

國立暨南國際大學通訊工程研究所

碩 士 論 文

H.264 視訊串流之資訊隱藏技術

Real-time Steganography in Video Streaming
over H.264

指導教授：吳坤熹 博士

研究生：賴韋立

中華民國 九十九年 六月

國立暨南國際大學通訊工程研究所

碩 士 論 文

H.264 視訊串流之資訊隱藏技術

Real-time Steganography in Video Streaming
over H.264

指導教授：吳坤熹 博士

研究生：賴韋立

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 賴韋立 所提之論文

H.264 視訊串流之資訊隱藏技術

Real-time Steganography in Video Streaming over H.264

(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

許靜芳

委員兼召集人

吳坤熹

委員

李佩君

委員

中 華 民 國 99 年 6 月 17 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)

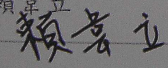
本授權書所授權之論文為授權人在 暨南國際大學 通訊工程研究所 _____ 組 98 學年度第二學期取得 碩士學位之論文。

論文題目：H.264視訊串流之資訊隱藏技術
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：賴章立

簽名：  _____

中華民國 99 年 08 月 05 日

致謝

能夠順利完成此篇論文，同時要感謝我的家人、同學和老師，不管是家人的支持或同學的幫助，還是老師的指導，都讓我能夠順利的完成學業。在這兩年間很幸運的能接受到他們的幫助，尤其是我的指導老師吳坤熹博士，老師在研究上嚴謹的指導與啟蒙，讓我更了解許多網路方面的專業知識，不管是在專業領域或是為人處事上，都給了我莫大的幫助。另外要感謝實驗室的各位同學，當我在研究遇到困難時，也給了我許多幫助，因為有你們我才能順利得完成此論文，並且讓我的碩士生涯過的多采多姿，充滿了許多的回憶。

論文名稱：H.264 視訊串流之資訊隱藏技術

校院系：國立暨南國際大學科技學院通訊工程研究所

頁數：51

畢業時間：99 年 6 月

學位別：碩士

研究生：賴韋立

指導教授：吳坤熹 博士

摘要

在即時通訊系統之中，即時視訊隱藏是將秘密視訊藏匿於掩護視訊之中的一項技術。藉由隱藏秘密視訊到掩護視訊之中，我們可以獲得一個隱密視訊；此視訊為一段有意義的畫面，並且和掩護視訊畫面看起來非常相似。藉由這種保護方式，當我們在進行視訊通話時，就算不幸遭受到有心人士竊取封包，竊取者也不會看到掩護視訊的畫面，而不會注意到隱匿在裡面的秘密視訊。

在本篇論文中，我們提出一種在網路電話（Voice over Internet Protocol，簡稱 VoIP）上隱藏視訊的方法，其方法我們稱之為即時視訊隱藏（Real-time Video Hiding，簡稱 RVH）技術，並進一步藉由 H.264/AVC 編碼技術來驗證我們的方法。H.264/AVC 編碼技術為目前最新的數位視訊編解碼器標準，同時已普遍的運用在網路電話上。經由我們的實驗證明，RVH 將能有效增加網路電話的安全性。

關鍵字：人類視覺系統，最低有效位元，資訊隱藏，H.264/AVC

目錄

| | |
|---|------|
| 致謝..... | I |
| 摘要..... | II |
| Abstract | III |
| 目錄..... | IV |
| 圖目錄..... | VI |
| 表目錄..... | VIII |
| 第 1 章 資訊隱藏技術簡介..... | 1 |
| 1.1 前言和動機..... | 1 |
| 第 2 章 現有文獻探討..... | 4 |
| 2.1 最低有效位元 (Least Significant Bit)..... | 4 |
| 2.2 資訊隱藏技術..... | 4 |
| 2.3 H.264/AVC 簡介..... | 7 |
| 第 3 章 系統架構..... | 10 |
| 3.1 視訊隱藏空間..... | 11 |
| 3.2 壓縮秘密視訊..... | 16 |
| 3.3 隱藏秘密視訊資料..... | 17 |
| 3.4 提取秘密視訊資料..... | 19 |
| 第 4 章 系統實作..... | 20 |
| 4.1 Linphone 系統架構..... | 20 |
| 4.2 mediastreamer2 系統架構..... | 22 |
| 4.3 程式撰寫流程..... | 24 |
| 第 5 章 實驗結果..... | 27 |

| | | |
|-------|---|----|
| 5.1 | 實驗成果..... | 27 |
| 5.2 | Peak Signal-to-Noise Ratio (PSNR)比較分析 | 30 |
| 第 6 章 | 結論和未來工作..... | 33 |
| 參考文獻 | | 35 |
| 附錄 A | Linphone 安裝 | 37 |
| A.1 | 目的..... | 37 |
| A.2 | 檔案下載..... | 37 |
| A.3 | 安裝說明..... | 37 |
| 附錄 B | 程式碼..... | 41 |
| B.1 | 目的..... | 41 |
| B.2 | 程式碼下載..... | 41 |
| B.3 | 程式碼說明..... | 41 |

圖目錄

| | |
|--|----|
| 圖 1.1 傳輸檔案示意圖 | 1 |
| 圖 1.2 資訊隱藏示意圖 | 3 |
| 圖 2.1 最低有效位元 | 4 |
| 圖 2.2 掩護圖像 | 5 |
| 圖 2.3 秘密圖像 | 6 |
| 圖 2.4 NAL unit stream 結構層 | 8 |
| 圖 3.1 傳送端之流程圖 | 10 |
| 圖 3.2 接收端之流程圖 | 11 |
| 圖 3.3 選擇視訊隱藏的像素 | 12 |
| 圖 3.4 選擇視訊隱藏空間 | 13 |
| 圖 3.5 選擇視訊隱藏空間($b = 2, p = 1$) | 14 |
| 圖 3.6 選擇視訊隱藏空間($b = 1, p = 2$) | 15 |
| 圖 3.7 選擇視訊隱藏空間($b = 4, p = 1$) | 16 |
| 圖 3.8 壓縮秘密視訊 | 16 |
| 圖 3.9 隱藏秘密視訊到掩護視訊($b = 2, p = 1$) | 17 |
| 圖 3.10 隱藏秘密視訊到掩護視訊($b = 4, p = 1$) | 18 |
| 圖 4.1 Linphone 系統架構 | 21 |
| 圖 4.2 mediastreamer2 呼叫程式流程 | 22 |
| 圖 4.3 video_stream_start 執行序列 | 23 |
| 圖 4.4 mediastreamer2 資料流程 | 23 |
| 圖 4.5 msx264 執行程序 | 25 |
| 圖 4.6 msx264 修改後執行程序 | 26 |

| | |
|------------------------------|----|
| 圖 5.1 原始掩護視訊 | 28 |
| 圖 5.2 原始秘密視訊 | 28 |
| 圖 5.3 RVH_4bit 掩護視訊..... | 29 |
| 圖 5.4 RVH_4bit 還原後的秘密視訊..... | 29 |

表目錄

| | |
|------------------------|----|
| 表 3.1 視頻解析度規格表 | 14 |
| 表 5.1 實驗器材規格 | 27 |
| 表 5.2 視訊 PSNR 比較 | 32 |

第1章 資訊隱藏技術簡介

1.1 前言和動機

在視訊電話系統中，視訊需要編碼成數位封包，然後在網際網路上傳輸。但由於現今的網際網路是一個開放式的環境，當封包在網際網路上傳輸時，可能會遭受到有心人士的窺探或監聽，如下圖 1.1 所示。為此我們需要透過一些方式來保護那些重要的資料，而最常見的方式就是使用密碼學(Cryptography)，例如：數據加密標準 (Data Encryption Standard，簡稱 DES) 或高階加密標準 (Advanced Encryption Standard，簡稱 AES)。密碼學可改變檔案或訊息的內容，讓除了目標收件者之外的任何人都無法得知內容，只有目標收件者握有「金鑰 (Key)」，可以解開加密檔案的鎖，並檢視檔案。但加密的訊息並不是隱藏的，在網際網路上傳輸時都有可能被偵測或監視。

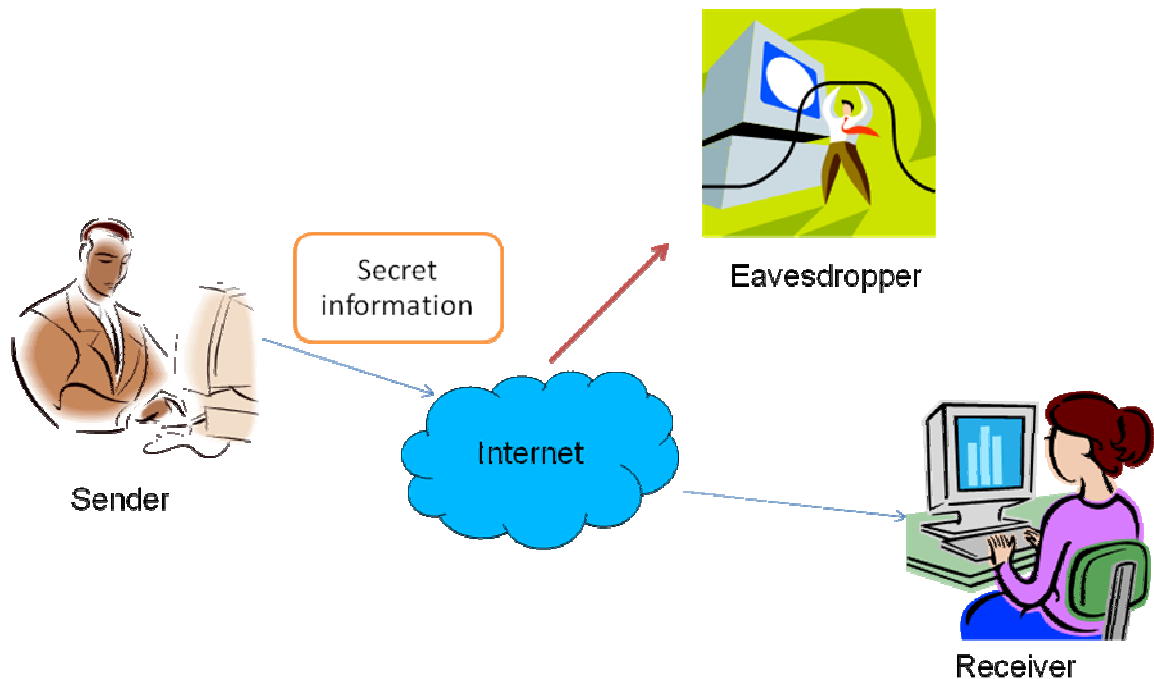


圖 1.1 傳輸檔案示意圖

雖然密碼學可以保護視訊內容，但是會使有心人士注意到此視訊是經過加密的資料。同時因為加密的信息通常都包含重要的秘密訊息，這將鼓勵有心人士投入更多的資源進行破解，所以我們需要藉由資訊隱藏(Steganography)來保護視訊內容。密碼學和資訊隱藏是兩種不同隱藏訊息的方法，雖然可以相輔相成，兩者卻是不同的技術。我們可以把資訊隱藏想成一種非常「低調」的偽裝方式，而它隱藏訊息的方式，會讓觀察者察覺不到有訊息的交換。它和密碼學不一樣，資訊隱藏是不易被察覺到的，所以某種角度而言，它比資料加密的方法可提供更好的安全性。

資訊隱藏技術已廣泛的運用在各個方面，例如：文字的訊息 [1]、WAV 或 MP3 格式的聲音檔案 [2][3]、BMP 格式的圖像檔案[4]、JPEG 格式的圖像檔案[5]或是 AVI 格式的影像檔案[6]。

在這篇論文，我們提出一種在即時網路電話系統下隱藏資訊的方法，其方法我們稱之為即時視訊串流隱藏 (Real-time Video Hiding, 簡稱 RVH) 技術。如下圖 1.2 所示，我們將秘密視訊隱藏到一段有意義的掩護視訊中，便可以獲得一段隱密視訊並透過網際網路傳輸。在理想的情況下，一個好的演算法將產生一個和掩護視訊相差無幾的隱密視訊。

由於人類視覺系統 (Human Visual System, 簡稱 HVS) 所能感受到的視覺訊號有限，視頻畫面在很小失真的情況下，一般是不會被察覺內容有所變更的。在此前提下，我們定義一個視訊隱藏空間來隱藏秘密資料。在適當的範圍內，此視訊不會有嚴重的失真。如此一來，我們就可以在人眼無法察覺的情況下隱藏資訊到掩護視訊裡面。一般來說，在視頻數據取樣中，修改其最低有效位元 (Least Significant Bit, 簡稱 LSB) 將導致最小的失真。這是一個常使用的方法，因為它簡單且容易實現。本篇論文所提出的 RVH 方法就是利用 LSB 技術來達到隱藏資訊的目的。

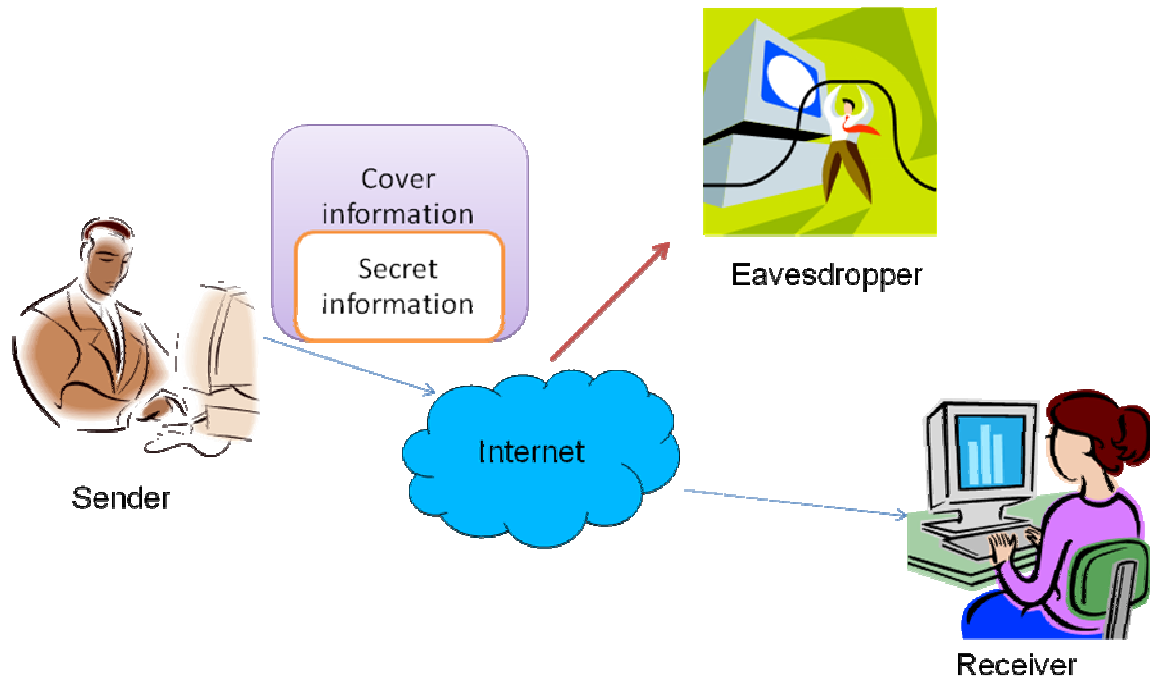


圖 1.2 資訊隱藏示意圖

在接下來的章節中，第二章介紹本論文所研究主題的一些背景知識，像是最低有效位元、資訊隱藏技術、H.264 編解碼器。第三章描述本篇論文所提出的方法，此章節也是此本論文的核心部份。第四章介紹如何實現我們所提出的方法，透過實作將可驗證我們所提出的方法之可行性。最後的兩個章節將會詳細說明此論文的實作結果，並且作一個總結以及未來工作的說明。

第2章 現有文獻探討

2.1 最低有效位元 (Least Significant Bit)

LSB 通常指二進位數字中最右邊的位元，可以用以檢測數字的奇偶性；與之相反的稱之為最高有效位元 (Most Significant Bit，簡稱 MSB)。LSB 代表了二進位數中最小的單位，換句話說就是對數字變化量影響最小的單位，如下圖 2.1 所示，數字 177 的二進位可表示成 10110001，其中最右邊的位元即為最低有效位元。

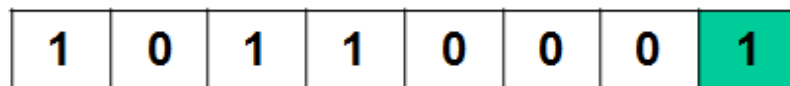


圖 2.1 最低有效位元

2.2 資訊隱藏技術

資訊隱藏是一門關於信息隱藏的技術，也被稱為隱寫術。資訊隱藏這一個概念早在幾百年前已經存在了，在遠古時期的古希臘時代，將文字寫於木板上，再將木板以蠟封住，如此一來就沒有人知道其中的文字內容，直到將蠟括除才能得知訊息。到了二十世紀，被廣泛使用的隱形藥水，像是牛奶、橘子汁、醋等等，都被用來隱藏信息於書信上；除非受到特殊的處理，否則其資訊是不會顯示出來的。例如，當將信紙加熱後，以這些液體書寫其上的隱形文字轉變為黑色時，就可以讀取其信息內容。時空背景轉換到現代，我們所要使用的技術也十分相似，其用意都是

將秘密訊息隱藏在其它信息之中，來達到提昇資訊的安全。

在現代的資訊隱藏技術中，我們藉由「掩護」這個媒介用於隱藏秘密信息，被隱藏的信息可以是一段文字、圖像或是任何可以用二進制代表的訊息。而掩護媒介可以是許多常見的訊息，如文字、聲音、圖片和視頻。例如在一個 24 位元的數位圖像中每一個像素的三種顏色份量（紅、綠、藍）各使用 8 個位元來表示；假設我們只考慮藍色的話，那麼將會有 2^8 種不同的數值可以表示深淺不同的藍色。而像 11111111 和 11111110 這兩個值所表示的藍色，人眼幾乎是無法區分的。因此，這個最低有效位元就可以用來存儲顏色之外的信息，而且在某種程度上幾乎是不會被察覺到的。以下藉由一個簡單的實例作為說明，圖 2.2 乍看之下只是一張普通的樹木照片[7]，但實際上卻隱藏著另一幅圖像在裡面。如果把每一個顏色份量和數字 3 進行邏輯運算，即可得到圖 2.3 的貓照片[7]，這就是在圖像中進行資訊隱藏的一種技術。



圖 2.2 掩護圖像

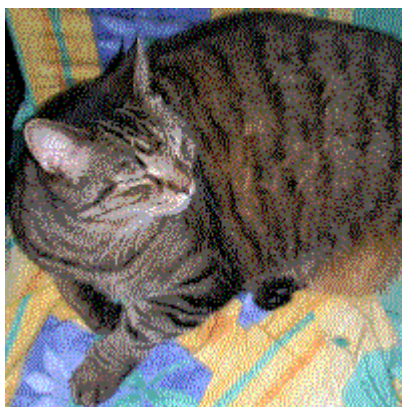


圖 2.3 秘密圖像

目前在數位圖像中隱藏資料的常見方法包含下列三種，以下我們只做簡略的介紹，其中 LSB 為我們 RVH 會使用到的技術之一。

1. LSB：這是一個最常使用的方法，因為它簡單且容易實現；本論文也是使用此方法進行資訊隱藏。在此方法中，圖像中每一個位元組的 LSB 將用於儲存秘密數據，所以產生的失真將會很低，人們的眼睛將不會察覺到失真的問題。但這種技術有一個缺點，由於它使用圖像的每一個像素儲存資料，因此只有在無損壓縮格式中（如 BMP 或 GIF），可以順利運作；但若有損壓縮格式中（如 JPEG），一些隱藏的信息可能會丟失。
2. Masking and filtering：此隱藏信息的方法類似浮水印，例如透過修改圖像亮度的部份。雖然這個技術可以比 LSB 隱藏更多的信息，但是缺點是其失真幾乎是顯而易見的，很容易就會被人發覺此檔案有問題。
3. Transformations：這是在圖像隱藏信息技術中一個很複雜的方式，在論文 [8][9] 中就是使用此方法。此方法需要在圖像在進行轉換處理時才能隱藏信息，例如：基於離散餘弦轉換（Direct Cosine Transform，簡稱 DCT）的資訊隱藏，就是其中一種方法。離散餘弦轉換是進行圖像有損數據壓縮用的，除了離散餘弦轉換，在圖像處理中的快速傅立葉轉換或其他方法，都可以用於隱藏訊息。

2.3 H.264/AVC 簡介

H.264/AVC/MPEG-4 Part10 [10] 是由 ITU-T 視訊編碼專家小組 (VCEG) 和 ISO/IEC 動態圖像專家小組 (MPEG) 聯合組成的聯合視訊團隊 (Joint Video Team) 提出的高度壓縮數位視訊編解碼器標準。ITU-T 的 H.264 標準和 ISO/IEC MPEG-4 Part10 (正式名稱是 ISO/IEC 14496-10) 在編解碼技術上是相同的，此編解碼技術也被稱為進階視訊編碼 (Advanced Video Coding, 簡稱 AVC)。該標準第一版的最終草案已於 2003 年 5 月完成。H.264 是 ITU-T 以 H.26x 系列為名稱命名的標準之一，同時 AVC 是 ISO/IEC MPEG 所命名的稱呼，這個標準通常被稱之為 H.264/AVC，以下我們統一簡稱為 H.264。相關研究顯示 H.264 與 MPEG-2 及 MPEG-4 相較之下，無論是壓縮率或視訊品質皆有大幅度的提升。H.264 也首次將視訊編碼層 (Video Coding Layer, 簡稱 VCL) 與網路提取層 (Network Abstraction Layer, 簡稱 NAL) 的概念涵蓋進來。以往視訊標準只著重於壓縮效能部份，而 H.264 包含了內建的 NAL 網路協定層，藉由 NAL 來提供網路的處理，可以讓 VCL 擁有更好的編解碼能力，使得 H.264 更適用於多媒體串流等相關應用，例如視訊電話就將 H.264 作為視訊的編解碼器。

NAL 是 H.264 影像編碼標準的一部分，以 NAL-unit 為單位的形式做為 VCL 的運算單位。在 NAL-unit Header 中的 NAL-unit Type 欄位記載此封包的類型，每種形式分別對應到 VCL 中不同的編解碼工具，藉由 NAL 來提供網路的狀態，讓 VCL 有更好的編解碼更錯能力。如圖 2.4 所示，一個完整的 H.264 位元串流是由多個 NAL-unit 所組成，所以位元串流也稱之為 NAL unit stream，一個 NAL unit stream 可以包含多個壓縮視訊序列 (codec video sequence)，一個壓縮視訊序列即代表的一個視訊影片，而壓縮視訊序列則是由多個 access units 所組成。當接收端收到一個 access units 後，就可以完整地解碼成單張的畫面；而每一個壓縮視訊序列的第一個 access unit 必須為 Instantaneous Decoding Refresh (IDR)。IDR access unit

的內容全是採用 Intra-prediction 編碼，即本身就可以完全解碼，不需參考其它 access unit 的資料。Access unit 亦是由多個 NAL-units 所組成，標準中總共制定了 12 種的 NAL-unit 型式，此型式進一步可分類成 VCL NAL-unit 和 non-VCL NAL-unit 兩種類型。VCL NAL-unit 純粹只有壓縮影像的內容，而 non-VCL NAL-unit 則有兩種：Supplemental Enhancement Information (SEI) 與 Parameter Sets，SEI 可以存放影片簡介或使用者自行定義的資料；Parameter Sets 主要是存放整個壓縮視訊序列的參數，例如：長寬比例、相關解碼的參考...等等。這些資料非常重要，倘若在傳送過程中發生錯誤，將導致影片無法解碼。因此，H.264 將這些資料採用 out-of-band 的方式傳送，以保證傳輸的正確性。

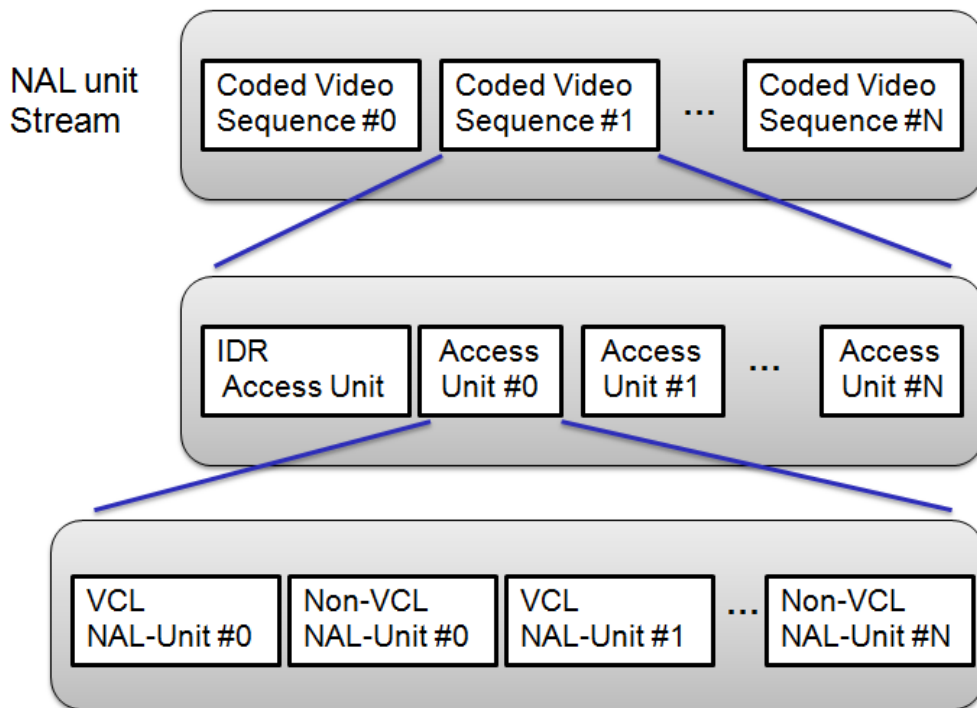


圖 2.4 NAL unit stream 結構層

視訊編碼層的壓縮原理是利用影像在時間與空間上的相似性，這些相似的資料經過壓縮演算法處理之後，可以將人眼無法感知的部份抽離，這些部份稱之為視覺

冗餘(Visual redundancy)。在這些部份被去除之後，就可以達到視訊壓縮的目的，其技術核心包括動作估計、轉換編碼、預測編碼、去區塊效應濾波、熵編碼等等，詳細資料可參考標準文件[10]。

第3章 系統架構

本章節主要在介紹 RVH 方法的架構，RVH 的處理流程可分為傳送端和接收端兩個部份。圖 3.1 表示為傳送端的處理流程，圖 3.2 表示為接收端的處理流程。

傳送端之處理流程：

1. 定義掩護視訊為 C，從掩護視訊中分配一個視訊隱藏空間。分配出來的視訊隱藏空間以 HS 表示，它將使用於儲存秘密視訊的資料。
2. 定義秘密視訊為 S，藉由壓縮 S 我們可以得到一個壓縮過視訊資料 S'，並且將其隱藏到 HS 中。
3. 藉由隱藏 S' 到 HS 中，我們可以獲得一段有意義的隱密視訊，表示為 D。緊接著將 D 透過網際網路傳送給接收端。

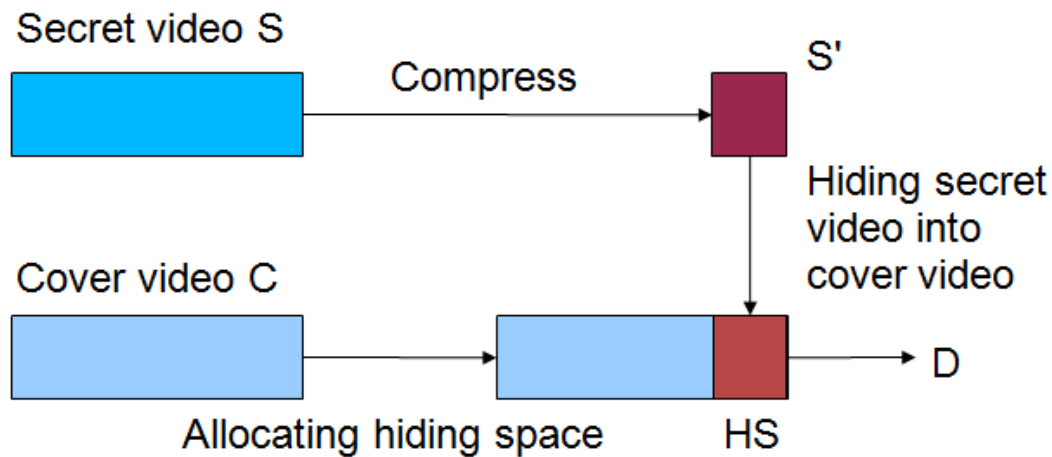


圖 3.1 傳送端之流程圖

接收端之處理流程：

1. 當接收端接收到隱密視訊 D 之後，我們可以從 HS 中提取壓縮過後的資料 S'。
2. 將 S' 經由解壓縮後可獲得秘密視訊 S''。因為壓縮資料和視訊編解碼皆會造成資料的損壞，所以 S'' 會不同於原先的秘密視訊 S。

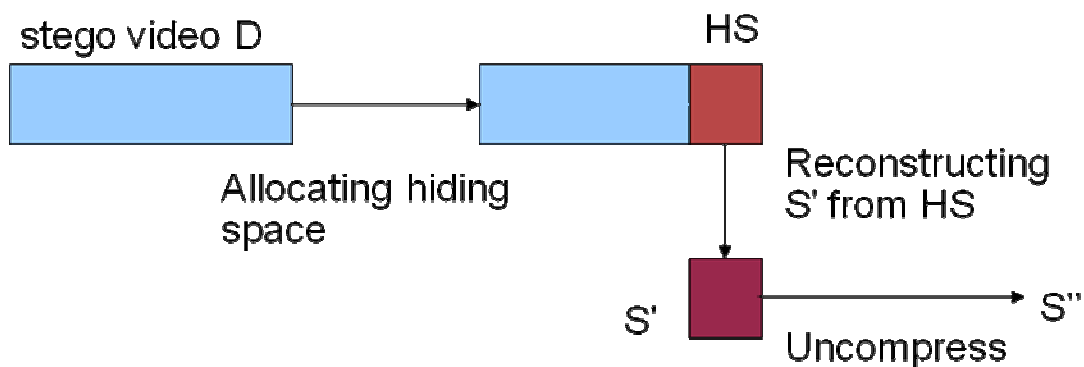


圖 3.2 接收端之流程圖

3.1 視訊隱藏空間

決定視訊隱藏空間是非常重要的，此空間主要是用於隱藏秘密視訊資料。若是空間過大。可能造成掩護視訊嚴重地失真，並讓竊聽者注意到藏於隱秘視訊中的秘密視訊。

為了避免資訊隱藏技術對掩護視訊造成嚴重失真，首先我們選擇掩護視訊的每一個像素作為隱藏空間，如下圖 3.3 所示。

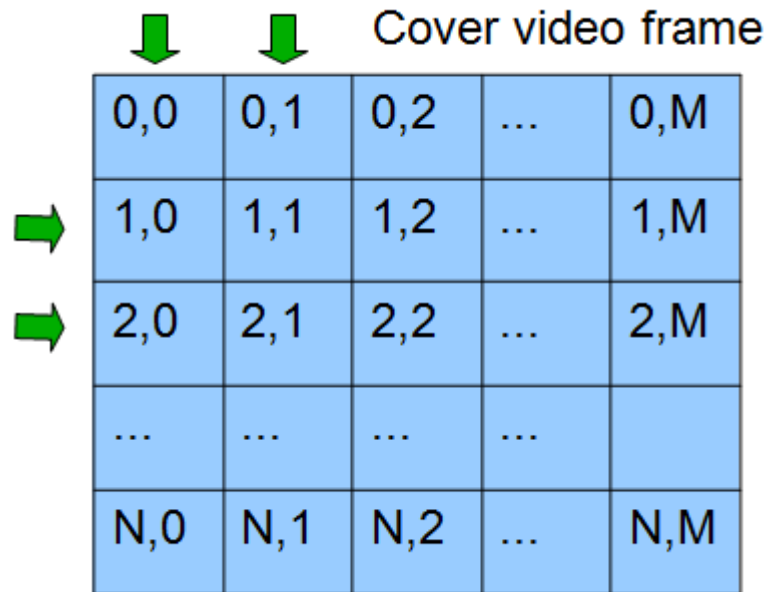


圖 3.3 選擇視訊隱藏的像素

接著在每一個隱藏的像素中，我們選擇最低有效位元來儲存秘密視訊。藉著適當的選擇，我們可以確保掩護視訊在最小失真的情況下隱藏秘密視訊。取樣的這些位元將會被用於儲存秘密視訊，如下圖 3.4 所示。這裡需注意一點，就是在選擇隱藏空間時並非一定要是最低有效位元，因為在 H.264 編解碼器有損壓縮的情況下，最低有效位元的內容很容易受到破壞，所以隱藏空間需要在確保掩護視訊在最小失真和秘密視訊品質的情況下慎重選擇。

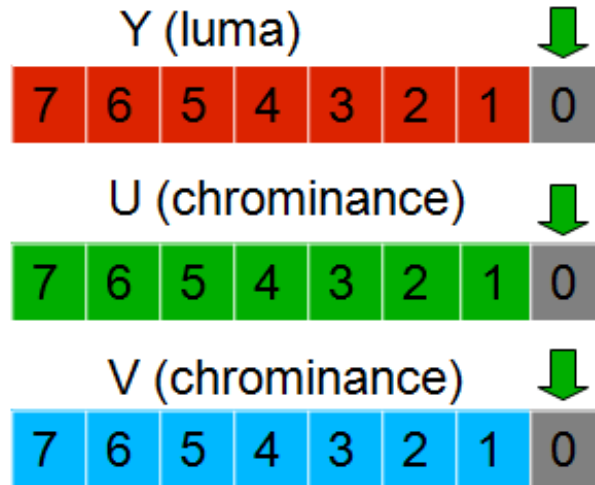


圖 3.4 選擇視訊隱藏空間

另外為了決定需要隱藏的位元數目，我們可以透過下列公式計算所需要的值。首先我們定義： N_s 為秘密視訊的解析度大小， N_c 為掩護視訊的解析度大小。依照公式 (1) 可計算出一個像素需要的隱藏空間位元數 b 以及需要選擇的隱藏像素 p (即表示我們需要選擇掩護視訊每 p 個像素中的 b 個位元作為隱藏空間)。另外必須注意此公式，在 N_s 解析度等於 N_c 解析度時將視為特殊情況處理，例如只取秘密視訊的最高有效位元來作為隱藏的資料。

$$\frac{b}{p} = \begin{cases} \frac{N_s \times 8}{N_c} & \text{if } N_s < N_c \\ \frac{4}{1} & \text{if } N_s = N_c \end{cases} \quad (1)$$

b 和 p 的數值大小主要取決於視頻規格，像是 CIF、QCIF 等。在 ITU-T 的 H.323[11]標準中制定了不同的規格，如常用於視訊電話會議中的 CIF 即為其中之一，其它的規格可參考表 3.1。

表 3.1 視頻解析度規格表

| 格式 | 視訊解析度 |
|-------|-------------|
| SQCIF | 128 x 96 |
| QCIF | 176 x 144 |
| CIF | 352 x 288 |
| 4CIF | 704 x 576 |
| 16CIF | 1408 x 1152 |

情況一：假設秘密視訊的規格為 QCIF，掩護視訊的規格為 CIF；根據表 3.1 所示，Ns 為 176 × 144，Nc 為 352 × 288。因為 Ns 小於 Nc，所以依照公式 (1) 我們可以計算出 b = 2 和 p = 1，即表示我們需要選擇掩護視訊每 1 個像素中的 2 個位元作為隱藏空間，如圖 3.5 所示。

$$\frac{b}{p} = \frac{N_s \times 8}{N_c} = \frac{176 \times 144 \times 8}{352 \times 288} = \frac{2}{1}$$

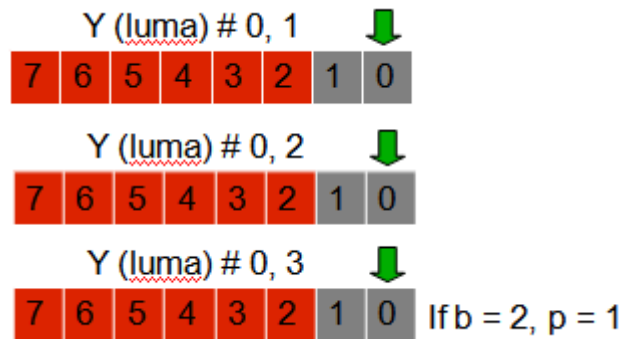


圖 3.5 選擇視訊隱藏空間(b = 2, p = 1)

情況二：假設秘密視訊的規格為 QCIF，掩護視訊的規格為 4CIF；根據表 3.1 所示，Ns 為 176 × 144，Nc 為 704 × 576。因為 Ns 小於 Nc，所以依照公式 (1) 我們可以計算出 b = 1 和 p = 2，即表示我們需要選擇掩護視訊每 2 個像素中的 1 個位元作為隱藏空間，如圖 3.6 所示。

$$\frac{b}{p} = \frac{N_s \times 8}{N_c} = \frac{176 \times 144 \times 8}{704 \times 576} = \frac{1}{2}$$

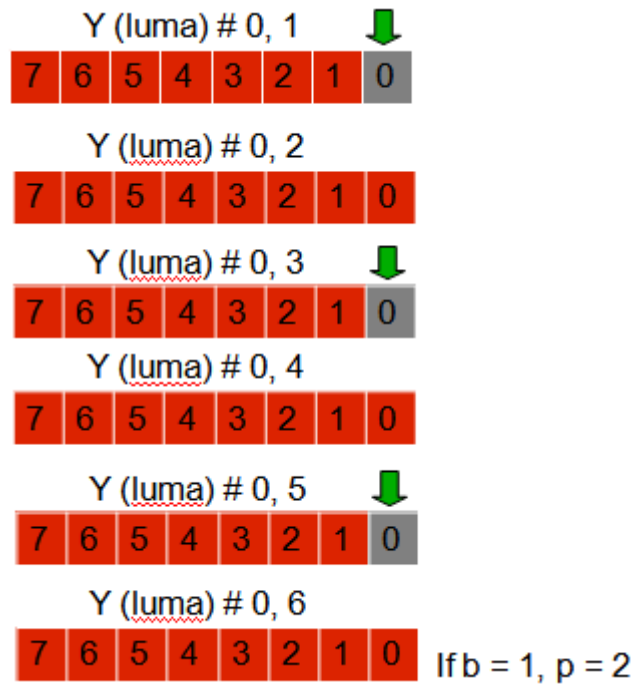


圖 3.6 選擇視訊隱藏空間(b = 1, p = 2)

情況三：此為特殊的情況，假設秘密視訊和掩護視訊的規格皆為 CIF；根據表 3.1 所示， N_s 為 352×288 ， N_c 為 352×288 。因為 N_s 等於 N_c ，所以依照公式 (1) 我們可以得知 $b = 4$ 和 $p = 1$ ，即表示我們需要選擇掩護視訊每 1 個像素中的 4 個位元作為隱藏空間，如圖 3.7 所示。

$$\frac{b}{p} = \frac{4}{1}$$

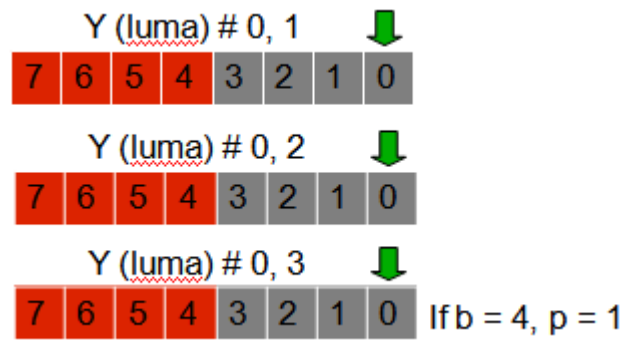


圖 3.7 選擇視訊隱藏空間($b = 4, p = 1$)

3.2 壓縮秘密視訊

為了將秘密視訊藏匿到隱藏空間中，我們必須對秘密視訊進行資料壓縮。資料壓縮主要是為了改變秘密視訊解析度大小，以減少需要隱藏的資料量。例如 CIF (352×288) 轉變為 QCIF (176×144) 時，其資料量約可縮為四分之一，圖 3.8 為改變解析度的示意圖。

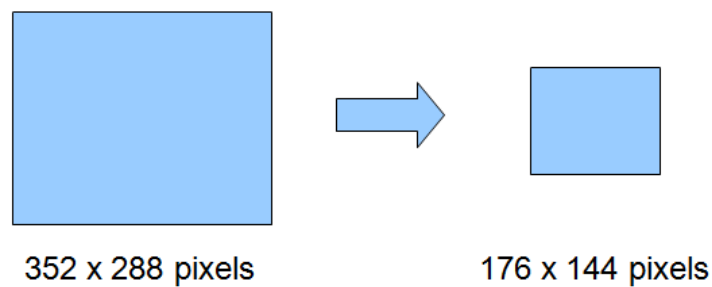


圖 3.8 壓縮秘密視訊

3.3 隱藏秘密視訊資料

在這個章節，將會說明在兩種不同的情況下，如何將秘密視訊藏匿到掩護視訊的隱藏空間中。

1. 當我們計算出 $b = 2$ 與 $p = 1$ 時，圖 3.9 說明了秘密視訊隱藏到掩護視訊的程序。首先，我們在傳送端壓縮秘密視訊資料後，經由公式(1)計算後得知 $p = 1$ ，所以我們選擇掩護視訊中所有的像素作為隱藏空間，像素位置 $cover\#0,0 \dots cover\#N,M$ 將利用於儲存秘密資料。緊接著因為 $b = 2$ ，所以我們依序將 S' 中的 2 個位元藏入到隱藏空間的位置，也就是掩護視訊中每 1 個像素的 2 個最低有效位元。

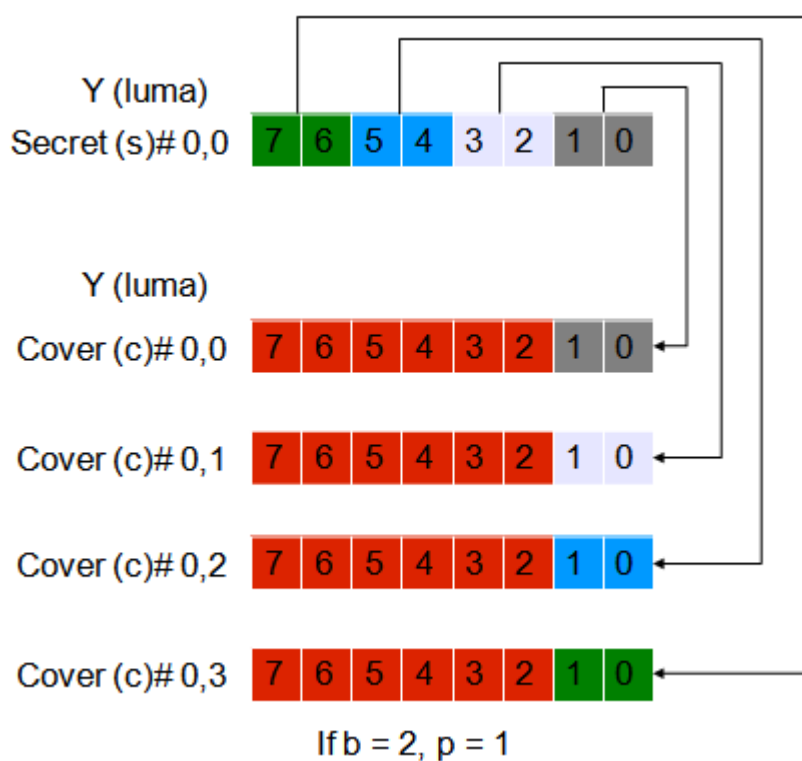


圖 3.9 隱藏秘密視訊到掩護視訊 ($b = 2, p = 1$)

2. 當 N_s 等於 N_c 時，我們透過公式(1)可得知 $b = 4$ 與 $p = 1$ 。圖 3.10 說明了秘密視訊隱藏到掩護視訊的程序。首先，我們在傳送端壓縮秘密視訊資料後，經由公式(1)計算後得知 $p = 1$ ，所以我們選擇掩護視訊中所有的像素作為隱藏空間，像素位置 $cover\#0,0 \dots cover\#N,M$ 將利用於儲存秘密資料。緊接著因為 $b = 4$ ，所以我們依序將 S' 中的 4 個位元藏入到隱藏空間的位置，也就是掩護視訊中每 1 個像素的 4 個最低有效位元。此外，和情況一不同的是，在這個特殊的情況中，我們只選擇秘密視訊中每 1 個像素的 4 個最高有效位元作為隱藏的資料。

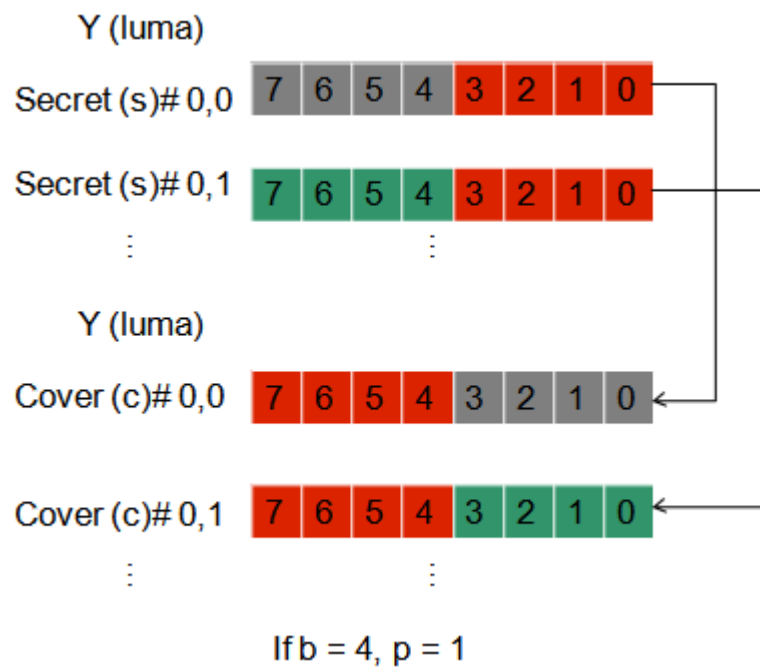


圖 3.10 隱藏秘密視訊到掩護視訊 ($b = 4, p = 1$)

3.4 提取秘密視訊資料

當接收端接收到隱密視訊資料後，進行解碼的動作，即可得到傳送端傳送的秘密視訊。首先我們必須在隱藏空間的位置提取出資料，然後重新組合成壓縮過後的 S' ，並進行解壓縮後，即可得到秘密視訊 S'' ，流程如圖 3.2 所示。除此之外，因為在進行 RVH 時，會使用到 H.264 編碼技術和壓縮技術，而這些技術都會將原有視訊內容輕微破壞，造成秘密視訊 S'' 的失真，因此取得的秘密視訊 S'' 和傳送端原始的秘密視訊 S 是不相同的；但是只要透過適當的參數選取，令此失真不嚴重到讓接收端看不清楚內容即可。

第4章 系統實作

4.1 Linphone 系統架構

為了驗證 RVH 的效能，我們將提出的 RVH 演算法實作於 Linphone [12]，這套開放源代碼的軟體。Linphone 符合了 IETF 制定的標準會議初始協定(Session Initiation Protocol，簡稱 SIP) [13]，所以它能與大多數支援 SIP 的網路電話進行通訊連線建立。它可以用於傳送和接收語音、視訊和文字訊息等多媒體訊息。同時所支援的語音編解碼器包含了 G.711、GSM 以及 iLBC...等等，也支援視訊編解碼器 H.264、MPEG4 以及 theora...等等。在操作介面上使用了簡單的 GNOME/GTK+；此外，也提供命令提示模式讓使用者用文字下指令。

Linphone 主要是由幾個模組套件所組成，如圖 4.1 所示

- eXosip 是一個基於 GNU oSIP 協定所建立的應用程式介面(Application Program Interface，簡稱 API)，它主要是處理 SIP 通話連線的建立工作。
- oRTP 組件是利用即時傳輸協定(Real-time Transport Protocol，簡稱 RTP) [14] 傳送和接收語音或視訊 RTP 封包的一個應用程式介面。
- mediastreamer2 在 Linphone 中是最重要的部分之一，包含了處理 oRTP 的語音封包和視訊封包輸入和輸出功能。此外，它同時具備了多種語音和視訊的編解碼器。
- msx264 是基於 ITU-T H.264/AVC 協定的一個 mediastreamer2 的套件。H.264/AVC 是一個高度壓縮數字視頻編解碼器標準。H.264/AVC 編解碼器提供了良好的視訊品質。

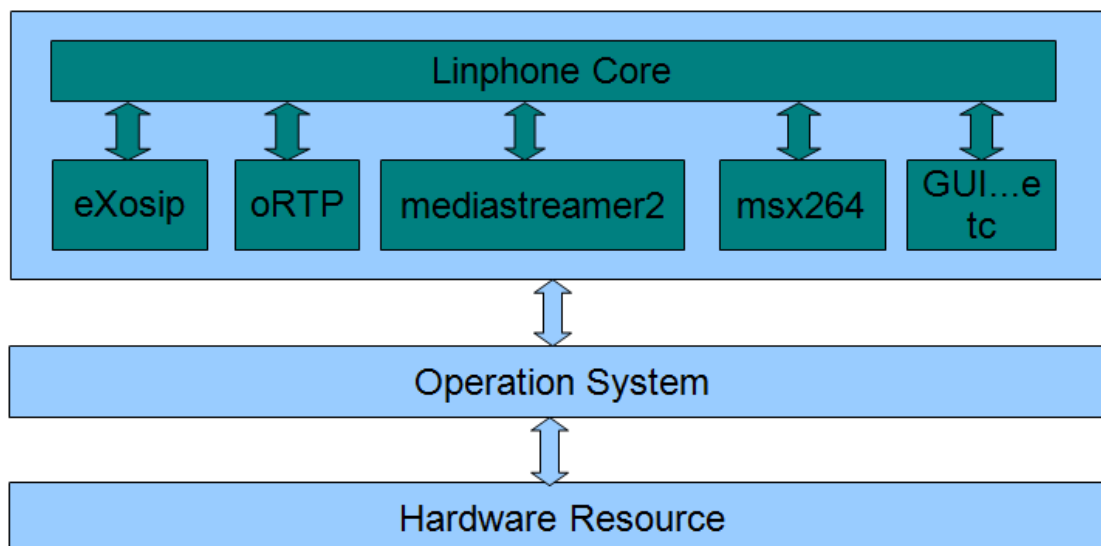


圖 4.1 Linphone 系統架構

為了驗證 RVH 方法的可行性，我們將 RVH 技術實作在 Linphone 上。若想知道 Linphone 是如何處理網路電話的流程，我們可透過圖 4.2 來了解。首先當雙方通話連線建立後，信令協定會指定雙方都共同支援的視訊編解碼器，這時 Linphone 將會創立相對應的結構去處理視訊的編碼和解碼。編解碼器的結構是定義在 mediastreamer2 的函式庫中，它支援多種的編解碼器，包含 H.264/AVC、MPEG4 以及其它可選的編解碼器。mediastreamer2 同時也創立了負責處理視訊輸入和輸出的結構，例如：從網路攝影機輸入影像到電腦或輸出影像到螢幕上。

此外，mediastreamer2 還創建處理傳送和接收 RTP 封包的結構，此結構負責處理將視訊資料封裝成 RTP 封包。換句話說，在 Linphone 軟體中，mediastreamer2 是一個處理所有關於視訊的程序的重要部分。

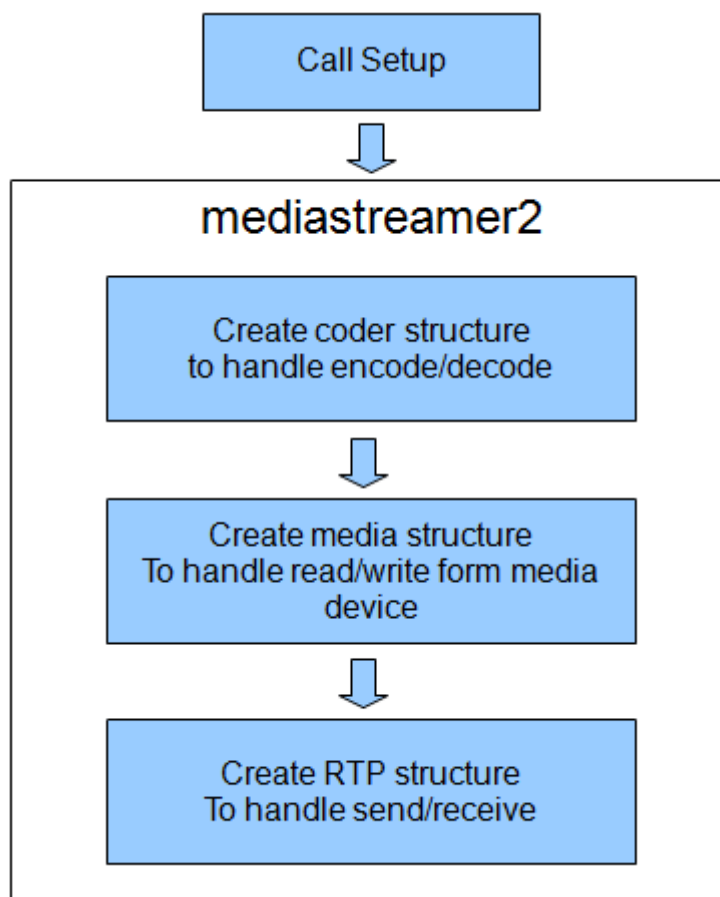


圖 4.2 mediastreamer2 呼叫程式流程

4.2 mediastreamer2 系統架構

如圖 4.3 和圖 4.4 所示，當一個通訊連線建立，mediastreamer2 將使用在 mediastreamer2/src/videostream.c 中的 video_stream_start 函數創建圖 4.2 所提到的六個結構，然後依照下面的流程進行。我們定義讀取視訊陣列資料的結構為 MCs，編碼視訊陣列資料的結構為 CEs，傳送 RTP 封包的結構為 RSs。

首先，在 MCs 從網路攝影機讀取視訊畫面資料後，將其資料傳遞給 CEs。接著 CEs 將其資料進行視訊編碼，在編碼後再將其資料給予 RSs。RSs 在接收到 CEs 的資料後，將其資料封裝成 RTP 封包傳送給接收端。

接著以相反的方向來看，我們定義寫入視訊陣列資料的結構為 MPs，解碼視訊陣列資料的結構為 CDs，接收 RTP 封包的結構為 RRs。首先，當 RRs 接收到 RSs 傳送的 RTP 封包後，將裡面的視訊資料給予 CDs 進行解碼成視訊畫面資料。緊接著，將解碼的視訊資料傳遞給 MPs，MPs 在接收資料後，將其視訊畫面撥放於螢幕上。

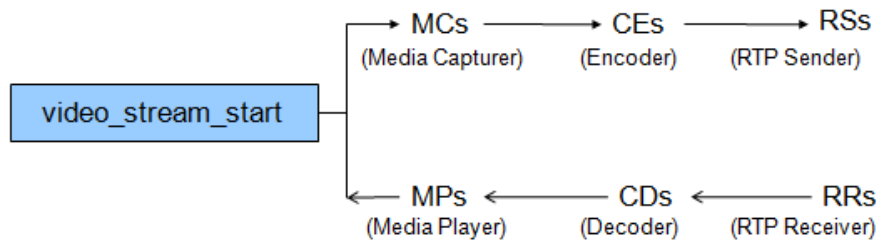


圖 4.3 video_stream_start 執行序列

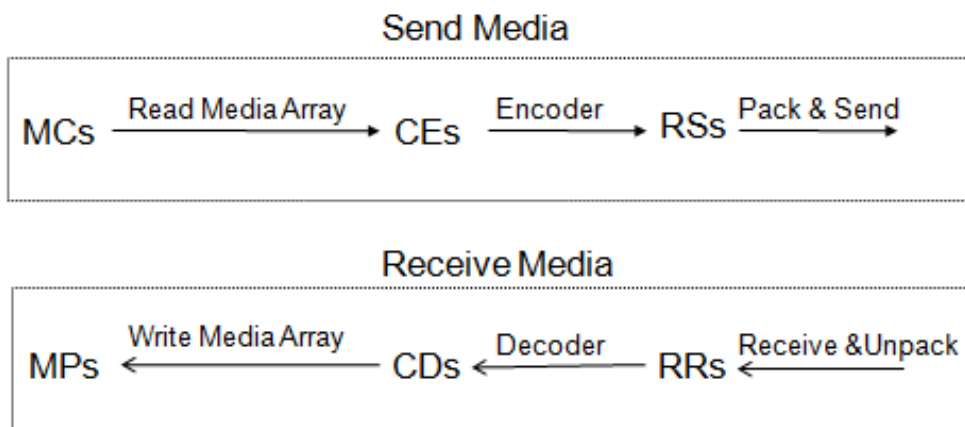


圖 4.4 mediastreamer2 資料流程

4.3 程式撰寫流程

因為 RVH 是架構在視訊編解碼器上，所以我們只需要修改在 Linphone 處理編解碼器的函式即可。在 Linphone 系統中，mediastreamer2 的套件 msx264 將會創建兩個結構去處理 H.264/AVC 編碼跟解碼，這部分的工作將會使用到 msx264/src/msx264.c 中的函式來完成。

在 msx264.c 的編解碼函式分別為 enc_process(...)和 dec_process(...)兩種。在 enc_process(...)中，此函式將會在 MCs 讀取視訊資料後將它進行編碼，然後寫入到 RSs 的 queue 進行資料傳送。而在 dec_process(...)中，其函式將會在 RRs 接收到視訊資料後將它進行解碼，然後寫入到 MPs 的 queue 中顯示到螢幕上。

有關於原始 msx264.c 其流程圖可由圖 4.5 說明，在 enc_process(...)部分，從網路攝影機讀取的視訊資料以符號 C 表示，而當 C 經由 H.264/AVC 編碼後的資料則表示為 D。在 dec_process(...)部分，透過 RTP 接收的視訊資料用 D 作為表示，而當經由 H.264/AVC 解碼後的資料則表示為 C'。另外要注意的是，因為 H.264/AVC 視訊編解碼器的壓縮是屬於有損性質的，所以將會使 C'和 C 的資料不盡相同。

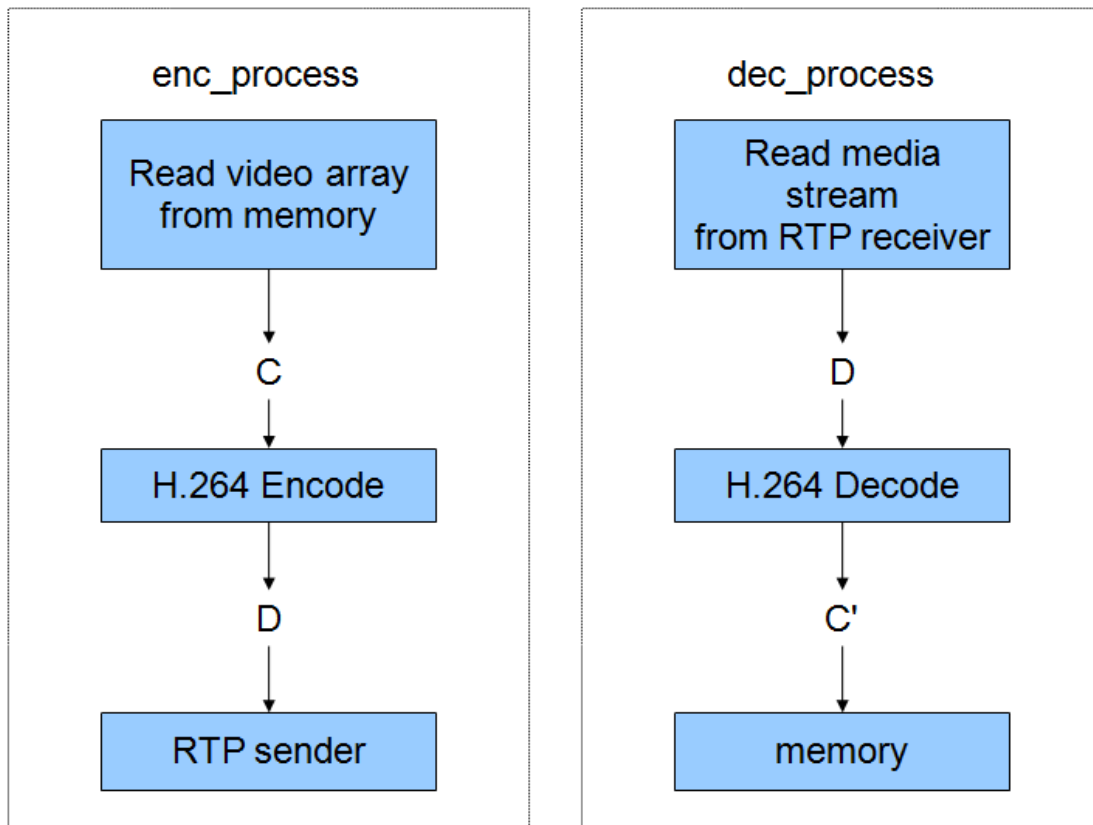


圖 4.5 msx264 執行程序

在瞭解了原始的 msx264 程式流程後，接下來將會介紹該如何修改程式來達成我們的目標。在 `enc_process(...)` 部份，我們由先前的圖 3.1 和圖 4.5 可得知整個傳送端的系統架構。以下將以圖 4.6 進行流程說明，首先我們選擇一個視訊資料作為掩護視訊 C，接著我們從網路攝影機讀取秘密視訊資料，然後我們將其秘密資料進行壓縮後可以得到 S'。接下來從掩護視訊 C 中選擇合適的隱藏空間 HS 來隱藏資料。其隱藏方式如同圖 3.9 將 S' 隱藏到 HS 中。最後我們將視訊資料 C 透過 H.264/AVC 進行編碼，在編碼後可得到欲傳送的隱密視訊 D。

在瞭解上面修改的流程後，我們可得知在 `dec_process(...)` 中，我們也只需要修改在解碼後的程式即可完成提取秘密視訊的動作。我們在 D 解碼後從 HS 中提取出資料，將其資料重新組合即可得到壓縮過的秘密視訊 S'。之後將 S' 解壓縮後就可以獲得傳送端的秘密視訊 S''，此處的秘密視訊 S'' 和原始的秘密視訊 S 必須視為不

同的視訊，這是因為 S'' 是經過編解碼程序後所產生的視訊，和 S 會有些微的不同。

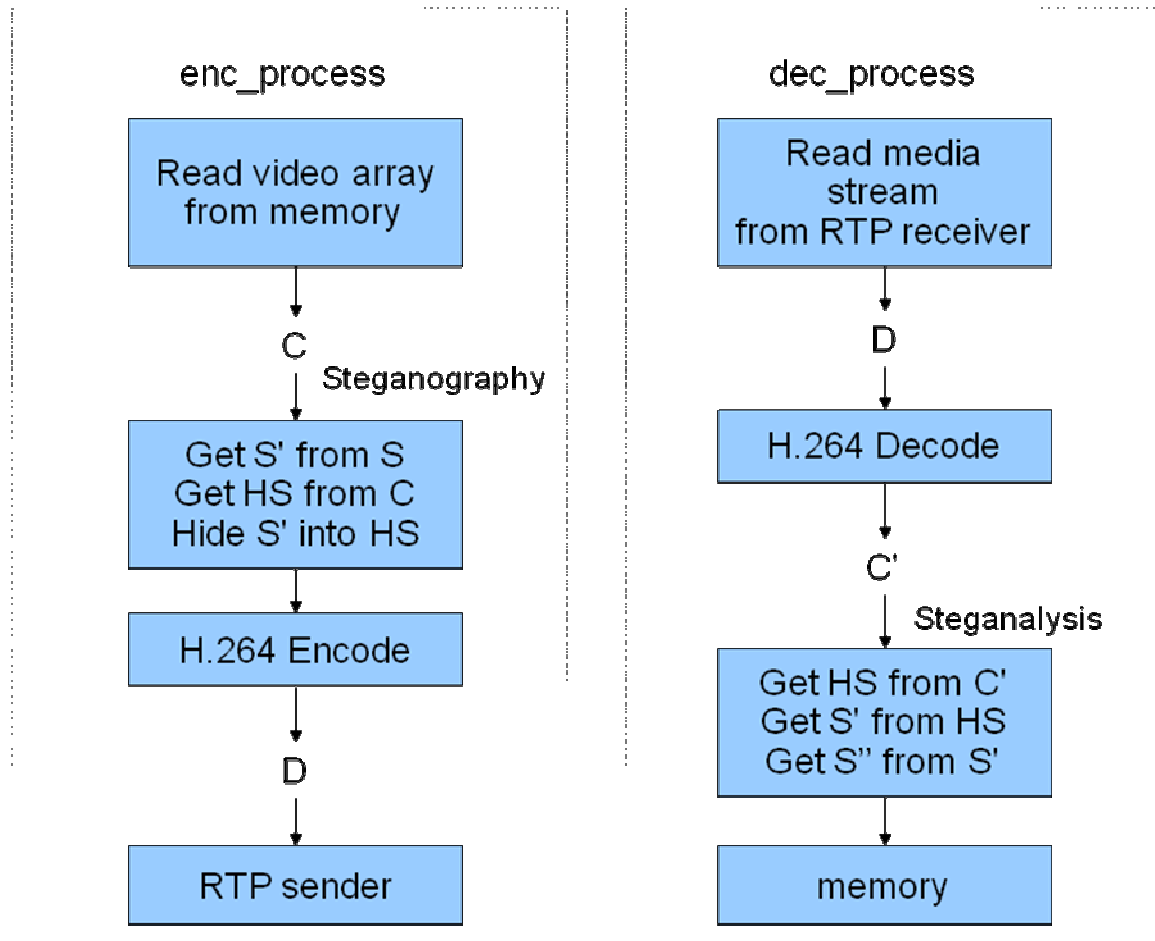


圖 4.6 msx264 修改後執行程序

第5章 實驗結果

視訊品質的量測是運行在兩台不同電腦上，它們各自代表著不同的用戶端，如傳送端和接收端。其配備分別為 Intel E2140 1.6GHz CPU、2.5GB 記憶體和 Intel Core2 Duo T7250 2.0GHz CPU、3.0GB 記憶體。使用的作業系統分別為 Linux Fedora 11 和 Linux ubuntu 9.10，兩部電腦位在同一區域網路(Local Area Network)中。我們使用網路攝影機錄製一段大小為 297KB 的 AVI 影像檔作為掩護視訊，其長度約為 30 秒。

表 5.1 實驗器材規格

| | | |
|-------|--------------------|--------------------------|
| 作業系統 | Fedora 11 (用戶端 A) | Ubuntu 9.10 (用戶端 B) |
| 中央處理器 | Intel E2140 1.6GHz | Intel Core2 T7250 2.0GHz |
| 記憶體 | 2.5GB | 3.0GB |

5.1 實驗成果

在這個章節中我們將主觀性的呈現我們的實驗結果。圖 5.1 顯示了我們所提供的掩護視訊的畫面截圖，圖 5.2 顯示了原始秘密視訊的畫面截圖。圖 5.3 顯示了使用 RVH_4bit 方法後的掩護視訊畫面截圖，圖 5.4 顯示了使用 RVH_4bit 方法還原後的秘密視訊畫面截圖。從下面的幾張截圖中我們可以看出經由 RVH_4bit 方法後，雖然掩護視訊和秘密視訊品質都降低了，但是我們還是能清楚看出視訊顯示的畫面。



圖 5.1 原始掩護視訊



圖 5.2 原始秘密視訊



圖 5.3 RVH_4bit 掩護視訊



圖 5.4 RVH_4bit 還原後的秘密視訊

5.2 Peak Signal-to-Noise Ratio (PSNR)比較分析

PSNR 是一個表示信號最大可能功率和影響它表示精度的破壞性雜訊功率比值的工程術語。由於信號有非常大的動態範圍，PSNR 常用對數分貝(dB)單位表示。通常在經過影像壓縮之後，輸出的影像都會有某種程度與原始影像不一樣。為了客觀的衡量經過處理後的影像品質，我們通常會參考 PSNR 值來認定某個處理程序是否令人滿意。圖像壓縮中典型的 PSNR 值約在 30 至 50dB 之間，其值越高代表品質越好。

在計算 PSNR 值之前，通常會先定義均方差(MSE，簡稱 Mean Squared Error)來計算 PSNR 所需要的參數值。

均方差定義為:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2 \quad (2)$$

其中 m 和 n 分別為視訊畫面的解析度大小，如'CIF (352 × 288)'即表示 m = 352、n = 288；I 和 K 分別為原始視訊和經處理後的視訊；i 和 j 分別為畫面中像素的位置，如 i = 0、j = 0 時即表示為畫面中最左上方的像素位置(0,0)。

PSNR 定義為:

$$PSNR = 10 \times \log \left(\frac{MAX^2}{MSE} \right) = 10 \times \log \left(\frac{255^2}{MSE} \right) \quad (3)$$

其中 MAX 是表示圖像點顏色的最大數值，在視訊中每個取樣點都是使用 8 個位元來表示，所以數值就是 $2^8 = 255$ 。

在了解 PSNR 的定義後，就可以透過此定義來計算章節 5.1 實驗的 PSNR 值。在 PSNR 值的計算中，我們分別定義了以下幾個數值。首先是 m 和 n ，由於我們的視訊解析度固定使用 'CIF (352 × 288)' 的大小，所以 $m = 352$ 、 $n = 288$ 。接著 I 定義為原始的秘密視訊或掩護視訊，如圖 5.1 或圖 5.2 的視訊； K 定義為經處理後的視訊，像是只有使用 H.264 邊解碼器處理後的視訊，或者是使用 H.264+RVH 方法後的視訊，如圖 5.3 或圖 5.4。在我們的實驗中，考慮了以下三種情況來衡量 PSNR 值。其中秘密視訊的 PSNR 計算，將使用原始的秘密視訊和經處理後的秘密視訊作為計算的參考數值；而在掩護視訊的計算中，則是使用原始的掩護視訊和經處理後的掩護視訊作為參考數值。

1. 首先我們只考慮使用標準 H.264 編解碼器，而不套用 RVH 視訊隱藏方法，也就是在一般情況下進行視訊電話。經測試後秘密視訊和掩護視訊的 PSNR 值約為 48dB，即表示在 H.264 編解碼前後的視訊畫面幾乎沒有差異；此數值將作為與其它情況比較的基準。
2. 我們考慮使用 RVH($b = 4, p = 1$)時的情況，掩護視訊和秘密視訊解析度大小相同時，其掩護視訊的解析度為 'CIF (352 × 288)'，秘密視訊解析度為 'CIF (352 × 288)'，掩護視訊每一個像素最低 4 個位元將用於隱藏秘密視訊每一個像素的最高 4 個位元。經測試後 PSNR 值分別為秘密視訊 23.4171dB、掩護視訊 27.2792dB，和上一個情況比較，其雜訊值都明顯的提高了。所以這個情況很容易被有心人士發覺此視訊檔有異。
3. 我們考慮使用 RVH($b = 2, p = 1$)時的情況，其掩護視訊的解析度為 'CIF (352 × 288)'，秘密視訊解析度為 'QCIF (176 × 144)'，掩護視訊每一個像素最低 2 個位元將用於隱藏秘密視訊。經測試後 PSNR 值分別為秘密視訊 17.2196dB、掩護視訊 30.5587dB，和情況二作比較，因為使用了較少的隱藏空間，所以掩護視訊 PSNR 明顯的提高了，因此更加不容易被有心人士發覺此視訊檔有異，但

是其秘密視訊 PSNR 值卻下降了，這表示秘密視訊的品質將會比情況二還要差。

表 5.2 視訊 PSNR 比較

| 隱藏方法 | 秘密視訊的 PSNR (dB) | 掩護視訊的 PSNR (dB) |
|------------------|-----------------|-----------------|
| H.264(未作隱藏) | 48 | 48 |
| H.264 + RVH_4bit | 23.4171 | 27.2792 |
| H.264 + RVH_2bit | 17.2196 | 30.5587 |

第6章 結論和未來工作

在本篇論文中，我們提出了在即時通訊系統（視訊電話）中進行視訊隱藏的一個方法，我們簡稱此方法為 RVH (Real-time Video Hiding)。RVH 方式是藉由隱藏秘密視訊到掩護視訊之中，來達到保護秘密視訊的一個方法。在此方法下，即使有心人士竊取了我們的資料封包，但也只會看到掩人耳目的掩護視訊內容，而不會注意到隱匿於其中的秘密視訊，如此一來將可有效增加網路電話的安全。

在 RVH 方法中，資料壓縮扮演了重要的角色。我們透過壓縮秘密視訊，可以得到比原先更小的數據資料，這將有效的幫助我們隱藏資料到掩護視訊中。而資料壓縮的方法將會影響我們隱藏的方式，進而影響視訊的品質。在我們的實驗中，實驗結果表明，如果秘密視訊解析度較大，則掩護視訊需要提供較多的掩藏空間用以隱藏資料，這將會使掩護視訊的品質降低較多。然而，如果解析度低的秘密視訊，其品質將會降低，但相對地掩護視訊就只需要提供較少的隱藏空間用以隱藏資料，如此一來掩護視訊的品質將會比較好。

當前即時通訊系統（網路電話）的應用程序通常包含了文字訊息、語音和視訊。在現有的研究文獻中，在網路電話系統上，使用資訊隱藏提升安全性的研究可謂數量不多，在視訊方面則更加稀少，這是因為在視訊電話中隱藏視訊有一定的困難度存在。我們成功在網路電話上實現了即時隱藏視訊程序，RVH 方法比起[6]所提出的方法，在選擇隱藏空間上有更好的作法。在[6]的方法中是採用固定的隱藏空間，每一個像素提供了 8 位元的隱藏空間，而在 RVH 方法中則是依照視訊解析度來變換隱藏空間的大小，每一個像素可提供 12 位元或 6 位元等不同的隱藏空間。因此我們可以選擇對掩護視訊失真程度較小的隱藏方法，這樣將有助於讓有心人士更不容易察覺到我們隱藏的秘密資料。

目前我們提出的方法並不能隨著網路品質自動調整資訊隱藏方式，如果在撥打

視訊電話時，系統能自動偵測 PSNR 值來判斷所需要的 b 值，這將有助於增加視訊的品質。此外，在 RVH 方法中，因為使用了編解碼器和壓縮等有損性質的方法，秘密視訊的品質都會因此下降。而在網路電話中，無論是語音或視訊電話，其通話品質都是非常重要的。因此在未來的工作中，如何增加視訊品質將是一項重要的研究目標。

參考文獻

- [1] Christian Grotho, Krista Grotho, Ludmila Alkhutova, Ryan Stutsman, and Mikhail Atallah, “Translation-Based Steganography”, Proceedings of Information Hiding Workshop (IH 2005), pp.213-233. Springer-Verlag, 2005.
- [2] P. Bao and Xiaohu Ma, “MP3-resistant music steganography based on dynamic range transform”, IEEE International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS 2004), pp.266-271, 18-19 Nov. 2004.
- [3] Chungyi Wang and Quincy Wu, “Information Hiding in Real-time VoIP Streams” Multimedia. IEEE International Symposium on Ninth (ISM 2007), pp.255-262, 10-12 Dec.2007.
- [4] EasyBMP, [<http://easybmp.sourceforge.net/steganogeaphy.html>].
- [5] Dorian A. Flowers, “Investigating Steganography”, Proceedings of the 42nd annual ACM Southeast regional conference (ACM-SE42),2-3 April 2004.
- [6] Mohamed Elsadig, Miss Laiha Mat Kiah, Bilal Bahaa Zaidan and Aos Alaa Zaidan, “High Rate Video Streaming Steganography”, IEEE Information Management and Engineering (ICIME 2009), pp.550-553, 3-5 April 2009.
- [7] Wikipedia, [<http://en.wikipedia.org/wiki/Steganography>].
- [8] Spyridon K. Kapotas, Eleni E. Varsaki and Athanassios N. Skodras, “Data Hiding in H.264 Encoded Video Sequences”, IEEE Multimedia Signal Processing (MMSP 2007), pp.373-376, 1-3 Oct. 2007.
- [9] Yang Hu, Chuntian Zhang and Yuting Su, “Information Hiding Based On Intra Prediction Modes For H.264/AVC”, IEEE Multimedia and Expo (ICME 2007), pp.1231-1234. 2-5 July 2007.
- [10] ITU-T Recommendation H.264. Advanced video coding for generic audiovisual

services, Nov.2003.

- [11] ITU-T Recommendation. H.323. Packet-based multimedia communications systems, Nov 1996
- [12] Linphone, [<http://www.linphone.org/>].
- [13] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley and E. Schooler, “SIP: Session Initiation Protocol”, IETF, RFC3261, June 2002.
- [14] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, “RTP: A Transport for Real-Time Applications”, IETF, RFC3550, July 2003.

附錄 A Linphone 安裝

A.1 目的

Linphone 安裝流程說明。以下安裝方式皆適用於 Fedora 和 Ubuntu 作業系統。

A.2 檔案下載

FFmpeg (相關套件):

http://ms11.voip.edu.tw/~weili/weili_thesis/FFmpeg_install.zip

Linphone (相關套件):

http://ms11.voip.edu.tw/~weili/weili_thesis/Linphone_install.tar.gz

A.3 安裝說明

FFmpeg 安裝說明

FFmpeg 是一套錄製、轉換各種語音視頻各式的工具集，同時也提供 SDK 的編解碼器介面，所以在安裝 Linphone 之前必須先安裝 FFmpeg。主要包含 ffmpeg、ffplay、ffserver、libavcodec 和 libavformat 等等，其中很多編解碼庫依賴於協力廠商，比如 xVid、lampMP3 等等。

1. 安裝 yasm：安裝 x264 前必須安裝，一種編譯器，可使 x264 運行效率較好。

➤ `#wget http://www.tortall.net/projects/yasm/releases/yasm-0.8.0.tar.gz`

➤ `#tar zxvf yasm-0.8.0.tar.gz`

➤ `#./configure --prefix=/usr/local/yasm`

- #make
- #make install
- #export PATH="\$PATH:/usr/local/yasm/bin"
- #vi /etc/profile
 - ◇ 最後一段寫入 export PATH="\$PATH:/usr/local/yasm/bin"

2. 安裝 x264：H.264 應用程式介面，要安裝 x264 必須要使用 yasm 來編譯，所以要先安裝 yasm，再安裝 x264。

- #wget http://ms11.voip.edu.tw/~weili/weili_thesis/FFmpeg_install.zip
- #unzip FFmpeg_install.zip
- #cd FFmpeg_install/x264
- #./configure --prefix=/usr/ --enable-shared
- #make
- #make install

3. 安裝 FFmpeg：

- #wget http://ms11.voip.edu.tw/~weili/weili_thesis/FFmpeg_install.zip
- #unzip FFmpeg_install.zip
- #cd FFmpeg_install/ffmpeg
- #./configure --prefix=/usr --enable-gpl --enable-shared --enable-libx264
- #make
- #make install

Linphone 安裝說明

以下安裝程序在 Fedora11 和 Ubuntu9.10 作業系統下皆能順利安裝完成，若使用其他版本則可能需再調整所需連結函式庫等參數。各套件最好放置 /usr/local/src 下再進行安裝。安裝的時候，最好把 ./configure-help 中的參數看一看，指定安裝在 /usr 下，可以省去很多麻煩。./configure 腳本如果沒有顯式指定安裝路徑(--prefix=)，則預設安裝到 /usr/local 目錄下。

注意：安裝在 /usr 下，會把系統中原有的舊標頭檔替換掉，如果你還想使用它們，還是安裝在 /usr/local 目錄下。由於各 Linphone 版本的不同，可能會有不同的需求，安裝過程不一定正確，僅此做為參考。以下安裝過程盡可能使用提供的檔案進行安裝。

1. 安裝 libosip2：支援 osip

- #wget http://ms11.voip.edu.tw/~weili/weili_thesis/Linphone_install.tar.gz
- #tar zxvf Linphone_install.tar.gz
- #cd Linphone_install
- #tar zxvf libosip2-3.3.0.tar.gz
- #cd libosip2-3.3.0
- #./configure
- #make
- #make install

2. 安裝 libeXosip2：支援 eXosip

- #wget http://ms11.voip.edu.tw/~weili/weili_thesis/Linphone_install.tar.gz
- #tar zxvf Linphone_install.tar.gz
- #cd Linphone_install

- #tar zxvf libeXosip2-3.3.0.tar.gz
- #cd libeXosip2-3.3.0
- #./configure
- #make
- #make install

3. 安裝 Linphone：網路電話軟體

- #wget http://ms11.voip.edu.tw/~weili/weili_thesis/Linphone_install.tar.gz
- #tar zxvf Linphone_install.tar.gz
- #cd Linhpone_install
- #unzip linphone-3.1.1.zip
- #cd linphone-3.1.1
- #./configure --prefix=/usr/
- #make
- #make install

4. 安裝 msx264：Linphone 擴充套件（支援 H.264）

- #wget http://ms11.voip.edu.tw/~weili/weili_thesis/msx264.tar
- #tar xvf msx264.tar
- #cd msx264
- #./configure --prefix=/usr/
- #make
- #make install

◇ 如有出現”[avcodec_decode_video] is deprecated”錯誤，請修改
configure 檔，將 Werror 取消。

附錄 B 程式碼

B.1 目的

說明 RVH 方法的程式碼作用，有利於了解程式的實作。

B.2 程式碼下載

msx264 : http://ms11.voip.edu.tw/~weili/weili_thesis/msx264.zip

B.3 程式碼說明

- `ffmpeg.h` : 處理載入掩護視訊。
- `rvh.h` : 控制 RVH 方式和計算 PSRN 值之程式碼。
- `msx264.c` : 主要程式，包含轉換解析度大小、RVH 等程式碼。以下只顯示新增部份，完整程式碼請參考 `msx264.c`。

程式碼中斜粗體的字皆為 RVH 新增部份，是原始 `msx264` 程式碼所沒有的。

`msx264/src/ffmpeg.h`

```
/** add */
AVFormatContext *ifc;
AVCodecContext *icc;
AVPacket packet;
AVFrame *iframe;

//功能：開啟一段欲使用的視訊，同時載入編解碼相關參數。
static void ffmpeg_input_video(const char *input_file_name)
{
    av_register_all();
    AVCodec *ic;
    int video_index = -1;
    int i;
```

```

ifc = avformat_alloc_context();
if(av_open_input_file(&ifc, input_file_name, NULL, 0, NULL) != 0)
{
    printf("can't open the file %s\n", input_file_name);
    exit(1);
}
if(av_find_stream_info(ifc) < 0)
{
    printf("can't find suitable codec parameters\n");
    exit(1);
}
for(i = 0; i < ifc->nb_streams; i++)
{
    if(ifc->streams[i]->codec->codec_type == CODEC_TYPE_VIDEO)
    {
        video_index = i;
    }
}
if(video_index == -1)
{
    printf("can't find video stream\n");
    exit(1);
}
icc = ifc->streams[video_index]->codec;
ic = avcodec_find_decoder(icc->codec_id);
if(ic == NULL)
{
    printf("can't find suitable video decoder\n");
    exit(1);
}
if(avcodec_open(icc, ic) < 0)
{
    printf("can't open the video decoer\n");
    exit(1);
}
iframe = avcodec_alloc_frame();
}

```

msx264/src/rvh.h

```

/** add */
static int rvh_enc_mode = 0;
static int rvh_dec_mode = 0;
char *returnMessage = "setEncode_loop fnuction is over\n";

```

//功能：切換即時視訊隱藏的方式，如一般模式、解碼模式、編碼模式。

```
static void *select_rvh_mode(void *ptr)
{
    while(1)
    {
        int tmp = 0;
        system("clear");

        printf("RVH encode mode\t=\t%i\n", rvh_enc_mode);
        printf("RVH decode mode\t=\t%i\n", rvh_dec_mode);
        printf("(1)Encoder set (2)Decoder set (3)Exit\n");
        scanf("%i", &tmp);

        if(!(tmp>=1 && tmp <=3))
            continue;

        system("clear");
        switch(tmp)
        {
            case 1: //encode
                printf("RVH encode mode\t=\t%i\n", rvh_enc_mode);
                printf("Normal mode(0)\t\tRVH encode mode (1)\n");
                scanf("%i", &tmp);

                if(tmp>=0 && tmp <=1)
                    rvh_enc_mode = tmp;
                break;
            case 2: //decode
                printf("RVH decode mode\t=\t%i\n", rvh_dec_mode);
                printf("Normal mode(0)\t\tRVH decode mode (1)\n");
                scanf("%i", &tmp);

                if(tmp>=0 && tmp <=1)
                    rvh_dec_mode = tmp;
                break;
            case 3:
                exit(0);
        }
    }

    pthread_exit((void *)returnMessage);
}
```

//功能：計算視訊經由 RVH 方法後的 PSNR 數值。

```
static double rvh_psnr(uint8_t *In_data[], uint8_t *Pn_data[], int FrameSize)
{
```



```

int x;
int In, Pn;
int MSE_tmp = 0;
double MSE = 0;
double PSNR = 0;

for(x=0; x<FrameSize; x++)
{
    In = *(In_data[0]+x);
    Pn = *(Pn_data[0]+x);
    MSE_tmp = MSE_tmp + (pow((In-Pn), 2));
}
MSE = MSE_tmp / FrameSize;

if(MSE == 0)
{
    PSNR = 48;
}
else
{
    PSNR = 10 *log10(pow(255, 2) / MSE);
}

return PSNR;
}
/** end ***/

```

```

msx264/src/msx264.c

/** add ***/

#include <libavformat/avformat.h>
#include <libavutil/avutil.h>
#include "ffmpeg.h"
#include "rvh.h"

/** end ***/

static void enc_init(MSFilter *f){
    EncData *d=ms_new(EncData,1);
    d->enc=NULL;
    d->bitrate=384000;
    d->vsize=MS_VIDEO_SIZE_CIF;
    d->fps=30;
    d->keyframe_int=10; /*10 seconds */
    d->mode=0;

```

```

d->framenum=0;
d->generate_keyframe=FALSE;
f->data=d;

//功能：載入掩護和秘密視訊和線程的參數設定。
/** add */

    const char *input_cover_video = "/home/weili/cover.avi";
    ffmpeg_input_video(input_cover_video);

    const char *input_secret_video = "/home/weili/secret.avi";
    ffmpeg_input_video_2(input_secret_video);

    pthread_t thread;
    int iret;
    iret = pthread_create(&thread, NULL, select_rvh_mode, (void
*)returnMessage);

/** end */
}
...

static void enc_process(MSFilter *f){
    EncData *d=(EncData*)f->data;
    uint32_t ts=f->ticker->time*90LL;
    mblk_t *im;
    MSPicture pic;
    MSQueue nalus;
    ms_queue_init(&nalus);

//功能：轉換解析度大小的預先設定。
/** add */

    const int in_width = 352;
    const int in_height = 288;
    const int out_width = 384;
    const int out_height = 288;
    const int ysize = 384*288;
    const int uysize = ysize/4;

    int x,y;
    int finished;
    static struct SwsContext *img;

    uint8_t *out_buf[4];
    int out_linesize[4] = {out_width, out_width/2, out_width/2, 0};
    out_buf[0] = malloc(out_width*out_height);

```

```

out_buf[1] = malloc(out_width*out_height >> 2);
out_buf[2] = malloc(out_width*out_height >> 2);
out_buf[3] = NULL;

img = sws_getContext(in_width, in_height, PIX_FMT_YUV420P,
out_width,out_height, PIX_FMT_YUV420P, SWS_FAST_BILINEAR, NULL,
NULL, NULL);

/** end **/

while((im=ms_queue_get(f->inputs[0]))!=NULL){
    if (yuv_buf_init_from_mblk(&pic,im)==0){
        x264_picture_t xpic;
        x264_picture_t oxpic;
        x264_nal_t *xnals=NULL;
        int num_nals=0;

        //send I frame 2 seconds and 4 seconds after the beginning
        //if (d->framenum==(int)d->fps*2 || d->framenum==(int)d->fps*4)
            d->generate_keyframe=TRUE;

        if (d->generate_keyframe){
            xpic.i_type=X264_TYPE_IDR;
            d->generate_keyframe=FALSE;
        }else xpic.i_type=X264_TYPE_AUTO;
        xpic.i_qpplus1=0;
        xpic.i_pts=d->framenum;
        xpic.img.i_csp=X264_CSP_I420;
        xpic.img.i_plane=3;

//功能：讀取視訊內容並進行即時視訊隱藏。
/** add **/

av_read_frame(iff, &packet) ;
avcodec_decode_video2(icc, iframe, &finished, &packet);
av_read_frame(iff_2, &packet_2) ;
avcodec_decode_video2(icc_2, iframe_2, &finished, &packet_2);

sws_scale(img, pic.planes, pic.strides, 0, in_width, out_buf, out_linesize);

/* secret_384*288_most_4bit hiding into cover_384*288_least_4bit*/
if(rvh_enc_mode == 1)
{
    int bit_tmp = 0;
    for(x=0;x<yssize;x++)
    {

```

```

        bit_tmp = *(out_buf[0]+x) &0xf0;
        bit_tmp = bit_tmp >> 4;
        *(iframe->data[0]+x) = *(iframe->data[0]+x) &0xf0 | bit_tmp;
    }
    for(x=0;x<uvsize;x++)
    for(y=1;y<3;y++)
    {
        bit_tmp = *(out_buf[y]+x) & 0xf0;
        bit_tmp = bit_tmp >> 4;
        *(iframe->data[y]+x) = *(iframe->data[y]+x) &0xf0 | bit_tmp;
    }
}

if(rvh_enc_mode == 1)
{
    xpic.img.i_stride[0] = iframe->linesize[0];
    xpic.img.i_stride[1] = iframe->linesize[1];
    xpic.img.i_stride[2] = iframe->linesize[2];
    xpic.img.i_stride[3] = 0;
    xpic.img.plane[0] = iframe->data[0];
    xpic.img.plane[1] = iframe->data[1];
    xpic.img.plane[2] = iframe->data[2];
    xpic.img.plane[3] = 0;
}
else
{
    xpic.img.i_stride[0] = pic.strides[0];
    xpic.img.i_stride[1] = pic.strides[1];
    xpic.img.i_stride[2] = pic.strides[2];
    xpic.img.i_stride[3] = 0;
    xpic.img.plane[0] = pic.planes[0];
    xpic.img.plane[1] = pic.planes[1];
    xpic.img.plane[2] = pic.planes[2];
    xpic.img.plane[3] = 0;
}
}
/** end **/

    if (x264_encoder_encode(d-
>enc,&xnals,&num_nals,&xplic,&oxpic)==0){
        x264_nals_to_msgb(xnals,num_nals,&nalus);
        rfc3984_pack(&d->packer,&nalus,f->outputs[0],ts);
        d->framenum++;
    }else{
        ms_error("x264_encoder_encode() error.");
    }
}
freemsg(im);
}

```

```

}

static void dec_init(MSFilter *f){
    DecData *d=(DecData*)ms_new(DecData,1);
    ffmpeg_init();
    d->yuv_msg=NULL;
    d->sps=NULL;
    d->pps=NULL;
    d->sws_ctx=NULL;
    rfc3984_init(&d->unpacker);
    d->packet_num=0;
    dec_open(d);
    d->outbuf.w=0;
    d->outbuf.h=0;
    d->bitstream_size=65536;
    d->bitstream=ms_malloc0(d->bitstream_size);
    f->data=d;

//功能：讀取視訊內容和線程的參數設定。
/** add **/

    const char *input_cover_video = "/home/weili/cover.avi";
    ffmpeg_input_video(input_cover_video);

    const char *input_secret_video = "/home/weili/secret.avi";
    ffmpeg_input_video_2(input_secret_video);

    pthread_t thread;
    int iret;
    iret = pthread_create(&thread, NULL, select_rvh_mode, (void
*)returnMessage);

/** end **/
}

static void dec_process(MSFilter *f){
    DecData *d=(DecData*)f->data;
    mblk_t *im;
    MSQueue nalus;
    AVFrame orig;
    ms_queue_init(&nalus);

//功能：轉換解析度大小的預先設定。
/** add **/

    const int in_width = 192;
    const int in_height = 144;

```

```

const int out_width = 384;
const int out_height = 288;
const int ysize = 384*288;
const int uvsiz = ysize/4;
int x,y;
int finished;

static struct SwsContext *img;
uint8_t *in_buf[4];
//int in_linesize[4] = {in_width, in_width/2, in_width/2, 0};

in_buf[0] = malloc(in_width*in_height);
in_buf[1] = malloc(in_width*in_height >> 2);
in_buf[2] = malloc(in_width*in_height >> 2);
in_buf[3] = NULL;

img = sws_getContext(in_width, in_height, PIX_FMT_YUV420P, out_width,
out_height, PIX_FMT_YUV420P, SWS_FAST_BILINEAR, NULL, NULL, NULL);

/** end **/

while((im=ms_queue_get(f->inputs[0]))!=NULL){
    /*push the sps/pps given in sprop-parameter-sets if any*/
    if (d->packet_num==0 && d->sps && d->pps){
        mblk_set_timestamp_info(d->sps,mblk_get_timestamp_info(im));
        mblk_set_timestamp_info(d->pps,mblk_get_timestamp_info(im));
        rfc3984_unpack(&d->unpacker,d->sps,&nalus);
        rfc3984_unpack(&d->unpacker,d->pps,&nalus);
        d->sps=NULL;
        d->pps=NULL;
    }
    rfc3984_unpack(&d->unpacker,im,&nalus);
    if (!ms_queue_empty(&nalus)){
        int size;
        uint8_t *p,*end;
        bool_t need_reinit=FALSE;

        size=nalusToFrame(d,&nalus,&need_reinit);
        if (need_reinit)
            dec_reinit(d);
        p=d->bitstream;
        end=d->bitstream+size;
        while (end-p>0) {
            int len;
            int got_picture=0;

```

//功能：進行視訊解碼

```

/** add **/

    av_read_frame(iff, &packet);
    avcodec_decode_video2(icc, iframe, &finished, &packet);

    av_read_frame(iff_2, &packet_2);
    avcodec_decode_video(icc_2, iframe_2, &finished, packet_2.data,
packet_2.size);

/** end **/
        avcodec_get_frame_defaults(&orig);
        len=avcodec_decode_video(&d-
>av_context,&orig,&got_picture,p,end-p);
        if (len<=0) {
            ms_warning("ms_AVdecoder_process: error %i.",len);
            break;
        }

//功能：進行 RVH 方法的反解
/** add **/

if(rvh_dec_mode == 1)
{
    int bit_tmp;
    for(x=0;x<ysize;x++)
    {
        bit_tmp = *(orig.data[0]+x) & 0x0f;
        bit_tmp = bit_tmp << 4;
        *(iframe->data[0]+x) = bit_tmp;
    }
    for(x=0;x<uvsize;x++)
    for(y=1;y<3;y++)
    {
        bit_tmp = *(orig.data[y]+x) & 0x0f;
        bit_tmp = bit_tmp << 4;
        *(iframe->data[y]+x) = bit_tmp;
    }
}

/** end **/

        if (got_picture) {

//功能：將顯示的內容資料傳送到輸出端。
/** add **/
                if(rvh_dec_mode == 1){
                    ms_queue_put(f->outputs[0],get_as_yuvmsg(f,d,iframe));}
                else{

```

```
ms_queue_put(f->outputs[0],get_as_yuvmsg(f,d,&orig));}
/** end **/
    }
    p+=len;
  }
}
d->packet_num++;
}
}
```