國立暨南國際大學通訊工程研究所

碩士論文

克服在 CDES 傳送機制上的問題

Overcoming the suspicion in transmission scheme based on CDES

指導教授：吳坤熹博士

研究生：潘韋勳

中華民國九十九年六月

國立暨南國際大學碩（博）士論文考試審定書

通訊工程研究所

研究生 **潘韋勳** 所提之論文

克服在 CDES 傳送機制上的問題

Overcoming the suspicion in transmission scheme based on CDES

(中、英文題目)

經本委員會審查，符合碩（博）士學位論文標準。

學位考試委員會

_____ 委員兼召集人

_____ 委員

_____ 委員

中　華　民　國　99　年　7　月　7　日

博碩士論文電子檔案上網授權書

（提供授權人裝訂於紙本論文書名頁之次頁用）

本授權書所授權之論文為授權人在 暨南國際大學 通訊工程研究所 _____組 98 學年度第 二 學期取得 碩士 學位之論文。

論文題目： 克服在CDES傳送機制上的問題
指導教授： 吳坤熹

茲同意將授權人擁有著作權之上列論文全文（含摘要），非專屬、無償授權國家圖書館及本人畢業學校圖書館，不限地域、時間與次數，以微縮、光碟或其他各種數位化方式將上列論文重製，並得將數位化之上列論文及論文電子檔以上載網路方式，提供讀者基於個人非營利性質之線上檢索、閱覽、下載或列印。

- 讀者基非營利性質之線上檢索、閱覽、下載或列印上列論文，應依著作權法相關規定辦理。

授權人： 潘韋勳
簽　名： 潘韋勳 _____ 中華民國 99 年 08 月 26 日

致謝

兩年前有幸在研究所的甄試當中，順利了進入了國立暨南大學通訊工程研究所，成為吳坤熹老師的門生之一，也成為 R409(Next Generation Telephony Laboratory)這個大家庭中的一員，但兩年後的我，即將離開這個地方…

在本研究生涯當中，體悟到了什麼叫做「研究生」，所謂的研究生要懂得培養檢視、獨立思考、解決、驗證問題以及反問自己的能力，並且對於碰到許多困難時能夠按部就班、不疾不徐。美國二戰名將巴頓將軍曾經說過：「當每個人的想法一致時，就表示根本沒有人在用大腦」。對於「世上任何一件事情」都保有自己的看法與解讀分析，但也要懂得聆聽旁人的意見，對於不懂得會吝嗇對人的”知識”，要懂得貪婪。最後則是個人的創造力，創意來自於在大腦內不斷塗鴉，這樣的你，總會得到一幅夢想的作品，人唯有懂得思考和想像，才能讓世界更進步。謹記著吳老師總是常叮嚀我們的「人要懂得為成功找方法，不要為失敗找理由」等等…，以上是我在這兩年中最大的收穫，也會陪伴著我的人生旅程，讓我能勇敢面對服完兵役後的我所遭遇到的「社會」。

感謝吳坤熹老師在我研究上的所有幫助，再一次感謝老師對我的鼓勵。

感謝 R409 的所有人，嘉裕、韋霖、韋立、霓雅、筱婷、Alex、信富兄、以及阿搞、胖達、阿嘉、麗雯、dog 的陪伴，本研究生涯中有你們真好。

感謝通訊工程研究所的所長、婉婷總是辛苦處理著研究生們的所有事務。

感謝室友們，哲睿，小張、邱哥在幫了我許多生活上的問題。

感謝其他實驗室的同學，一起打通訊所的籃球比賽和排球比賽，希望有機會還能再與你們交手，但是冠軍還是我們 R409 的。

感謝暨大提供健身房和田徑場，讓我研究之餘還能做我喜歡的運動。

最後，感謝我的父母永遠相信著我以及怡君、兄弟姊妹對我的鼓勵與陪伴。

謝謝我所有認識的人及認識我的人，謝謝!

論文名稱：克服在 CDES 傳送機制上的問題

校院系：國立暨南國際大學科技學院通訊工程研究所　　頁數：78

畢業時間：2010 年/6 月　　學位別：碩士

研究生：潘韋勳　　指導教授：吳坤熹 博士

摘要

CDES (Confused Document Encrypting Scheme) 是一項提供資訊隱藏(Information Hiding 或是稱之為 Steganography)的文件保護機制。與傳統加密機制不同的是，它並不直接傳送加密過的訊息，而是將秘密訊息隱藏在正常並具有意義的文件當中。傳送端分別傳送多份欺敵文件(Cheating text)以及一份經過加密的索引檔案(Plaintext index file)給接收端，每份欺敵文件的內容皆為有意義的文字所組成，就算被人截取，也很難察覺出，哪些文件只是一般訊息，哪些才隱藏著大量的秘密訊息。但是，本研究發現在這樣的傳送機制下，反而會讓正在竊聽網路封包的竊聽者(Eavesdropper)起疑心；尤其是對那份唯一加密過的索引檔案，不免懷疑這些檔案之間的關聯性。為了改善 CDES 在實際應用上的這個問題，本論文中，除了基於 CDES 原有文件保護機制，並透過圖像隱藏技術的加入，將索引檔案隱藏在圖像中，藉以達到保護秘密訊息的目標。研究中並實際將此一方式運用在電子郵件服務上，透過改善 CDES，讓訊息傳送過程中，其傳遞內容更具意義以及合理化，讓竊聽者不會對內容產生懷疑。

關鍵詞：欺人耳目的加密文件機制,電子郵件,資訊隱藏,隱匿法

Title of Thesis： Overcoming the suspicion in transmission scheme based on CDES

Name of Institute：Graduate Institute of Communication Engineering,

College of Science and Technology,

National Chi Nan University Pages：78

Graduation Time：06/2010 Degree Conferred：Master

Student Name：Wei-Xun Pan Advisor ：Dr. Quincy Wu

Abstract

Confused Document Encrypting Scheme (CDES) is a document protection technique for data hiding (Steganography), which uses meaningful cheating texts to confuse eavesdroppers. It is a simple and effective file protection scheme. However, CDES has a problem that its plaintext index file (PIF) was transmitted in encrypted form, which looks suspicious. In this thesis, we proposed an improved messaging protection framework based on CDES. As many users include emoticons like smiling faces in their instant messages, it is also natural for many users to include pictures and photographs in their emails. By applying the image-hiding technique to hide the PIF in an emoticon or a photo, emails protected by CDES looks more natural and less suspicious, which could provide a better protection to the secret texts.

Keywords：Confused Documents Encrypting Schemes(CDES), Email services, Information Hiding, Steganography

目錄

圖目錄

表目錄

1. 導論

1.1 現況

由於現今網路的快速發展以及個人電腦的大量普及，人與人之間的文字訊息傳遞方式，也有了極大的轉變。如圖(一)所示，人類通訊的方式，從古代的飛鴿傳書到現今的電話簡訊，以及現今最常用的網路服務電子郵件(Email)與即時訊息(Instant Message)，許多玲瑯滿目的通訊方式，實現了溝通無國界的夢想，縮短了人與人之間的距離。但當有兩個人正在進行極機密的通訊時，基於人類心中的「好奇」慾望，難保不會有第三個人、第四個人正在竊取往來於其間的祕密訊息。

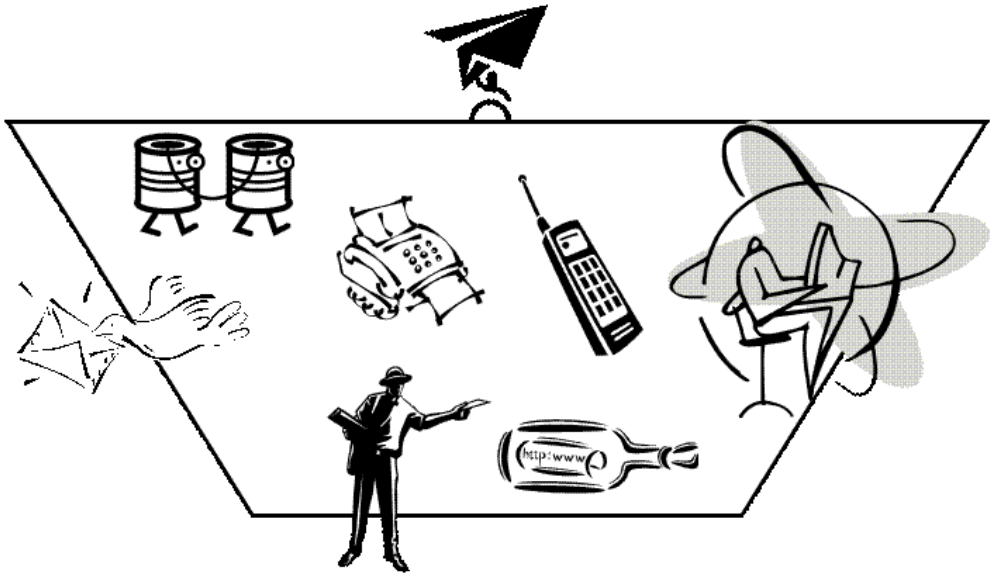

圖 一、通訊方式的演進

因此，為了保護彼此間的通訊是安全無慮的，Information Hiding(資訊隱藏)以及 Cryptography(密碼學)這兩個技術因而誕生，提供使用者一個安全的通訊環境。但對於那些竊聽者(Eavesdropper)們來說，這些保護方式並不像銅牆鐵壁般的牢靠，還是必須透過許多專家不斷的改進以及發展許多的保護機制，來扼止有心人士竊取訊息的意圖。

1.2 研究動機

隨著網際網路的高速發展，Information Security(資訊安全)在現今越來越受到重視，基於這個因素，在本研究中，將設計一個能應用於電子郵件服務上以增加其訊息安全性的架構。

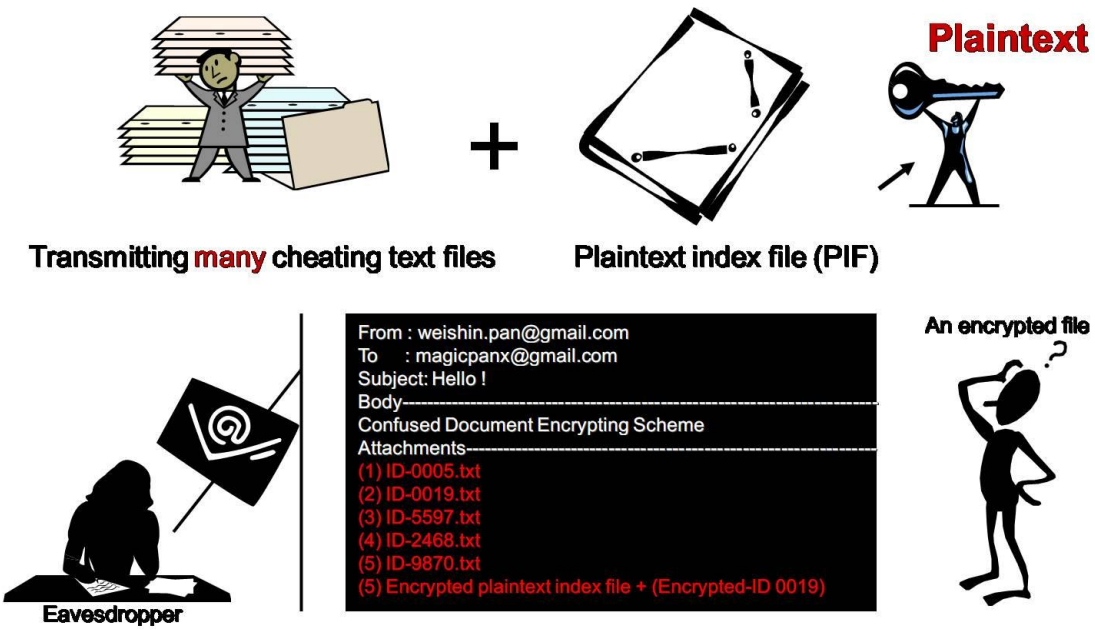

圖 二、傳統 CDES 運作機制以及實際應用示意圖

這個方法的概念是基於 Confused Document Encrypting Scheme (CDES) [1]這個文件保護機制，如圖(二)所示。CDES 文件保護機制主要概念為不直接傳送加密過的秘密文字訊息，而是以一份或多份有意義的欺敵文件(Cheating text)和一份加密過後的索引檔案來替換。發送端在傳送秘密訊息(Plaintext)的時候，會分別傳送每一份欺敵文件代號以及內容；接著再送出加密過後的欺敵文件代號和加密過的索引檔。當接收端收到後，會先解開已加密欺敵文件的代號，當收到正確的欺敵訊息後，就可以解開索引檔案的內容，進而得到其中的秘密訊息。

從上頭可以看出，在這樣的傳送結果下，任何意圖竊取訊息的人，很難發現這些文件中，是哪些文件中隱藏了大量的秘密訊息，或者哪些文件中，又只是存放著一般的訊息。CDES 的特點，就在於用有意義的訊息內容來混

淆敵人，讓人覺得這是份稀鬆平常的訊息。

但是從圖(二)中可以看到，當在電子郵件服務上使用這個保護機制時，可以發現在傳統的 CDES 傳送過程中，需要傳送多份的 Cheating text (欺敵訊息)和一份加密過後的 Plaintext index file (PIF)，其中欺敵訊息是以明文傳送，但 PIF 是以加密後的密文傳送。在這個傳送過程中會讓竊聽者感到懷疑，尤其是那唯一的加密 PIF 文件，更顯示出此次傳送的資料並非尋常，可說是「欲蓋彌彰」。雖然說 PIF 文件已經加密過了，但這會讓有心人士更想要破解出這份文件的秘密以及這些欺敵訊息與它的關聯性，因此在實際應用上 CDES 仍不夠完善。在本研究當中，將改善在 CDES 保護機制中的 PIF 文件的傳送問題。

1.3 現代人網路聊天模式之觀察

大部分的人們，不論是學生或是上班族，在外頭最常用的可能是手邊的手機簡訊服務，加上現在無線行動上網的風氣越來越風行，像是常用的 Wi-Fi,、3.5G,、WiMAX，在手機上進行電子郵件的收發或是即時訊息的傳送都不成問題。但是最輕鬆的方式，還是在放學和下班後，回到家，打開電腦，開啟瀏覽器，看看今天信箱中有沒有收到新的電子郵件(圖三)，工作列的右下角則開啟著即時通訊(Instant Message)軟體(如圖四)，舒舒服服的坐在沙發椅上與人聊天， 並快樂的吃著手邊的零食。

在圖(三)和圖(四)中，可以發現一點有趣的特性，就是許多表情符號加上圖片的多媒體應用。在以前的聊天模式，可能都是以許多文字的組成來當作表情符號。例如說: Orz (表示一個人下跪後採體前屈的姿勢，表示認輸或是失意)，或是在外國人聊天中很常見的 LOL (laugh out loudly)，是採英文單字縮寫後所組成的表情符號，表示大聲的笑。還有更具表達力的方式，

就是透過許多字元組成的表達符號，這在一般的電子佈告欄(Bulletin Board System)中最為常見(圖六)。但這些表情符號並未因為網際網路的高速發展而消失，它們已經在網路族群中佔有不可撼動的地位。

在現今網路族群的組成大都是以年輕人佔絕大部分，所以在聊天方式也趨近年輕化，要求更多采多姿，在表情符號上面來看，根據圖(四)，如一般常見的 Windows Live Messenger (MSN)，在畫面當中可以看出人們喜歡在一句話的最後，加上一個表情符號，而且是越花俏可愛越好，更能增加在聊天時的氛圍，也大幅取代了之前使用文字組成的表情符號。這個方式能夠清楚表達與你聊天的人目前的情緒起伏，例如生氣時，就送出一個正在生氣的表情圖案，對方就知道你現在是處於心情不好的狀態。

以往現實生活中，我們是透過對方的肢體語言和臉上表情來判斷一個人的心情；在虛擬的網路中，也能因為這些花俏的表情符號，更拉近了網路世界中所有人的距離。但缺點則是畫面變得過於花俏，假如彼此的訊息交換速度很快，在訊息判讀上很容易搞錯。另外在訊息交換方面，考慮到了安全性的問題，以即時通訊軟體為例(圖四)，當透過封包擷取軟體 Wireshark，可以從圖(五)看到，文字訊息是以明碼傳送，只要有心都可以看到這些訊息；在電子郵件上也有此疑慮，畢竟網路並不是安全無虞的 ！

圖 三、常用 Web-based mail 使用畫面截圖

圖 四、常用即時通訊軟體操作畫面截圖

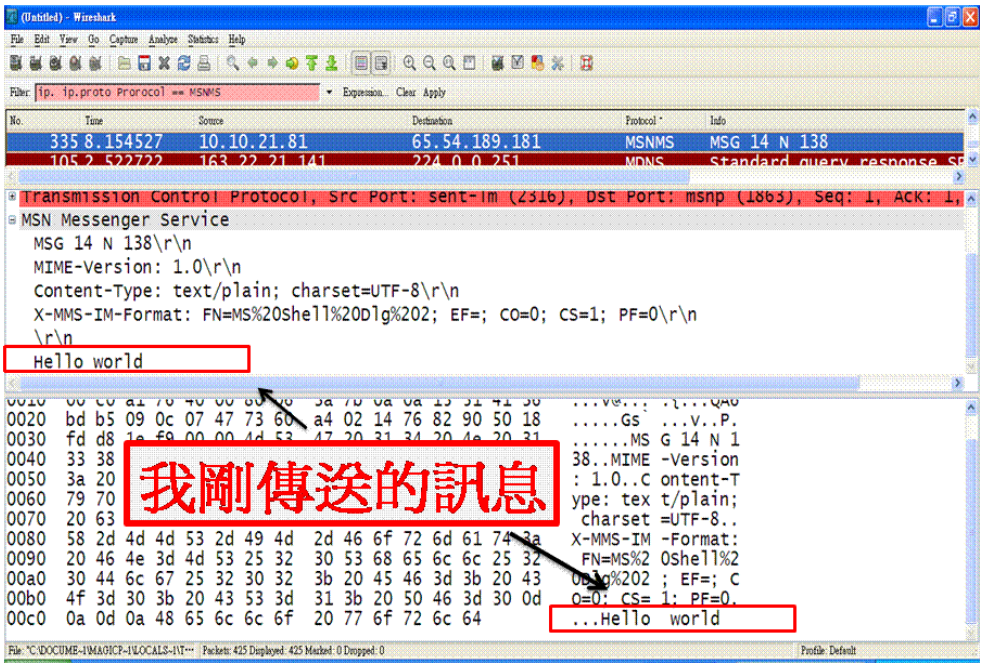

圖 五、透過 Wireshark 來監測 Instant-Message 封包之畫面

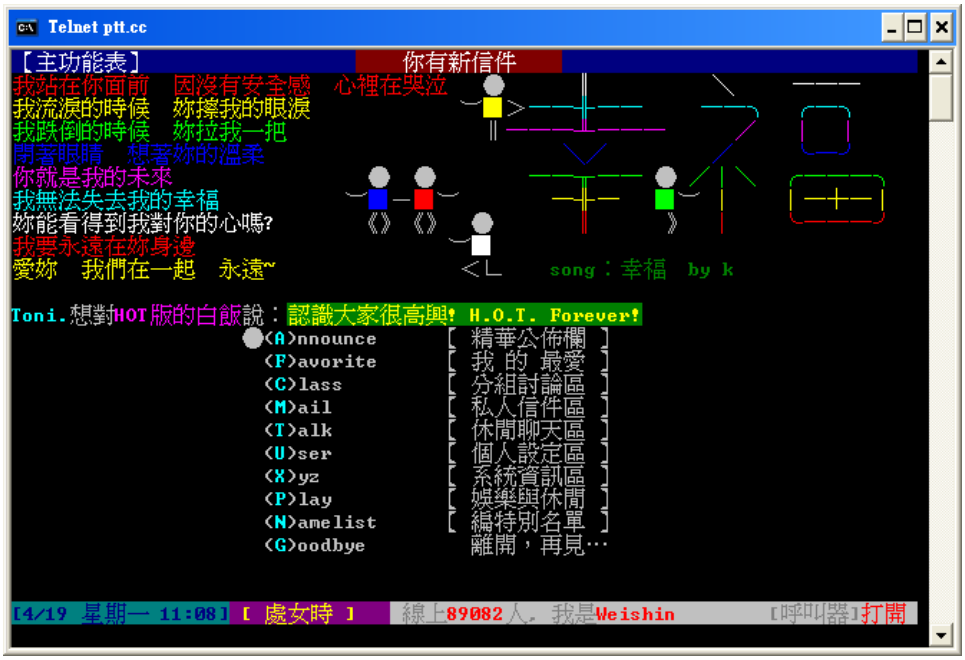

圖 六、電子佈告欄(BBS)常使用的表情符號

1.4 系統概念與目標

所以根據 1.3 節中所觀察到的現象，在電子郵件或是即時文字訊息的過程，有個很特別的地方在於大家使用表情符號的頻率變高了，在一句話的後面加上一個表情，代表著目前的心情，讓整個交談過程更加生動活潑，不再是冷冰冰的文字表達。

所以在本研究所設計的系統中，最主要概念即為結合 Image Hiding(圖像隱藏)的技術，以及為了能夠讓這個保護機制在傳送秘密訊息的過程中可以更加快速，以及解決中文支援的問題，採用了其他研究者在 CDES 上的改善方式，並將它實作在電子郵件這個網路服務上。此外，根據前面章節所觀察到的現象，將 CDES 裏頭重要的 PIF 文件隱藏在圖像檔案或是表情符號中，也不再需要傳送多份欺敵文件的傳遞動作，從外人眼中看似一般正常的聊天行為，不藉由一般的訊息加密傳遞方式，卻能隱藏著許多秘密資訊，為本論文所提出的方法所著重達到的目標。

如圖(七)所示，本系統將在電子郵件的服務中，透過文字加上圖像的型態，將 cheating text 中有意義的訊息內容放置在信件主體之中，將加密過後

的 PIF 文件隱藏在簽名檔圖案或是任何附加的影像檔案，接著寄給接收端後，他就會根據其訊息內容和附加的影像檔案，解開真正的秘密訊息。這一切的傳送行為，對於竊聽者來說，感到並無異狀，並不會對其傳送內容感到懷疑。但是，看似一般的電子郵件交換過程，這裡頭卻能夾帶著大量的秘密訊息。

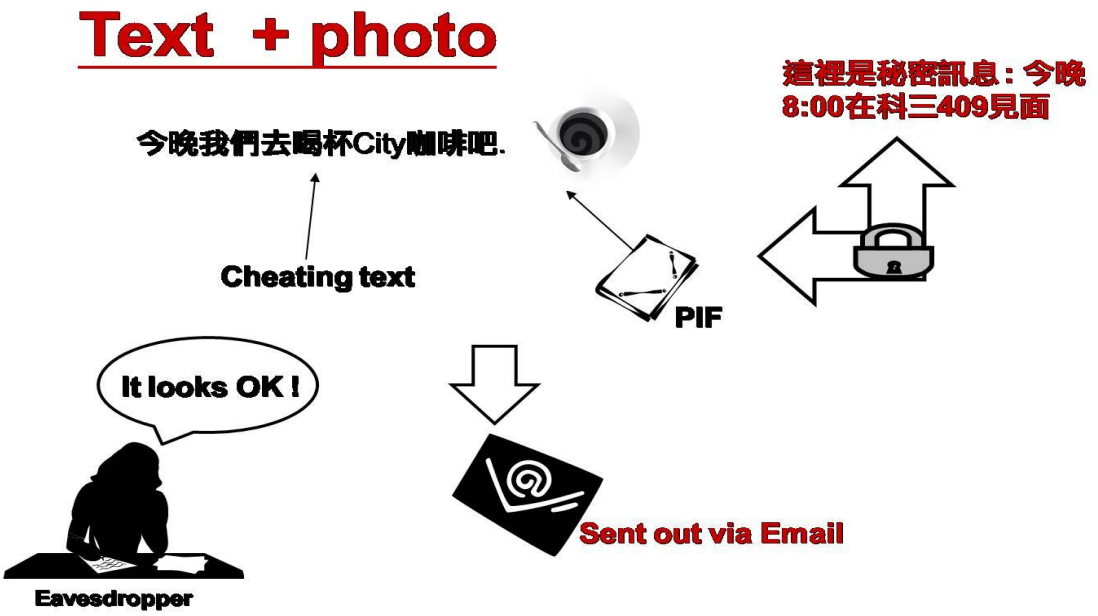

圖 七、系統概念示意圖

2. 背景知識與現有文獻探討

2.1 資訊隱藏技術的演進

大家在武俠小說中是否常看到「無字天書」？它是透過隱形墨水的使用，將絕世武功藏在裡頭；要讀取文字就拿到火上烤一烤，資訊也就出現了。這個概念也就是資訊隱藏技術的核心，將你的秘密訊息隱藏在任何可紀錄的媒介上，透過它到達目的地，並透過特定的方式將秘密訊息還原。如圖(八)所示：

圖 八、資訊隱藏技術所使用之媒介類型

在大部份生活都是數位化的時代，在數位資訊隱藏方面可以參考 Katzenbeisser 以及 Petitolas 所編著[2]和 Bauer 所著[3]，在這本書中他們去對資訊隱藏技術的目的與功能作了一個分類，其中提到 watermarking 與 steganography 這兩大技術：

- **Watermarking**： 嵌入的秘密資訊不可能被移除或是修改，也就是指強韌性，像是現在各國流通的錢幣都有使用這個技術。在許多攝影師的作品中，也都會加上個人版權及浮水印，避免被盜用。
- **Steganography**： 也稱作「偽裝法」強調秘密資訊跟容量與安全性之關係，例如將重要資訊偽裝在風景圖片中，一般人只會看到這個風景圖片，並不會想到秘密就在裡頭。

  Information hiding 的分類架構如圖(九)所示：

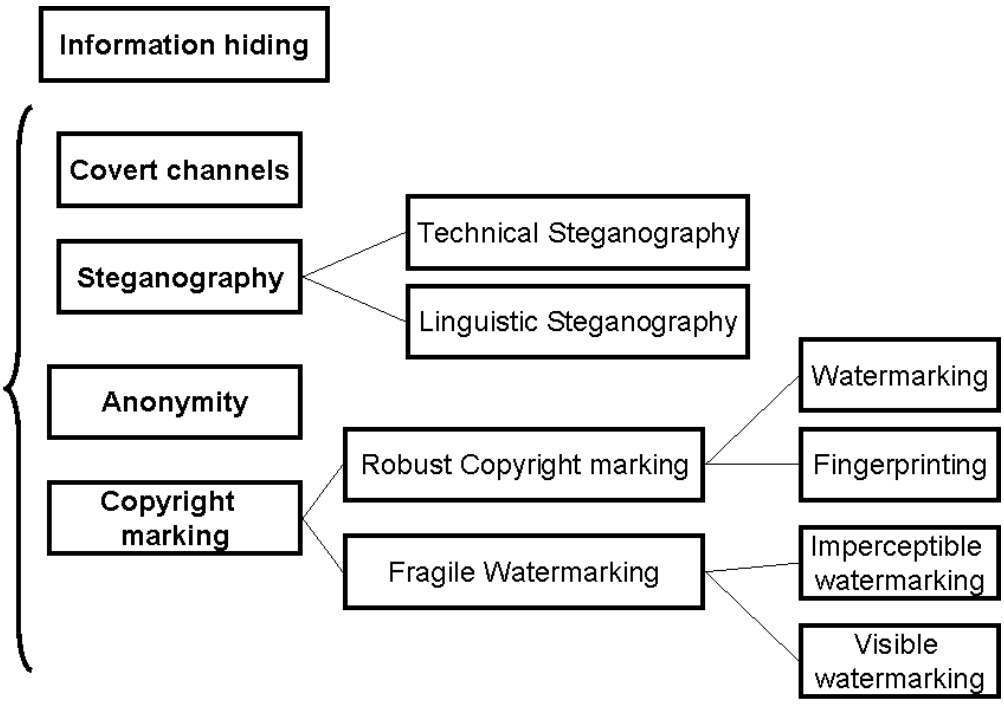

圖 九、Information Hiding 技術的分類表

當要隱藏一段訊息時，必須要考慮到下列幾點：

- 容量(Capacity)： 代表著可隱藏的資料量。
- 安全性(Security)： 當秘密資訊被隱藏在媒介，必須讓別人無法偵測並修改其內容資訊。
- 強韌性(Robustness)： 媒介在經過秘密資訊嵌入的過程中，可承受的資訊變化量。

綜合上面所述，在本研究中，運用了偽裝法(Steganography)，來達成目的，在下面 舉一個例子，它是最典型的文字型的偽裝法：

Apparently neutral's protest is thoroughly discounted and ignored. Isman hard hit. Blockade issue affects pretext for embargo on by-products, ejecting suets and vegetable oils.

這是一份在二戰期間的密文[4]，將文章內每個單字的第二個字母取出 合併後，就可以得到秘密訊息：

Pershing sails from NY June 1.

這個方法的缺點在於無法隱藏夠多的文字資訊，你必須讓要進行隱藏的媒介有足夠的資訊量，字詞要夠多，才能將秘密訊息解開。所以不論是圖片、音樂、文件等何種媒介，資訊量都必須大於秘密資訊所占的儲存空間。

2.2 密碼學

密碼學(Cryptology)，意謂著如何達到資訊的秘密性[5][6]，可解決資訊在傳送過程被竊取的問題。在一開始，密碼學泛指加解密演算法，將明文(Plaintext)透過加密演算法和加密金鑰，轉換成人類無法辨認的內容，也就是密文(cipher text)，接著再透過解密演算法和解密金鑰，解出原來的明文，下面圖(十)是典型的密碼系統架構：

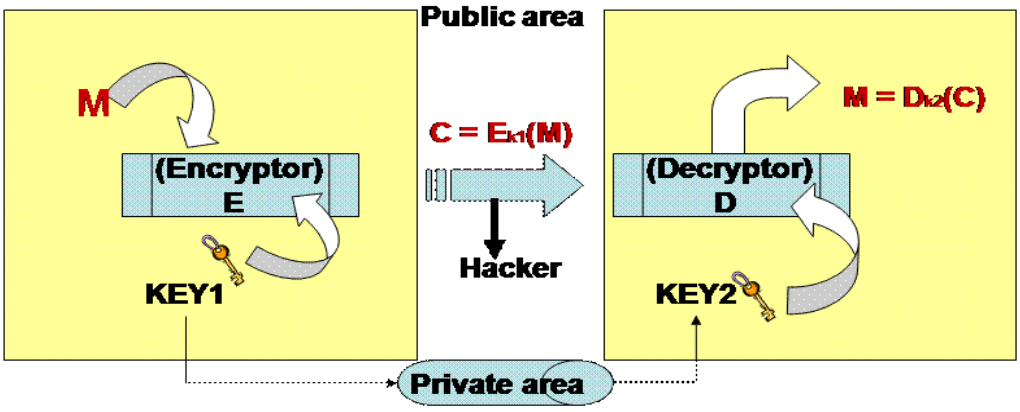

圖 十、對稱式加密流程圖

根據圖(十)，我們可以得知傳送端要將明文 M 送出，必須先透過加密器 Encryptor 與加密金鑰 KEY1，透過下列的加密公式產生密文 C：

加密公式為：$C = E_{k1}(M)$

接著密文 C 會在公共區域上傳送，重要的是解密金鑰必須透過秘密的途徑將它送給對方。雖在公共區域上傳送的密文可能遭擷取，但若缺乏解密金

鑰，它無法被立即解開；然而，這不代表密文 C 永遠不能轉回明文 M，只是必須花費許多時間透過其他方式來進行破解。這部份屬於破密學(Cryptanalysis)的範疇，在本論文中不再多加詳述。

當接收端收到密文 C 後透過 Decryptor 與解密金鑰 KEY2 後，就可以還原明文 M：

解密公式為：$M = D_{k2}(C)$

一個密碼系統的建構，必須基於以下幾點：

- 私密性 (Privacy)：必須能夠阻止不合法的接收者(Eavesdroper)得到明文。
- 可驗證性(Authenticity)：必須確定所接收的資料是否來自正確合法的發送端，而不是由惡意的第三方所偽造。
- 完整性(Integrity)：必須確定接收到的資料沒有經過惡意的第三方進行竄改。
- 不可否認性(Non-repudiation)：發送端對於傳送過的資料，無法否則是由其所送出。

在典型的密碼系統架構中，只有合法的傳送端與接收端共享同一把秘密金鑰，這架構稱之為對稱式加密系統(symmetric Cryptosystem)，或是稱之為秘密金鑰加密系統(Secret-key Cryptosystem)。它的特色在於加密和解密的速度很快，但安全性上面，例如以不可否認性來說，假如接收端有筆資訊，因為雙方都知道對方的金鑰，因此傳送端可以說他根本沒傳過任何資訊，因為接收方擁有金鑰可以去進行竄改以及偽造資訊的動作，除非有第三方介入，否則無法確保其不可否認性。

另外在金鑰的傳送部分，也必須透過秘密的傳送方式交至對方手裡。但要如何達成？這在密碼學中屬於 Key Distribution 的範疇，有相當多的特殊技巧。不過在本研究中，預設為金鑰已經由某種安全的機制傳送到接收端。有興趣的讀者可參考相關的文獻。

相較於另一種架構「非對稱加密系統」(Asymmetric Cryptosystem)，又可稱為公開金鑰加密系統(public-key Cryptosystem)。它與對稱式加密系統最大的差異在於使用不同把金鑰進行加密與解密，但是加密/解密的運算速度也比對稱式加密系統來的慢。這類型的加密系統的概念源至 1976 年 Diffie 和 Hellman 所提出[7]，雙方各自產生一組 Key(Public-key 和 private-key)，之後雙方互相交換其 public-key，接著就透過對方的 public-key 加上自己的 private-key 產生一把共同金鑰，密文就可以透過這把共同金鑰去產生明文，明文則可以透過對方的 public-key 來產生密文。這樣的加密程序解決了對稱式加密系統中共同金鑰的運作機制，也不用擔心這把共同金鑰落入第三個人的手中，因為在非對稱式系統中，資訊竊取者得到 public-key，但關鍵的 private-key 卻在傳送的雙方身上，因此無法產生可以解開密文的金鑰。

2.3 Blowfish Algorithm

在 2.2 節密碼學的部分介紹了傳統加密系統與金鑰之間的基本概念。在本研究中，採用了對稱式加密系統的架構，並使用了由 Schneier 在 1993 年所發表的免費加密演算法 Blowfish [8][9]，它是採區塊式加密法(block cipher)，對一個固定大小區塊的明文或者是密文作加密/解密，其區塊大小為 64 個位元，金鑰長度在 32 至 448 個位元之間。在加密結構上則是屬於在 1970 年，Horst Feistel 所提出的 Feistel network[10]。

另外，Blowfish 演算法當初會被設計出來，是想達到易於實作以及運算速度快的特性，並取代舊有的 DES(Data Encryption Standard)[11] 和 IDEA (International Data Encryption Algorithm)[12]。Blowfish 演算法會進行 16 回合的加密運算，如圖(十一)所示：

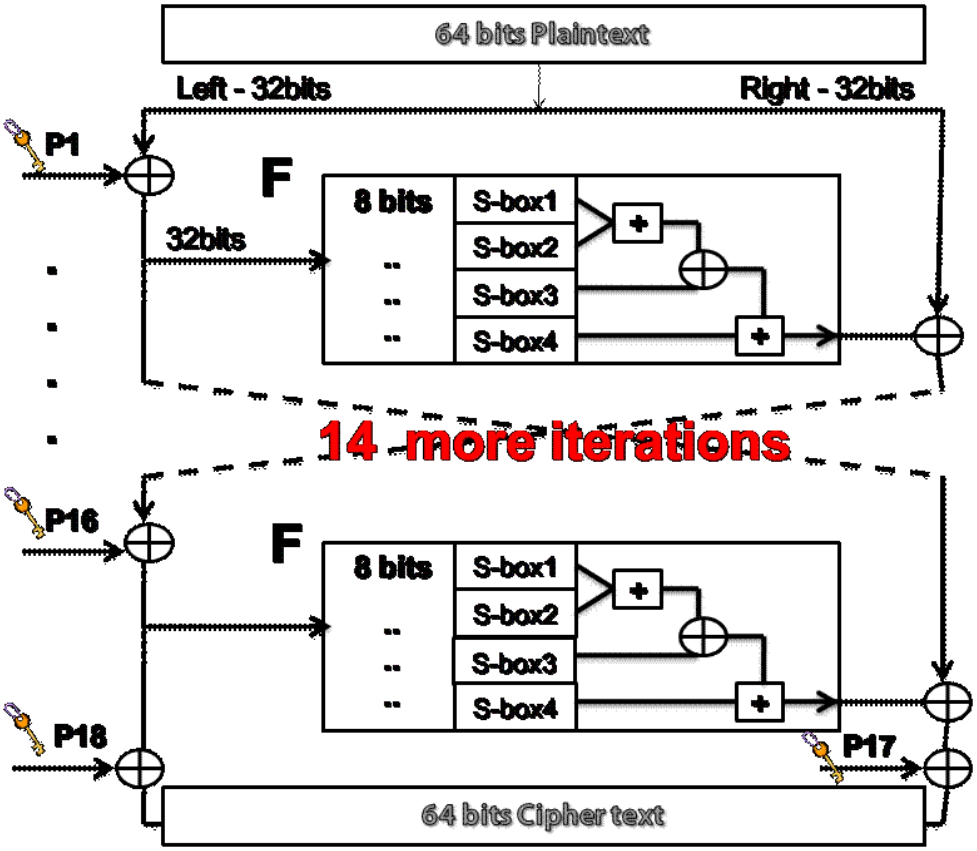

圖 十一、Blowfish 演算法加密過程

輸入區塊大小為 64 個位元的明文，分成左右兩邊 32 個位元進行，接著金鑰(原先欲設定的密碼)經過 sub-key generator 後會被用來產生 18 把 32 位元的 sub-key，左右兩邊各 32 個位元的資料會與 sub- key 進行加密運算，每跑一回合就使用一把 sub-key，直到全部用完，接下來會進入 F(function)這個運算式，在裡頭將分成四個區塊各 8 個位元，進入裡面的 S-BOX (Substitution Boxes)後，會進行 ADD 以及 XOR 的加密運算，再與另外半邊的 32 個位元的資料算進行 XOR 運算，運算結果再與下一回合的另一邊 32 個位元的資料進行運算，重複 16 回合的 F 函數加密運算後，以及與最後兩把 sub-key 做 XOR 加密運算，運算完的結果，就可以得到 64 位元的密文。另外，在加解密的運算中所使用的加法與 XOR，在 Blowfish 中，並不具有代數的交換性，所以可以增強其演算法的安全性。

2.4 LZMA Algorithm

資料壓縮(Data Compression)這個技術，以日常生活來比喻的話，鋁箔包如果不去壓扁，垃圾桶很快就滿出來，所以為了提升有限容量的可用存放空間，就必須做壓縮(Compress)這個動作。在電腦上來說，為了節省儲存空間，我們會透過壓縮軟體去對資料做壓縮，或是進一步去做加密的動作，提升資料的安全性。在本研究中，為確保資料壓縮過後的正確性，採用無失真壓縮演算法 LZMA。

LZMA(Lempel-Ziv-Markov chain-Algorithm)，在 2001 年發展出來，屬於字典型編碼法(dictionary coding)，例如著名的 Open source software 7-Zip 這套解壓縮軟體中的.7z 格式就是採用 LZMA 來作為它的壓縮方式，有著高壓縮率以及快速的解壓縮速度。另外為了推行 LZMA，7-zip 官方網站也提供了 LZMA software development kit (SDK)，讓更多軟體開發者能快速的使用此套免費的壓縮技術。

字典編碼法的概念，以現實生活的例子來說，要對一組字串 hello world 作編碼時，當我們手中有一本字典，假設我們可以在字典的第 298 頁的第 8 個單字找到 hello，在第 197 頁的第 2 個單字可以找到 world，之後就以代碼(298, 8) (197,2) 來表示 hello world。當對方收到這組代碼後，依據同一本字典，就能順利對這組代碼作解碼，並取得原字串。

在字典編碼中，運作模式可區分為靜態和動態。靜態就是，在字串壓縮前先建立好固定不變的英漢字典，兩個人都必須使用同一本字典，這種方式最大的好處在於，可以針對你所要壓縮的資料來建立一本字典。缺點是都需要買同一家出版社發行的英漢字典，在電腦上運作時，也就需要將建立好的字典檔傳給對方；未來字典檔有任何修訂，都必須同步更新。

而動態的方式，在一開始沒有預設好的字典檔。壓縮一組字串時，首先切割其字串，並讀入第一個單字，接著從一本即時產生的空白字典找尋有沒有這個單字。假如該單字已存在，就取得此單字在字典中的代碼，若不存在，就將它加入字典中，接著再繼續讀下一個單字。最後編碼成新的字串再送給接收端，接收端並不需要切割字串，也不需要與字典作比對，只需將輸入的字串分別轉換為字典中的代碼或是未經編碼的純字串，再將新的字串加入字典中，最後將代碼轉換成單字輸出。

大多數的字典型編碼法，像是著名的 LZ 系列 LZ77[13]和 LZ78[14]都是屬於動態建立字典檔，LZMA 也屬於這種模式。

LZ77 是由 Ziv 和 Lempel 在 1977 年發表，它是一邊讀取資料一邊建立字典檔，然後新加入的資料比對前面建立的字典檔的方式。LZ77 是使用滑動視窗(sliding window)的機制，如圖(十二)所示。假設有個人常搭火車，他透過一個窗戶去看外面的風景，接著大腦會將看到的畫面記下來；當他哪天又再經過一次，腦海瞬間會浮現那時所看到的，說著：「這景色我看過了，這裡是台中!!」。

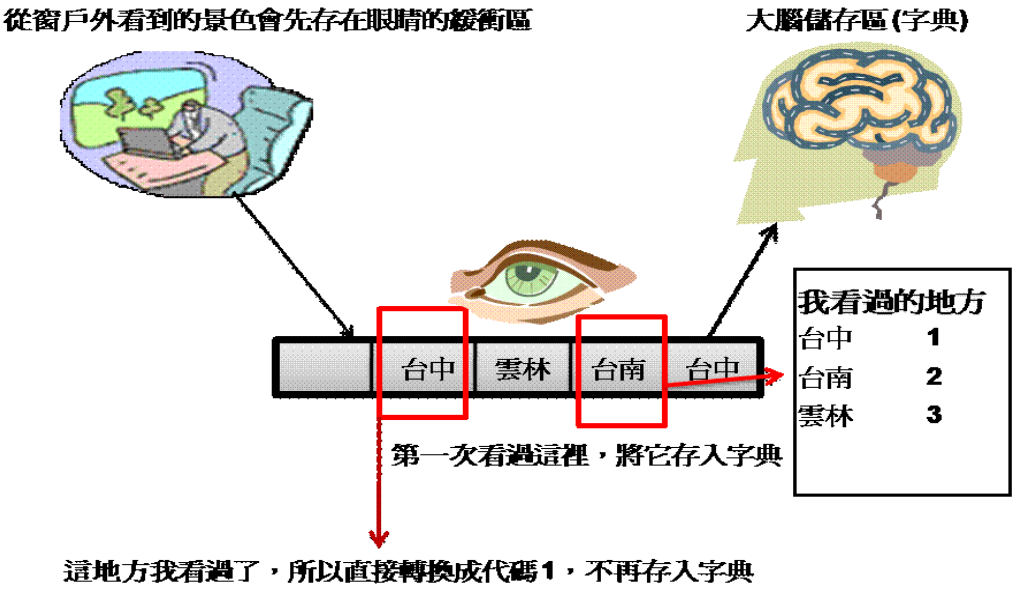

圖 十二、滑動視窗機制示意圖

使用前面剛讀取的輸入，假如單字是第一次出現就放進字典，再來又輸入了重複的單字就以字典內中的代碼取代它，完成壓縮編碼的動作。

LZMA 是基於 LZ77 所發展出來，但加入區間編碼(Range coder)的技術。區間編碼(Range encoder)是 7-zip 用來避免運算編碼法(Arithmetic coding) [15]的專利問題所開發出來的。壓縮原理為將一整個字串視為一個介於 0 與 1 之間的區間來表示([0,1))，接著透過每個字元的機率分佈，並切割成許多子區間。所以當訊息越長，子區間會越短，相對來說就需要更多的位元數來表示這個子區間，對於字元出現的機率差異越大的資料效能越好。這兩者的結合也創就了 LZMA 的優異壓縮效能。

2.5 CDES

Confused Document Encrypting Scheme，簡稱 CDES，這個文件保護機制在 1998 年被 Lin & Lee 所提出[1]，透過多份的欺敵訊息(cheating text)，混淆想從中攔截訊息的有心人士，最重要的秘密訊息 (plaintext)則是透過特定的欺敵訊息和索引檔反解回來。以下敘述 CDES 的運作機制。

- CDES 系統架構的重要元素
  - Plaintext (秘密訊息)： 要隱藏的真正資訊。
  - Cheating text(欺敵訊息)：用來混淆敵人的訊息。
  - CPT (Character position table)：藉由欺敵訊息所得到的字元頻率以及索引值範圍的對應表格
  - PIF (Plaintext index file)：秘密訊息透過 CPT 位置索引值所形成的索引檔，也是未來要傳送給接收端的重要檔案。
  - KEY： 金鑰 1 負責 PIF 的加密，金鑰 2 負責 ID 的加密。

- CDES 在傳送端的運作流程

1. 首先輸入祕密訊息和包含足夠字元的欺敵訊息

首先設定想要的 Plaintext(秘密訊息)內容：

Plaintext : Cat is my Pet.

{C, a , t , i , s , m , y , p , e , . , □}

接著輸入想要的 Cheating text(欺敵訊息)內容：

Cheating text : Computer security is important.

{C, o , m , p, u , t , e , r , s , c , i , y , a , n , . , □}

透過欺敵訊息裡頭的所有字元產生 CPT，以字元的出現順序去依序給予其欄位 Position record 一個值，如表(一)所示：

表 一、Character's position table

Character	Position record
C	1
o	2 , 25
m	3 , 23
p	4 , 24
u	5 , 13
t	6 , 16 , 27 , 30
e	7 , 11
r	8 , 14, 26
s	10 , 20
c	12
i	15 , 19 , 22
y	17

a	28
n	29
.	31
□	9 , 18 , 21

2. 接著 Plaintext 透過 CPT 產生 PIF，隨機從裡頭取得位置索引值，另外使用任何壓縮演算法去壓縮 PIF，其內容如下：

1 28 16 21 15 20 18 3 17 9 24 7 6 31

3. 隨機產生 ID 給每一份欺敵訊息。因為有多份欺敵訊息，但只有其中一份會用以進行加解密。這份欺敵訊息的 ID，在本論文中是使用 128 位元的 IDEA 對稱式加密演算法，以金鑰 2 去對 ID 作加密後，加入 PIF 裡頭：

ID ： 236785

Computer security is important.

4. 使用金鑰 1，對壓縮過的 PIF 進行加密。
5. 送出所有的已設定 ID 的欺敵訊息以及加密過後的 PIF 檔案。

圖(十三)為 CDES 在傳送端的處理流程：

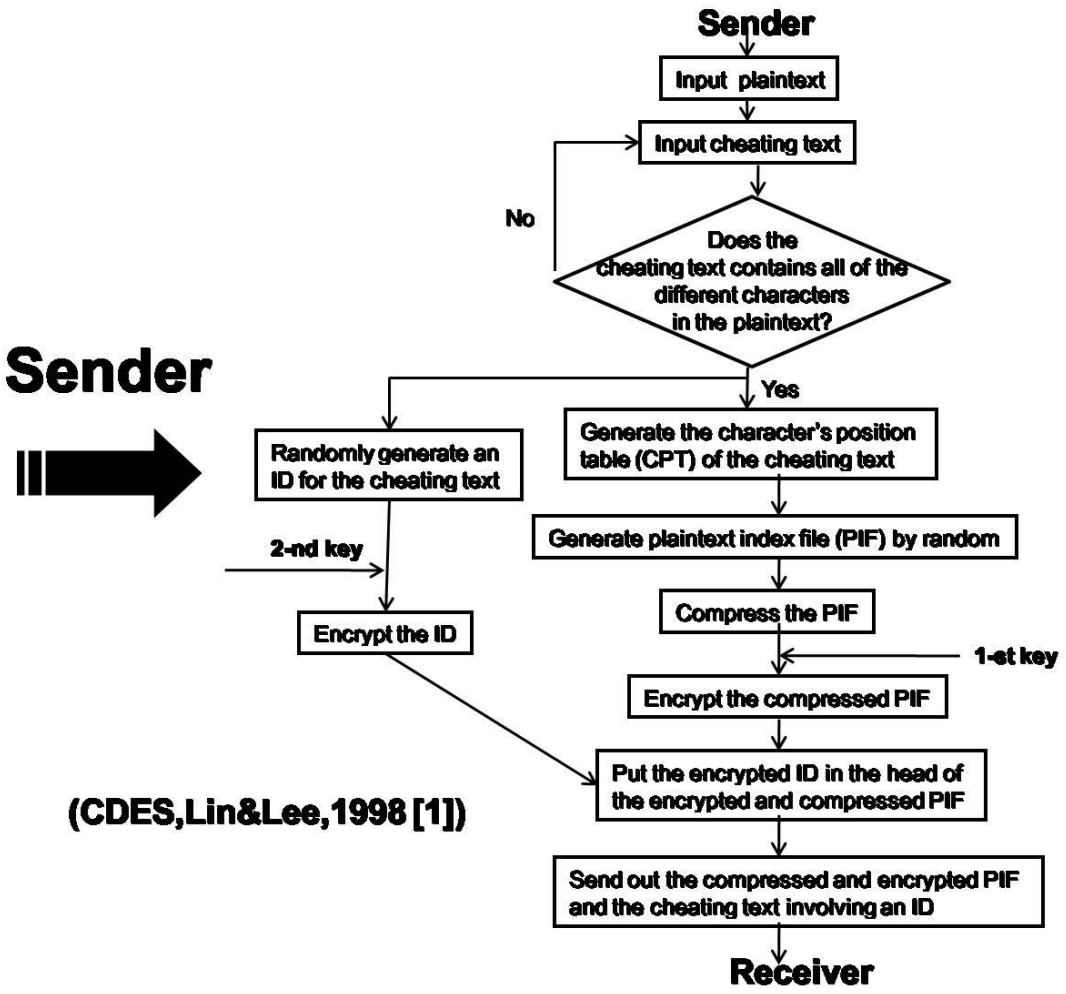

圖 十三、傳送端流程圖

- CDES 在接收端的運作流程，如圖(十四)所示：
    1. 對收到的 PIF 進行解密，先使用金鑰 2 將裡頭的欺敵訊息 ID 解開。
    2. 接著搜尋接收到的欺敵訊息中，有沒有符合這組 ID 的檔案，有的話，使用金鑰 1 去對 PIF 解密。
    3. 透過對應到 ID 的欺敵訊息，去產生 CPT

       Cheating text : Computer security is important.
    4. 對 PIF 進行解壓縮
    5. PIF 透過之前產生的 CPT 將原來的秘密訊息解出，接著就

可得到：

Plaintext : Cat is my Pet.

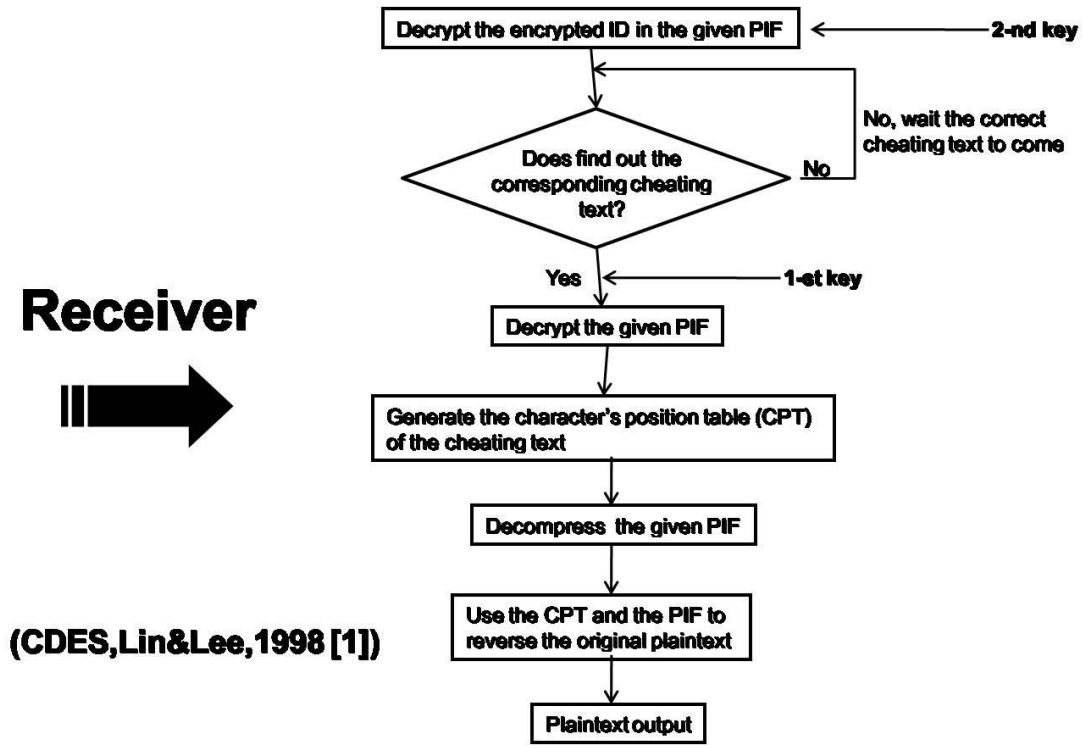

圖 十四、接收端流程圖

2.6 CDES 之應用評估與相關改善研究

最初，在 Lin & Lee 所提出的 CDES 當中，一直存在著幾個問題，所以有許多研究者提出了許多改善方式讓這個文件保護機制更臻完美：

1. **只支援英文字型**，因為在 CDES 中，它所涵蓋的字元範圍是 0~127 (ASCII)，而中文字型(BIG5)常用字約有 5,401 個，次常用國字 7,653 個，符號 471 個，共計 13600 個字。後來在 Yen & Hwang[16][17] 的改善研究中，藉由使用中文內碼的特性，將中文字轉換成十六進位碼來進行處理，所以在 CPT 所出現的字元範圍被簡化成 0~9 以及 A~F 之間，也成功解決 CDES 原先無法支援中文的問題。
2. **欺瞞訊息中必須包含秘密訊息的所有字元**，因此在[18]中，Liang 等人提出了一個改進方式，在產生 CPT 的過程中，以十六進位碼去進行處理，並自動補齊所缺少的字元(A~F - 0~9)。不管欺敵訊

息選擇為何，都能包含秘密訊息的所有字元，就不會有缺字的問題存在。

3. **PIF 檔案過大的問題**，在 CDES 這個文件保護機制中，一直存在著的問題，就是 PIF 大小是原始秘密訊息的 4~5 倍以上，所以必須藉由壓縮 PIF，來減少 PIF 的儲存空間，後來 Yao[19]所提出的方法，成功將 PIF 大小縮減為原始資料的 2 倍，他在檔案處理的部分並沒有使用 file I/O，而是使用 binary file I/O 來產生 PIF，並在 CPT 的產生方式中，忽略字元出現頻率大於 16 的索引值，以將索引值的範圍成功地限制在 0 至 255 之間。
4. **在欺敵訊息上的取得問題**，前面提及到的[1][16][17][18]，都是採一份加密過的 PIF 和多份欺敵訊息分開傳送的方式，藉此達到混淆敵人的目的，在[19]則是接收端必須透過傳送端所提供的加密後的 URL(Uniform Resource Locator)中，去取得欺敵訊息文件。
5. **實際運用的難易度**，因為 CDES 都必須傳送多份欺敵文件加上一份加密過後的 PIF 文件，這唯一的 PIF 文件的傳送行為對想要竊取訊息的人來說，是很可疑的，在許多欺敵文件傳送時也會造成網路頻寬的浪費，在實際運用上也多了許多疑慮。

3. 系統架構

3.1 系統基本模型

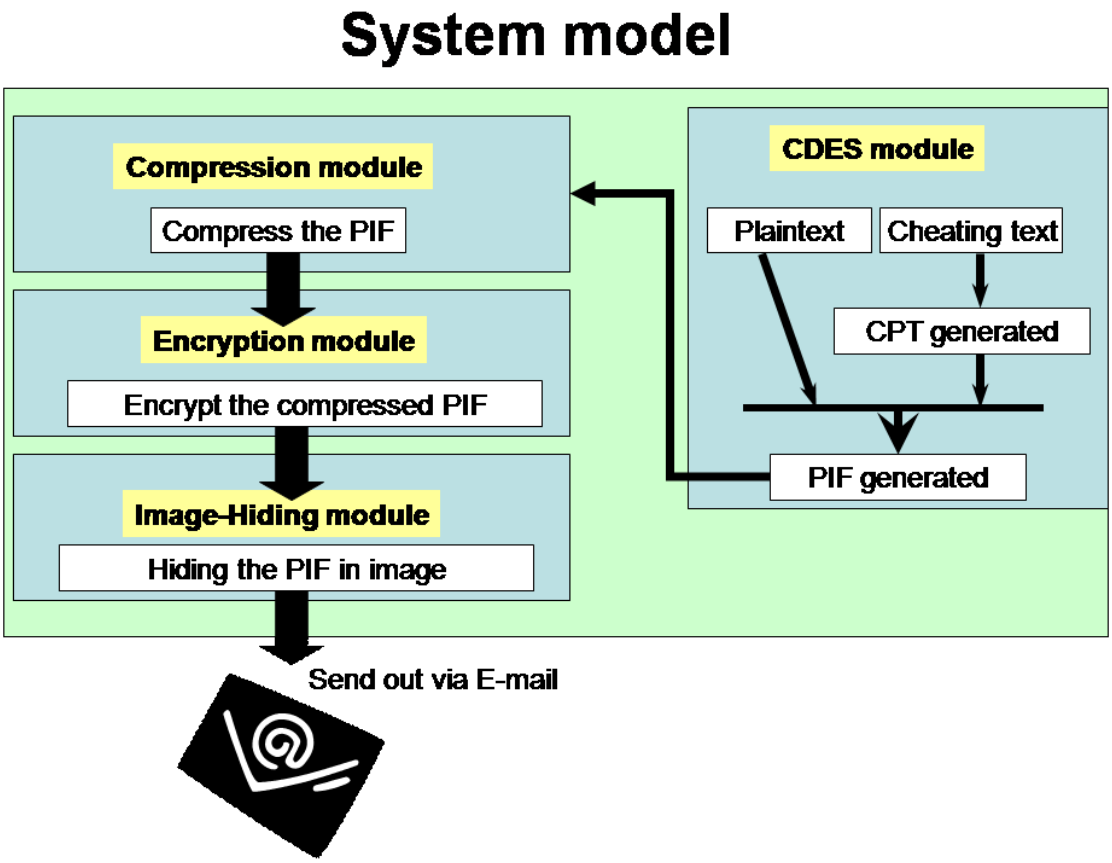

圖 十五、系統模組架構圖

如圖(十五)所示，本論文所提出的系統，由四個模組所組成，依序為：CDES module、Compression module、Encryption module、Image-Hiding module。

傳送端從 CDES module 開始處理 Cheating text(欺敵訊息)和 Plaintext(秘密訊息)的內容，並產生 CPT 和 PIF，接著再進入 Compression 和 Encryption 模組中，進行壓縮和加密，最後就是 Image-Hiding module，這模組會將 PIF 文件隱藏在圖片或表情符號中。當前面這些工作都完成後，再透過郵件使用者代理程式(Mail User Agent, MUA)將圖片附加在信件當中，讓它看起來像是個平常的簽名檔，或是一般的夾帶圖片，並且將 Cheating text 輸入到信件的主體(Body)當中。撰寫完成並寄出後，接收端再透過這個系統所提供的 CDES 模組，將秘密訊息解密出來。

3.2 系統模組介紹

3.2.1 CDES Module

這個模組是基於原始的 CDES 文件保護機制[1]以及後來所其他研究者提出 CDES 的改善方式[16][17][18][19]，去進行開發修改的。另外，本研究中並沒有去對欺敵文件作一個隨機編號的動作，因為欺敵文件只需要一份，就能包含秘密訊息的所有字元，所以不需像原始的 CDES [1]採用許多份欺敵文件。

相關的程式碼檔案如以下表(二)所列：

- 傳送端

表 二、傳送端相關檔案

Source code	Function & Note
Encryption.cpp	系統起始點
CPT.cpp	**`CPTcreate();`** 實際執行處
CPT.h	void **CPTcreate**(unsigned char *buf ,unsigned char *buf2, int length,int length2) //產生 CPT

如表(三)與接收端有關的程式碼檔案的有：

- 接收端

表 三、接收端相關檔案

Source code	Function & Note
Decryption.cpp	**`Decrypting();`** 實際執行處
Encryption.cpp	系統起始點
DECRYPTION.H	`void` **`Decrypting();`**

程式運作流程如下：

1. 設定 Cheating text file 所要顯示的欺敵訊息。
2. 讀取 Cheating text file 並轉換成十六進制碼(大寫)處理。
3. 設定 Plaintext file 所要隱藏的秘密訊息。
4. 讀取 Plaintext file 並轉換成十六進制碼(大寫)處理。
5. 透過 Cheating text 產生 CPT。
6. Plaintext 內的所有十六進制的字元，透過與 CPT 的字元比對，取得隨機的索引值。
7. 取得各個字元的隨機索引值後，將其存入 PIF 檔案中。

首先，在 Character position table 的產生方法上基於[17][18][19]所發展出來，有幾個條件限制。

例如在下面的範例中，輸入的 Cheating text 內容為：

哈囉! Weishin Pan,今天天氣很好呢.

再來將 Cheating text 內容轉換成十六進制(大寫字)

ABA2C56F21205765697368696E2050616E2CA4B5A4D1A4D1AEF0ABDCA66EA94F2E

下面表(四)是此段 Cheating text 所產生的 CPT 表格：

表 四、Character Position Table(CPT)

字元	字元出現頻率	最小索引範圍值	最大索引範圍值
A	9	1	9
B	3	10	12
2	6	13	18
C	3	19	21

5	5	22	26
6	10	27	36
F	3	37	39
1	4	40	43
0	4	44	47
7	2	48	49
9	3	50	52
3	1	53	53
8	1	54	54
E	5	55	59
4	4	60	63
D	3	64	66

根據表(四)可以得知：

- 在字元這個欄位組成由 0~9 和 A~F 所組成，若在 Cheating text 缺少任一個字元，則自動補進欄位。
- 在字元出現頻率這個欄位，出現次數必須小於等於 16，假如超過 16 就除以 16 取餘數作為其值。
- 在最大與最小索引值範圍這兩個欄位，範圍由於出現次數被限制的關係(16 個字元*最大字元出現次數 16)，所以索引範圍值在 1~256 之間。
- 這樣的 CPT 產生方式，用意是在於將 Cheating text 轉換成十六進位碼表示後，就能夠包含 0 到 9 和 A 到 F，就能隱藏所有的秘密訊息字元。至於限制字元的出現頻率以及索引值範圍大小的結果，則會控制到根據 Plaintext 所產生的 PIF 文件大小。

再來是 Plaintext index file 的產生：

1. 在 Plaintext 的部分，輸入

今晚到科三 409 見,ok?

2. 再來將 Plaintext 轉換成十六進制(大寫字元)

A4B5B1DFA8ECACECA454343039A8A32C6F6B3F

3. 與前面 Cheating text 所產生的 CPT 作比較後，每處理一個符合的字元就給予一個範圍內隨機取得的索引值，每次輸出為一個索引值並以 Tab 格開，輸出至 PIF 中：

8	61	11	23	11	41	65
38	8	…(以下省略)				

3.2.2 Compression Module

這個模組提供了壓縮檔案的能力，主要是針對占用龐大容量的 PIF 檔案進行壓縮。實作中選擇 LZMA 這個字典型壓縮演算法的原因，在於 PIF 內的元素，大都是有機率會是相同的值，也就是同個字元的情況下，有可能是相同的值。所以當 Plaintext 中越多重複的字串，越能發揮其高壓縮率的特性。

在這部分的程式開發，是以 Lloyd 的 easylzma project[20]進行修改，它是屬於開放式原始碼，可在 windows 平台進行開發。

不過要提到一點就是，實際上在 LZMA 的壓縮和解壓縮都是透過 7-zip (開發作者是 Igor Pavlov)所提供的 LZMA SDK[21]來進行呼叫，它所使用的版本為舊的 4.32 版，目前官方網站已經發布 9.12 版，提供了新的壓縮演算法。不過舊版的 LZMA SDK 已能滿足本研究目前的壓縮需求，並不需要一些額外壓縮演算法的支援，所以並沒有去針對這部分

取得最近的 SDK。相關程式檔案如表(五)所列。

表 五、壓縮模組相關檔案

Source code	Function & Note
compress.h	LZMA API
decompress.h	LZMA API
common.h	定義壓縮/解壓縮函式的標頭檔
LZMA.h	`void` **`doCompress();`** // **壓縮** **`void doDecompress();`** **// 解壓縮**
easylzma.dll	動態連結程式庫

3.2.3 Encryption Module

這個模組主要提供檔案加密的功能，使用 Blowfish 這套對稱式加密演算法，它具有加密速度快，容易實作，配備等級要求低的優點，目前，也有許多研究將這個演算法運用在嵌入式系統。

在開發上，則是基於 Ivan Vecerina 的 coopfish project[22]，它主要是針對 blowfish 的檔案加密/解密開放式原始碼。

另外，在加密金鑰的長度上，由於 blowfish 的密碼長度在 32-448 位元，所以最少必須要有 8 個以上的英文或數學字元。在本研究中，對於金鑰的設置以及如何取得，都已經採用預設值(固定金鑰，並且透過一個安全的途徑，將金鑰傳給接收端)，以下不再贅述。

相關的程式碼檔案如下表(六)所列：

表 六、加密模組相關檔案

Source code	Function & Note
blowfish.cpp	**encrypt_CBC();decrypt_CBC();執行處**

blowfish.h	**//區塊處理模式預設使用 CBC mode**
	void **encrypt_CBC**(Pad const& pad, void const* src, void* dst, size_t byteSize, Block* pChain=00);
	void **decrypt_CBC**(Pad const& pad, void const* src, void* dst, size_t byteSize, Block* pChain=00);

Encryption.cpp	enum { BF_NONE, BF_ENCRYPT_MODE, BF_DECRYPT_MODE } bfmode = BF_NONE; **//設定 bfmode 是要加密還是解密** int **blowfish_MODE**(char mode)

在這裡有一點要說明，根據上面的 blowfish.h，可以看到有兩個加解密函式，分別是 decrypt_CBC()和 encrypt_CBC()，在區塊處理模式中[5]，可分為 ECB 模式(Electronic Codebook Mode)、CBC 模式(Cipher Block Chaining mode)、CFB 模式(Cipher Feedback mode)，OFB 模式(Output Feedback Mode) 。在 coopfish 中也提供了前面三種模式，其中以 ECB 和 CBC 模式最常被使用，下面將詳細說明這兩種模式。

要談 CBC 之前必須先提到 ECB 這個最基本的模式。

- ECB mode

  明文 M 表示成許多獨立的區塊 M = m1+m2+m3+….+mN，其每個區塊固定大小為 64 個位元。若一部分的位元組無法湊滿 64 個位元，則採取填補(Padding)的機

制，在最後的部分加入空白字元或是其他更複雜的填補方式。在每個獨立的區塊中，是透過 Key 與明文 M 的運算產生了密文 C，而每個區塊都是獨立運作，皆沒有關聯性。缺點是，假如同時出現相同的區塊內容，這樣表示加密後的結果，兩者也會相同，也容易被進行分析和破解密碼的動作。

- CBC mode

  CBC 的出現，是為了解決 ECB 在上面所提及的問題，明文 M 表示成許多獨立的區塊 M = m1+m2+m3+….+mN，其中每個區塊固定大小為 64 個位元，與 ECB 模式一樣，都具有填補(padding)的機制。

  透過區塊串接(Block chaining)的方式將每個獨立區塊連結在一起。產生一組起始向量的隨機數字 IV(Initial Vector)，在起始區塊中與明文 m1 作 XOR 運算後，再透過金鑰去產生密文 C。

  下一塊區塊要處理時，則是透過前一個區塊所產生的密文 C 與下一個明文 mN 進行 XOR 運算，再透過金鑰再去產生下一個密文 C，以此類推。

3.2.4 Image-Hiding Module

影像隱藏模組(Image-hiding module)提供了將重要的 PIF 檔案隱藏在圖片中的能力，在這裡透過了由 Allan Latham 開發的 JPHS[23]，它提供了圖片類型為 Jpeg 的檔案隱藏以及資訊取出的功能，它也是一套開放原始碼的工具，也是使用了 Blowfish 這套加密演算法。其相關程式檔案如表(七)所列。

表 七、影像隱藏模組相關檔案

Source code	Function & Note
Jphide.c	static int **jphide** (const char* infilename, const char* outfilename, const char* hidefilename)
Jpseek.c	static int **jpseek** (const char* infilename, const char* seekfilename)
encryption.cpp	此部分程式已將 JPhide 和 JPseek 整合進去

在 JPHS 這個工具中主要是透過 JPhide 和 JPseek 這兩個程式所組成。JPhide 的功用在於將檔案隱藏在圖片中，JPseek 則是將隱藏於圖片內的檔案取出。JPHS 的視窗版 JPHSWin 當中，會建議圖片可隱藏的資訊量，例如 ncnu_original.jpg，檔案大小為 33kb, 長寬比為 350 乘以 350 像素(pixels)，JPHSWin 會計算最大可容納資訊為 5kb, 建議值則是 3kb，實際的操作畫面如圖(二十四)所示。

JPHS 程式處理流程如下，首先看到的是 JPhide 和 JPseek 的指令參數

jphide.exe input-jpeg-file output-jpeg-file hide-file
jpseek.exe input-jpeg-file output-seek-file

1. 選擇一張你要將 PIF 隱藏在裡頭的圖片，這張圖片大小必須大於你所要隱藏的祕密資訊
2. 接著使用 JPhide 對圖片進行 PIF 的隱藏動作，先輸入一組金鑰進行 blowfish 的加密動作，輸入完之後，再進行一次金鑰確認，即可隱藏成功。
3. 當要解出這圖片裡頭的 PIF 時，使用 JPseek 來進行取出的動作，也必須輸入加密時的金鑰，才能解開它。

4. 實驗與環境設定

4.1 建置實驗環境

本章說明實驗環境。系統設置如下列表(八)所示：

表 八、實驗環境設定

Hardware
PC-1 CPU ：Intel Pentium 4 - 3.00GHZ RAM：2GB
Operating system(OS)
Microsoft windows XP professional version 2002 Service Pack 3
Integrated Development Environment(IDE)
Microsoft Visual studio 2005 Professional edition(VC 8.0) ,C language Mirosoft .NET Framework 2.0
Development Open source software
Compression module：easylzma project (easylzma-0.0.7.tar.gz), 7-Zip(4.65)
Encryption module：coopfish project(version 1.1)
Image-Hiding module：JPHS project (jphs-0.3-(source code), jphs-0.5 (windows execute file)
Mail User Agent: Mozilla Thunderbird 3 (3.0.4)、Gmail (Web-based)
Photos for stegangraphy ：ncnu_original.jpg (type: jpeg)

4.2 實驗流程

在實驗當中，預設兩隻秘密金鑰都已交到對方手上。

首先，圖(十六)為傳送端的的實驗流程：

■ 實驗流程圖

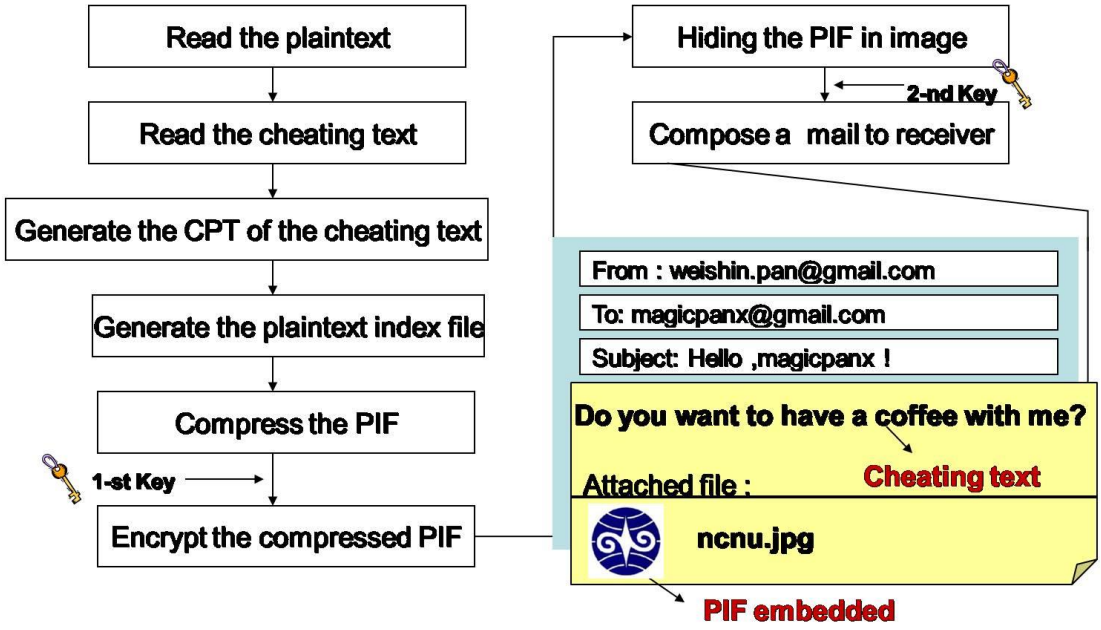

圖 十六、傳送端流程圖

1. 輸入 Plaintext 裡頭的內容
2. 輸入 Cheating text 裡頭的內容
3. 讀取 Plaintext 和 Cheating text 中的所有內容轉成十六進位碼的大寫字元，再來進行後續的文字處理
4. 根據 Cheating text 的內容來產生 Character position table(CPT)
5. Plaintext 藉由 CPT 來產生 Plaintext index file(PIF)
6. 對 PIF 進行壓縮的動作
7. 使用第一隻金鑰(1^{st} - key)去對 PIF 進行加密
8. 將加密後的 PIF 隱藏在要隱藏的圖片(ncnu_original.jpg)中，並產生一個新的圖片(ncnu.jpg)並使用第二隻金鑰(2^{nd} - key)去對 ncnu.jpg 的內容進行加密
9. 撰寫一封電子郵件給接收端，其文字部分輸入 cheating text(Do you want to have a coffee with me?)的內容，在附加檔案加上已隱藏 PIF 的影像檔(ncnu.jpg)
10. 寄出這封電子郵件

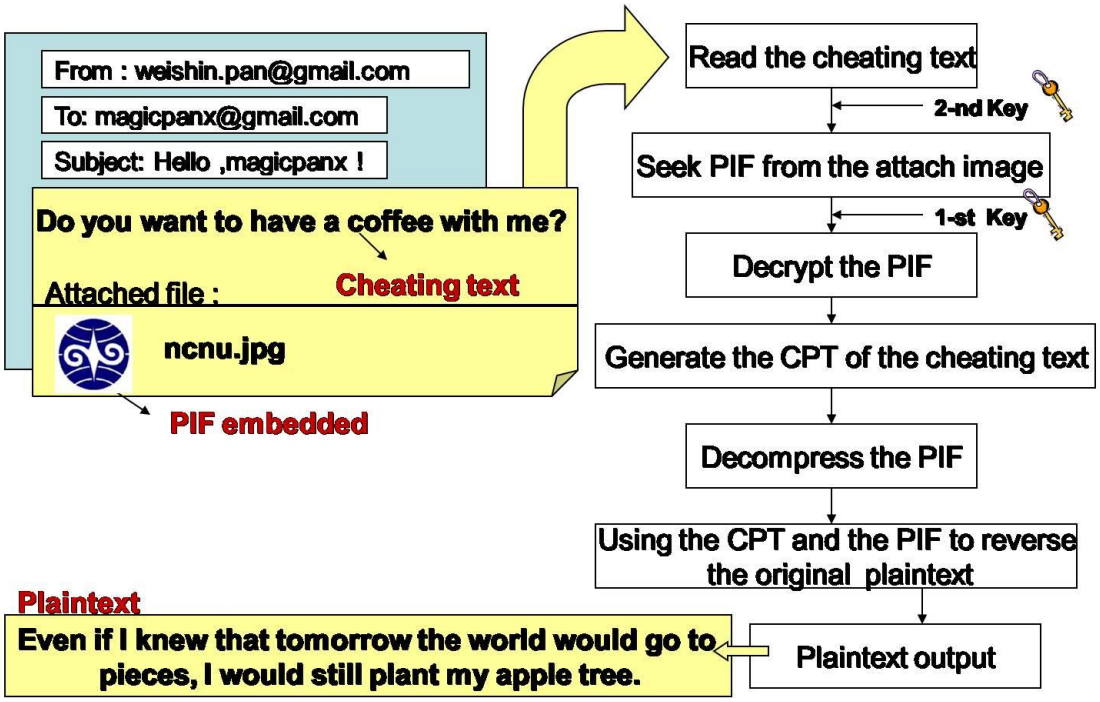

圖 十七、接收端流程圖

接著，圖(十七)為接收端的流程圖：

1. 接收端收到新信
2. 讀取信件 Body 的訊息內容，作為 Cheating text 的內容
3. 使用第二隻金鑰(2^{nd} - key)對已經接收到的圖片(ncnu.jpg) 進行解密並取得隱藏於其中的已加密的 PIF 文件
4. 使用第一隻金鑰(1^{st} - key)去對已加密 PIF 文件進行解密
5. 將 Cheating text 中的所有內容轉成十六進位碼的大寫字元後，開始進行處理
6. 透過 Cheating text 產生 CPT
7. 對已壓縮過後的 PIF 進行解壓縮
8. 使用 PIF 中所提供的資訊，也就是 Plaintext 所含字元的位置索引值還原 Plaintext 的內容並輸出到畫面上，可以看到以下訊息。

Even if I knew that tomorrow the world would go to pieces, I would still plant my apple tree.

4.3 實驗成果展示

根據前面的流程圖(十六)與圖(十七)所描述的運作流程，本節展示實際應用的操作畫面，並對每個步驟進行詳細說明：

1. 如圖(十八)所示，執行傳送端的程式後，系統會先要求你輸入你想要的欺敵訊息 Cheating text，而這個訊息也是未來用於電子郵件 Body 的內容。在這裡輸入了下面的句子範例為：

   Do you want to have a coffee with me?

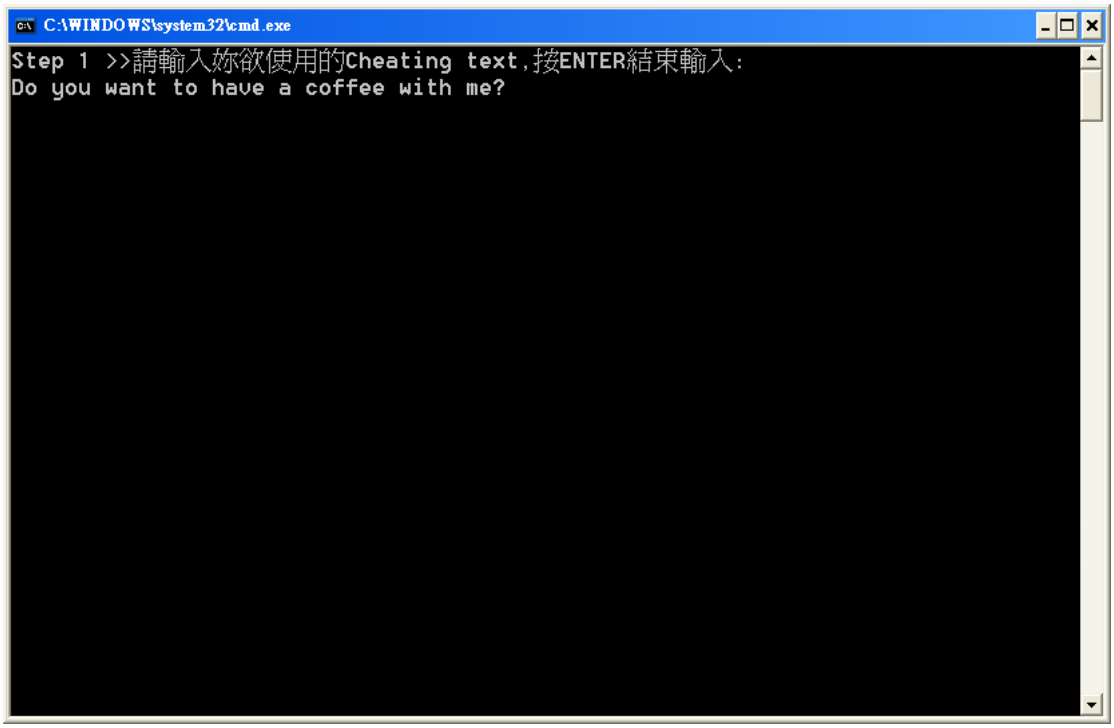

圖 十八、Cheating text 輸入畫面

2. 如圖(十九)所示，這時候系統會要求輸入欲傳送的秘密訊息(Plaintext)，在這裡輸入下面的句子：

   Even if I knew that tomorrow the world would go to pieces, I would still plant my apple tree.

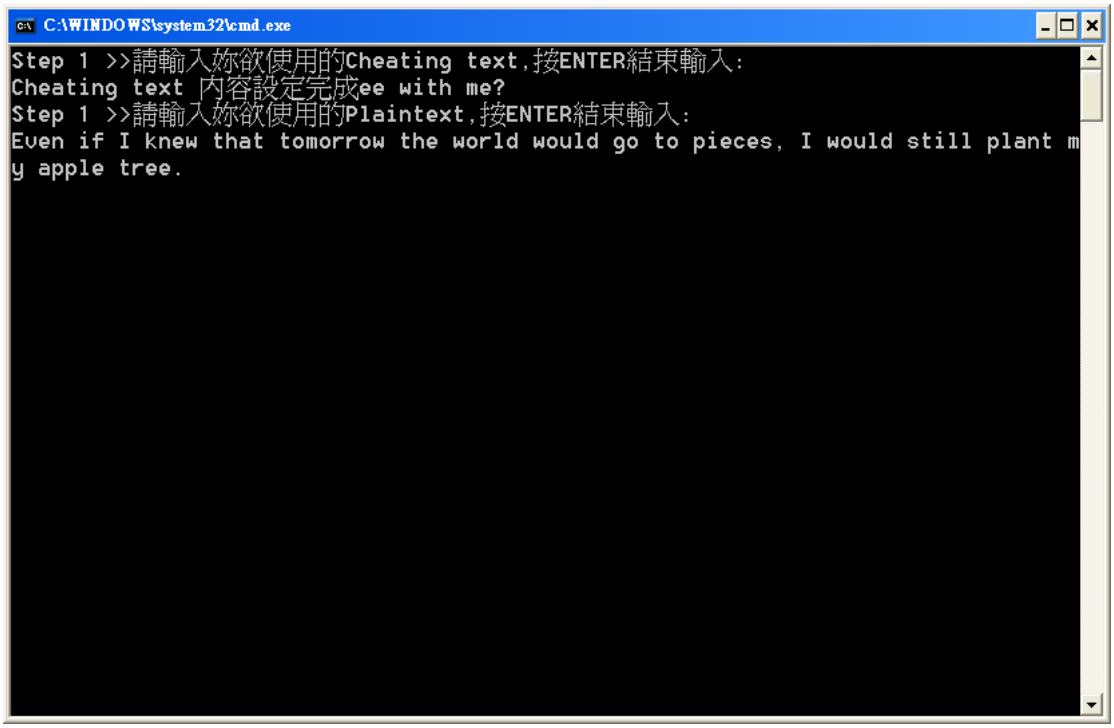

圖 十九、Plaintext 輸入畫面

3. 根據圖(二十)，表示正在傳送端進行壓縮以及加密 PIF 文件的動作,會得到"**bfpif**"這個檔案。

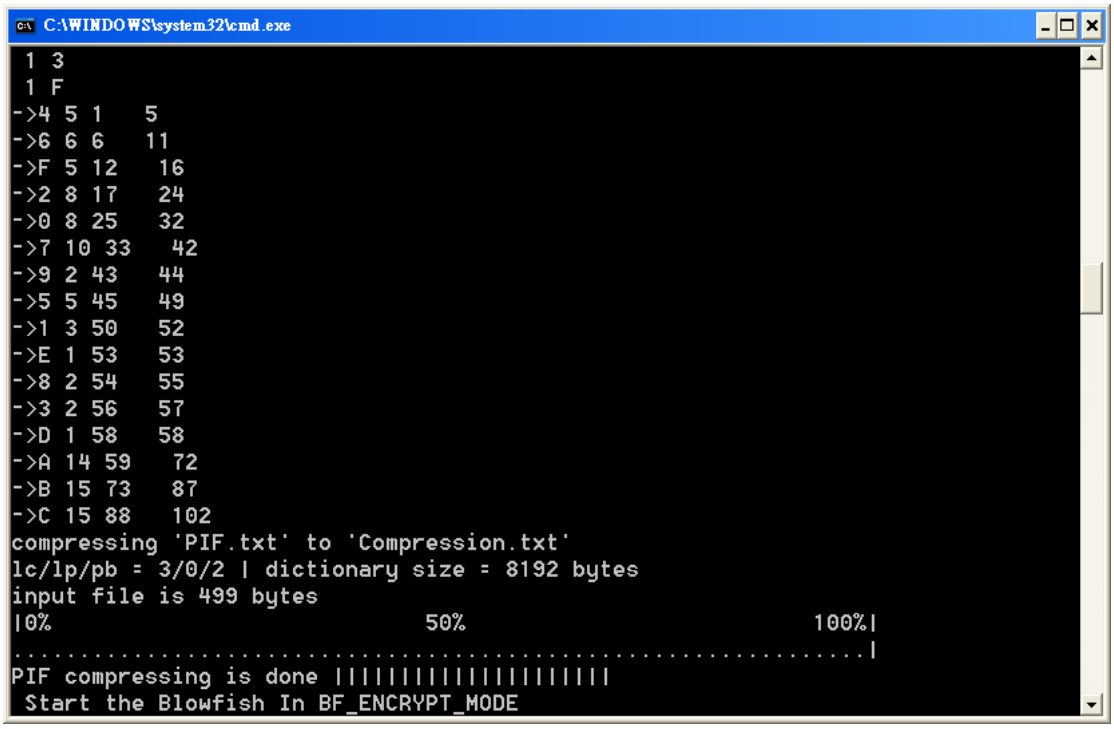

圖 二十、壓縮和加密 PIF 的操作畫面

4. 如圖(二十一)所示，系統這時候會執行隱藏圖片的程式 JPhide，將檔案 bfpif 放進所要隱藏的圖片 ncnu_original.jpg 中，並輸入金鑰。

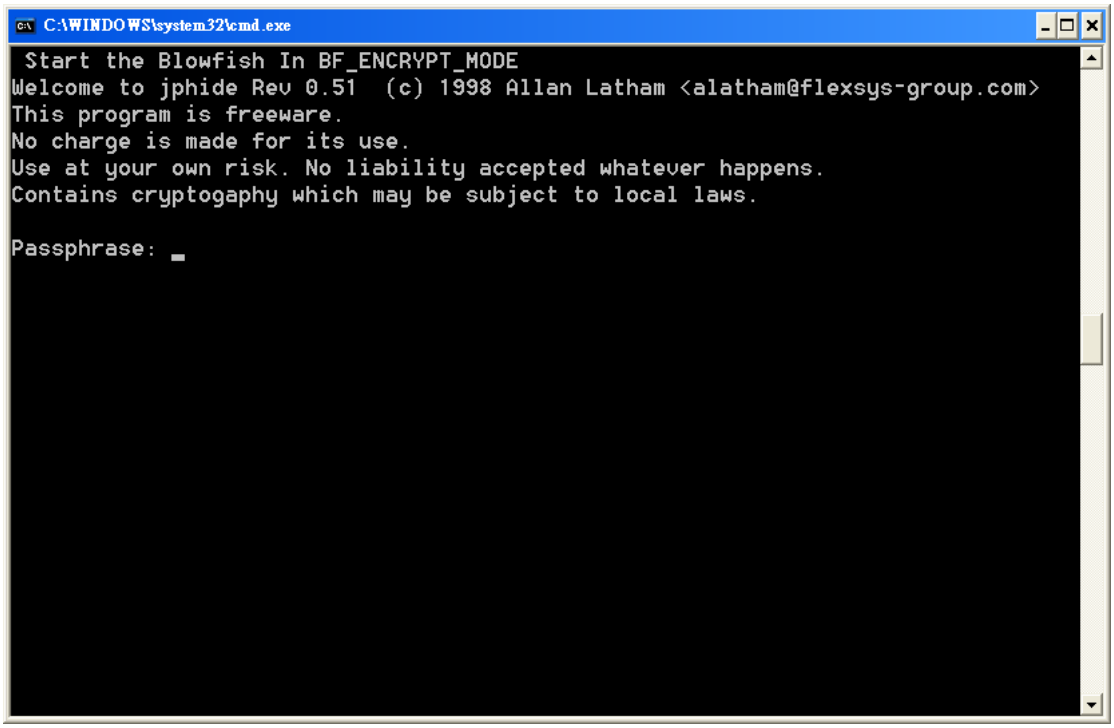

圖 二十一、輸入加密金鑰並將 PIF 文件隱藏在圖片

5. 在圖(二十二)的部分，表示再重新輸入一次金鑰，確認是否輸入有誤。

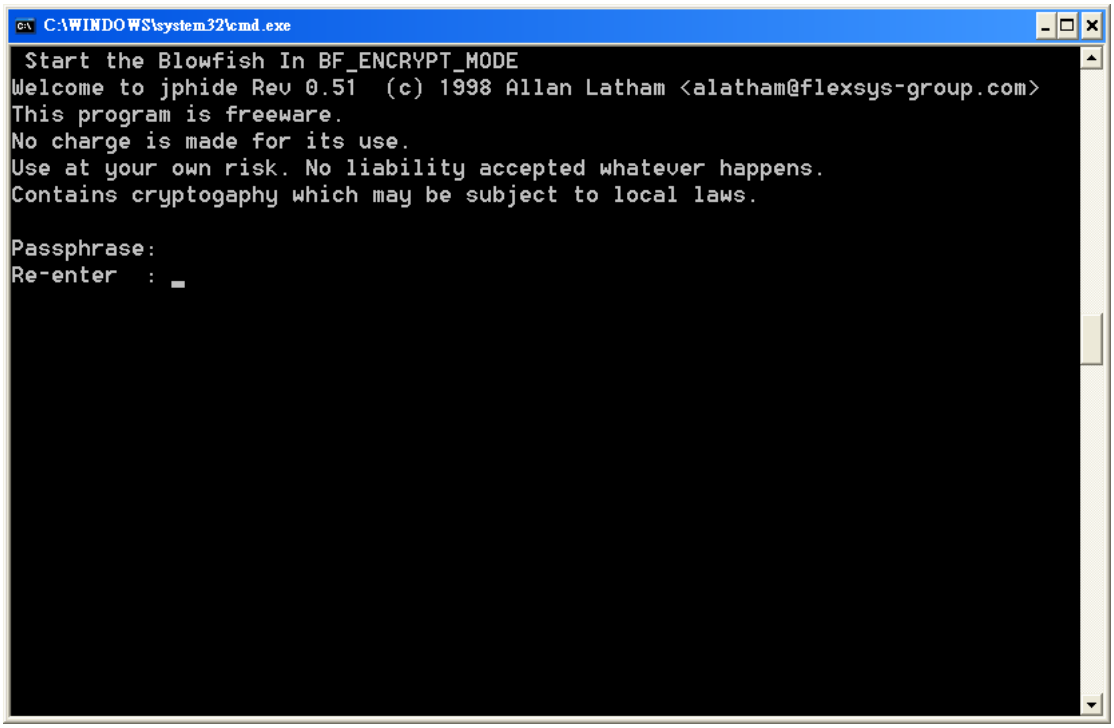

圖 二十二、再次輸入密碼並將 PIF 隱藏在圖片

6. 圖(二十三)為輸入完欲設定的金鑰後，就表示系統已經將 PIF 藏在新產生的 ncnu.jpg 裏頭。要確認是否成功藏入，可以與 ncnu_original.jpg 的檔案大小進行比較：ncnu-original.jpg 大小約為 33kb,隱藏資訊後的

ncnu.jpg 則是約為 13k。或是以圖(二十四) JPHSwin 來進行驗證的動作，透過 JPHSwin 將 PIF(filename：bfpif)隱藏在欲傳送的表情圖片，另外 JPHS 會在視窗的第一個欄位中的兩個值 **Approximate max** 和 **recommanded** 顯示使用者所選用的圖片最大可隱藏多少資訊以及建議隱藏的資訊大小。如此可避免圖案太小與資訊太多而造成隱藏失敗的問題。接著選擇 **Open jpeg**，接著選擇 **Hide** ,並在彈出對話盒中輸入密碼以及確認密碼。

圖 二十三、將 PIF 成功隱藏在 ncnu.jpg 圖片中

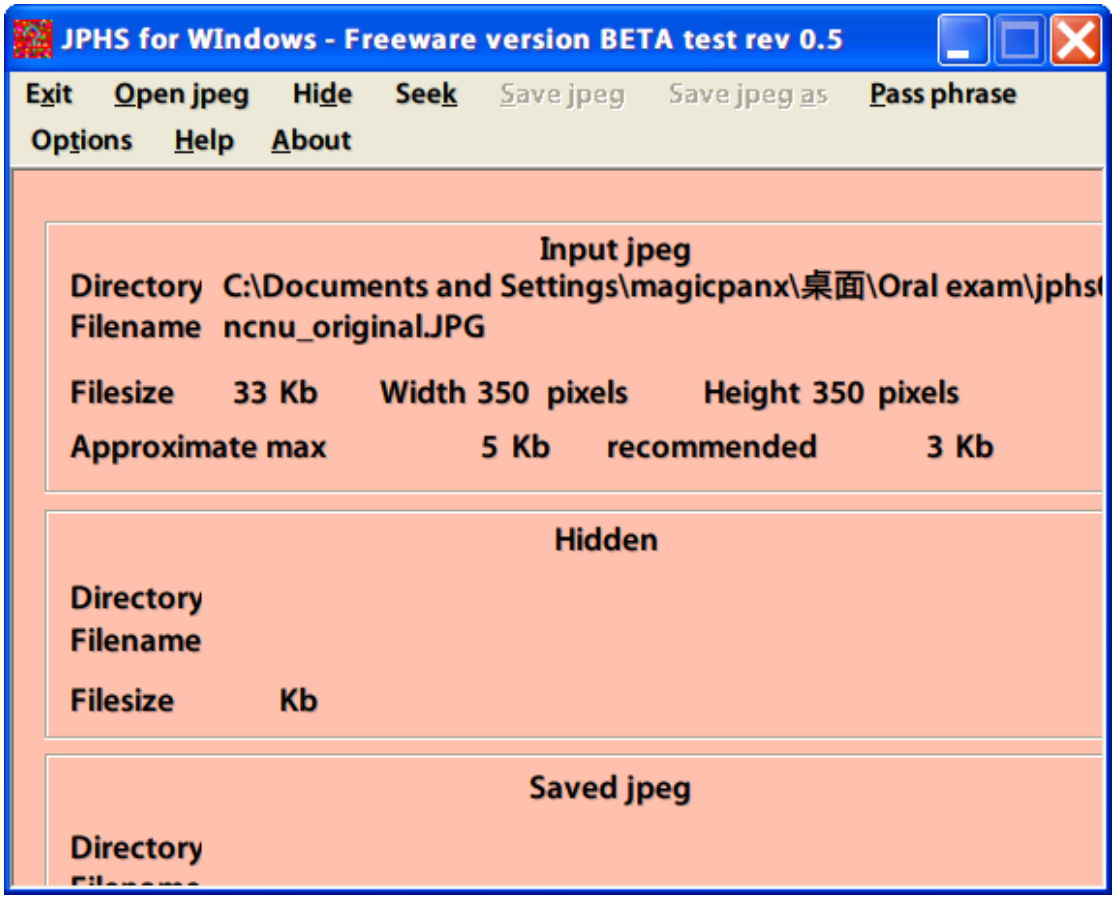

圖 二十四、透過 JPHSwin 來進行隱藏及驗證

7. 根據圖(二十五)的操作畫面可以看到，在 Windows 系統下，透過免費的 Mozilla Thunderbird 這套 MUA 來寄出夾帶圖片檔案或是表情符號的信件。

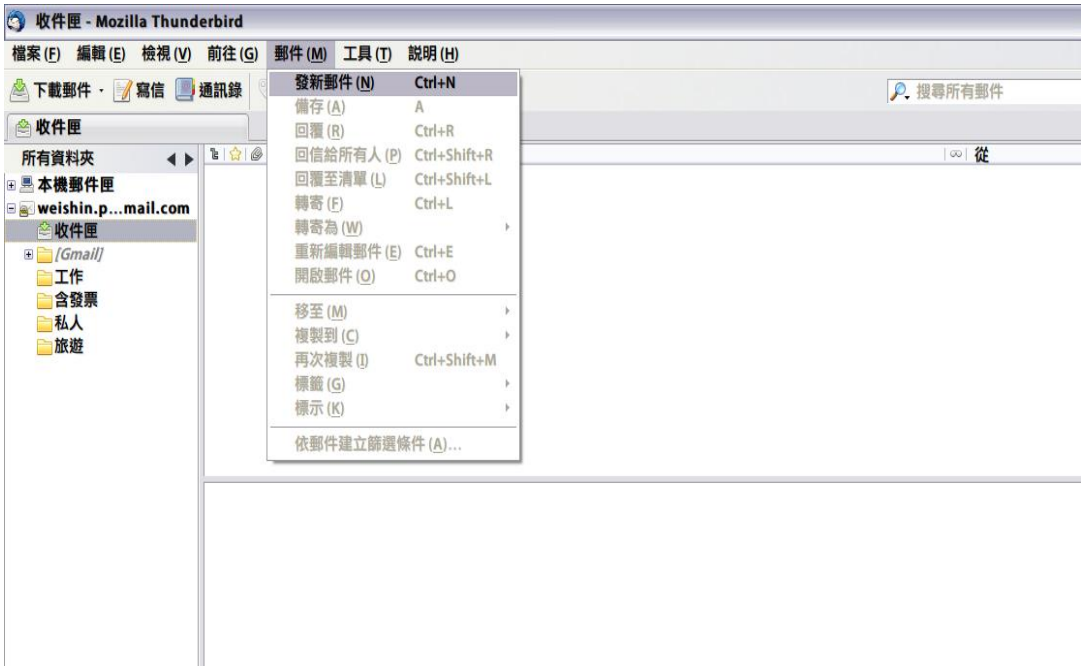

圖 二十五、Mozilla Thunderbird 主畫面

8. 圖(二十六)，傳送端 (weishin.pan@gmail.com)輸入接收端的電子郵件信箱位址 (magicpanx@gmail.com)，接著信件主旨任意輸入內容後，再點選「附件」按鍵，將 ncnu.jpg 夾帶進這封電子郵件當中。在信件的 Body 中輸入 Do you want to have a coffee with me? 後確認內容無誤，就可以寄送此封信件。

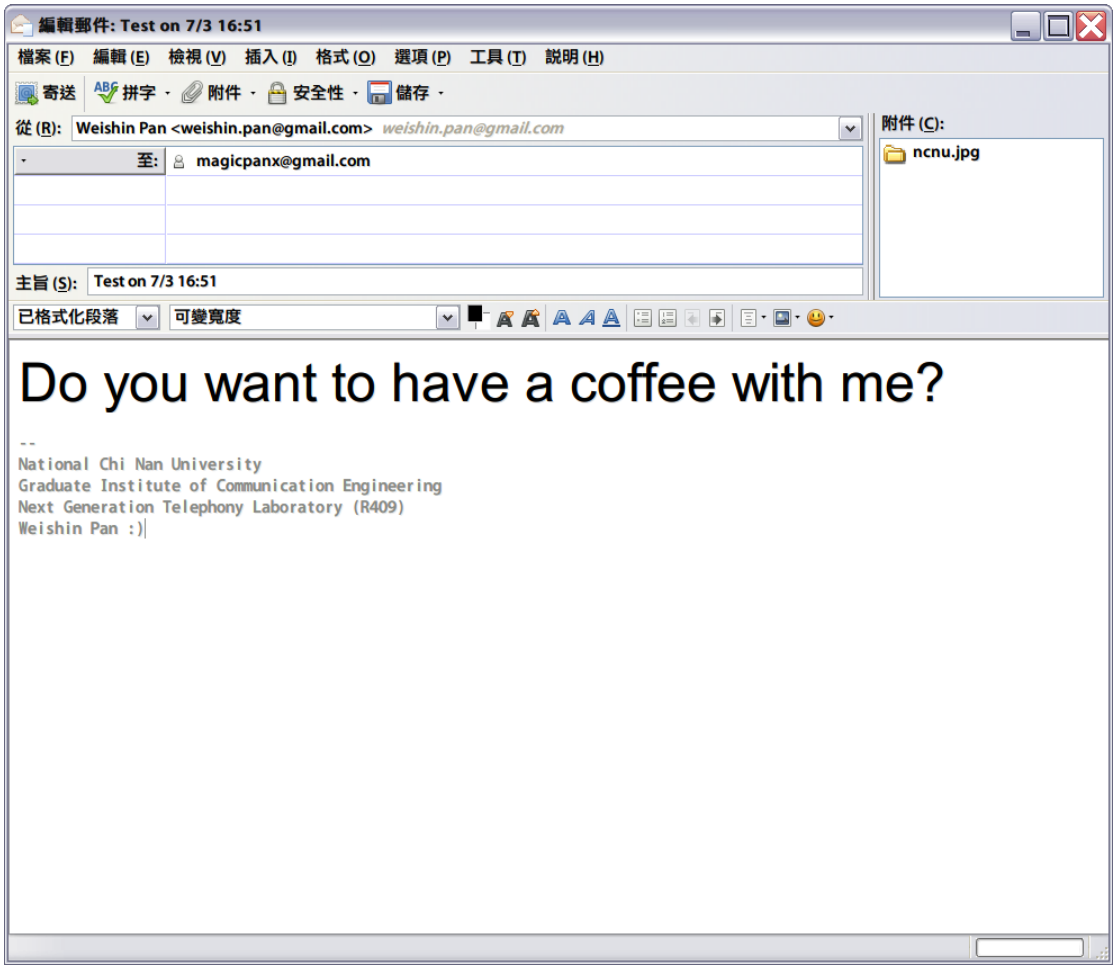

圖 二十六、輸入接收端的電子郵件位址

9. 圖(二十七)為任一電子信箱的收信畫面，在信件內取得 Body 內容(Do you want to have a coffee with me?)與附加的圖片(ncnu.jpg)之後，接著開啟接收端的解密程式。

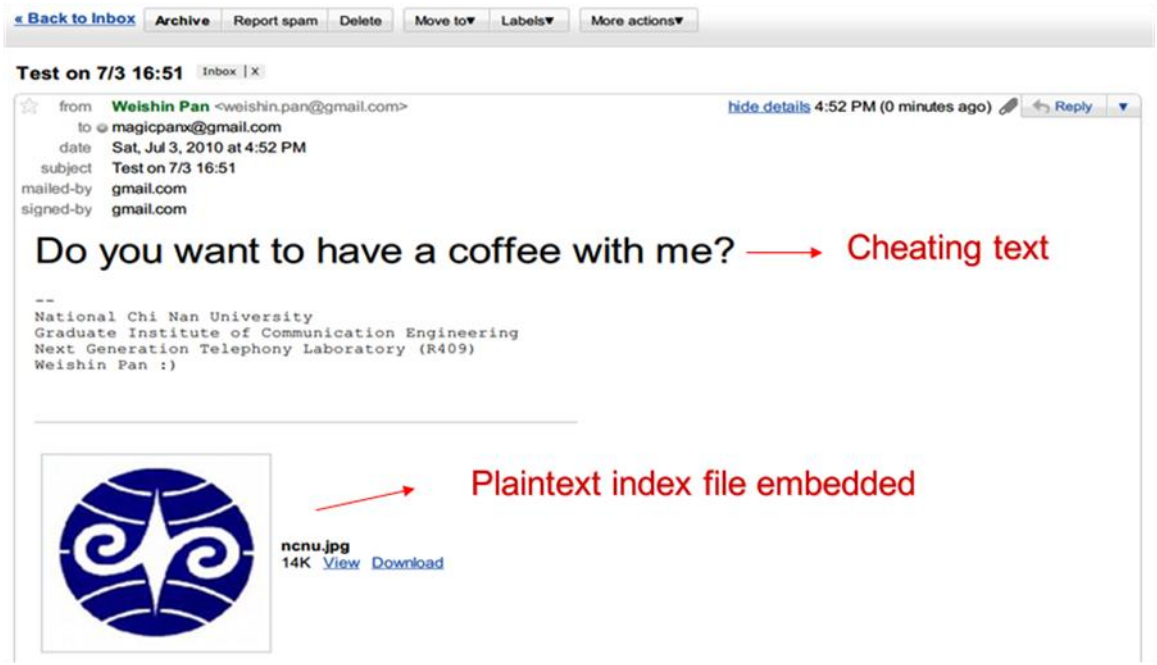

圖 二十七、接收端在信箱中收到電子郵件的畫面

10. 圖(二十八)，開啟接收端程式後，可以先看到系統會執行 JPseek 並詢問你的圖片解密金鑰，或用 JPHSwin 的操作畫面(圖二十九)，將附加於 ncnu.jpg 之中的 bfpif 文件取出來，選擇 **Seek**，並且輸入解密的密碼,解開後以 bfpif 這個檔案名稱來進行存檔。

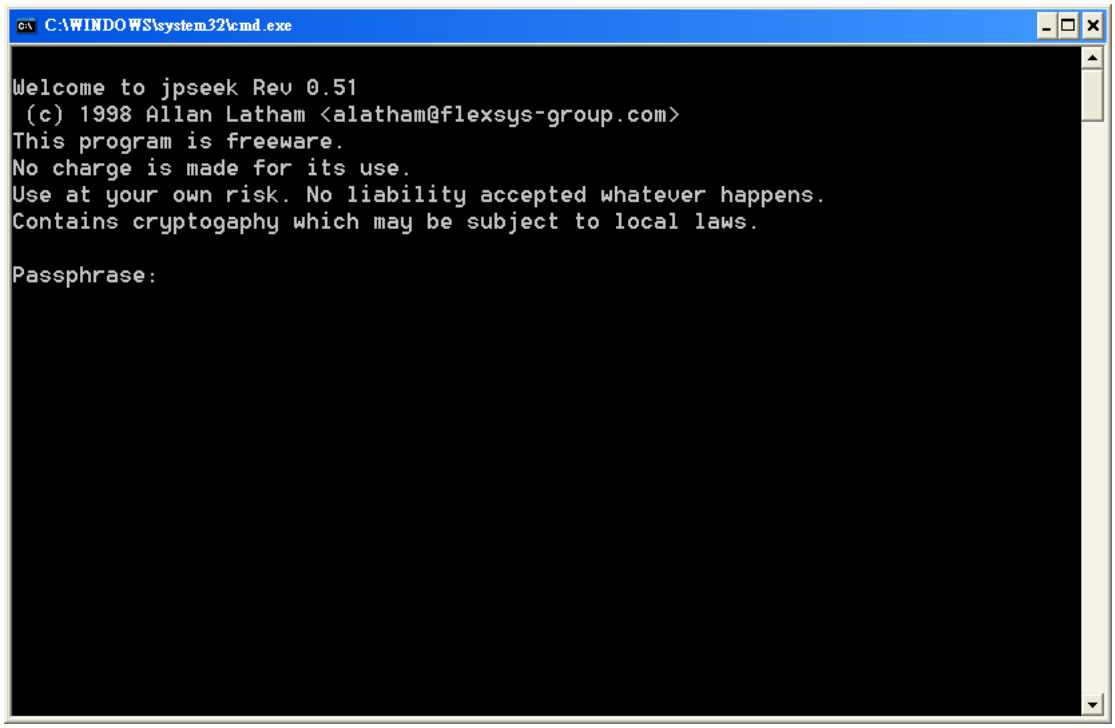

圖 二十八、輸入圖片解密金鑰的操作畫面

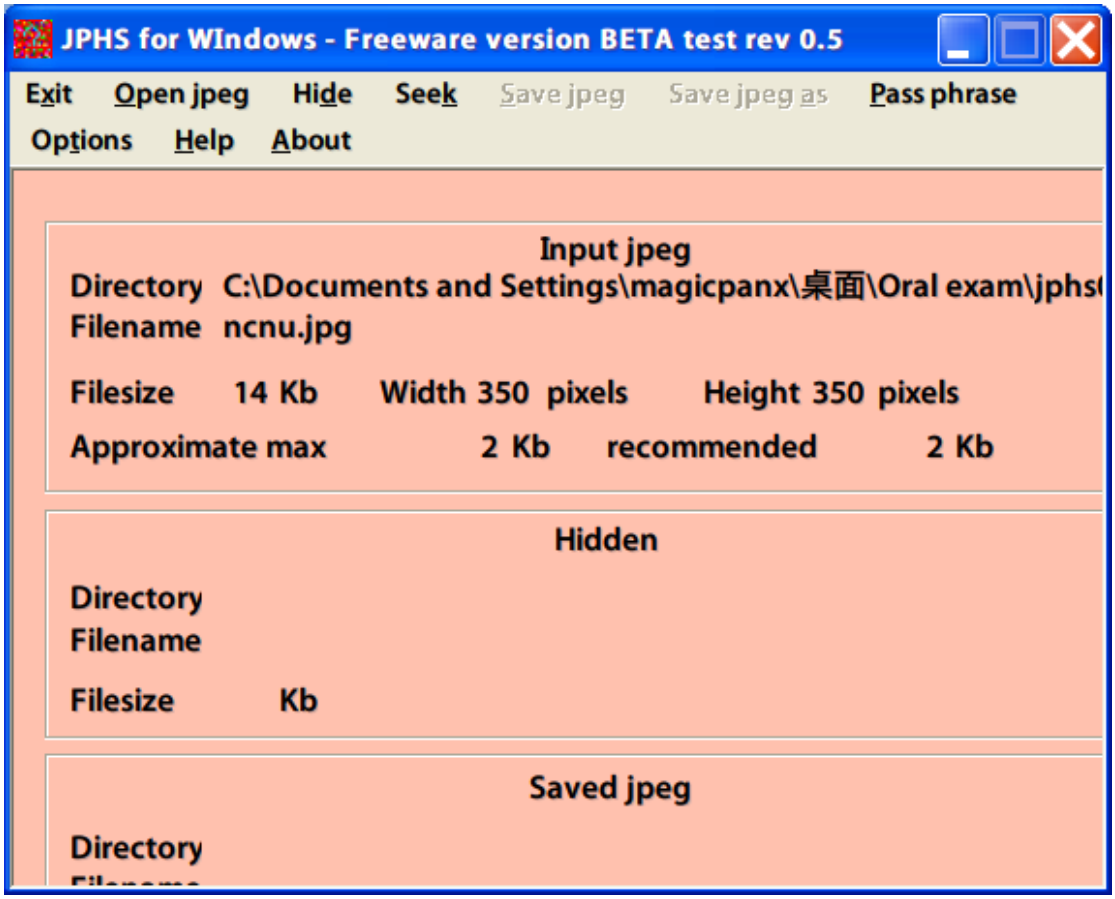

圖 二十九、透過 JPHSwin 來取出 PIF 文件

11. 圖(三十)為系統詢問 Cheating text 要設定的內容畫面，接著輸入剛剛在信件中所收到的 Body 內容，如下：

Do you want to have a coffee with me?

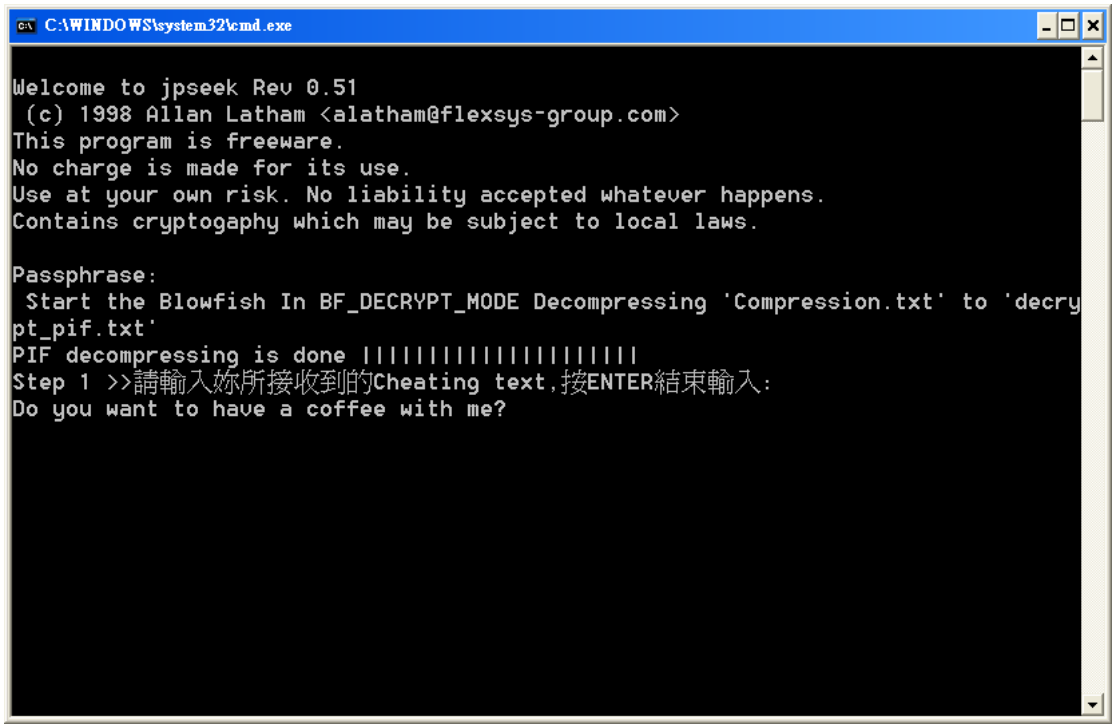

圖 三十、輸入所收到信件中的 body 內容

12. 圖(三十一)為最後的結果畫面，可以看到秘密訊息成功被解密出來，如下列訊息：

Even if I knew that tomorrow the world would go to pieces, I would still plant my apple tree.

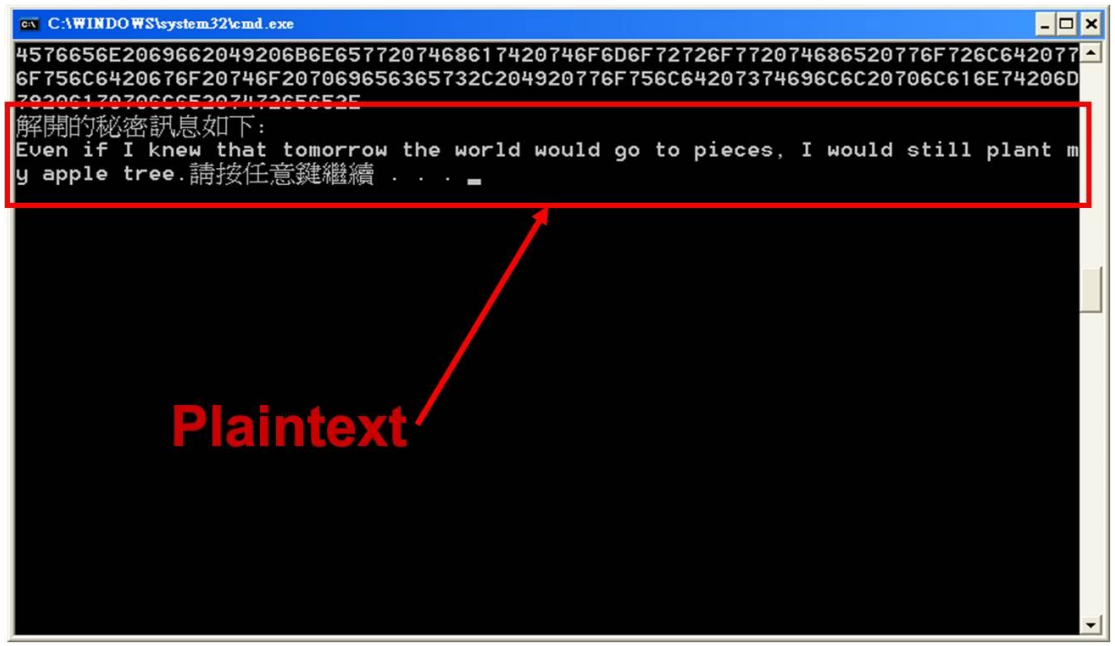

圖 三十一、顯示秘密訊息的內容

5. 結論

這次的研究中，透過了在平常生活中，觀察一般使用者在網際網路上的訊息交換，啟發了此次研究的概念，並藉以改善傳統 CDES 傳送 PIF 過程時令人疑竇的動作。

傳統的 CDES 傳送過程中，需要分開傳送多份欺敵訊息文件加上一份加密過後的 PIF 文件來達到混淆敵人的目的。但是，這份唯一加密過的檔案，對於竊聽者來說，他會對開始這樣的傳送動作感到疑惑，思考著這些檔案之間的關聯性，

本研究成果，成功地應用在電子郵件這個網路訊息服務，並且加入了圖像隱藏的技術，將 PIF 文件隱藏在附檔中，偽裝成一般的簽名檔或是被夾帶的圖像，而欺敵訊息的內容轉換為在信件的訊息主體中。這樣的保護機制，讓每次的傳送行為及內容更加有意義，竊聽者無法察覺在這些內容中的奧妙之處。因為不管從哪個角度去看這封郵件，都很難看出哪個部分有問題。

由實驗結果可以看出，這個系統提供了簡單有效又不會讓人懷疑的方式來達到交換秘密訊息的目的，這也增加了在電子郵件服務的安全性。此外，因為傳統 CDES 還有一些問題存在著，造成實際應用上的困難點，所以本研究加入了其他學者對於傳統 CDES 的改善方法，改善了中文支援性和欺敵訊息必須包含秘密訊息所有字元的問題，讓整個保護機制更加快速，秘密訊息可以傳送的更快速更安全。綜合以上所述，未來本架構可以很快速地應用在需增強安全性的網路訊息服務上面。

6. 未來方向

根據本次研究所提出的系統架構，未來希望能將這個訊息安全保護架構運用在 Instant Message(即時訊息)的服務上，像是目前熱門的 Windows Live Messenger、Yahoo Messenger。因為在現代人聊天模式中，除了即時的

文字訊息之外，表情符號和許多圖像的大量傳遞的特性，已經是現代人聊天時不可或缺的元素了。對大多數使用者來說，這種聊天方式已經很普遍和合理化了。

另外，本次研究最後結果，尚未加入的特色是，在這次系統設計當中並未整合相關的寄發電子郵件專屬模組，而是透過手動操作來達到實驗的目的。未來如果能將本系統成功整合至免費的 Thunderbird extensions，變成一個能提供秘密訊息傳送的程式再自動透過 Thunderbird 來寄出信件，這個擴充套件將能造福許多在網際網路上的使用者，加強在他們對於秘密訊息的傳遞需求，又能跳脫傳統加密傳送在操作上的不便。

7. 參考文獻

[1] C. H. Lin and T. C. Lee, "A Confused Document Encrypting Scheme and its Implementation", Computer & Security, Vol.17, No.6, pp.543-551,1998

[2] S. Katzenbeisser and F. A. P. Petitolas, "Information Hiding Techniques for Steganography and Digital Watermarking", Artech House, Boston, U. S. A., 2000

[3] F. L. Bauer, "Decrypted Secrets - Methods and Maxims of Cryptology", Berlin, Heidelberg, Germany, Springer-Verlag, 1997

[4] Neil F. Johnson and S. Jajodia , "Steganography: Seeing the Unseen", IEEE Computer, pp. 26-34, Feb. 1998

[5] 王旭正,柯宏叡, “資訊與網路安全：秘密通訊與數位鑑識新技法”, 博碩文化, 2006

[6] 賴溪松,韓亮,張真誠, “近代密碼學及其應用”, 旗標出版, 2003

[7] W. Diffie and M. E. Hellman,"New Directions in Cryptography", IEEE Transactions on Information Theory, Vol. 22, No.6, pp.644-654, Nov.

1976

[8] B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)", Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, pp. 191-204, Dec. 1993

[9] B. Schneier, "The Blowfish Encryption Algorithm - One Year Later", Dr. Dobb's Journal, Sep. 1995

[10] H. Feistel, "Cryptography and Computer Privacy", Scientific American, Volume 228, No. 5, pp. 15-23, May 1973

[11] NBS FIPS PUB 46, "Data Encryption Standard", National Bureau of Standards, U.S. Department of Commerce, Jan. 1997

[12] X.Lai and J.Massey, "A proposal for New Block Encryption Standard" , in Proceeding of EUROCRYPT '90 (Springer-Verlag, Berlin), pp.389-404, 1911

[13] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compressoion", IEEE Transactions on Information Theory, Vol. IT-23, No.3, pp. 337-343,1977

[14] J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding", IEEE Transactions on Information Theory, Vol. 24, Issue: 5, pp. 530 – 536,1978

[15] I.H. Witten, R.M. Neal and J.G. Cleary, "Arithmetic Coding for data Compression", Commications of the ACM, Vol. 30,No.6,pp. 520-540, Jun. 1987

[16] W. H. Yeh and J. J. Hwang, "Hiding Digital Information Using a Novel System Scheme", Computers and Security, Volume 20, Number 6, pp. 533-538, Sep. 2001

[17] W. H. Yeh and J. J. Hwang, "A scheme of hiding secret Chinese

information in confused documents", Journal of Information Management, Vol.7 (2), pp. 183-191, 2001

[18] B. F. Liang, etc, “On the study and implementation for confused document encrypting scheme of data hiding”, Technical Report, Department of Information Management, Ta Hwa Institute of Technology, R.O.C.,2002

[19] T. J. Yao and Quincy Wu, "On the Study of Overhead Reduction for Confused Document Encrypting Schemes", International Conference on Multimedia Computing and Information Technology (MCIT 2010) University of Sharjah(UoS), Sharjah, United Arab Emirates (UAE), Mar. 2-4, 2010

[20] Lloyd, http://lloyd.github.com/easylzma/

[21] 7-zip , http://www.7-zip.org/sdk.html

[22] Coopfish, http://ivan.vecerina.com/code/coopfish/

[23] JPHS, http://linux01.gwdg.de/~alatham/stego.html

8. 附件

附件 A. EasyLZMA project

1. 在環境設置的部分，使用 Microsoft Visual studio 2005 Professional Edition(VC 8.0)作為主要程式開發環境
2. 首先，連結至 http://lloyd.github.com/easylzma/取得 EasyLZMA 的原始碼，下載檔案名稱為：easylzma-0.0.7.zip
3. 將 EasyLZMA 解壓縮至 C:/ 目錄底下。
4. 作者為了讓這個專案能夠順利在 Visual Studio 下編譯，將原來所提供的 C 原始碼檔案，轉換成 Visual Studio 下的專案檔。

5. 在 EasyLZMA 目錄下，可以看到一個名稱為 CMakeLists.txt 的文件，可以得知作者使用 CMake 這個軟體來產生 Visual Studio 的專案檔。
6. 至 http://www.cmake.org/ 下載 CMake，請選擇 win32 的版本，下載完成後安裝它。
7. 進入到命令提示字元模式下，進入 EasyLZMA 目錄下的 build 資料夾，並且輸入”cmake ..”，假如執行成功後，並無任何錯誤時，你將可以在 build 這個資料夾下看到 Visual Studio 的專案檔。
8. 打開 EasyLZMA 的 Visual Studio 專案，並且執行編譯的動作，成功後，可以在專案裡面找到未來會使用到的 DLL 和 LIB 文件。

附件 B. Coopfish project

1. 在環境設置的部分使用 Microsoft Visual Studio 2005 Professional Edition(VC 8.0)作為程式開發環境。
2. 首先，前往 http://ivan.vecerina.com/code/coopfish/ 下載 coopfish project 的原始碼。
   (http://ivan.vecerina.com/code/coopfish/coopfish.zip)
3. 解壓縮 coopfish 後，開啟一個 Visual Studio 的新專案，將其原始碼加入專案，再進行編譯的動作。

附件 C. JPHS

1. 在環境設置的部分使用 Microsoft Visual studio 2005 Professional edition(VC 8.0)作為程式開發環境。
2. 首先，前往 http://linux01.gwdg.de/~alatham/stego.html 下載

JPHS 的原始碼，它有 Llinux 版和 windows 版：

Linux：

(ftp://ftp.gwdg.de/pub/linux/misc/ppdd/jphs-0.3.tgz)

Windows：

(ftp://ftp.gwdg.de/pub/linux/misc/ppdd/jphs_05.zip)

3. 解開這兩個壓縮檔後，可從裏頭取出原始碼和視窗程式 JPHSWin 還有兩個重要系統執行檔 JPhide 和 JPseek，分別用來在圖片中隱藏資訊和取出圖片內的隱藏資訊。

附件 D. Mozilla Thunderbird 3

1. 在 Windows 系統下，至官方網站 http://moztw.org/thunderbird/ 下載目前版本為 3.04 的 Thunderbird (http://download.mozilla.org/?product=thunderbird-3.0.4&os=win&lang=zh-TW)
2. 下載並進行安裝，成功安裝後點選 Thunderbird 圖示執行
3. 為可寄發信件請先新增一個可用的使用者帳戶，依序操作(檔案->開新檔案->郵件帳號->輸入結束->建立帳號)

附件 E. Source code

Encryption.cpp

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <math.h>
#include <cstring>      // 引入字串函數標題檔
#include <ctype.h>      // 引入字元測試與轉換函數標題檔
#include "CPT.h"
#include "DECRYPTION.h"
#include "LZMA.h"
```

```
#include "blowfish.h"
#include <conio.h>
//#pragma comment(lib, "easylzma.lib")
//#pragma comment(lib, "easylzma_s.lib")
//#pragma comment(lib,"ws2_32.lib")
using namespace std;
#define SIZE 80000000
#define SIZE2 80000000

FILE *IN , *OUT , *IN2 , *OUT2;
char    ch , ch2;
char    temp_ch , temp_ch2;
unsigned char buf[SIZE];
unsigned char buf2[SIZE];
unsigned char temp_buf[SIZE2];
unsigned char temp_buf2[SIZE2];
int status , count, count2 ,i,j;
int len , len2;
// For CPT
CPT table;
// For Decrtptimg
DECRYPT de;
// For blowfish
//File header used to identify the file format
static const char magicHead[] = {'c','f','s','h'};
enum { BF_NONE, BF_ENCRYPT_MODE, BF_DECRYPT_MODE } bfmode
= BF_NONE;
static const char* exeName; ///< Name of this executable, obtained from
command line
static FILE* src = 00; ///< Source file buffer
static FILE* dst = 00; ///< Destination file buffer
int blowfish_MODE(char mode);

int main(void)
{
   FILE *input_fp;
   char input_ch[100];
   char input_ch2[100];
```

```
  input_fp = fopen("cheatingtext.txt","w");
 // printf("Step 1 >>請輸入你欲使用的 Cheating text,按 ENTER 結束輸入:\n");
 /*
  while((input_ch = getche()) != '\r')
       putc(input_ch,input_fp);*/                              ////fix on 20100809


input_fp = fopen("cheatingtext.txt","w");
//printf("Step 1 >>請輸入你欲使用的 Cheating text,按 ENTER 結束輸入:\n");

puts("Step 1 >>請輸入你欲使用的 Cheating text,按 ENTER 結束輸入:");
gets(input_ch);
puts("輸入的字串為：");
puts(input_ch);
for(int i=0 ; i<100;i++)
       {
            if (input_ch[i] == '\0')
                 break;
           fprintf(input_fp,"%c",input_ch[i]);
        }
  fclose(input_fp);
  //printf("Cheating text 內容設定完成\n");
  printf("\n");
 // printf("Step 2 >>請輸入你欲使用的 Plaintext,按 ENTER 結束輸入:\n");
  //printf("\n");

 input_fp = fopen("plaintext.txt","w");
puts("Step 2 >>請輸入你欲使用的 Plaintext,按 ENTER 結束輸入:");
gets(input_ch2);
puts("輸入的字串為：");
puts(input_ch2);
for(int i=0 ; i<100;i++)
       {
            if (input_ch2[i] == '\0')
                 break;
           fprintf(input_fp,"%c",input_ch2[i]);
        }
```

```
/* while((input_ch = getche()) != '\r')
        putc(input_ch,input_fp);*/                          //fix on 20100809

  fclose(input_fp);
  //printf("Cheating text 內容設定完成\n");
  //It tanslates cheating text to Hex code and put them to temp.txt
  IN = fopen("cheatingtext.txt" ,"r");
  OUT= fopen("temp.txt" , "w");

  if (IN != NULL)
  {
      while((ch = getc(IN)) != EOF)
      {
            count++;
            buf[i++] = ch;
            printf("%c",ch);

      }
      i = 0;
      fclose(IN);
      cout << "INPUT 總共有" << count << "個字元" << endl;
      count = 0 ;

  }
  else
       cout << "Can't open the file" << endl;

  for(int i=0 ; i<SIZE;i++)
      {
            if (buf[i] == '\0')
                  break;
           // printf("0x%02x\n",buf[i]);
            fprintf(OUT,"%x",buf[i]);
            // write HEX code to temp.txt
      }
// len = strlen((const char*)buf2);
// printf("%d" ,len);
  //計算字串長度
```

```
fclose(OUT);

//---------------------------------------------------------------------------
OUT= fopen("temp.txt" , "r");

if (OUT != NULL)
{
    while((temp_ch = getc(OUT)) != EOF)
    {
          count++;
          temp_buf[j++] = temp_ch;
        // printf("%c\n",temp_ch);

    }
    j = 0;
    cout << "Temp 總共有" << count << "個字元" << endl;
    count = 0;

}
else
     cout << "Can't open the file" << endl;

fclose(OUT);

//---------------------------------------------------------------------------

len = strlen((const char*)temp_buf);                          // 取得字串長度
for (int i = 0; i <= len; i++)                    // 轉成大寫迴圈
{
  if (islower(temp_buf[i]) != 0)                          // 若為小寫字元
     temp_buf[i] = toupper(temp_buf[i]);                        // 轉成大寫字元
}
cout << "轉換大寫後：" << temp_buf << endl      // 顯示轉換後字串
       << endl;

cout << "temp_buf_size =" << sizeof(temp_buf) << endl;
cout << "----------以下是給 plaintext 轉換用的----------" << endl;
```

```
//--------------------------------------------------------------------------
//
//      For plaintext    ^_^!!!!!!!!!!
//
//--------------------------------------------------------------------------
IN2 = fopen("plaintext.txt" ,"r");
OUT2 = fopen("temp2.txt" , "w");

if (IN2 != NULL)
{
     while((ch2 = getc(IN2)) != EOF)
     {
          count2++;
          buf2[i++] = ch2;
          printf("%c",ch2);

     }
     i = 0;
     fclose(IN2);
     cout << "INPUT 總共有" << count2 << "個字元" << endl;
     count2 = 0 ;

}
else
      cout << "Can't open the file" << endl;

for(int i=0 ; i<SIZE;i++)
     {
          if (buf2[i] == '\0')
               break;
          // printf("0x%02x\n",buf2[i]);
          fprintf(OUT2,"%x",buf2[i]);
          // write HEX code to temp.txt
     }
// len = strlen((const char*)buf2);
// printf("%d" ,len);
//計算字串長度
fclose(OUT2);
```

```
  //-------------------------------------------------------------------------------
  OUT2= fopen("temp2.txt" , "r");

  if (OUT != NULL)
  {
      while((temp_ch2 = getc(OUT)) != EOF)
      {
            count2++;
            temp_buf2[j++] = temp_ch2;
            //printf("%c\n",temp_ch2);

      }
      j = 0;
      cout << "Temp2 總共有" << count2 << "個字元" << endl;
      count2 = 0;

  }
  else
       cout << "Can't open the file" << endl;

  fclose(OUT2);

  //-------------------------------------------------------------------------------
  len2 = strlen((const char*)temp_buf2);                          // 取得字串長
度
  for (int i = 0; i <= len2; i++)                    // 轉成大寫迴圈
  {
     if (islower(temp_buf2[i]) != 0)                          // 若為小寫字元
        temp_buf2[i] = toupper(temp_buf2[i]);                       // 轉成大寫字
元
  }
  cout << "轉換大寫後：" << temp_buf2 << endl       // 顯示轉換後字串
        << endl;

  cout << "temp_buf_size =" << sizeof(temp_buf2) << endl;
  cout << "以下是創造 CPT 的步驟" << endl;
  //-------------------------------------------------------------------------------
```

```
  // printf("%c" , temp_buf[0]);

    table.CPTcreate((unsigned char*)&temp_buf,(unsigned char*)&temp_buf2 ,
len,len2);
    //產生 CPT
    //將 plaintext 與 cpt 做比對產生 PIF 的內容
    //doCompress();
  runmode = RM_COMPRESS;
  doCompress();       // LZMA 中的壓縮函式
  //runmode = RM_DECOMPRESS;
  //doDecompress();
  //de.Decrypting();
  bfmode = BF_ENCRYPT_MODE;
  blowfish_MODE(bfmode);
// bfmode = BF_DECRYPT_MODE;
  //blowfish_MODE(bfmode);
//doDecompress();
  //_sleep(1000);
  //de.Decrypting();
//    printf("%c" , (1 << 24));
system("jphide.exe ncnu_original.jpg ncnu.jpg bfpif");
//system("pause");
return 0;
}
uint32 roundUp( uint32 value, unsigned rounding )
{
    value += rounding-1;
    value -= value % rounding;
    return value;
}
/// Obtain the byte length of @a file using the standard library.
long fileSize( FILE* file )
{
    long const savedPos = ftell(file);
    fseek( file, 0, SEEK_END );
    long const size = ftell(file);
    fseek( file, savedPos, SEEK_SET );
```

```
    return size;
}
/// Error handling routine. Displays message and usage info, then aborts the
application.
void fail( const char* msg )
{
    printf("\n\n"
           " ERROR: %s\n\n"
           " Usage:\n"
           "    To encrypt: %s [key] [EncryptedOutputFile] [InputFile]\n"
           "    To decrypt: %s [key] [EncryptedInputFile]\n"
           " The name of the original file is stored in the encrypted file, and
restored accordingly.\n\n"
          , msg, exeName, exeName );
    if( src ) fclose(src);
    if( dst ) fclose(dst); //NB: also try to delete the destination file in this case ?
    exit(EXIT_FAILURE);
}

/// Write a 4-byte word to the output file in big-endian format
void putWord( uint32 word )
{
    uint8 buf[4] = { uint8(word>>24), uint8(word>>16), uint8(word>>8),
uint8(word) };
    if( ! fwrite( buf, 4, 1, dst ) ) fail("Error writing output file");
}

/// Read a 4-byte word from the source file in big-endian format
uint32 getWord()
{
    uint8 buf[4];
    if( ! fread( buf, 4, 1, src ) ) fail("Error reading input file");
    return (uint32(buf[0])<<24) | (uint32(buf[1])<<16) | (uint32(buf[2])<<8) |
buf[3];
}

void putByte( uint8 byte )
{
```

```
    if( ! fwrite( &byte, 1, 1, dst ) ) fail("Error writing output file");
}

uint8 getByte()
{
    uint8 ans;
    if( ! fread( &ans, 1, 1, src ) ) fail("Error reading input file");
    return ans;
}

int blowfish_MODE(char mode)
{
  switch( mode ) {
    case    BF_ENCRYPT_MODE: printf(" Start the Blowfish In
BF_ENCRYPT_MODE "); break;
    case    BF_DECRYPT_MODE: printf(" Start the Blowfish In
BF_DECRYPT_MODE "); break;
  };

    const char* const key = "1234567890";
    int const keyLen = strlen(key);
    if( keyLen<8 || keyLen>56 )
        fail("Key length must be between 8 and 56 characters");

    blowfish::Pad const    pad = blowfish::generatePad( key, keyLen );
    blowfish::Block          chain( 0xC009F158, 0x17EC17EC ); // some dummy
initializer -- best would be to use a stored random number instead

    enum { kBufSize = 32*1024 };
    char iobuf[kBufSize];

    if( mode == BF_ENCRYPT_MODE ) // encryption
    {
        const char*             srcName = "Compression.txt";
        const char* const dstName = "bfpif";
        src = fopen( srcName, "rb" );
        if( ! src ) fail("Could not open input file");
        dst = fopen( dstName, "wb" );
```

```
        if( ! dst ) fail("Could not open output file");
        long const srcSize = fileSize( src );

        { // srcName might be a full/relative path -- we only want to store the file
name itself
            const char* scan = srcName;
            while( char const c = *scan++ )
            {
                if( c=='/' || c=='\\' || c==':' ) // take all path delims of
unix&windows...
                    if( !! *scan ) // only if isn't a trailing char...
                        srcName = scan;
            }
        }

        fwrite( magicHead, 4, 1, dst );
        int const srcNameLen = strlen(srcName);
        if( srcNameLen != uint8(srcNameLen) )
            fail("Input file name is too long - limit is 255 chars");
        putByte( srcNameLen );
        fwrite( srcName, srcNameLen, 1, dst );
        putWord( srcSize );

        long sizeLeft = srcSize;
        while( sizeLeft > 0 )
        {
            long sizeRead = fread( iobuf, 1, kBufSize, src );
            sizeLeft -= sizeRead;
            if( sizeLeft == 0 )
                sizeRead = roundUp( sizeRead, blowfish::kBlockSize );
            else if( sizeRead != kBufSize )
                fail("Error while reading input file");
            blowfish::encrypt_CBC( pad, iobuf, iobuf, sizeRead, &chain );
            fwrite( iobuf, 1, sizeRead, dst );
        }

        chain = blowfish::core::encipherBlock( pad, chain );
        putWord( chain.L );
```

```
        putWord( chain.R );
    }
    else // decryption
    {
        const char* const srcName = "bfpif";
        src = fopen( srcName, "rb" );
        if( ! src ) fail("Could not open input file");

        fread( iobuf, 4, 1, src );
        if( strncmp( iobuf, magicHead, 4 ) )
            fail("Incorrect file type was passed as decryption source");
        unsigned const dstNameLen = getByte();
        fread( iobuf, dstNameLen, 1, src );
        iobuf[dstNameLen] = '\0';
        dst = fopen( iobuf, "wb" );
        if( ! dst ) fail("Could not open output file");

        uint32 const dstSize = getWord();
        uint32 const srcSize = roundUp( dstSize, blowfish::kBlockSize );

        long sizeLeft = srcSize;
        while( sizeLeft > 0 )
        {
            long sizeReq = sizeLeft;
            if( sizeReq > kBufSize ) sizeReq = kBufSize;
            long sizeRead = fread( iobuf, 1, sizeReq, src );
            sizeLeft -= sizeRead;
            blowfish::decrypt_CBC( pad, iobuf, iobuf, sizeRead, &chain );
            if( sizeLeft == 0 )
                sizeRead -= ( roundUp( dstSize, blowfish::kBlockSize ) -
dstSize );
            fwrite( iobuf, 1, sizeRead, dst );
        }

        chain = blowfish::core::encipherBlock( pad, chain );
        uint32 L = getWord();
        uint32 R = getWord();
        if( chain.L != L    ||    chain.R != R )
```

```
            fail("Invalid checksum while decrypting destination file");
    }

    fclose(src); src = 00;
    fclose(dst); dst = 00;
    return EXIT_SUCCESS;
}
```

CPT.cpp

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <math.h>
#include <cstring>      // 引入字串函數標題檔
#include <ctype.h>      // 引入字元測試與轉換函數標題檔
#include "CPT.h"
#include "time.h"

using namespace std;

void CPT::CPTcreate(unsigned char * buf , unsigned char *buf2 ,int length,int
length2)
{
      FILE *OUT ;
      OUT = fopen("cpt.txt" ,"w");
      for(int i=0;i<16;i++)
      {
            for(int j=0;j<4;j++)
            {
            table[i][j] = 0 ;
            original_temp = 'Z';
            table[i][0] =("%c",original_temp);
            }
       }

       original_temp = buf[0];
       table[0][0] = original_temp;
```

```
for(int i=0; i<length ; i++) //for-1
{
        if ( buf[i] != '\0')
        {

                original_temp = buf[i];
                cout <<" 1 "<<original_temp << endl;

                for (int k=0;k<16;k++)
                {
                  diff_temp = table[k][0];
                    if (original_temp == diff_temp)
                      {
                            table[k][1] = table[k][1] + 1;
                            break;
                      }
                    else if(diff_temp == 'Z')
                      {
                              table[k][0] = original_temp;
                              break;
                      }
          }
   }
        else
        {
              break;
        }
} //endof for-1


//-------------------------------------------------------------

for(int i=48;i<58;i++)
{
      for(int j=0;j<16;j++)
      {
            if( i == table[j][0])
            {break;}
```

```
                else if ( table[j][0] == 'Z')
                {
                        table[j][0] = i;
                        table[j][1] = j;    //-1
                        break;
                }
        }
  }
//-------------------------------------------------------------
for(int i=65;i<71;i++)
{
        for(int j=0;j<16;j++)
        {
                if( ("%c",i) == table[j][0])
                {break;}
                else if ( table[j][0] == 'Z')
                {
                        table[j][0] = i;
                        table[j][1] = j;    //-1
                        break;
                }
        }
  }
//-------------------------------------------------------------
    table[0][1] = table[0][1] -1;
    for(int i=0 ; i<16;i++)
          table[i][1] = table[i][1] + 1;
    //-------------------------------------------------------------

    for(int i=0; i<16;i++)
    {
          if(table[i][1] > 16)
              table[i][1] = table[i][1]%16;
          if(table[i][1] == 16)
              table[i][1] = table[i][1] - 1;
    }
  //-------------------------------------------------------------
```

```
        table[0][2] = 1;
        for(int i=0;i<16;i++)
        {
              frequency = table[i][1];
              t1 = table[i][2];
              t2 = t1 + frequency - 1;
              t3 = t1 + frequency;
              table[i][3] = t2;
              if((i+1) < 16)
                 table[i+1][2] = t3;
              else
                    break;
        }
      //-------------------------------------------------------------------
        for(int i=0;i<16;i++)
        {
             fprintf(OUT,"%6c",table[i][0]);
              for(int j=1;j<4;j++)                                    //原本是 0-4 導
致多秀出[X][0]的十進位
                    fprintf(OUT,"%6d",table[i][j]);
              //printf("\n");
              fprintf(OUT,"\n");

        }
          fclose(OUT);
        for(int g=0;g<16;g++)
           printf("->%c %d %d      %d
\n" ,table[g][0],table[g][1],table[g][2],table[g][3]);

        //-------------------------------------------------------------------
        //
        //     CPT
        //
        //-------------------------------------------------------------------

        OUT = fopen("PIF.txt" ,"w");

        for(int i=0;i<length2;i++)
```

```
        {
            if(buf2[i] != '\0')
            {
                original_temp2 = buf2[i];
                srand(time(NULL));
                for(int j=0;j<16;j++)
                {
                    diff_temp2 = table[j][0];
                    if(original_temp2 == diff_temp2)
                    {
                        //printf("%d == %d\n",original_temp2    ,
diff_temp2);

                        random_index = (rand() % (table[j][3]-table[j][2]
+1 )) + (table[j][2]);
                       // fprintf(OUT," ");
                        fprintf(OUT,"%d\t",random_index);
                       /* if(random_index >0 && random_index <10)
                        {fprintf(OUT,"000");
fprintf(OUT,"%d",random_index); break;}
                        if(random_index >=10 && random_index <100)
                        {fprintf(OUT,"00");
fprintf(OUT,"%d",random_index); break;}
                        if(random_index >=100 && random_index <256)
                        {fprintf(OUT,"0");
fprintf(OUT,"%d",random_index);break ;} */
                    }
                }

            }
        }
        fclose(OUT);
}
```

Decryption.cpp

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
```

```
#include <math.h>
#include <cstring>       // 引入字串函數標題檔
#include <ctype.h>       // 引入字元測試與轉換函數標題檔
#include "DECRYPTION.h"
#include "time.h"
void DECRYPT::Decrypting()
{
   FILE *input_fp;
   char input_ch[100];

   input_fp = fopen("cheatingtext.txt","w");
// printf("Step 1 >>請輸入妳所接收到的 Cheating text,按 ENTER 結束輸
入:\n");
//    while((input_ch = getche()) != '\r')
//         putc(input_ch,input_fp);
// fclose(input_fp);
// printf("Cheating text  內容設定完成\n");

puts("Step 1 >>請輸入你欲使用的 Cheating text,按 ENTER 結束輸入:");
//fix20100809
gets(input_ch);
puts("輸入的字串為：");
puts(input_ch);
for(int i=0 ; i<100;i++)
        {
             if (input_ch[i] == '\0')
                  break;
            fprintf(input_fp,"%c",input_ch[i]);
         }
   fclose(input_fp);

    De_IN = fopen("cheatingtext.txt" ,"r");
      De_OUT= fopen("decrypt_temp.txt" , "w");
   if (De_IN != NULL)
   {
       while((De_ch = getc(De_IN)) != EOF)
       {
             De_count++;
```

```
            De_buf[m++] = De_ch;
            printf("%c",De_ch);
        }
        m = 0;
        fclose(De_IN);
        cout << "INPUT 總共有" << De_count << "個字元" << endl;
        De_count = 0 ;
    }
    else
         cout << "Can't open the file" << endl;
    for(int i=0 ; i<De_SIZE;i++)
        {
            if (De_buf[i] == '\0')
                break;
           // printf("0x%02x\n",De_buf[i]);
            fprintf(De_OUT,"%x",De_buf[i]);
            // write HEX code to temp.txt
        }
  // len = strlen((const char*)buf2);
  // printf("%d" ,len);
   //計算字串長度
   fclose(De_OUT);
//---------------------------------------------------------------------------
   De_OUT= fopen("decrypt_temp.txt" , "r");

   if (De_OUT != NULL)
   {
        while((De_temp_ch = getc(De_OUT)) != EOF)
        {
            De_count++;
            De_temp_buf[n++] = De_temp_ch;
           // printf("%c\n",De_temp_ch);
        }
        n = 0;
        cout << "Temp 總共有" << De_count << "個字元" << endl;
        De_count = 0;
   }
   else
```

```
        cout << "Can't open the file" << endl;
  fclose(De_OUT);

  //-------------------------------------------------------------------------
  De_len = strlen((const char*)De_temp_buf);                    // 取得字
串長度
  for (int i = 0; i <= De_len; i++)                  // 轉成大寫迴圈
  {
    if (islower(De_temp_buf[i]) != 0)                    // 若為小寫字元
      De_temp_buf[i] = toupper(De_temp_buf[i]);                  // 轉成大
寫字元
  }
  cout << "轉換大寫後：" << De_temp_buf << endl      // 顯示轉換後字串
       << endl;
  cout << "temp_buf_size =" << sizeof(De_temp_buf) << endl;
 // cout << "----------以下是給 plaintext 轉換用的----------" << endl;
  //-------------------------------------------------------------------------
  //-------------------------------------------------------------------------
  //
  //    For plaintext   ^_^!!!!!!!!!!
  //
  //-------------------------------------------------------------------------
 /* De_IN2 = fopen("plaintext.txt" ,"r");
  De_OUT2 = fopen("decrypt_temp2.txt" , "w");
  if (De_IN2 != NULL)
  {
      while((De_ch2 = getc(De_IN2)) != EOF)
      {
          De_count2++;
          De_buf2[m++] = De_ch2;
          printf("%c",De_ch2);
      }
      m = 0;
      fclose(De_IN2);
      cout << "INPUT 總共有" << De_count2 << "個字元" << endl;
      De_count2 = 0 ;
  }
  else
```

```
        cout << "Can't open the file" << endl;
  for(int i=0 ; i<De_SIZE;i++)
      {
          if (De_buf2[i] == '\0')
              break;
          printf("0x%02x\n",De_buf2[i]);
          fprintf(De_OUT2,"%x",De_buf2[i]);
          // write HEX code to temp.txt
      }
 // len = strlen((const char*)buf2);
 // printf("%d" ,len);
  //計算字串長度
  fclose(De_OUT2); */
  //-------------------------------------------------------------------------------
  /*De_OUT2= fopen("decrypt_temp2.txt" , "r");

  if (De_OUT != NULL)
  {
      while((De_temp_ch2 = getc(De_OUT)) != EOF)
      {
          De_count2++;
          De_temp_buf2[n++] = De_temp_ch2;
          printf("%c\n",De_temp_ch2);
      }
      n = 0;
      cout << "Temp2 總共有" << De_count2 << "個字元" << endl;
      De_count2 = 0;
  }
  else
       cout << "Can't open the file" << endl;

  fclose(De_OUT2);
  */
  //-------------------------------------------------------------------------------
 /* De_len2 = strlen((const char*)De_temp_buf2);                    // 取
得字串長度
  for (int i = 0; i <= De_len2; i++)                    // 轉成大寫迴圈
  {
```

```
    if (islower(De_temp_buf2[i]) != 0)                    // 若為小寫字元
      De_temp_buf2[i] = toupper(De_temp_buf2[i]);                    // 轉成
大寫字元
  }
  cout << "轉換大寫後：" << De_temp_buf2 << endl      // 顯示轉換後字串
       << endl;
  cout << "temp_buf_size =" << sizeof(De_temp_buf2) << endl;*/
  cout << "以下是創造 CPT 的步驟" << endl;
  //-------------------------------------------------------------------------------
  //
  //                    For CPT
  //
  //-------------------------------------------------------------------------------
    De_OUT3 = fopen("decrypt_cpt.txt" ,"w");
    for(int i=0;i<16;i++)
    {
        for(int j=0;j<4;j++)
        {
        De_table[i][j] = 0 ;
        De_original_temp = 'Z';
        De_table[i][0] =("%c",De_original_temp);
        }
     }

     De_original_temp = De_temp_buf[0];
     De_table[0][0] = De_original_temp;

    for(int i=0; i<De_len ; i++) //for-1
    {
          if ( De_temp_buf[i] != '\0')
          {

              De_original_temp = De_temp_buf[i];
              cout <<" 1 "<<De_original_temp << endl;

              for (int k=0;k<16;k++)
              {
                De_diff_temp = De_table[k][0];
```

```
                    if (De_original_temp == De_diff_temp)
                       {
                            De_table[k][1] = De_table[k][1] + 1;
                            break;
                       }
                    else if(De_diff_temp == 'Z')
                       {
                             De_table[k][0] = De_original_temp;
                             break;
                       }
          }
    }
         else
         {
               break;
         }
} //endof for-1

//-------------------------------------------------------------

for(int i=48;i<58;i++)
{
       for(int j=0;j<16;j++)
       {
             if( i == De_table[j][0])
             {break;}
             else if ( De_table[j][0] == 'Z')
             {
                   De_table[j][0] = i;
                   De_table[j][1] = j; //-1
                   break;
             }
       }
  }
//-------------------------------------------------------------
for(int i=65;i<71;i++)
{
       for(int j=0;j<16;j++)
```

```
        {
            if( ("%c",i) == De_table[j][0])
            {break;}
            else if ( De_table[j][0] == 'Z')
            {
                De_table[j][0] = i;
                De_table[j][1] = j; //-1
                break;
            }
        }
    }
  //---------------------------------------------------------------
De_table[0][1] = De_table[0][1] -1;
    for(int i=0 ; i<16;i++)
        De_table[i][1] = De_table[i][1] + 1;
     //---------------------------------------------------------------

    for(int i=0; i<16;i++)
    {
        if(De_table[i][1] > 16)
            De_table[i][1] = De_table[i][1]%16;
        if(De_table[i][1] == 16)
            De_table[i][1] = De_table[i][1] - 1;
    }
   //---------------------------------------------------------------
  De_table[0][2] = 1;
    for(int i=0;i<16;i++)
    {
        De_frequency = De_table[i][1];
        De_t1 = De_table[i][2];
        De_t2 = De_t1 + De_frequency - 1;
        De_t3 = De_t1 + De_frequency;
        De_table[i][3] = De_t2;
        if((i+1) < 16)
           De_table[i+1][2] = De_t3;
        else
             break;
    }
```

```
      //---------------------------------------------------------------------
         for(int i=0;i<16;i++)
         {
             fprintf(De_OUT3,"%6c",De_table[i][0]);
             {for(int j=1;j<4;j++)     //原本是 0-4 導致多秀出[X][0]的十進位
                  fprintf(De_OUT3,"%6d",De_table[i][j]);}
              //printf("\n");
              fprintf(De_OUT3,"\n");
         }
          fclose(De_OUT3);
         for(int g=0;g<16;g++)
            printf("->%c %d %d      %d
\n" ,De_table[g][0],De_table[g][1],De_table[g][2],De_table[g][3]);
//-----------------------------------------------------------------------------------------------
         //
         //   開始進行對照
         //
//-----------------------------------------------------------------------------------------------
   /*
         De_IN4 = fopen("PIF.txt","r");
         De_OUT4 = fopen("decrypt_pif.txt","w");

   if (De_IN4 != NULL)
   {
       while((De_ch4 = getc(De_IN4)) != EOF)
       {
            De_count++;
            De_buf4[m++] = ("%s",De_ch4);
            printf("%s\n",De_ch4);

       }
       m = 0;
       fclose(De_IN4);
       cout << "INPUT 總共有" << De_count << "個字元" << endl;
       De_count = 0 ;
   }
   else
        cout << "Can't open the file" << endl;
```

```
        fclose(De_IN4);
 */

//-------------------------------------------------------------------------------------------
-------------------------------------------------------
  //
  //    最後的比對動作
  //    1. PIF.txt 的為 2 20 ...
  //    2. 透過 De_table[16][4] 之後可以得知原來的字元  5468....
  //    3. 接著將這些字元寫進 decrypt_index.txt
  //    4. 再來將檔案內的讀出來之後,兩個一組讀取, 透過位移的方式讓它
轉回 54 68 ...   ex: 十進位(5) * 16(2 的四次方) +  下一個值(4) = 54
  //
//-------------------------------------------------------------------------------------------
-------------------------------------------------------
    FILE *fp= NULL;
    FILE *fp2= NULL;
    FILE *fp3=NULL;
    char tmp[10] = {0};
    char buffer[MAX_NUM] = {0};
    int **V = NULL;
    int row = 0, column = 0, column_tmp = 0;
    int i = 0, j = 0;
    fp = fopen("decrypt_pif.txt", "r"); //PIF.txt
    fp2 = fopen("decrypt_index.txt","w");
    while( fgets(buffer, MAX_NUM, fp) != NULL )
    {
        row += 1;

        for(i=0, j=0; i<=(int)strlen(buffer); i++)
        {
            if( (buffer[i] == '\n') || (buffer[i] == '\t') )
            {
                column_tmp++;
                j = 0;
                memset(tmp, 0, sizeof(tmp));
            }
            else
```

```
            {
                tmp[j] = buffer[i];
                j++;
            }
        }

        if( column_tmp > column )
            column = column_tmp;

        column_tmp = 0;
        memset(buffer, 0, sizeof(buffer));

    }
    V = (int **)malloc(row * sizeof(int *));
    for(i=0; i<row; i++)
        V[i] = (int *)malloc(column * sizeof(int));
    for(i=0; i<row; i++)
        for(j=0; j<column; j++)
            V[i][j] = 0;
    row = 0;
    column = 0;
    memset(tmp, 0, sizeof(tmp));
    memset(buffer, 0, sizeof(buffer));
    fseek(fp, 0, SEEK_SET);
    while( fgets(buffer, MAX_NUM, fp) != NULL )
    {

        for(i=0, j=0; i<=(int)strlen(buffer); i++)
        {
            if( (buffer[i] == '\n') || (buffer[i] == '\t') )
            {
                V[row][column_tmp] = atoi(tmp);

                column_tmp += 1;
                j = 0;
                memset(tmp, 0, sizeof(tmp));
            }
            else
```

```
            {
                tmp[j] = buffer[i];
                j++;
            }
        }

        row += 1;

        if( column_tmp > column )
            column = column_tmp;

        column_tmp = 0;
        memset(buffer, 0, sizeof(buffer));

    }
    fclose(fp);
    int gg;
    int bb;
    //int cc;
    //unsigned char zz;
    //unsigned char ff;
      int m_len;
    //for(i=0; i<row; i++)
    //     for(j=0; j<column; j++)
    //          printf("V[%d][%d] = %d\n", i, j, V[i][j]);

    for(i=0; i<row; i++){
        for(j=0; j<column; j++){
            gg = V[i][j];
            for(int q=0;q<16;q++)
            {

                if(gg <= De_table[q][3] )
                {
                    bb = De_table[q][0];
                    //printf("%c--" , bb);
                    fprintf(fp2,"%c",bb);
```

```
                    /*
                        if(bb >= '0' && bb <= '9')
                            zz = ((bb - '0') << 4);
                        else
                            zz = ((bb - 'A' + 10) << 4);
                        printf("%X_", zz);

                        ff = zz - 27;
                        printf("%d_",ff);
                    */
                        break;
                    }
                }
                //printf("V[%d][%d] = %d\n", i, j ,V[i][j]);
            }
        }
        fclose(fp2);
        fp2 = fopen("decrypt_index.txt","r");
        fp3 = fopen("decrypt_text.txt","w");
        fscanf(fp2,"%s",&De_buf6);
        for(int i=0 ; i<De_SIZE ;i++){
                if(De_buf6[i] != '\0')
                    printf("%c",De_buf6[i]);
                else break;}
        printf("\n");
       // m_len = strlen((const char*)De_buf2);
       // printf("m_len == %d\n", m_len);
       // printf("The plaintext ==>>");

        for(int i=0 ; i<32767 ;i+=2)
        {
            if(De_buf6[i] != '\0')
            {

                for(int j=0 ; j<2;j++)
                {
                    m1 = De_buf6[i+j];
                    if(m1 == '0')
```

```
                { m2 = 0;}
                 else if (m1 == '1')
                 {m2 = 1;}
                 else if (m1 == '2')
                 {m2 = 2;}
                 else if (m1 == '3')
                 {m2 = 3;}
                 else if (m1 == '4')
                 {m2 = 4;}
                 else if (m1 == '5')
                 {m2 = 5;}
                 else if (m1 == '6')
                 {m2 = 6;}
                 else if (m1 == '7')
                 {m2 = 7;}
                 else if (m1 == '8')
                 {m2 = 8;}
                 else if (m1 == '9')
                 {m2 = 9;}
                 else if (m1 == 'A')
                 {m2 = 10;}
                 else if (m1 == 'B')
                 {m2 = 11;}
                 else if (m1 == 'C')
                 {m2 = 12;}
                 else if (m1 == 'D')
                 {m2 = 13;}
                 else if (m1 == 'E')
                 {m2 = 14;}
                 else if (m1 == 'F')
                 {m2 = 15;}

                 if(j == 0)       //前面 4 bits
                 { m_hex = m2 * 16;}
                 else if (j == 1) //  後面  4bits
                 { m_hex2 = m2;}
        } //for
                m_hex3 = m_hex + m_hex2; //組合起來
```

```
                        printf("%c",m_hex3);
                        fprintf(fp3,"%c",m_hex3);
                    }    //if
                else
                    break;
            } //for
        fclose(fp2);
        fclose(fp3);
}
```