逢 甲 大 學

資 訊 工 程 學 系 專 題 報 告

無線網路簡報管理系統

學　生　：吳豐年(四甲)
　　盧奕吉(四甲)
指導教授 ：李維斌

中華民國九十二年十二月

目錄

專題簡介

專題研究動機與目的

動機

數位時代的來臨與網際網路的興起，帶來了許多新興的商機。其中有一種常見的「線上電影院」的電子商店，是本次的專題研究的方向。

這類型的線上電影院，有時是收費的，有時是免費的。不過不論是收費還是免費，因爲智慧財產權的關係，不希望使用者能在未經授權的情形下自行散佈影片資料。現在網路上大部分的電子商店，對於防止未經授權的重製行爲的對策並不多。許多的電子商店是利用 RTP(Real-Time Transport Protocol)，來達到部分的功效。RTP 其實主要是用在即時的線上影音傳輸，例如視訊會議、實況轉播等等，使用這個傳輸模式的時候，影音的初始化和控制是由伺服端控制，而傳輸的資料是以串流的格式(Streaming media)來進行。因此並不能像網路上一般的檔案一樣方便地下載。但現在透過特殊的下載軟體支援，還是可以很方便地下載下來，因此我們就針對把影片加密做研究，希望能實作出較爲安全的影片發佈方式。

目的

透過對影片的加密，我們希望能達成一種新的影片購買機制。對於影片提供者來說，他們把加密處理過的影片放在網路上供使用者下載。當使用者希望能看影片的時候，他需要付錢給影片提供者，以買到一把解密用的金鑰。利用這把金鑰才能夠正確地播放加密後的電影。所以現在影片提供者不再需要費心控制使用者的下載，因爲透過加密的影片，如果沒有正確的金鑰，在別的播放器裡面是沒辦法正確地呈現的。

何謂影像播放程式

一個動作影像其實是一系列的圖片，以一定的速度換圖片的時候利用人眼的視覺暫留而產生連續動作的錯覺。
一個影片的產生有以下幾個流程：

1. 一系列構成影片的圖片。
2. 把這些影片依照影片格式的要求，轉成特定的影片格式。
3. 依照影片格式的壓縮演算法，把這一系列的影片製成 Video Track。
4. 依照同樣的順序，產生聲音的 Audio Track。
5. 根據影片格式的多工(Multiplex)演算法，把 Video tracks 和 Audio tracks 以同步的方式存放在一個檔案裡。

而這個產生出來的檔案，也就是我們常見的數位影片。常見的例子是 mpg 檔(Mpeg-1 格式)、mov 檔(QuickTime 格式)。

以 Mpeg-1 格式的影片來舉例：

1. 一系列的圖片。
2. 把這些圖片轉成 Jpeg 格式。
3. 把這些 Jpeg 圖片依照 Mpeg-1 壓縮演算法，以 I,P,B 三種 frame 來構成 Video Track。
4. 產生 Audio Track。
5. 把 Video Track 和 Audio Track 依照 Mpeg-1 格式做 Multiplex 到一個檔案。

所謂的影片播放程式，指的是一個程式，它依照一系列的步驟，把這個影片的檔案的內容，也就是一系列的圖片，以合理的速度在輸出裝置上呈現出來。而電影播放合理的速度通常是每秒 20 張以上的圖片(20fps)，才能讓因視覺暫留產生出的連續錯覺順暢。把影片播放出來的一系列的步驟可以定義如下：

1. 把檔案依照影片的格式，使用特定的解多工(Demultiplex)方法把影片還原

成 Video tracks 和 Audio tracks。
2. 根據影片的格式，使用正確的解壓縮演算法，把 Video tracks 和 Audio tracks 解開。
3. 根據影片的格式，使用解出來的資料來呈現聲音和影像。

因爲我們專題的程式負責了對於影片的播放，所以也是影片播放程式的一種。而影片播放並不一定是播放到螢幕、喇叭。廣義而言，輸出到另一個檔案，甚至另一種格式，只要經過以上的步驟，把特定格式的影片檔案分解成原始的資料，就是一種影片的播放。

何謂影像播放系統

所有與影片播放有關程式的集合的程式、工具建構成的影音播放、處理平台，就是影像播放系統。在我們的專題中，除了影像的播放，我們還依靠了加密及影像的處理，來達成安全的影像播放系統。

何謂安全的影像播放系統

我們對於安全的影像播放系統的定義是：影片的內容在此影像播放程式的支持下是安全的。
影片在沒有金鑰的情形下，雖然可以被其它的影像播放系統播放，但得到的卻是沒有意義的呈現效果。當然有可能另外寫出加入解密的影像播放系統，但是在缺乏金鑰的情形下，是沒辦法正確地播放影片內容的。

所謂的安全，其實並不是某樣特定的東西安全。安全常常是很多部分一同架構出來的，而且安全其實也只是一個等級，並沒有絕對的安全。關於我們專題的方向，其實和 DRM (Digital Right Management) 其中的加密影音有相同的方向。因此我們會在後續的討論中詳述 DRM 的技術。

成果簡介

我們的程式目前可以對特定格式影片檔(AVI)的 Video 的 Raw Data 做加密，使得這個影片檔可以在其它的播放程式下播放。但除非輸入了加密時產生的金鑰，否則並不能正確地呈現影片原有的影像內容。

程式其實分成兩個部分。一個部分是對於影片的加密和輸出， 一個部分是影片的解密播放。一般的使用者只需要影片的解密播放部分而已，而影片的提供者就

有影片的加密程式部分。爲了展示方便，因此這裡把這個程式的兩個部分都做在一起，並且提供一個較爲方便的介面來操作。

技術背景

影音處理架構

JAVA Media Framework (JMF) 是一個用途廣泛的 API。它提供 JAVA 程式設計師多樣化的方法來處理不同類型的媒體資料(time-based media)。因此 JMF 是主角而 Time-Based Media 則是與 JMF 產生相互作用的重要配角。此章節將依序介紹 Time-Based Media 的種種特色以及它在基本資料處理模型中所扮演的角色，並由此導入 JMF 在這影音處理、操控的架構。

Time-Based Media (Streaming Media)

任何隨著時間做有意義的變動的資料都可被視爲是 Time-Based Media。
Time-Based Media 的一個主要的特性就是它需要即時的傳輸以及處理。當 Time-Based Media 開始傳送的那一刻，它就必須要面對嚴密的截止時間限制，不管是在接收上或是呈現的階段上都是如此。因此 Time-Based Media 也可被稱爲 Streaming Media 因爲資料的傳送就像是一條穩定的資料流一般。

Media Streams

Media Streams 指的就是從本地的檔案目錄，或是透過網路，或是從麥克風、錄音機等裝置捕捉取得的資料。Media Stream 通常包含數個資料軌(track)。例如 QuickTime 檔案可能會含有一個音軌和一個影像軌。含有數個資料軌的 Media Stream 通常被稱爲(multiplexed)的 Media Streams。Demultiplexing 指的就是將各個資料軌從一個 Media Streams 中抽取出來的處理過程。

- 一個資料軌的類型(type)將能鑑別它所包含的資料種類。 例如音訊或影訊。
- 一個資料軌的格式(format)則是界定資料的所採用的結構方式。

Media Streams 可以以它傳遞資料的方式來區分成兩種類型：推和拉的資料傳輸。.

- 拉(pull)資料傳輸是由 client 端提出啓動的並且是由 client 端控制。 例如 http 和 File 都屬於拉的通訊協定。
- 推(push)資料傳輸是由 server 端啓動並且由 server 控制的。 例如 rtp 就是一個用來傳輸 Media Stream 的推的通訊協定。

Content Type

內容格式指的是 Time-Based Media 儲存的格式。例如 QuickTime, MPEG, 以及

WAV 都是內容格式的例子。 (內容格式其實就是檔案格式，只是因爲 Time-Based Media 的來源，通常不是在本地的檔案目錄中因此統稱爲內容格式)。

Common Media Formats

在選擇使用某個媒體格式作爲特定應用時必須考慮到某些因素：

- 格式的一些特性
- 目標電腦的執行環境 以及
- 你的目標使用者的需求

某些媒體格式當初就是以一些特定應用和需求爲而設計的，因此也特別適合在某些情況下使用。

Media Presentation

大部分的 Time-Based Media 不論影音都可以透過輸出裝置如喇叭或是螢幕呈現。這些裝置，就是媒體資料輸出最常見的目的地。Media Stream 還可以被傳遞到其他的目的地。 例如儲存到一個檔案中，或是透過網路傳送到另一端。媒體資料的輸出的目的地通常是可被稱爲一個*資料槽*(Data Sink)。

Presentation Controls

當一個 Time-Based Media 在播放時它通常會有如同錄放影機般的控制。 例如播放、停止、快速往前、等等的功能。

Presentation Quality

Time-Based Media 的呈現品質通常是仰賴於幾個關鍵的因素，其中包括：

- 壓縮的方式
- 播放系統的能力 以及
- 支援的頻寬 (針對透過網路傳遞的媒體資料流)

傳統上，品質越高，檔案的大小以及處理能力和頻寬就需要越大。頻寬通常是以 bit rate 爲單位。(也就是個時間單位所傳輸的 bit 數)。

要達到高品質的影像呈現，畫格(frame)在一段時間內顯示的次數(也就是所謂的 frame rate)，就必須愈高愈好。通常一部電影的以每秒三十個畫格的速率來播放，就大約跟電視品質不相上下了。

Media Processing

在大部分的情況下，媒體資料流中的資料會經過處理後才呈現給使用者。一般上，有一連串的處理運算會在呈現給使用者前發生。

1. 如果串流是被 multiplexed 那各個資料軌將被提取出來 demultiplex。
2. 如果各個資料軌是在壓縮狀態 那它們將被解碼(decode)。
3. 如果有需要，各個資料軌可以被轉成一個不同的格式。
4. 效果過濾器(effect filters)可以被應用在解碼後的資料軌上。(如果需要)

這些資料軌接著會被傳送到適當的輸出裝置並呈現出來。
但是如果 media stream 將被儲存起來而不是呈現在輸出裝置 情況則有些不同。如果你想將這些將從捕捉裝置的 video/audio 裝置輸出到檔案中則可分成底下這些步驟：

1. 影音軌的取得。
2. 效果過濾器可以被施用到原始軌上(raw track)。
3. 各個資料軌將被編碼(encode)。
4. 壓縮過後的資料軌將被 multiplex 成一個 media stream。
5. 然後這個 media stream 將被存到檔案中。

Demultiplexers and Multiplexers

一個 demultiplexer 的功能是從 media stream 中抽取出各個資料軌。一個 multiplexer 則是執行相反的動作。它將各個資料軌，合併成一個 multiplexed 的 media stream

Codecs

一個加解碼器(codec)扮演解壓縮的角色。當一個資料軌被編碼時，它將被轉換成一個適合儲存和傳輸的壓縮格式。當他被解碼時，它則被轉換成一個適合被呈現的原始(raw)格式。
每一個加解碼器都有它所支援的輸入以及輸出檔案格式。在某些情況下，會用一連串的加解碼器來轉換一個檔案格式到另一個檔案格式。

Effect Filters

一個效果過濾器會對資料軌做某些修改。效果過濾器，通常可分成兩種處理前效

果；pre-processing 以及處理後效果 post-processing。

- 處理前效果指的是施用在 codec 作用之前的效果。
- 而處理後效果則指實施在 codec 作用之後的效果。

通常效果過濾器都是對未壓縮的檔案(raw)作處理。

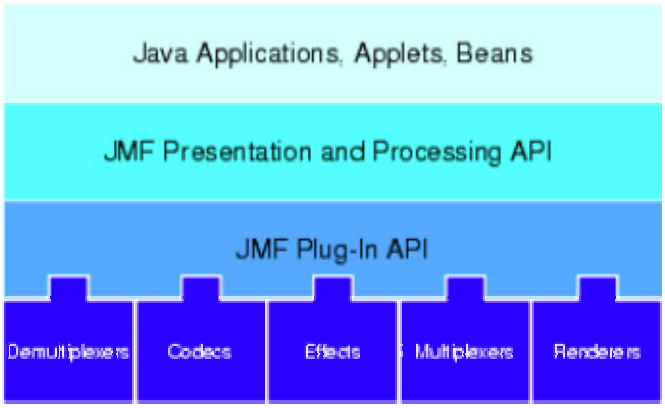
Renderers

呈現器(renderer)是一個呈現裝置的抽象的表示。對音訊來說，呈現裝置指的是電腦的音效卡，對影訊而言，呈現裝置則是電腦螢幕。

JMF 簡介

Java Media Framework (JMF)提供了 Time-Based Media 資料取得、處理、傳送的架構以及訊息傳送協定的統一化。幾乎任何的媒體資料的操控、使用和處理都是可能的。 JMF 支援媒體資料的捕捉並且致力於提供程式設計師對媒體資料額外的控制。它還提供一個 plug-in 的架構來直接存取媒體資料並使 JMF 能更輕易的被延伸以及客製化。

此章節將從 JMF 高階的 API 切入介紹它的階層架構，並一路解說到 JMF 的低階 API(也就 Plug-Ins)。之間會提到許多重要的概念和運作模式(例如時間架構、資料流處理架構、事件通報架構等等)，而這些都將有相當程度的描述。



High-Level Architecture

從一個 VCR 的運作環境中我們可以發覺到 JMF 與真實世界中的運作模式其事是

相當雷同的。一個裝置像是 VCR 提供了一個熟悉的構造模型來錄影(音)、處理、以及呈現 time-based media。當你使用 VCR 播放一個電影時，你是以錄影帶的方式提供 VCR 一個 media stream。VCR 將之讀入並分析、解譯錄影帶上的資料，並把適當的訊號送到你的電視和音響。

JMF 也是使用這個同樣的模型架構。一個 data source 會將 media stream 封裝起來，就有如錄影帶一般。而一個 player 會提供處理以及控制，類似 VCR 所提供的控制機制。使用 JMF 播放、和捕捉影音則需要適當的輸入輸出裝置像是麥克風、照相機、音響、以及螢幕等等。

Data Source 和 player 都是 JMF 高階 API 用來管理、捕捉、呈現、和處理 time-based media 時或不可缺的元件。

JMF 還另外提供了一個低階的 API 來支援客製化的處理元件以及其它的延伸。此階層提供了 Java 設計師一個簡易的 API 來結合、併入 time-based media 到 Java 程式中，同時並保有它原有的靈活性和延伸性來支援進階的 media 應用。

Time Model

JMF 是以奈秒的精確度來紀錄時間的。一段時間中的某一點通常是以一個 Time 物件來表示的。而支援 JMF 時間架構的類別會實作一個 clock 來保持一個 media stream 的時間進度。
Clock 介面宣告了用來控制 media 資料呈現的基本時間和同步化運算。
Clock 使用 TimeBase 來保持 media stream 進行時的時間進度的資訊。
TimeBase 提供一個規律的時脈來源，就好似手錶中的水晶振盪器一般。
TimeBase 只提供目前的時間，也稱爲 time-base time。這個 time-base time 不能被停止或是重新啓動。

一個 Clock 物件的 media time 代表了 media stream 目前的位置。Media stream 剛開始時 media time 是零，而當 media stream 結束時 media time 則代表了 media stream 的最尾端。整個 media stream 的長度則是 media time 從頭到尾的撥放時間表示。

爲了保持目前 media 的時間進度 clock 使用：

- Time-base 的開始時間
- Media time 的起始時間和

■ 播放速率 (也就是 clock 與 timeBase 的相對速度)
播放速率是 time-base 的一個比率係數。比方說播放速率爲 1 時代表正常播放速度，而播放速率 2 時，則代表影片會以兩倍的速率撥放。

當影片播放開始時 media time 將被對應到 time-base time 而此時 time-base time 的進度將被用來測量時間的進度。

Managers

JMF 的 API 主要是由許多個，描述物件間，用來捕捉、處理、以及呈現 time-base media 的行爲和互動關係的介面，所組成的。這些介面的實作是在 framework 的架構中運行。利用一種中介的物件 **managers**，JMF 使得現有的類別與主要介面的實作的結合變的簡單了。

Manager 負責 players、processors、dataSources、以及 dataSink 的建立。這種間接層次的實作是相當有利的。從 client 的角度來看，這些物件永遠是以同樣的方式建造的，不管他所要求的物件實作是內定的或是客製的。

要以 JMF 撰寫一個程式你會需要透過一個 manager create 的方法來建造一個 player、processor、dataSource、或是 dataSink。
稍後我們將會提到 Player、processor、dataSource 以及 dataSink 等 JMF 中重要的元件，並爲它們的功能加以解說。

Event Model

JMF 套用一個結構式的事件通報機制來報知 JMF 程式 media 系統目前的狀態，並讓 JMF 程式能夠對 media 所產生的錯誤狀況，加以回應。當一個 JMF 物件需要通報目前的狀況，它會發出一個 MediaEvent。

對於能夠發出 mediaEvent 的 JMF 物件 JMF 定義了一個相對的聆聽介面。如果當 MediaEvent 被發出時要能夠接收到通報，則須實作恰當的聆聽介面，並且登記將會發出通報的物件。

Data Model

JMF media 播放器通常使用 DataSource 來管理 media 內容的傳輸。一個 dataSource 將 media 的位置以及傳輸的通訊協定資料給封裝起來。

一個 dataSource 是由一個 JMF 的 mediaLocator 或是一個 URL(universal resource locator)所指定的。
mediaLocator 類似一個 URL 並且能從一個 URL 所產生，它們的差別在於 mediaLocator 不需系統事先有安裝指定的通訊協定處理器，而 URL 則需要，否則將無法建立一個 URL 物件。

一個 dataSource 管理一組 sourceStream 物件。一個標準的 data source 使用位元組陣列當作傳輸的單位。一個 buffer data source 使用一個 buffer 物件當作它的傳輸單位。

Push and Pull Data Sources

Media 資料可以由各式各樣的來源取得。例如本地或是網路所取得的檔案。
JMF 的資料來源可以以資料被傳遞的啓動方式來分類：

pull 資料來源－指的是一種由客戶端所啓動並且由客戶端所控制資料流動的資料來源。其中用來傳輸這類資料的較爲典型的通訊協定包括 HPPT 和 File。JMF 定義了兩種 pullDataSource；pullBufferDataSource 和 pullDataSource。

push 資料來源－指的是一種由伺服端所啓動及控制的資料傳遞。Push 資料來源包括廣播 media、multicast、media 以及 VOD(voice on demand)。RTP 是其中一個針對廣播資料所使用的通訊協定；而 SGI 則是一個用來作(VOD)的通訊協定。JMF 同樣也定義了兩個 pushDataSource；pushDataSource 和 pushBufferDataSource。

Data Formats

一個 media 的實際格式是由一個 format 物件所表示的。格式本身並不具有編碼的具體參數，它只形容格式本身編碼的名稱以及格是所接受的資料格式。

JMF 以繼承 Format 類別的方式定義了 audioFormat 以及 videoFormat 類別；
一個 audioFormat 描述了 音訊特有的屬性。例如 bits per sample 和頻道數等等。
一個 videoFormat 則是描述了影訊常有的屬性包括：
IndexedColorFormat
RGBFormat
YUVFormat
JPEGFormat
H261Format

H263Format

Controls

JMF 的 Control 提供了一個機制來設定以及查詢一個物件的屬性。Control 通常提供一個對應的介面元件給使用者對物件的屬性作???存取的控制。許多的 JMF 物件都具有 Controls 包括 Controller 物件 dataSource 物件 dataSink 物件以及 JMF Plug-Ins。

任何想要提供它相對應 Control 物件的 JMF 物件都可以實作 Controls 介面。Controls 介面定義了方法來取得關聯的 Control 物件。DataSource 以及 Plug-In 使用 Controls 介面來提供它們的 Control 物件存取。

Standard Controls

JMF 定義了標準的控制介面如下：

- CachingControl－使得下載進度能被顯示以及監督。
- GainControl 使得聲音大小能被控制。它也支援聲音變動的監督。
- 從 dataSource 讀取資料並將它寫出到目的地如檔案的 dataSink 或是 multiplexer 物件時可以實作 StreamWriteControl 介面。這個控制使得使用者得以限制創造出的串流的大小。
- FramePositioningControl 和 FrameGrabbingControl 將控制框的能力輸出給 players 和 processors。
- FramePositioningControl 提供在一個 player 以及 processor 中 準確的框定位。
- FrameGrabbingControl 則提供一個機制來從視訊流中捕捉一個靜態的影框。
- 有 format 的物件可以實作 formatControl 介面來提供 format 的存取。formatControl 還提供了方法來查詢以及設定格式。
- 一個 trackControl 是一種 FormatControl 它提供一個 processor 物件對一個特定的資料軌的控制機制。
- 利用 trackControl 介面所定義的方法將可以指定對於每個資料軌作想要作的格式轉換並且選擇想要 processor 所套用的 effect、codec 和 renderer plug-ins。
- BufferControl 提供使用者層(針對某種物件)的緩衝區控制。

JMF 還定義了幾個 codec 控制來提供軟硬體編解碼器的控制

- BitRateControl
- FrameProcessingControl
- FrameRateControl...

Extensibility

進階的開發者可以由兩個方法延伸 JMF 功能：

1. 藉由實作一個能夠替換調 JMF Processor 標準處理元件的客製化處理元件 (plug-ins)。
2. 藉由直接實作 Controller, Player, Processor, DataSource, 或是 DataSink 介面

實作一個 JMF plug-in 能讓你客製化或是延伸一個 processor 的原有的功能(使你不必從零開始)。當一個 plug-in 向 JMF 註冊之後，它就可被任何支援那個 plug-in 的 processor 作爲處理的選項。

JMF plug-ins 可以用來

- 延伸或是取代一個 processor 物件的處理能力。
- 存取位於資料流中某一點的 media 資料。例如 effect plug-ins 可以用來作事前和事後處理與一個 processor 相關聯的 media data。.
- 在 player 或是 processor 外處理 media 資料。

在需要更大的彈性的狀況下 JMF controller、player、processor、dataSource 或是 dataSink 介面可以客製化的去實作開發，並且加以結合。

Presentation

在 JMF 中，呈現的過程是由 Controller 介面所展示的。Controller 定義了讓物件控制、呈現以及捕捉 time-based media 的基本狀態(state)以及控制機制。它定義了一個 media controller 所需經過的階段並且提供控制這些階段間轉換的機制。因爲有一些在 media 資料能被呈現前需要執行的運算蠻好時間的，因此 JMF 允許利用程式來控制它們發生的時間。

一個 controller 會發出各式各樣針對 controller 特性的 MediaEvents 來提供它狀態變動的通知。要從一個 controller 例如 player 接收事件你必須實作 controllerListener 介面

JMF API 定義了兩種 controller；player 以及 processor.。一個 player 或 processor 是爲一個 dataSource 所建造的而且通常不能再重複使用來播放其它的 media 資料。

Players

一個 player 處理來源 media 資料串流並且在準確的時候呈現它。一個 DataSource 用來傳遞來源 media 串流到一個 player。呈現的目的地則是依照要表現的 media 的型態來決定。
player 並不提供任何它處理過程中的控制或是它呈現 media 資料的方式。

Player States

一個 player 可以是處在六個狀態中之一。Clock 介面定義了兩個主要的狀態；停止的和開始的。
爲了協助資源管理 controller 將停止的狀態又分成五個待命的狀態；未實現的、實現中的、實現完的、預取中的和預取完的。

在正常的運作下，一個 player 會逐步穿越這些狀態，直到達到開始的狀態。

- 當一個 player 在未實現狀態時它已經被實體化(已經宣告)但卻未得到任何關於 media 的資訊。當一個 player 最初被創造時就是在此狀態。
- 當 realize 方法被呼叫時，player 就從未實現狀態進入實現中狀態。一個正在實現中的 player 正在判斷資源的需求。在實現的過程中 player 將取得它只需要取得一次的資源。
- 當一個 player 完成實現時它將進入實現完的狀態。一個實現完狀態的 player 知道它需要的資源以及它要呈現的 media 種類的資訊。因爲一個在實現完狀態的 player 知道要如何呈現資料，此時它就可以提供畫面元件和控制。
- 當 prefetch 方法被呼叫，一個 player 將會從實現完的狀態進入預取中的狀態。一個預取中的 player 正將準備呈現它的 media。在這個階段 player 會預先載入它的 media 資料並且取得獨占的資源。
- 當 player 完成預取動作，它將邁進預取完的狀態。一個預取完的 player 已經準備好可以撥放了。
- 在乎叫 start 方法後，player 將跳到開始的狀態。一個開始的 player 物件的 time-base time 以及 media time 已經互相對應，且 clock 開始跑了。

一個 player 在當它從一個狀態跳到另一個狀態時會發出 TransitionEvent.。
ControllerListener 介面則會去聆聽 player 目前是在哪一個狀態並依此回應。

Processors

Processors 也可以用來呈現 media 資料。一個 processor 其實就是一個專門的 player。它們的差異在於 processor 允許並提供對來源資料處理的控制。Processor 支援所有 player 的呈現控制。

除了呈現 media 資料在目標裝置上 processor 還可以將 media 資料輸出到一個 dataSource 並使它在另一個 player 或是 processor 被呈現或是存到其他的目的地 (例如檔案)。

Controller Events

由一個 controller(例如 player、processor)發出的 ControllerEvent 可以分成三類：更改通知、結束通知以及狀態變動通知。

更改通知－事件像是 RateChangedEvent、DurationUpdateEvent 和 FormatChangedEvent 的發布表示某些 Controller 的參數被更改了，通常是因應某個方法的呼叫而產生的結果。

狀態變動通知－使得你的程式能夠對 Controller 物件的狀態作適當的反應。

ControllerClosedEvents 會由一個 controller 當它關閉後發出。當一個 controller 發出一個 controllerClosedEvent 它就不能在被使用了。

Processing

一個 processor 是一個 player。它是以 dataSource 爲輸入，在執行時會對 media 資料，針對某些使用者的要求而作處理，並將 media 資料輸出到一個 DataSource 或是呈現出來。如果資料是被送到一個 DataSource 則另一個 player 或是 processor 或是 dataSink 可以把它作爲資料來源。

雖然說 player 的處理過程是由實作者所事先訂定的，但是一個 processore 則由許可應用開發者來定義他們所想要的處理內容。這也就造就了即時的 effect、mixing 和 compositing 處理的可能性。

media 資料的處理被分爲幾個階段：

- 解多工(demultiplexing)是在將來源串流解譯的過程。如果串流是由多個資料軌所構成的，它們將在此被抽取出來，並且分開的輸出。

- 事前處理(pre-processing)是將 effect 演算法使用在以經分開的個別資料軌上的過程。

- Transcoding－這是將一個資料軌轉換成另一個格式的過程。當一個資料流是從一個壓縮的型態，被轉到一個非壓縮的型態時，通常被稱爲解碼。相反的如果它是由，未壓縮的型態被轉到一個壓縮的型態，則稱爲編碼。

- 事後處理(post-processing)是將 effect 演算法使用在解碼後的資料軌的過程。

- Multiplexing 是將多個資料軌結合回一個資料流的過程。

- Rendering 是將 media 呈現給使用者的過程。

每一個處理個過程都是由一個處理元件所負責執行的。這些處理元件就是 JMF plug-ins。如果 processor 支援 trackControls 則可選擇將哪一個 plug-ins 使用在哪一個資料軌上。JMF 總共有五種 plug-ins：

Demultiplexer
Effect
Codec
Multiplexer
Renderer

Processor States

- Processor 另外又提供了兩個待命的狀態；配置中以及配置完。它們是插在未實現和實現中狀態中間。
- 一個 processor 在呼叫 configure 後會進入配置中的狀態。當一個 processor 是在配置中的狀態它會與 dataSource 取得聯繫、demultiplex 來源資料流、並且讀取關於格式的資訊。
- 當 processor 邁進配置完的狀態，它已經與 DataSource 建立連線並且確認完畢資料格式。

Processing Controls

你可以透過 trackControl 控制你想要 processor 對某個資料軌作什麼樣的處理。透過 trackControl 你可以明確的指出你想要採用在你的資料軌上的 effect、codec 和 renderer plug-ins。

Data Output

getDataOutput 方法可以以 DataSource 的方式取得一個 processor 物件的輸出。這個 dataSource 則可以用來當作 player、processor 或是 dataSink 的輸入。

一個 processor 物件的輸出 dataSource 可以是任何的型態：pushDataSource、PushBufferDataSource、PullDataSource 和 PullBufferDataSource。

Media Data Storage and Transmission

DataSink 是用來從一個 dataSource 讀進 media 並且將它呈現到某個目的地的物件。如同 player，dataSink 物件是由 manager 利用一個 dataSource 所建構的。DataSink 可以使用 streamWriterControl 來提供而外的控制。

加解密原理與實作

密碼學簡介

密碼學(Cryptology)一字源自希臘文"krypto's"及"logos"兩字，直譯即爲"隱藏"及"訊息"之意。而其使用，可以追溯到大約四千年前。公元二千年，埃及人就將祭文刻在墓碑上。之後人們都是以書寫在紙張上的方式，用來傳秘密訊息。在二次大戰中，密碼更是扮演一個舉足輕重的角色，許多人認爲同盟國之所以能打贏這場戰爭完全歸功於二次大戰時所發明的破譯密文數位式計算機破解德日密碼。西元1949 年，Shannon 提出第一篇討論密碼系統通訊理論之論文，近代密碼學可說是濫觴於斯。直至西元1975 年，Diffie 與Hellman 提出公開金匙密碼系統之觀念，近代密碼學之研究方向，正式脫離秘密金匙密碼系統之窠臼，蓬勃發展，至今已近二十年。發展至今，主流分爲二大類密碼系統。第一類爲秘密金鑰密碼系統(Private Key Crytosystem，第二類爲公開金鑰密碼系統(Public Key Cryptosystem)。

秘密金鑰密碼系統 (Private Key Cryptosystem)

若加密金鑰 k1 只有合法的發送方知道，則此密碼系統稱爲秘密金鑰密碼系統。一而言，秘密金鑰密碼系統中之加密金鑰 k1 及解密金鑰 k2，具有「知道 k1 即知道 k2，反之亦然」的特性。在很多情況下 k1 等於 k2。因此秘密金鑰密碼系統又稱爲對稱金鑰密碼系統(Symmetric Key Cryptosystem)，或單一金鑰密碼系統(One-Key Cryptosystem)。由於傳統的密碼系統均使用秘密金鑰密碼系統，因此，此系統又可稱爲傳統密碼系統(Conventional Cryptosystem)。幾乎現在的軍事、外交及商業上的應用均採用此系統，以保護其機密資訊。

一個安全的秘密金鑰密碼系統，可以達到下列的功能：

1. 保護資訊機密：明文經加密後，除非擁有解密金鑰 k2(或加密金鑰 k1)，外人無從了解其內容。

2. 鑑定發送方之身份：接收方任意選擇一亂數 r，請發送方加密成密文 C，送回給接收方。接收方再將 C 解密，若能還原成爲來的 r，則可確知發送方的身份無誤，否則是第三者冒充。此乃由於只有發送方及接收方知道加密金鑰，因而才能將亂數 r 所對應的密文 C 求出。其它沒有金鑰的人是無法求出正確的 C 的。

3. 確保資訊完整性：在許多不需要隱藏資訊內容，但需要確保資訊內容不被更改的場合時，發送方可將明文加密後的密文附加於明文之後送給接收方，接收方可將附加之密文解密 (或將明文加密成密文) 然後對照是否相符。若相符合則表示明文爲正確，否則有被更動之虞。通常可利用一些技術，將附加密文之長度縮減，以減少傳送時間及記憶體容量。

然而，秘密金鑰密碼系統具有下列的缺點：

1. 收發雙方如何獲得其加密金鑰及解密金鑰？
   這個問題稱爲金鑰分配問題。若收發雙方互不認識時，此問題尤其嚴重。當我們暫不考慮金鑰分配問題時，可假設發送方與接收方有一條安全通道(Secure Channel)來傳遞金鑰。
2. 金鑰的數目太大。
   若網路中有 n 個人，則每一個人必須擁有 n-1 把金鑰。網路中共需有 n(n-1)/2 把不同的金鑰。當 n 等於 1000 時，每人需保管 999 把金鑰。網路中，共需有 499500 把不同的金鑰。何管理這麼多的金鑰也是一大問題。
3. 無法達到不可否認性服務。
   由於發送方與接收方都知道對方的金鑰，因此發送方可在事後否認他先前送過的任何資訊。因爲接收方可以任意地僞造或竄改，而第三者並無法分辨是發送方抵賴送過的資訊，或是接收方自己捏造的。因而在公開金鑰密碼系統

出現以前，人們認爲要在通訊網路上傳遞文件，而具有人手親筆簽名之同等效力是不可能的。因爲親筆簽名具有事後不可否認的特性。

以下是屬於秘密金鑰密碼系統：
AES，Blowfish，CAST128/CAST256，Cryption，DES/3DES，DESX，IDEA，MARS，RC2，RC5，RC6，SEED，Twofish，Square…

公開金鑰密碼系統 (Public Key Cryptosystem)

由於秘密金鑰的金鑰分配問題及無法達到不可否認性的問題，公開金鑰密碼系統於 1976 年誕生了。在公開金鑰密碼系統中，加密金鑰 k1 與解密金鑰 k2 是分開的。就算知道了 k1 還是無法得知 k2，反之亦然。而只有接收方擁有解密的金鑰 k2。或是反過來，只有接收方擁有加密金鑰可以加密，而任何人都可以解密。由於加密金鑰 k1 與解密金鑰 k2 的不同，因此此系統又稱爲雙金鑰密碼系統 (Two-Key Cryptosystem)，或非對稱密碼系統 (Asymmetric Cryptosystem)。

一個安全的公開金鑰密碼系統，可以達到以下的功能：
1. 保護資訊機密：任何人均可將明文加密成密文，此後只有擁有解密金鑰的人，才能解密。

2. 簡化金鑰分配及管理問題：網路上每一個人只需要一把加密 (公開) 金鑰及一把解密 (私有秘密) 金鑰。這些金鑰只要由接收方自己產生即可。除擁有更高的安全性外，更大大簡化金鑰之分配及管理問題。

3. 可達到不可否認性功能：由於只有接收方自己才擁有解密金鑰，若他先用解密金鑰將明文加密 (簽署) 成密文 (簽署文)，則任何人均能用公開金鑰將密文解密 (驗證) 成明文，並與原來的明文對照。由於只有接收方才能將明文簽署，任何人均能驗證無法僞造。提供此種功能之公開金鑰密碼系統，稱爲數位簽章。數位簽章可以達到確保資訊的鑑定性，完整性及不可否認性，但無法達到秘密性。此乃因爲任何人都可知道公開金鑰 k1，而對簽署文驗證 (解密) 。

然而公開金鑰也具有一些缺點存在。其最大的缺點，就在於加解密運算複雜，且速度緩慢。以著名的公開金鑰系統 RSA 與秘密金鑰系統 DES 比較，DES 的硬體製作可達到每秒 45Mb 之加解密，而 RSA 以現在的技術，仍很難達到每秒 50Kb，兩者相差將近 1000 倍。因此有人建議利用公開金鑰系統達成數位簽章，及解決秘密金鑰密碼系統之金鑰分配問題，而以秘密金鑰密碼系統對明文加解密，達到秘密性功能。此種密碼系統稱爲混合型密碼系統 (Hybrid Cryptosystem)。

公開秘密金鑰中較有名的當屬 RSA。它是由 Rivest-Shamir-Adleman 這三個數學家所創立出來的加密演算法。不過近年來對於 RSA 的研究顯示，RSA 的安全性可能不能達到預期，是一個有待克服的問題。

Java 對於密碼學實作

對於資訊安全，Java 訂定出了 Java Security。Java Security 包含了許多的 API 以達到權限控管、數位簽章、加解密等功能。而 Java Security 中有一個部分稱爲 JCA (Java Cryptography Architecture) 的 API 集合。裡頭包含了數位簽章、摘要值 (Message Digest)、驗證、加解密的相關 API。而在 JCA 的裡面，屬於加解密部分的 API，是叫做 JCE (Java Cryptographic Extension) 的 API 集合。

我們專題這次主要使用的是 JCE 的部分，因爲我們只使用到加解密，並沒有用到其它資訊安全的相關技術。

專題實作技術與討論

專題實作部分，我們分爲處理影片的 JMF 部分，和穿插在其中的加密部分。以下對這兩大部分做介給和說明實作的方法。而程式方面的寫法和架構，請參考附錄中的程式解說。

JMF 部分：

實作技術種類：

目前對於影片的加密，我們知道兩大類實行方法：
1. 對影片的檔案加密，然後播放的時候做即時解密。

這種處理方式是先把整個影片的檔案當成一般的檔案來做處理，把整個檔案都加密起來。然後等到要播放的時候，再即時解開送給播放程式播放。以這種方式來處理的影片加密，有以下優缺點：

優點：

a. 加密部分實作容易，而且加密快速：因爲加密的部分只需要直接讀檔加密，不涉及對影片做深入的處理，因此不但實作容易，而且加密的速度也非常快，以 DES 來說，加密的速度可以達到每秒數十 Mb 的資料量。
b. 加解密和影片格式無關：因爲只需要當成一般的檔案來加解密，不需要考

慮到特定的影片格式實作問題。

c. 不受加密演算法的限制：因爲加密演算法有不同的特性。公開金鑰密碼系統因爲屬於非對稱密碼系統，一般來說密文與明文大小並不相同。而對於秘密金鑰加密系統大部分是區塊加密，除了最後一個區塊之外每一個區塊密文和明文的大小都相同。因爲此種方法對於影片的加密是在最外層，因此任何大小特性的改變都不會對影片的播放造成影片，因爲影片在送給播放器之前就已經經過了解密的過程了。

缺點：

a. 解密部分實作不易：把影片整部解到硬碟上，再送交給播放器就失去了加密的意義，因爲攻擊者隨時可以把解出來的影片複製出來。有的實作式是一次解一小段到硬碟上，送交給播放器處理，但同樣的這種方式和把整部影片解出來在攻擊者的眼中並沒有多大的分別。較安全的一種方式是即時解壓後是由記憶體送給播放器處理，這樣也佔有了速度上的優勢。這種方式提高了安全性，因爲透過程式從記憶體中的 Buffer 側錄資料，需要對 Buffer 的格式、在記憶體中的位置鎖定、以及資料的存放方式都有很深的了解。對於 Java 程式來說，因爲透過了 JVM 的執行，所以記憶體是動態配製(Dynamic Memory Allocation)的。就算不是 Java 的程式，只要 Compiler 的編碼是以動態配製爲主，就很難知道資料是存在哪裡。然而以這種方式寫解密程式，會造成播放器無法和密文對應到正確時間的問題。因爲既然檔案已然被加密，那麼當使用者把影片時間調至下 10 分鐘時，我們並沒辦法得知，下 10 分鐘的資料是在密文中的哪一個部分。因此以這種方式實作的程式，只能循序漸近地看，沒有快轉、倒回、以及隨意指定時間的影片播放基本功能。

b. 若是以解至硬碟的方式來實作的話，有效率低落的問題。因爲周邊的存取往往是最耗時間的，而當影片的播放過程中需要大量地去存取周邊的時候，就會使整個系統的效能快速下降。既然影片播放中，FPS 值是很重要的一個要素，那麼速度的下降就直接影響到這個方法的可行性。

結論：

這種加密的方式的變型是非常適用於 RTP 影音的傳輸加密。因爲 Real Time 的影音並沒有快轉、倒回或是指定任意時間的功能，例如視訊會議這方面的應用。而且因爲影音是即時產生的，所以也沒有一整個檔案加解密產生的問題。只需要對產生的影音串流在 Buffer 裡面做加解密動作就完成了。這樣子不但得到了保密的功能，使得就算使用者使用支援串流通訊的下載軟體下載了串流影音檔，也沒有辦法播放。而且實作簡易、在因爲不涉及大量的硬碟存取的情形下，還沒有效率

的問題。是一個付出不高就可以達到一定安全補強效果的有潛力方法。

但對於不是即時產生影音，而是由一個影片檔播放的情形，不論是否使用 RTP，都會失去需多影片播放基本功能，例如快轉、指定播放點等而大大減低實用性。並不是一個合適的加密方式。

2. 只對影片內的內容物加密，但保存其格式，在播放的時候做即時解密。

這種方式和第一種直接對檔案加密不同的地方是我們只加密影音軌，或是只加密 Raw Data。對這兩者加密都需要先把資料依照資料格式解多工之後才能做進一步的處理。這種作法有以下的優缺點：

優點：

a. 安全性較高：加密的時候針對內容物加密，解密的時候就不需要一段一段地解密，而可以從需要解密的地方解密。
b. 較有彈性：不會因爲加密而失去了一些影片的播放控制功能。因爲加密的是內容物，而影片在播放的時候所需要的資訊，一般來說放在檔頭，是不會被加密的。因此對於播放系統來說，這個影片是可以識別的，只是播放系統可能不知道，內容是經過加密的。
c. 後續發展較有潛力：因爲加解密是針對影音軌或是 Raw Data，因此除了加密解之外，還可以往許多的用途發展。例如加入數位簽章或是加入 Water Mark。

缺點：

a. 加解密實作難度較高：因爲要在影片解開來之後對其做處理，並不如直接對影片加密直觀容易。而且因爲影片需要先被解多工，所以這就涉及了 JMF 支援格式的問題，不被支援的影片格式是沒有辦法處理的。沒有直接對檔案加密的通用性。
b. 可能會有失真的問題：如果加密的資料是在 codec 解開之後，那麼還有可能會有失真的問題。現行的編碼多半採用失真壓縮，以減小多媒體檔案的大小，但是失真壓縮就等於是破壞了資料的完整性，對於加密的資料來說，造成的結果就是無法還原。
c. 效能不佳：加解密的過程雖然沒有大量的周邊存取動作，而選用較快的加密法 (例如 DES) ，加密可以在非常短的時間內完成。可是要對 Raw Data 做處理，需要一連串的資料格式轉換動作，所以若是先經過 codec 再經 effect plugin 的話，需要效能較高的電腦，才能負荷較高的影片即時處理能力。否則會有影片不順的情形出現 (FPS 不足)。

結論：整體來說，雖然使用第二種方法技術難度較高，而且也有不少的問題有待

克服，但是後續的發展性比第一種方法來得好。除了加解密之外還可以做很多途的應用，而且現行所有的問題都是可以解決的，只是需要較多的心力來寫出支援格式的多工/解多工以及 codec 程式。著眼在後續的發展性上，且第一種方式已有人實作過，但第二種方式較少見，因此我們的專題針對第二種方式來加以實作研究。

實作方法：

我們使用了 JMF Extension 中的 Effect Plugin 架構來達成專題的目的。Effect Plugin 其實是一種自訂的 codec，它可以被設定在正常影音的 codec 之前或是之後來處理，也不限定只能有一個。這個部分是由 JMF 提供功能強大的自訂 codec 和 codec 執行順序功能達成的。當一個影音軌 (Video/Audio Track)經解多工出來後，會接著由特定的 codec 來對這些影音軌做解碼的動作。然而 Effect plugin 較不同的地方是，他的功用並不是對這些影音軌做編碼/解碼的動作，而是把一些自訂的效果，加到影音軌上，或是加到已解開的 Raw Data 上面，廣義的來說是另一種形式的編碼/解碼，所以也是 codec 的一種。

根據 Effect Plug 出現在正常影音軌 codec 的前後，可以分成在 codec 之前的 Pre-Process Effect，和 codec 之後的 Post-Process Effect。我們專題的目標是把影片加密，所以不論是在 Pre-Process Effect 或在 Post-Process 裡對資料做處理，都會使得影片無法被正常地播放，而達到我們預期的目的。然而雖然如此，在實務上這兩者有滿大的不同。

對於 Post-Process 來說，因爲大部分影片格式的 codec 在做編碼的時候，採取了失真壓縮以達到有效減少影片大小，而又沒有明顯的畫質減少的目的。因此對於加解密來說，這就形成了一個障礙。既然常用不論是秘密金鑰密碼系統或是公開金鑰密碼系統都有確保資料完整性的特定，密文任意一個位元的更改將造成解密的失敗。而這就和所謂的「失真」壓縮產生了衝突。然而若是沒有失真的問題，那麼加密後的影片將會有一個有趣的現象。因爲是針對影像的 Raw Data 做加密處理，所以其實這個影片可以被任何的播放器開啓，而且可以呈現出加密後的影像。但只有擁有金鑰的解密播放程式才能播放出正確的影像。

而在播放的時候，一樣是要以 Effect Plugin 在 Raw Data 部分做即時的解密，這樣才能讓使用者看到正確播放的內容。

對於 Pre-Process 來說，因爲會產生失真的 codec 編碼是在 Effect Plugin 之後，也就是說影音軌在 codec 外就經加密處理過了。因此解碼後再編碼並不會產生失真的效果，因爲解碼後的內容不變，自然編碼後的結果就是解碼前的結果-也就是我們加密的影音軌。以這種方式達成的影片加密，和 Post-Process 一樣的地方是

沒有正確的金鑰，這個影片檔案對使用者來說是沒有意義的檔案。然而在企圖用別的播放器播放的時候，並不會看到加密後的影像，而是無法播放。因爲影音軌已經經過加密，自然無法被原來這種格式的 codec 識別，進而無法對其做解碼的動作。

在播放的時候，要利用 Effect Plugin 在 codec 解碼之前插入，對影音軌做解密的動作，然後才把解密後的影音軌送交 codec 做解碼，才能繼續播放的動作。

加解密部分：

加解密的部分，我使用了 JCE 來實現。JCE 裡面已經有 Java 定義好的 API，已經可以滿足此專題對於加密的需求。我們的專題採用的是 DES 秘密金鑰密碼系統來加密。

DES (Data Encryption Standard)簡介：

DES 是在 1970 年代中期由美國 IBM 公司發展出來的，且被美國國家標準局公佈爲資料加密標準的一種區塊加密法 (Block Cipher)。直到今日，儘管 DES 已歷經了 20 個年頭，但在已知的公開文獻中，還是無法完全地、徹底地把 DES 給破解掉。換句話說，DES 這套加密方法至今仍被公認是安全的。

DES 屬於區塊加密法，而區塊加密法就是對一定大小的明文或密文來做加密或解密動作。而在 DES 這個加密系統中，其每次加密或解密的區塊大小均爲 64 位元，所以 DES 沒有密文擴充的問題。就一般資料而言，無論明文或密文，其資料大小通常都大於 64 位元。這時我們只要將明／密文中每 64 位元當成一個區塊而加以切割，再對每一個區塊做加密或解密即可。但是對明文做區塊切割時，可能最後一個區塊大小會小於 64 位元。此時，我們要在此區塊之後附加「0」位元，直到此區塊大小成爲 64 位元爲止。

另一方面，DES 所用的加密或解密金鑰 (Key)也是 64 位元大小，但因其中有 8 個位元是用來做錯誤更正 (Error Correction) ，所以 64 位元中真正是金鑰效用的只有 56 位元。而 DES 加密與解密所用的演算法除了子金鑰的順序不同之外，其它的部分則是完全相同的。

DES 擁有加密速度快的優點，但是也有一個安全性不足的缺點。
安全性的不足有兩個原因：

1. 金鑰的長度過短：因爲 DES 的金鑰長度只有 56 位元，因此許多人認爲這樣子的長度，並不足以應付攻擊。
2. S 盒子的暗門：DES 的加密過程中，一共有 16 回合的重覆運算，每個運

算利用 XOR 邏輯函數及所謂的「S 盒子」對數據做非線性的處理。因爲對於 S 盒子的分析評估，至今仍列爲機密。許多人擔心這些 S 盒子隱藏了秘密的暗門 (Trapdoor)。NSA (美國國安局)能夠經由這些暗門，無需密鑰就能破解密文。根據 IBM 中參與設計 DES 的人表示，NSA 對 DES 的原設計唯一修改之處就是這些 S 盒子，因爲根據 NSA 做的分析，發現原設計中的 S 盒子具有某些弱點。

1994 年 Wiener 提出了一種「已知明文攻擊 (Known-Plaintext Attack)」。Wiener 首先以 Pipeline 結構設計出一片可以每秒尋找五千萬支秘密金鑰的晶片。再以十萬元的代價設計一台裝有 5760 塊上述晶片的秘密金鑰破解機。預估在 35 個小時內，可以解出 DES 的秘密金鑰。而更有人說 NSA 擁有可以在 10 分鐘內破解 DES 的密碼破解機器。因此對於需要高安全性的資料，DES 似乎是不能滿足需求的。

專題加解密實作

我們的專題選用 DES 來做加解密密碼系統，有以下幾點原因：

3. DES 的 key 長度只有 56bit，對於需要高安全度的東西不太足夠，但是對於影片的加密已經足夠，因爲破解 key 需要的成本遠大於一部影片的價格。
4. DES 加密的速度很快。他可以達到 45Mb/s 的加密速度，加密的速度對於影片加密的實作是非常重要的。如果使用 RSA 之類的公開金鑰密碼系統來做影片加密的話，那麼很容易就造成效能低落，進而使得影片的 FPS 不足，造成影片不流暢。而秘密金鑰密碼系統比公開金鑰密碼系統快得多，因此選定秘密金鑰密碼系統來加解密。
5. DES 是屬於對稱加密法，又是使用區塊加密碼。這樣的加密法有一個特性，那就是除了最後一個區塊外，一樣大小的區塊明文和加密出來的密文的長度是相同的。對於影片來說，我們可能會處理到 Raw Data，也就是一張一張的圖片(Frame)，保持加密前後的大小才能維持影片格式的正確性，以對影片產生的影響最小。而若是使用公開金鑰密碼系統來加密的話，往往密文會比明文來的大，每張圖片都變大累積起來，會造成影片比原來的影片大上許多。
6. DES 是區塊加密法。他有許多的模態可以供選擇。而其中我們選用 EBC 模態。因爲這個模態並不會把前後的資料使用連鎖加密，因此提高了影片的容錯程度。若是其中一個區塊壞掉了，不會導至所有同一把 Key 加密的部分全部損壞。預設的區塊是 64bit，換算成 24bit RGB 圖形大約是 3 個像素(Pixel)，而 3 個像素的錯誤對整張圖片來說，是可以接受的。若是使用 CBC 之類的連鎖模態，會造成整張圖片，甚至整部影片的損毀。因爲這個原因，所以我們選擇 DES 加密法。

加密的部分，我們分爲以下幾個步驟：

1. 設定金鑰的加密演算法以使用合適的格式。
2. 使用 Secure Random 產生安全的亂數金鑰。
3. 利用這把金鑰，把區塊加密器 (Cipher) 初始化，同時選定模態，加密或解密模式和 Padding 方式。
4. 使用初始化完畢的區塊加密器來對字元陣列 (Byte Array)做加解密。

在程式裡面我們把這些步驟封裝在一個類別裡面以簡化程序。

現有的議題

已知程式問題

目前對於此研究的程式實作已知的問題有兩方面，一方面是程式上的技術問題，一方面是此架構所衍生出來的問題。以下是這兩方面的問題的說明及討論。

程式技術問題

格式支援度問題

目前我們的程式只支援 AVI 格式的影片加解密。其中一個主要的原因是因爲 JMF 只支援了少數的格式。而目前常見的影片格式裡頭，只有 AVI 和 QuickTime MOV 格式支援了部分的編碼和解碼兩種功能。其餘的不是沒有編碼功能，就是完全不支援。以下是 JMF 對於多媒體格式詳細的支援列表：

- **D** indicates the format can be decoded and presented.
- **E** indicates the media stream can be encoded in the format.
- **read** indicates the media type can be used as input (read from a file)
- **write** indicates the media type can be generated as output (written to a file)

Media Type			
JMF 2.1.1 Cross Platform Version	**JMF 2.1.1** Solaris/Linux Performance Pack	**JMF 2.1.1** Windows Performance Pack	
AIFF (.aiff)	**read/write**	**read/write**	**read/write**

8-bit mono/stereo linear	D,E	D,E	D,E
16-bit mono/stereo linear	D,E	D,E	D,E
G.711 (U-law)	D,E	D,E	D,E
A-law	D	D	D
IMA4 ADPCM	D,E	D,E	D,E
AVI (.avi)	**read/write**	**read/write**	**read/write**
Audio: 8-bit mono/stereo linear	D,E	D,E	D,E
Audio: 16-bit mono/stereo linear	D,E	D,E	D,E
Audio: DVI ADPCM compressed	D,E	D,E	D,E
Audio: G.711 (U-law)	D,E	D,E	D,E
Audio: A-law	D	D	D
Audio: GSM mono	D,E	D,E	D,E
Audio: ACM**	-	-	D,E
Video: Cinepak	D	D,E	D
Video: MJPEG (422)	D	D,E	D,E
Video: RGB	D,E	D,E	D,E
Video: YUV	D,E	D,E	D,E
Video: VCM**	-	-	D,E
GSM (.gsm)	**read/write**	**read/write**	**read/write**
GSM mono audio	D,E	D,E	D,E
HotMedia (.mvr)	**read only**	**read only**	**read only**
IBM HotMedia	D	D	D
MIDI (.mid)	**read only**	**read only**	**read only**
Type 1 & 2 MIDI	-	D	D
MPEG-1 Video (.mpg)	**-**	**read only**	**read only**
Multiplexed System stream	-	D	D

Video-only stream	-	D	D
MPEG Layer II Audio (.mp2)	**read only**	**read/write**	**read/write**
MPEG layer 1, 2 audio	D	D,E	D,E
QuickTime (.mov)	**read/write**	**read/write**	**read/write**
Audio: 8 bits mono/stereo linear	D,E	D,E	D,E
Audio: 16 bits mono/stereo linear	D,E	D,E	D,E
Audio: G.711 (U-law)	D,E	D,E	D,E
Audio: A-law	D	D	D
Audio: GSM mono	D,E	D,E	D,E
Audio: IMA4 ADPCM	D,E	D,E	D,E
Video: Cinepak	D	D,E	D
Video: H.261	-	D	D
Video: H.263	D	D,E	D,E
Video: JPEG (420, 422, 444)	D	D,E	D,E
Video: RGB	D,E	D,E	D,E
Sun Audio (.au)	**read/write**	**read/write**	**read/write**
8 bits mono/stereo linear	D,E	D,E	D,E
16 bits mono/stereo linear	D,E	D,E	D,E
G.711 (U-law)	D,E	D,E	D,E
A-law	D	D	D
Wave (.wav)	**read/write**	**read/write**	**read/write**
8-bit mono/stereo linear	D,E	D,E	D,E
16-bit mono/stereo linear	D,E	D,E	D,E
G.711 (U-law)	D,E	D,E	D,E
A-law	D	D	D
GSM mono	D,E	D,E	D,E

DVI ADPCM	D,E	D,E	D,E
MS ADPCM	D	D	D
ACM**	-	-	D,E

就算是 AVI 和 MOV，對於特定的影像格式還是不支援編碼。而最常用到的 Mpeg 格式則是全面不支援編碼，只支援了 Mpeg-1 影像部分的解碼。這使得我們目前只能先實作出 AVI 部分的加解密，其餘的尚不能實作出來。

架構衍生問題

失真問題

我們此專題是使用 JMF Effect Plugin post effect 的架構來對影片的圖片 (Frame)，也就是 Raw Data 的部分加解密。在此架構下，影片處理的流程是在進入 Effect Plugin 之前先由 codec 解碼之後做加密，然後再由 codec 編碼後交由多工演算法輸出檔案。在由 codec 編碼的時候會造成圖片的失真，也就是我們加密後的檔案會遭遇到些許的破壞。此一特色使得加解密沒辦法很直接地實行，因爲不論破壞的程度如何，非完整的密文是沒辦法解密的。

我們採用 AVI 格式來實作程式，主要的考量是我們使用的影像軌 (Video Track) 的格式是 MJPG，在經過轉型成爲 RGB 之後輸出編碼是不會失真的。雖然如此，他所帶來的副作用就是檔案會變大許多。以 1.5mb 的範例影片來說，加密後的輸出會大至 36mb。

壓縮比下降問題

影像間壓縮的原理，主要是利用減少記錄圖片間不變的像素，而只記錄圖片間改變的「變化量」來達到壓縮的目的。而單一的圖像格式壓縮，大至上是利用除去人眼不敏感的資訊來達到壓縮的目的。然而經過壓縮的影像，會失去在圖片和圖片間的相關性，使得以減少記錄圖像間不變像素的方式難以發揮優勢。以 QuickTime MOV 實測結果，會使 4mb 的檔案增加至 7mb 的大小。當然增加的大小與原先的大小沒有關聯，而與 Frame Rate 和圖片間的相同程度有較大的關係，但可以確定的是，加密後的影片，一定會大於一般電影的影片。所以對影像的加密，會使得壓縮比下降。

解決方法

格式支援度問題

這個問題的解決方法是自行利用 Extend JMF 的方式來支援所需要的格式。因爲 JMF 明確定義了 codec，multiplexor 和 demultiplexor 的格式，所以只要實作特定的介面 (Interface)，就可以新增自己的 codec 和多工/解多工程式。而對原先完全不支援的格式，還要自行寫 Data Source，以支援此一格式的檔案當做輸入來源。

失真問題

對於失真問題的解決方法有兩個。

使用加密 Video Track 的架構

先前已討論過我們實作加密影片的方式有兩種。其一是加密 Video Track，也就是在 codec 解開之前就加以加密，其二就是我們目前採用的，加密 Raw Data 的方式。因爲加密不容許任何的失真，所以跳過會使其失真的部分，也就是 codec，就能根本地解決問題。而部分容許失真的 Water Mark，則可以使用加密 Raw Data 的架構，以滿足可以很方便地對圖片做修改的需求。而使用此一架構還有另一個好處就是因爲不是針對 Frame 加密，而是對尚未被 codec 解開的 Video Track 加密，所以較不會有壓縮比下降的問題。

使用自訂的 codec，只加密非失真部分圖像

因爲會造成失真的部分是 codec，因此若是我們知道哪些資訊會失真，就可以針對不會失真的資料做加密。例如在轉 JPEG 格式的過程中，由 DCT 餘弦轉換出來的資料，會有一部分被丟棄，這個部分是會失真的部分。若是我們只針對這些 DCT 出來不會被丟棄的資料來做加密，就可以解決了 JPEG 失真的問題。這個解決的方法也有其缺點，最大的兩個缺點是實作難度高，以及沒辦法根本解決問題。以下是其討論：

a. 實作難度較高：因爲需要涉及到較底層的運作，不但要自行寫 codec，還要由其中找出加密非失真資料的方法。這個部分因目前尚未實作，所以不知道其可行性如何。需要再對其進行研究。

b. 沒辦法根本解決問題：上面我們討論的是 JPEG 格式失真的可能解決方法，但是圖片不一定是以 JPEG 格式來存放。其它的格式解決失真的方法也不會一樣，若是以後出現了新的格式來存放圖像，例如 JPEG2000，那麼也要爲此新格式找出解決失真的問題。

雖然有這些缺點存在，但是因爲目前的主流是 Mpeg 格式的檔案。所以若是解決了此一失真問題，便可以在 Mpeg 格式的影片上做許多資訊安全的技術實作研究。例如：數位簽章、不容許失真的 Water Mark 技術 (例如存放執行檔、文字訊息等)。

壓縮比下降問題

壓縮比的下降，主要的導因是因爲一系列動作的影像因爲加密而失去了相似性，而使得壓縮的原理難以發揮。這個方面目前有兩個因應的對策：

使用加密 Video Track 的架構

此一架構因爲避開了 codec，因此並沒有壓縮比下降的問題。但採用此一架構，就失去了之前所提及使用加密 Raw Data 的優勢。

使用部分加密

當對整張圖像做加密的時候，會讓圖像間完全失去關聯性，但是如果加密的並不是整張圖像，就能保存些許的關聯性，從而讓影像的壓縮擁有一點優勢。我們加密的最終目的是不希望使用者在未授權的情形下得以看到影片的內容。若是我們能只加密一部分的影像，使得影片雖然可以被看到部分的內容，但是因爲加密部分導致的畫質下降而讓影片失去其價值，則就能達到我們的目的。然而加密多少才能導致影片失去觀看的價值，是一個非常主觀的問題。這個部分需要更深入的研究與各別的需求做評估才能達到一個可以接受的平衡。而部分加密的實作方法有許多種，如何才能加密最少的地方，達到對影片最大的保護，也需要再加以研究評估。

結論

目前探討的問題是已知的問題，而且大部分都尚未實作出解決方法。雖然如此，這些可以作爲未來的發展方向，甚至是未來有新的問題出現的時候，都可以再進一步的研究解決方案。

專題與目前資訊安全技術的關連

我們的專題與目前一項資訊安全的議題有很大的關係。這個議題就是數位智慧財產權的保護 (DRM , Digital Right Management)。以下是對 DRM 的介紹。

DRM 簡介

1. DRM 是什麼？

DRM 的全名是 Digital Right Management，也就是數位智慧財產權的管理。DRM 是一個可以把數位媒體與一些功能的的結合。這些功能包括了保護不被未經授權的複製，而且具有可以影響多媒體檔案播放的方式。DRM 是一項可以提供數位媒體內容可以在網路上安全的發佈、行銷還有販賣的技術。

2. DRM 爲什麼重要？

網際網路和個人電腦的發展已經顯著地改變了數位媒體內容-例如音樂、影片-的創造，發佈和使用的方式。網際網路的使用者現在依賴網路下載他們有興趣的多媒體內容，而非實際上去店家買他們喜愛的 CD 或是 DVD。檔案分享程式現在

也因爲提供了對於需求內容的立即存取而被網際網路使用者普遍地使用。數位媒體的檔案並不會因爲複製而減低品質，而且可以被更容易地發佈。因爲沒有安全的措拖，所以數位媒體檔案在網路上發佈已經造成了對於隱私權的隱憂，不論是經由授權還是非授權的管道。DRM 使得內容提供者可以控制他們內容物的發佈。針對每一個發行的內容物，他們可以設定一個使用權利規定來保護他們的智慧財產權。而且他們也可以對他們的產品如何使用或是發佈設定限制。這些設定權利的過程也使得網路管理者能受到一些來自於顧客的重要資訊，而這些資訊可以供提供者做爲一個改進的依據，進而提供品質更好的服務。

3. DRM 有什麼用途？

a. 網路管理者可以搜集關於使用者使用數位媒體產品的資訊，例如他們看了什麼影片？什麼時間看的？在什麼地點觀看？電腦內的設定是什麼？等等…。獲得這些客戶市場的資訊對於數位內容提供者非常重要。
b. 當使用者在使用多媒體檔案的時候，可以設定一個使用時間的限制，使得他們只能使用此一內容物特定的時間，例如觀看 20 分鐘的影片或是 7 天。一但時間到期了，使用者就必需要購買可以繼續使用的授權。
c. 你可以在網路上透過檔案分享系統，例如一些 P2P 軟體，來發佈你的影片。而當使用者下載而觀看的時候，就可以把他們導向一個可以購買授權或是申請試用的網頁。以這種方式可以方便地實現 Pay-Per-View 的模式，而顧客可以只買他們想要看的部分。
d. 它是提供給組織內部內容物的一個好用的工具。當你需要關掉某位成員的帳號的時候，你同時也關掉了他對於內容物的存取權利。

當然 DRM 的用途不止這些，然而隨著技術發展，我們對於數位產品的保障就就多。

4. DRM 有什麼經濟利益？

除了因爲保護數位內容物而得到的利益之外，DRM 也可以透過例如減少頻寬使用的來達到經濟利益的目的。例如當網路上的影片是 Pay-Per-View 的模式的時候，使用者就不一定要下載整個內容物，而可以只購買下載他們所需要的部分。DRM 的技術可以防止一些不斷試用的投機客，DRM 把對於內容物的控制權利交回到提供者的手中，而不是一但數位媒體放到網路上，就對它無計可施。

結論

DRM 的技術還不斷地在發展中。有許多的研究正在進行，許多的技術正在開發。最有名的例子是微軟 (Microsoft)的 Windows Media DRM 9 Series。希望能在不久的將來，使得每位數位媒體的創作者的心血都受到保護。

專題和 DRM 的關係

我們的專題是對影片做加密，並以此方式限制使用者的複製，同時也可以對發佈媒體的方式做控制。目前短期的目標是以加密的方式，達到可以讓使用者買到他們需要的部分的片段來觀看。這需要把我們原始的加密模式外，再加上一層影片的分割。如此一來不但可以節省頻寬的使用，對媒體提供者提供了智慧財產權的保障，也讓使用者可以選擇只購買需要的部分，減少了浪費。而影片的加密，其實只是 DRM 技術中的一環，通常光是靠單一的技術並沒有辦法全面地對內容物有所保障，而需要許多的技術或是平台來加以支援。希望後續的研究，可以對影片的 DRM 技術有更多的發展。目前微軟的 Windows Media DRM 9 Series 已經可以做到利用對影片的加密達到某些權限控制，而我們希望能在 Java 上開發此一技術，以使得此技術可以應用到非 Windows 的平台上。

參考資料

http://java.sun.com/products/java-media/jmf/2.1.1/index.html
SUN 的 JMF 網頁

http://www.hut.fi/~then/multimedia.html
一個連向許多多媒體資源的網頁

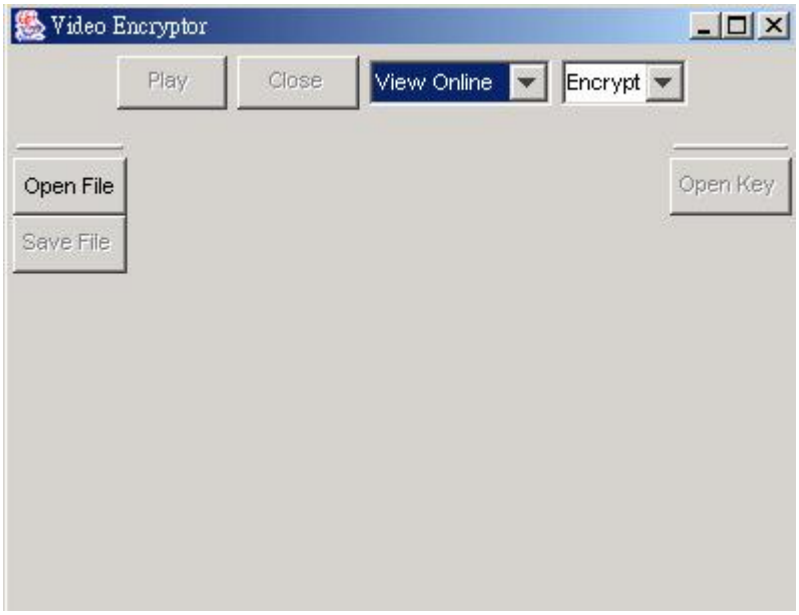
http://www.hn.is.uec.ac.jp/~arimura/compression_links.html#standards
影像壓縮技術相關資源連結

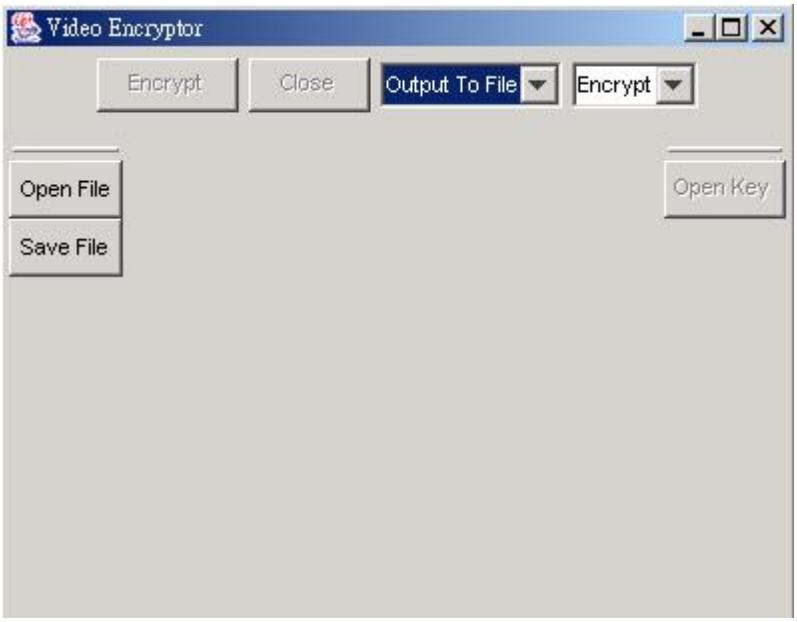
附錄

系統配置-以 Windows 平台爲例

1. 安裝 Java SDK
2. 安裝 JMF SDK
3. 依照 Sun 的安裝說明設定 CLASS_PATH
4. 直接執行 EncryptEffect.jar，或在命令模式打 java –jar EncryptEffect.jar

執行後程式開啓程式之圖片





程式操作

1. 觀看影片加密效果
   選擇 View Online，Encrypt→Open File 選定所要加密影片→按 Play→開始。
2. 將影片加密
   選擇 Output To File，Encrypt→Open File 選定要加密影片→Save File 選擇要輸出檔案名及路徑→(選擇 Open Key 以此把 Key 加密，若無的話會自動產生)→按 Encrypt 開始。Key 會產生在與輸出檔同地方，檔名與輸出檔同，副檔名爲 des。
3. 播放加密影片
   選擇 View Online，Decrypt→Open File 選定已經加密之影片→按 Open Key 以選擇解密用的 Key→按 Play 開始。

全文完