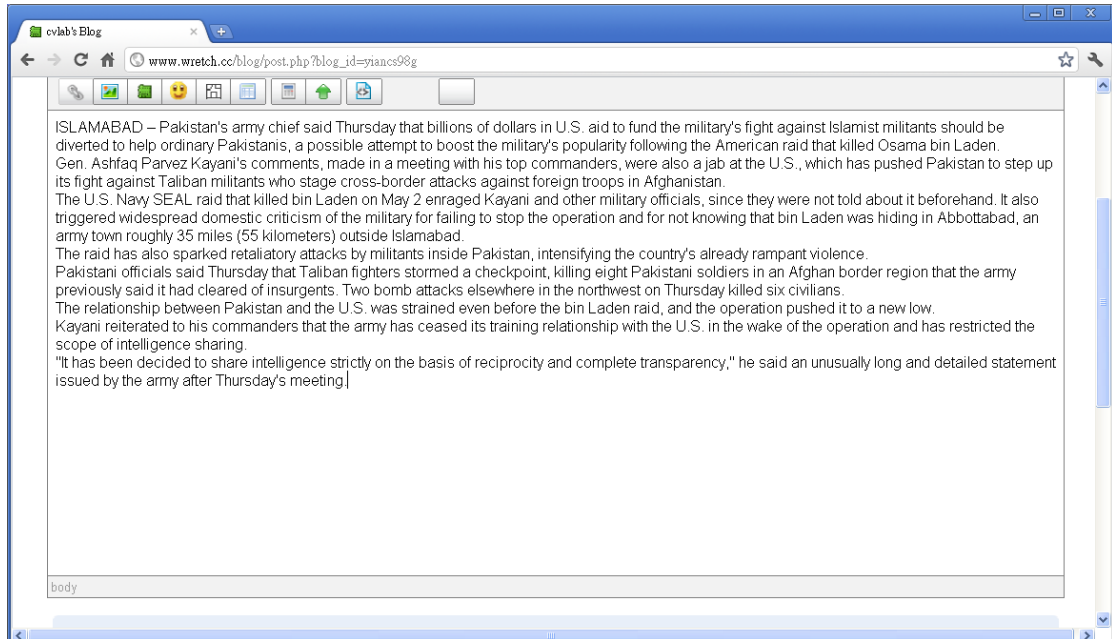


(a)

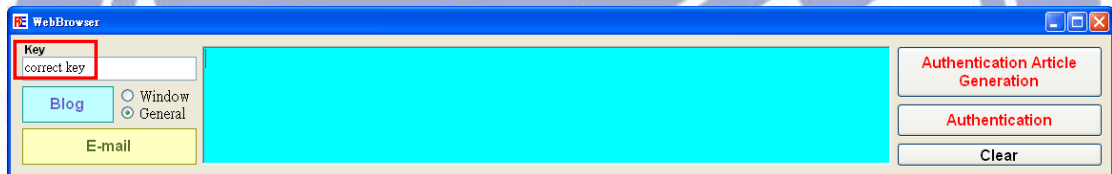


(b)

Figure 6.5 The appearances of (a) a protected email and (b) its highlighted form displayed on IE



(a)

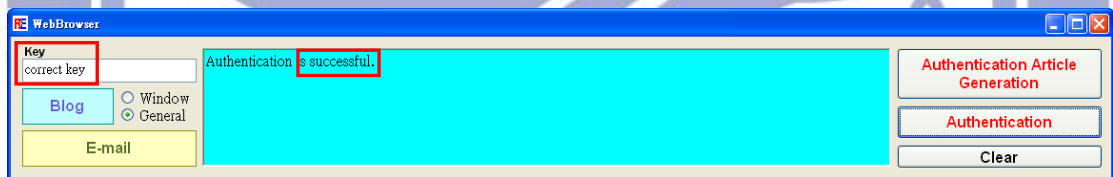


(b)

Figure 6.6 An example of experimental results. (a) An original article will be publish on blog. (b) Our program with a secret key typed. (c) A protected blog article. (d) The authentication result by using the correct secret key. (e) A protected blog article tampered by replacing a word. (f) The authentication result of the tampered blog article.

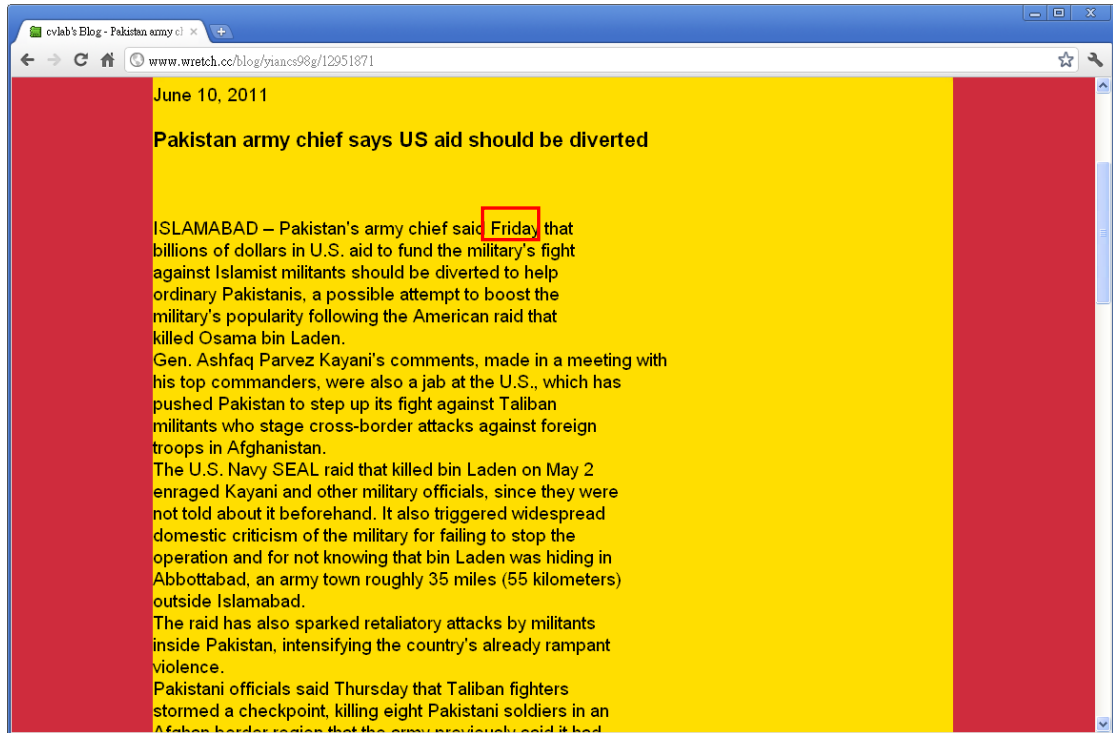


(c)

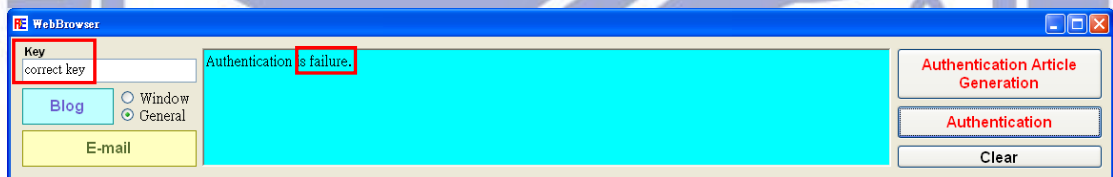


(d)

Figure 6.6 An example of experimental results (continued). (a) An original article will be publish on blog. (b) Our program with a secret key typed. (c) A protected blog article. (d) The authentication result by using the correct secret key. (e) A protected blog article tampered by replacing a word. (f) The authentication result of the tampered blog article.



(e)



(f)

Figure 6.6 An example of experimental results (continued). (a) An original article will be publish on blog. (b) Our program with a secret key typed. (c) A protected blog article. (d) The authentication result by using the correct secret key. (e) A protected blog article tampered by replacing a word. (f) The authentication result of the tampered blog article.

Chapter 7

Conclusions and Suggestions for Future Works

7.1 Conclusions

In this study, we have proposed several new data hiding techniques for Internet applications, including the blog, BBS, and email. These techniques are useful for applications like covert communication, authentication, etc.

For the blog, we have proposed a new article authentication method by the use of the proposed data hiding technique which uses invisible ASCII control codes for authentication signal generation. Specifically, the signal is generated from a folded cover article with a secret key, and is embedded in a distributional hiding order into a folded version of the input article to obtain a verifiable blog article. Then, by comparing the extracted authentication signal with a verification signal computed from the stego-article, a given blog article can be authenticated to decide whether it has been tampered with or not.

For the BBS, two hiding methods have been proposed. One is based on the use of invisible Big-5 codes, and the other on the use of special Big-5 space codes. We use the two methods to encode a secret message and embed the resulting secret symbols into a folded article to achieve the goal of covert communication via the BBS. According to the experimental results, the secret message hidden in a BBS article is not observable from the appearance, and it was also proposed to enhance the security of the proposed method by adding a user-defined secret key to randomize the content of the secret message, so that a malicious user cannot easily extract the secret even

when he/she knows the proposed algorithm. Furthermore, we also use the two proposed methods to accomplish BBS article authentication, and our experimental results prove the feasibility of the proposed methods.

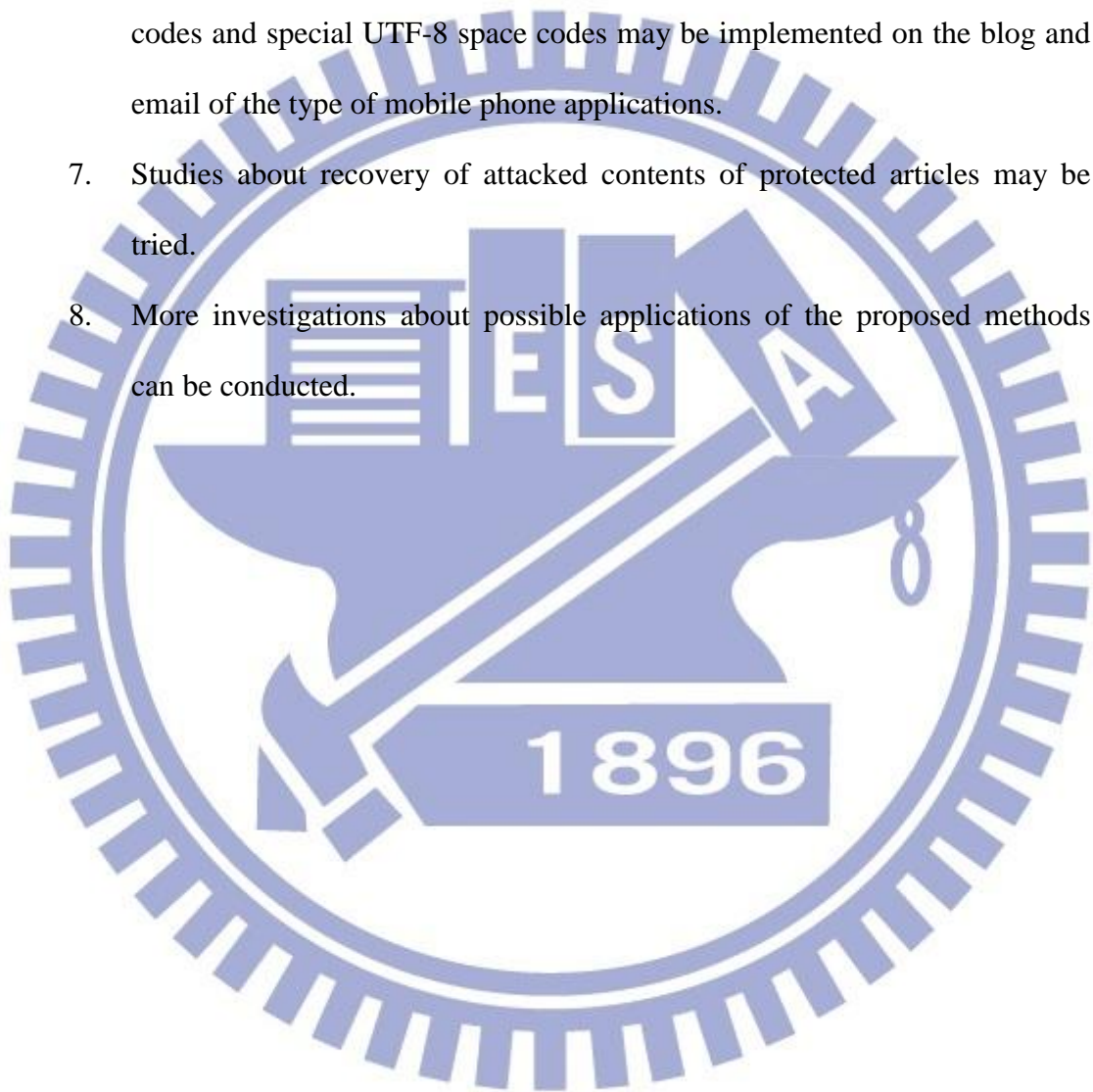
For email, an authentication scheme based on a method which uses special UTF-8 space codes has been proposed. The embedded secret symbols devised by combining the used UTF-8 space codes appear at text line ends with some white spaces, if the article contents of emails are highlighted by a mouse. However, these symbols are still idoneous to be used for email authentication, because they do not disturb users to read emails at all. And even if malicious users work out the proposed hiding method, they still cannot arbitrarily tamper with a protected email and escape from the verification. The authentication process is similar to that used for blog article authentication. And through our experimental results, it is proven that the proposed data hiding method can be used for authentication of not only the email but also other Internet applications such as the blog.

7.2 Suggestions for Future Works

According to our experience obtained in this study, several suggestions for future works are listed in the following.

1. The data hiding methods proposed in this study may be improved by randomizing the embedding order of the secret symbols at text line ends.
2. The data hiding methods proposed in this study may be improved by randomizing the mapping mode between bit strings of a secret message and the used secret symbols.
3. The data hiding methods proposed in this study can be used for more applications, such as metadata association, secret sharing, and so on.

4. The data hiding method for email authentication proposed in this study can be used on more Internet applications like the facebook and twitter.
5. Multimedia contents such as pictures and videos allowed on the blog or email could be used as cover channels to design new data hiding techniques.
6. The proposed authentication methods by the use of invisible ASCII control codes and special UTF-8 space codes may be implemented on the blog and email of the type of mobile phone applications.
7. Studies about recovery of attacked contents of protected articles may be tried.
8. More investigations about possible applications of the proposed methods can be conducted.



References

- [1] I. S. Lee and W. H. Tsai, "Data hiding in emails and applications by unused ASCII control codes," *Journal of Information Technology and Applications*, Vol. 3, No. 1, pp. 13-24, Sept. 2008.
- [2] BBS browser: <http://pcman.openfoundry.org/>. PTT address: <telnet://ptt.cc>, founded in Sept. 1995.
- [3] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding," *IBM System Journal*, Vol. 35, Nos. 3 & 4, Feb. 1996.
- [4] G. Cantrell and D. D. Dampier, "Experiments in hiding data inside the file structure of common office documents: a steganography application," *Proceedings of 2004 International Symposium on Information and Communication Technologies*, pp. 146-151, Las Vegas, Nevada, U. S. A., 2004.
- [5] T. Y. Liu and W. H. Tsai. "Quotation authentication: a new approach and efficient solutions by data hiding and cascaded hashing techniques," *IEEE Transactions on Information Forensics and Security*, Vol. 5, No. 4, pp. 945-954, Dec. 2010.
- [6] P. Wayner, "Strong theoretical steganography," *Cryptologia*, Vol. XIX/3, pp. 285-299, 1995.
- [7] S. Inoue, K. Makino, I. Murase, O. Takizawa, T. Matsumoto, and H. Nakagawa. "A proposal on information hiding method using XML," *Proceedings of 1st NLP and XML Workshop*, Tokyo, Japan, Nov. 2001.
- [8] "Extensible Markup Language (XML) 1.0 (Second Edition)," <http://www.w3.org/TR/REC-xml>, Feb. 2001.
- [9] Y. C. Lai and W. H. Tsai, "Covert communication via PDF files by new data hiding techniques," *Proceedings of 2009 Conference on Computer Vision*,

Graphics and Image Processing, Nantou, Taiwan, Aug. 2009.

- [10] I. S. Lee and W. H. Tsai, "A new approach to covert communication via PDF Files," *Signal Processing*, Vol. 90, No. 2, pp. 557-565, Feb. 2010.
- [11] I. S. Lee and W. H. Tsai. "Security protection of software programs by information sharing and authentication techniques using invisible ASCII control codes," *International Journal of Network Security*, Vol. 10, No. 1, pp. 1-10, Jan. 2010.
- [12] I. S. Lee and W. H. Tsai, "Secret communication through web pages using special space codes in HTML files," *International Journal of Applied Science and Engineering*, Vol. 6, No. 2, pp. 141-149, Nov. 2008.
- [13] K. L. Huang and W. H. Tsai. "Secret sharing with steganographic effects for HTML documents," *Proceedings of 2004 Conference on Computer Vision, Graphics and Image Processing*, Hualien, Taiwan, Aug. 2004.
- [14] Y. H. Chang and W. H. Tsai, "A steganographic method for copyright protection of HTML documents," *Proceedings of 2003 National Computer Symposium*, Taichung, Taiwan, Dec. 2003.
- [15] S. Zhong, X. Cheng and T. Chen, "Data hiding in a kind of PDF texts for secret communication," *International Journal of Network Security*, Vol.4, No.1, pp. 17-26, Jan. 2007.
- [16] American National Standard for Information Systems - Coded Character Sets - 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII), ANSI X3.4-1986, American National Standards Institute, Inc., March 26, 1986.
- [17] CP950 to Unicode table:
<http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP950.TXT>, Jul. 2000.