私立東海大學資訊工程與科學系

碩士論文

指導教授：林 祝 興 博士、王　　凡 博士

(Dr. Chu-Hsing Lin、Dr. Farn Wang)

應用 LDAP 於公開金鑰基礎架構之研究與實作

Design and Implementation of LDAP for Public Key Infrastructure

研 究 生：鐘 子 杰　撰

(Tzu-Chieh Chung)

中 華 民 國 九 十 一 年 六 月

致　　謝

本篇論文能順利的完成，要致謝的人非常多，首先要感謝是我的指導老師，林祝興老師。在進入東海大學資訊工程與科學系攻讀碩士學位的過程中，無論在資訊專業知識領域或為人處世的觀念與態度上，都獲得您相當多的指導與協助；您學者智慧與風範，皆是足以學習仿效之表率，更是成為我不斷地要求自己，絲毫不懈怠的驅動力。

其次，非常榮幸能邀請到王凡、洪國寶、黃胤傅、葉義雄四位老師（依姓名筆劃排列）擔任我的口試委員，對於本論文中未臻完整之處，提供了許多寶貴的意見，由衷的感謝您們。

再則要非常感謝交通大學資訊工程學系賴威聲學長的不吝指教，讓我得以在學習的瓶頸中自我突破，確立本論文之研究主題與方向。另外，謝謝資訊安全實驗室的同儕，李正隆、李維貞、江峰菖及楊國鴻，無論對於本論文或平日的研究，能夠相互討論、教學相長，對我撰寫本論文助益良多；最後要感謝李家豪、黃國榮等學弟在口試期間的協助及本系助教葉錦菁對於本論文的校稿，讓本論文能順利完成。

我想，未來的成就能不負大家期望，更勝於在此諸多感謝，這也是我完成此論文後對自我的期許。

鐘子杰　謹誌　於　東海大學

中華民國九十一年七月十日

摘　　要

在公開金鑰基礎架構下的網路環境，針對網路使用者身分鑑別的問題，以非對稱式加密方法，建立出以密碼學為基礎的安全機制。透過具有公信力且可信賴的第三者－憑證中心，來發行及維護使用者的公鑰憑證，使得公鑰系統更具正確性及可靠性。

目錄服務提供分散式系統中應用程式或服務對資料庫內部資料快速的搜尋、存取等功能，「要求－回應」的機制將結果即時地回應給用戶端。LDAP 是以 X.500 目錄服務為基礎發展出來的網路協定，簡化的機制，提供了更高的可用性與便利性。

隨著使用者對網際網路服務的需求日益頻繁，身分鑑別及存取控制等安全問題也益發重要，如何讓安全機制更便捷、更有效率的應用在網際網路的環境中，成為相當值得研究之課題。利用具快速存取特性的 LDAP 於公開金鑰基礎架構環境中，有效的降低了憑證中心的工作負荷。加上屬性憑證來做為存取控制之依據，運用這樣的安全機制，將使得網路安全的問題獲得更好的改善。

關鍵字：公開金鑰基礎架構、憑證中心、公鑰憑證、目錄服務、LDAP、屬性憑證。

Abstract

In the public key infrastructure network environment, the solution of authentication of the internet users is to build a security architecture, grounded on the knowledge of cryptographic, by using asymmetric ciphers. The third party – certification authority, a trustworthy organization, is responsible for issuing and managing the public key certificates of the end entities. Thus, it gives assistance in the public key system to be more accuracy and reliability.

Directory service provides the applications or services to search and access data items rapidly in the distributed system. The "request-response" mechanisms supply the immediate results to the end users. LDAP is a new network protocol developed based on X.500 directory service. The simplistic scheme offers more availabilities and conveniences.

Several security issues such as authentication and access control are neatly arranged in the wake of the frequency and dependence on the internet services for the end users. It comes to be a great course how to implement the security in the internet in facilitate and efficiency way. Taking advantage of the characteristics of the LDAP, high-speed data access, the workload of the certification authority is reduced noticeably. The network security issues will be improved by applying the architecture above, appending the access control with the attribute certificates.

Keyword: Public key infrastructure; Certification authority; Public key certificate; Directory service; LDAP; Attribute certificate.

目　　錄

圖表目錄

第壹章　序論

第一節 研究背景

隨著網際網路的發達，對於網路服務的使用亦日益盛行，在網路上進行交易、收發郵件、閱讀隱私資料已儼然成為日常生活的一部份；在我們依賴著網路上便利與快速的服務同時，所衍生的網路安全問題卻是更需要探討的課題。在電子化的身分認證系統中，常面臨到一個共同的問題，便是身分認證的關係只建立在使用者與特定的電腦系統之間，而無法將這種關係延伸到更多的使用者團體與更廣的網路系統。

在使用這些網路服務的同時，為了安全的顧慮，通常使用者必須經過身分的驗證，驗證成功後才會被賦予使用這些服務的權限，而身分的驗證機制，最傳統也最為廣泛使用的便是利用使用者帳號與密碼。但是，在網路上使用不同的伺服器、不同的網路服務，往往會用到不只一組以上的帳號與密碼，在一般的情況下，記憶這麼多組的密碼，反而容易造成安全上的漏洞，例如將容易忘記的帳號密碼紀錄在紙上而遭竊取，因此，這樣的身分認證系統並無法成為真正的公共身分認證系統。

公開金鑰基礎架構(Public Key Infrastructure, PKI)提供了公共的身分認證系統環境，目的在維護每個憑證政策(Certification Policy, CP)

與憑證實作準則(Certificate Practice Statement, CPS) ，規範其運作規定與作法，一方面讓用戶瞭解在使用上的作業規定，另一方面則藉此表明其在安全性及公正性上的可信賴度。在公開金鑰基礎架構下的網路環境，將使用者身分的認證的問題，利用非對稱式加密方法，建立出以密碼學為基準的安全機制，並透過具有公信力且可信賴的第三者，憑證中心(Certification Authority, CA)，來發行及維護使用者的數位憑證。

隨著電子簽章法的通過，公開金鑰基礎架構的必然性，與憑證中心的重要性，已成為最重要的課題。然而，隨著公開金鑰基礎架構日益普及的趨勢，憑證中心勢必發行、維護更多的使用者憑證，而使整個憑證中心的運作變的龐大而缺乏效率。

目前在網際網路上用來整合眾多不同伺服器帳號與密碼的認證服務，以目錄存取服務的應用最為出色。目錄服務的功能並不只侷限於此，它提供了易於查詢、存取的資料庫系統；其資料結構與服務協定更具層次分明的組織架構之特性。如何將目錄服務的這些優點，應用在公開金鑰基礎架構的環境下，規劃出更完整、更具效率的，成為本論文的研究主題重心。

第二節 論文架構

本論文除了本章序論外，第二章將提出相關的資訊安全技術研究，包含公開金鑰基礎架構(PKI, Public Key Infrastructure)、X.509 認證服務、安全通道層(SSL, Secure Socket Layer)、傳輸層安全(TLS, Transport Layer Security)等。第三章針對適合應用於公開金鑰基礎架構下的存取控制機制的目錄服務及 LDAP (Lightweight Directory Access Protocol)進行研究與介紹。第四章則說明在公開金鑰基礎架構環境下，規劃並設計出應用 LDAP 目錄服務在公開金鑰環境的系統。第五章為結論，針對第四章提出之應用系統做一回顧與相關問題討論，並對本論文做一總結及提出未來可再研究發展之方向。

第貳章　相關技術研究

第一節 公開金鑰基礎架構

在電子化的身分認證系統中，通常面臨到一個共同的問題，便是身分認證的關係只建立在使用者與特定的電腦系統之間，而無法將這種關係延伸到更多的使用者團體與更廣的網路系統；也就是說，這樣的身分認證系統並無法成為真正的公共身分認證系統。

公開金鑰基礎架構提供了公共的身分認證系統，目的在維護每個使用者的憑證的正確性，也就是使用者公開金鑰的正確性，能夠證明此公開金鑰是屬於某一位特定使用者所擁有。

公開金鑰基礎架構由下列幾個主要角色所組成(圖 2-1)：

- 終點實體(End Entity)：使用憑證的的主要對象，終點實體可視為一般使用者、團體、公司行號等。
- 憑證中心(CA, Certification Authority)：可信任的公正第三者，負責憑證、憑證廢止清冊之發行與管理等工作。
- 註冊中心(RA, Registration Authority)：位於憑證中心之前端，終端實體可透過註冊中心申請憑證。
- 憑證/憑證廢止清冊資料庫(Certificate/CRL Repository)：公開金鑰基礎架構中的資料儲存體，負責儲放憑證、憑證廢止清冊

之資料。

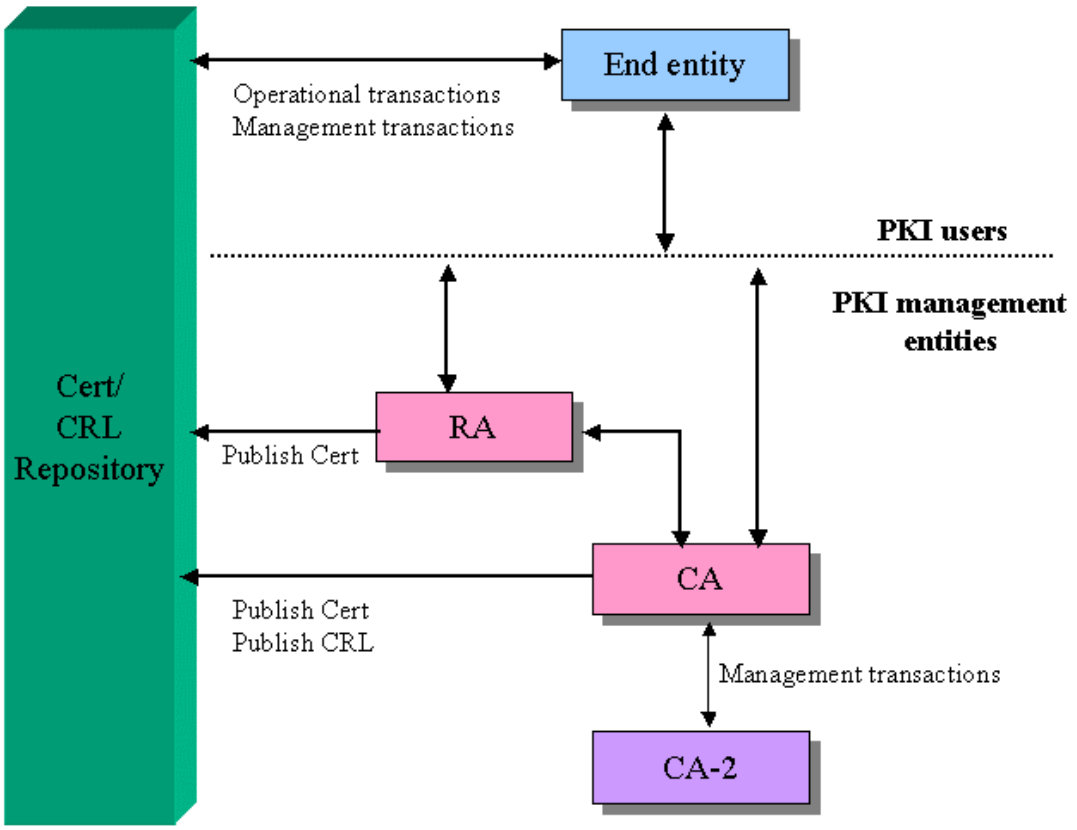

圖 2-1 公開金鑰基礎架構之憑證管理

一、憑證中心

憑證中心是一種組織，在公開金鑰基礎架構中扮演的是可信任的公正第三者，目的在對個人及機關團體提供認證及憑證簽發管理等服務，以建立具有機密性(Confidentiality) 、鑑別(Authentication) 、完整性(Integrity) 、不可否認性(Non-repudiation) 、存取控制(Access control)及可用性(Availability)的資訊通訊安全環境與機制。主要工作是負責憑證與憑證廢止清冊之發行、憑證與公開金鑰的管理以及處理與其他認證中心相互之間的信任關係。

然而，在實際的運作上，憑證中心不可能完全地處理每一個使用者申請憑證的作業，因為憑證中心不可能驗證所有申請憑證之使用者的身分，所以必須透過前端的註冊中心來當協助處理身分驗證的機制，再交由憑證中心發行憑證；如果註冊中心本身是一個可自行發行憑證的組織單位，亦可以由註冊中心代為發行憑證。註冊中心的角色並不一定為一特定的人或單位來執行，只要能夠確認使用者身分資料的單位，都可以是註冊中心。

在建置營運憑證中心時，須依憑證中心之營運政策及策略，制訂憑證政策(Certification Policy，CP)與憑證實作準則(Certificate Practice Statement，CPS) ，規範其運作規定與作法，一方面讓用戶瞭解在使用上的作業規定，另一方面則藉此表明其在安全性及公正性上的可信賴度。

二、憑證發行機制

憑證的發行機制可分成兩種，基本驗證機制(basic authenticated scheme)與集中管理機制(centralized scheme)[8]。

■ **基本驗證機制**(basic authenticated scheme)：

由申請憑證之使用者產生金鑰對(key pair)**及憑證要求**(certificate request)，**將憑證要求以憑證中心之** IAK(Initial Authentication Key)**加密傳回憑證中心進行驗證，再依憑證要求產生使用者之憑證並發行**(圖 2-2)。

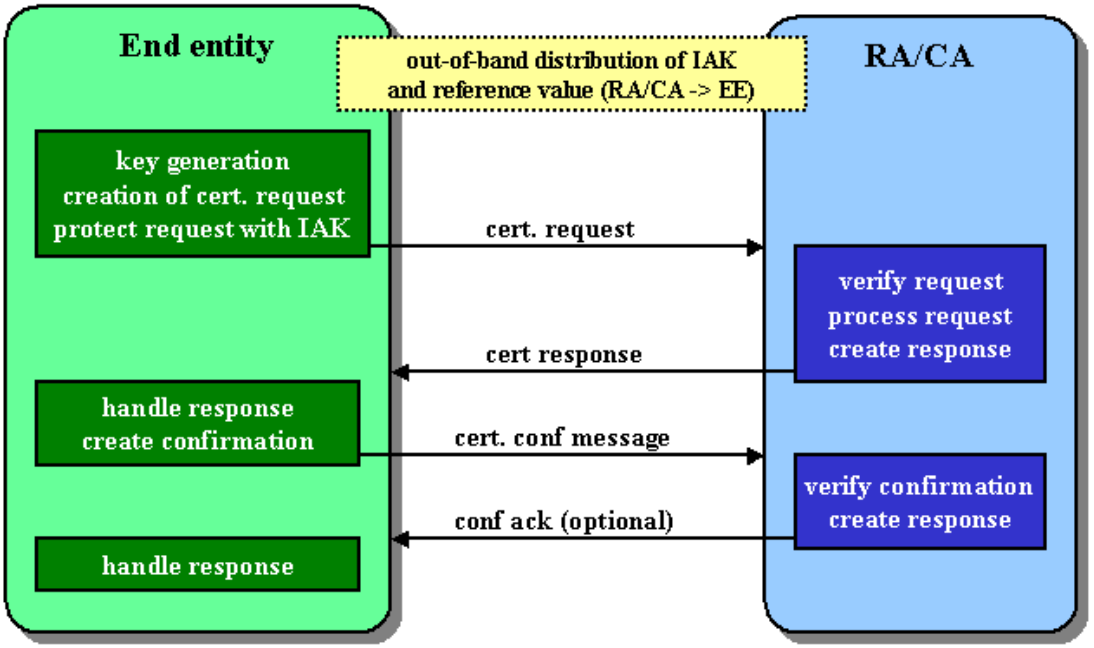

圖 2-2 憑證發行－基本驗證機制

■ **集中管理機制**(centralized scheme)：

使用者將 PSE(Personal Security Environment)**資料送至憑證中心，由憑證中心產生金鑰對及憑證，並將憑證發行。**

第二節 X.509

電子化作業在網際網路上的應用日益廣泛，同時衍生原本傳統人

工作業上所沒有的資訊安全問題，其中，網路上的身分認證(Authentication)機制便是一項重要課題。為了在開放式網路架構下認證遠端使用者的身分，ITU (International Telecommunication Union)於 1988 年提出"開放性系統互連－目錄服務：認證架構"(Open System Interconnection－The Directory：Authentication Framework) X.509，之後此標準為國際標準組織 ISO(International Standard Organization)所採用為 ISO 9594-8 認證標準。目前幾乎所有的公開金鑰建設(PKI)標準都是架構在 X.509 此標準所發展出來的。

一、X.509

在 ITU 提出之 X.509 規格中，對於認證(Authentication)提出兩種不同安全度的認證層級：「簡單認證」與「強認證」；X.509 規格中並描述了公開金鑰憑證(Public Key Certificate)格式[10]、憑證管理[11]、憑證路徑、目錄資料結構及金鑰產生與管理[12]等，其中並包含憑證中心間交互承認的憑證之儲存，以減少憑證驗證時必須從目錄服務中獲得憑證資訊。X.509 規格內容可歸納包含下列項目：

- 簡單認證程序(Simple Authentication Procedure)：

  此部份所定義的驗證程序是使用一般最常見的密碼(password)認證的技術來辨識通訊的對方。

- 強認證程序(Strong Authentication Procedure)：

此部份內容，提出一個高安全度的身分認證機制，就是使用公開金鑰密碼學的技術，來辨識通訊的對方。在 X.509 提出的認證架構，並非要建立一個通用的認證架構，整個認證協定只有描述資料項。

- **金鑰及憑證管理**(Key and Certificate Management)：

  因為強認證程序需使用公開金鑰密碼系統的特性來進行認證工作，因此，此部份針對金鑰與憑證的管理與正確性做一重點摘要，並定義了憑證廢止清冊(Certificate Revocation List，CRL)的內容。

- **憑證擴充及憑證廢止清冊**(Certificate Extensions and CRL)：

  由於 1988 年版的 X.509 中對憑證及憑證廢止清冊的定義並不是很完整，所以在 1995 年針對這些問題，提出了 X.509 修正案，對於這兩部分做一些修正與補充。

二、X.509 憑證

1. 在 X.509 規格中所定義的憑證(Certificate)必須要符合下列兩點特性：

   - 任何可由憑證中心取得公開金鑰的使用者，都可以找到已經通過認證的公開金鑰。
   - 除了憑證中心以外，任何人修改憑證的動作都會被偵測、察覺。

2. X.509 **憑證除了符合上述兩個特性外，憑證內容必須包含以下資訊欄位(圖** 2-4)**：**

- **版本(Version)：此憑證發行所依循的版本。**
- **序號(Serial number)：憑證中心對此憑證所發予唯一的序號。**
- **簽章演算法(Signature algorithm identifier)：憑證中心發行此憑證使用之簽章演算法，用以驗證簽章時使用。**
- **發行者名稱(Issuer name)：發與此憑證之憑證中心名稱。**
- **有效期限(Period of validity)：憑證之有效期限，包含起始日期與終止日期。**
- **主旨名稱(Subject name)：此憑證之擁有人之使用者名稱。**
- **公開金鑰資訊(Public key information)：包含此憑證之公開金鑰與公開金鑰相關資訊。**
- **簽章(Signature)：憑證中心以其私密金鑰對此憑證之簽章。**

了解上述符合 X.509 **規格之憑證需包含的內容資訊後，憑證可以下列格式表示如下：**

CA<<A>> = CA{V, SN, AI, CA, T_A, A, Ap, (UCA), (UA)}

其中各個符號所代表的意義如下：

A ：使用者名稱。

V ：憑證版本。

SN：憑證序號。

AI：簽章演算法。

CA：憑證中心名稱。

T_A：憑證之有效期限。

A_p：憑證之公鑰，包含公鑰相關資訊。

(UCA)：憑證中心識別碼(選擇性使用)。

(UA) ：使用者識別碼(選擇性使用)。

Y<<X>>：憑證中心 Y 核發使用者 X 之憑證

Y{I}：憑證中心 Y 對訊息 I 所做的簽章，其中包含訊息 I 及簽章。

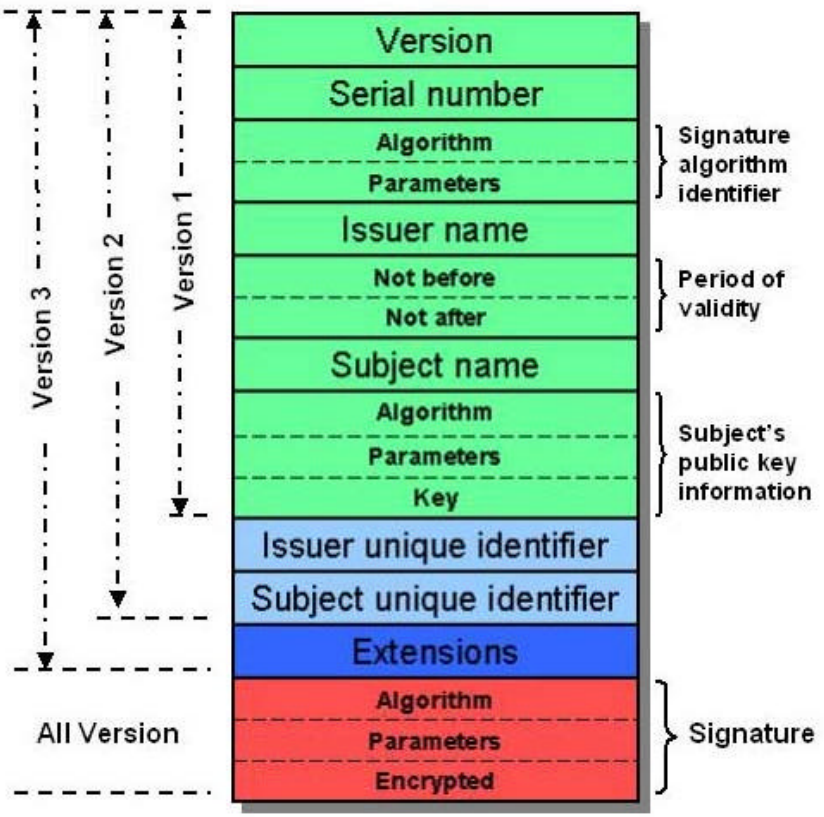

圖 2-4　X.509v3 **憑證格式**

憑證中心利用自己的私密金鑰來對憑證做簽章，如果使用者知道對應的公開金鑰，就可以確認此憑證是否確實為該憑證中心所簽署的憑證。

三、X.509 憑證廢止清冊

在憑證內容中定義了有效期限，如信用卡一樣，在舊憑證快要過期之前，我們就要發出新的憑證。除此之外，在某些情況下，我們會在憑證到期之前就廢止它。可能原因如下：

- 使用者的私密金鑰被破解。
- 使用者不再由此憑證中心來認證。
- 憑證中心的憑證被破解。

因此，憑證中心必須維護一張清冊，清冊單記錄的是已經被廢止但是尚未過期的憑證；這些憑證可能是發給使用者或其他的憑證中心。此憑證廢止清冊(Certificate Revocation List)經由發行者的簽署，內容包含發行者的名稱、清單的產生日期、下一張憑證廢止清冊的預定核發日期，以及所有被廢止的憑證(圖 2-5)。廢止的憑證記錄中包含憑證的序號及廢止日期，因為序號對憑證中心而言是唯一的，所以利用序號就足以辨識這些被廢止的憑證。

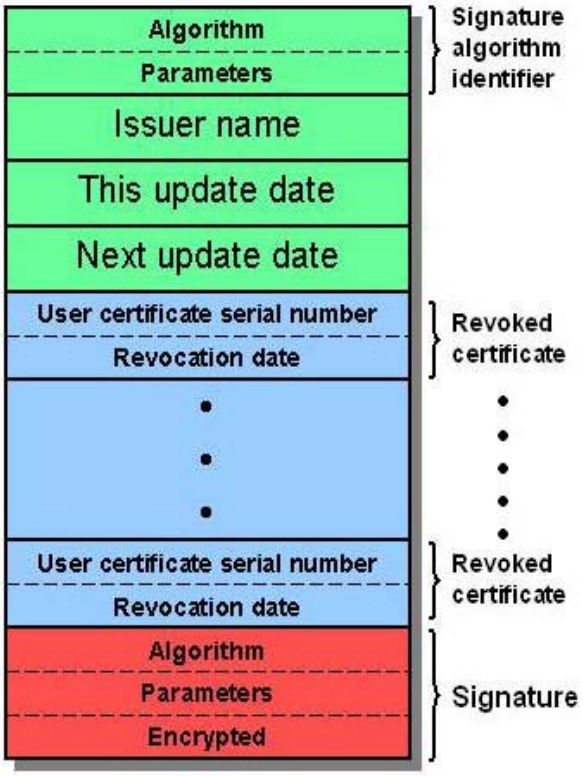

圖 2-5　X.509 憑證廢止清冊格式

四、X.509 v3 – 憑證/憑證廢止清冊擴充

由於 1988 **年版的** X.509 **規格中對憑證及憑證廢止清冊的定義並不是很完整，對後來的設計與實作角度來看，以** X.509 **第二版的格式並無法傳送足夠所需要的資訊，所以在** 1995 **年針對這些問題，提出了** X.509 **修正案，對於這兩部分做一些修正與補充。除了對上述問題提出解決方案外，為了避免因為增加固定格式的欄位而必須做版本的修改，修正的** X.509 **第三版提出了選擇性的擴充資料欄位，每個擴充欄位項目包含了一個擴充識別碼、一個關鍵指標及一個擴充值。**

X.509 v3 **定義的憑證與憑證廢止清冊中的擴充項目，依功能可分**

為三類：金鑰與策略的資訊(key and policy information)**、憑證擁有者與憑證發行者的屬性**(certificate subject and certificate issuer attributes)**，及憑證路徑的限制**(certification path constraints)**。**

1. **金鑰與策略的資訊**
   - **金鑰識別碼**(Authority key identifier)
   - **使用者金鑰識別碼**(Subject key identifier)
   - **金鑰用途**(Key usage)
   - **私密金鑰使用期限**(Private-key usage period)
   - **憑證策略**(Certificate policies)
   - **策略對映**(Policy mapping)
2. **憑證擁有者與憑證發行者的屬性**
   - **使用者別名**(Subject alternative name)
   - **發行者別名**(Issuer alternative name)
   - **使用者目錄屬性**(Subject directory attributes)
3. **憑證路徑的限制**
   - **基本限制**(Basic constraints)
   - **名稱限制**(Name constraints)
   - **策略限制**(Policy constraints)

五、憑證路徑驗證

利用憑證來進行身分確認的動作，如果所有使用者都採用同一個憑證中心的話，那他們就會同時信任這個憑證中心，因此也可以直接信任其他使用者的憑證。

當有很多使用者時，通常讓所有使用者採用同一個憑證中心的情況，並不是很實際的做法。在使用者的憑證為不同憑證中心所簽發的情況下，如何驗證對方憑證的正確與合法性，我們必須透過憑證路徑驗證來達到此一目的。在我們要驗證一張個別的憑證時，必須先建立其憑證路徑(Certificate Path)或憑證鏈(Certificate Chain)，也就是將該憑證與其關聯的憑證中心之憑證建立階層式架構，一直到我們可以信賴的第三公正單位或憑證中心。憑證路徑的驗證必須滿足下列條件(假設憑證路徑中共有 n 張憑證{1, .., n})：

- 在{1, . „. n-1}的憑證路徑中，所有的憑證 x，其憑證的主旨(subject)必須是憑證 x+1 的發行者名稱(issuer)。
- 憑證 1 必須是一個可信賴的公正單位。
- 憑證 n 必須是終端使用者之憑證(非可簽發憑證的憑證中心)。
- 在{1, . „. n-1}的憑證路徑中，所有憑證 x 必須都是在有效期限之間。

憑證路徑之驗證演算步驟如下(圖 2-6)：

(1).初始化(initialization)

(2).**基本憑證驗證處理**(basic certificate processing)

(3).**下一張憑證處理準備**(preparation for next certificate)

(4).**包裝**(wrap-up)

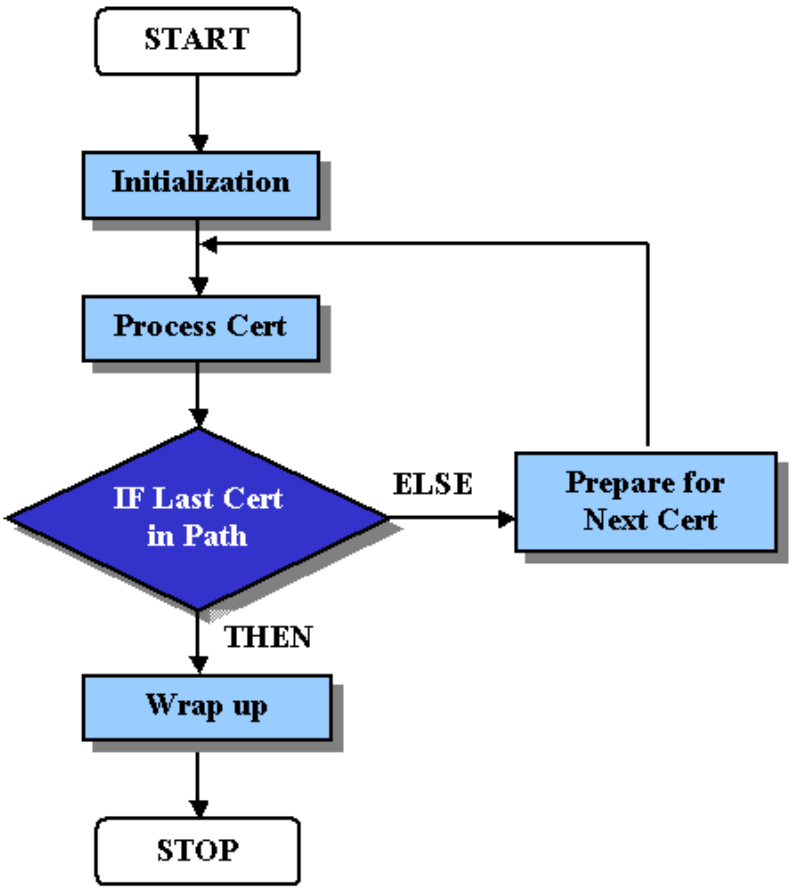

圖 2-6 憑證路徑之驗證演算步驟

六、憑證路徑驗證演算法

前一章節定義了憑證路徑的驗證，並且說明了驗證作業包含四個主要步驟：初始化、基本憑證驗證處理、下一張憑證處理準備、包裝，此一章節針對這四個步驟，說明每一步驟的主要功能及細節。

1. **輸入**(Input)：

憑證路徑驗證演算法的包含了七個輸入值：

(1).**憑證路徑：憑證路徑長度為** n**，亦即內容包含** n **張憑證**(1~ n)**。**

(2). ***T*****：目前日期與時間。**

(3). ***user-initial-policy-set*****：使用者憑證策略，若不使用憑證策略可放入一個** any-policy **值。**

(4).***Trust anchor information*****：憑證路徑中可信賴第三者(憑證中心)之資訊，包含發行者名稱**(issuer name)**、公開金鑰演算法**(public key algorithm)**、公開金鑰參數**(public key parameters)**。**

(5). ***initial-policy-mapping-inhibit*****：指示可否使用** policy-mapping**。**

(6). ***initial-explicit-policy*****：指示可否使用** explicit-policy**。**

(7). ***initial-any-policy-inhibit*****：指示可否使用** any-policy**。**

2. **初始化**(Initialization)

初始化的步驟中，主要在建立憑證路徑驗證演算法所使用之變數；根據輸入的七個參數值，建立十一個狀態變數(state variables)**：**

- valid_policy_tree
- permitted_subtrees
- excluded_subtrees
- explicit_policy
- inhibit_any-policy
- policy_mapping
- working_public_key_algorithm

- working_public_key
- working_public_key_parameters
- working_issuer_name
- max_path_length

3. **基本憑證驗證處理** (Basic Certificate Processing)

完成初始化步驟後，開始針對 1~ n **之憑證進行驗證處理，其中驗證步驟又可細分如下：**

(1).**驗證憑證之基本資料**

檢查及驗證憑證內容之基本資料的正確性，驗證的內容包含下列作業：

- **簽章是否正確。**
- **憑證是否在有效期限內。**
- **憑證是否被列入憑證廢止清冊。**
- **發行者名稱是否合法。**
- **發行者識別碼是否合法。**

(2).**驗證主旨名稱**(subject name)

(3).**驗證憑證策略擴充**(certificate policies extension)

4. **下一張憑證處理準備** (preparation for next certificate)

在基本憑證驗證處理步驟完成後，如果基本憑證驗證處理的憑證並不是憑證路徑的最後一個(第 n **張憑證)，則需要在處理下一個憑證**

驗證作業之前，建立一些狀態變數。假設目前完成基本憑證驗證作業的為憑證 i (i=1, 2, 3, ….., n-1)**，下一張憑證驗證處理準備作業步驟如下：**

(1).**如果憑證** i **有** policy mapping extension**，驗證** any-policy **是否出現在** issuerDomainPolicy **或** subjectDomainPolicy **中。**

(2).**如果憑證** i **有** policy mapping extension**，則修改** ***valid_policy_tree*****。**

(3).**將憑證** i **之主旨名稱**(subject name)**放入** ***working_issuer_name*** **變數中。**

(4).**將憑證** i **之公開金鑰放入** ***working_public_key*** **變數中。**

(5).**將憑證** i **之公開金鑰參數放入** ***working_public_key_parameters*** **變數中。**

(6).**將憑證** i **之公開金鑰演算法放入** ***working_public_key_algorithm*** **變數中。**

(7).**如果憑證** i **有** name constraints extension**，則修改相關之狀態變數(*permitted_subtrees*、*excluded_subtrees*)。**

(8).**如果發行者與主旨名稱不一致，則檢查**

- **如果** explicit_policy **值大於** 0，explicit_policy **值減** 1。
- **如果** policy_mapping **值大於** 0，policy_mapping **值減** 1。

◆ **如果** inhibit_any-policy **值大於** 0，inhibit_any-policy **值減** 1。

(9).**如果憑證** i **有** policy constraints extension，**則修改相關之狀態變數(*explicit_policy*、*policy_mapping*)。**

(10).**如果憑證** i **有** inhibitAnyPolicy extension，**且其值小於** ***inhbit_any- policy*，則將** ***inhbit_any-policy*** **值設為** inhibitAnyPolicy。

(11).**驗證憑證** i **是否為一張** CA **的憑證。**

(12).**如果憑證** i **並非自己簽發**(self-issued)，**則檢查*max_path_length*值是否大於** 0，**若值大於** 0，**則將** ***max_path_length*** **值減** 1。

(13).**如果憑證** i **有** pathLengthConstraint，**且其值小於** ***max_path_length*，則將** ***max_path_length*** **值設為** pathLengthConstraint。

(14).**如果憑證** i **有** key usage extension，**將** keyCertSign **這個位元設定為** 1。

(15).**確認及處理憑證** i **中其他關鍵性的擴充參數。**

5. **包裝** (Wrap-up)

完成憑證 i **到憑證** n-1 **的基本憑證驗證處理與下一張憑證處理準備後，相同的，憑證** n(**使用者憑證**)**也經基本憑證驗證處理，但完成後便進入包裝的步驟。包裝的過程如下：**

(1).**如果憑證** n **並非自行簽發**(self-issued)**，且** ***explicit_policy*** **值大於** 0**，則將** ***explicit_policy*** **值減** 1**。**

(2).**如果憑證** n **有** policy constraints extension **及** requireExplicitPolicy**，且** requireExplicitPolicy **值為** 0**，則將** ***explicit_policy*** **設定為** 0**。**

(3).**將憑證之主旨名稱**(subject name)**放入** ***working_issuer_name*** **變數中。**

(4).**將憑證之公開金鑰放入** ***working_public_key*** **變數中。**

(5).**將憑證之公開金鑰參數放入** ***working_public_key_parameters*** **變數中。**.

(6).**確認及處理憑證** n **中其他關鍵性的擴充參數。**

(7).**計算** valid_policy_tree **及** user-initial-policy-set**。**

6. **輸出** (Output)

輸出主要在表現整個憑證路徑驗證處理過程為成功或失敗。若是驗證失敗，則輸出驗證失敗的代表值及驗證失敗的警訊；若是驗證成功，除了輸出驗證成功的代表值外，同時輸出下列參數的最終值(處理憑證路徑中憑證 n **的值)：**

- valid_policy_tree
- working_public_key
- working_public_key_algorithm

- working_public_key_parameters

七、屬性憑證

目前 ITU **根據** X.509v3 **之規格，針對存取控制擬定了屬性憑證** (Attribute Certificate)**標準，並公佈為網際網路草案**(inernet draft)**，目的在取代現有的** X.509v3 **規格。屬性憑證是從使用者的公鑰憑證(為了方便區分憑證類型，我們將上述的** X.509 **憑證稱為公鑰憑證)衍生而來的一種分散式結構；公鑰憑證主要用途是包裝主體的公鑰，而屬性憑證則用來存放主體的相關屬性，因此屬性憑證並不包含主體的公鑰。一個公鑰憑證的主體可能會有一個以上的屬性憑證，而這些屬性憑證與其公鑰憑證均具有關聯性。公鑰憑證和屬性憑證並不一定要由相同的機構所發行，否則當憑證的發行與管理工作頻繁，將造成發行機構的負擔。在不同的發行機構的環境下，憑證中心發行的公鑰憑證和屬性憑證中心**(Attribute Authority, AA)**發行的屬性憑證會以不同的私鑰來做簽章。如果發行公鑰憑證的** CA **和發行屬性憑證的** AA **是在同一個發行機構下，最好將此發行機構視為屬性憑證而非標示公鑰憑證。屬性憑證內容如下：**

- **版本(Version)：屬性憑證的版本。**
- **持有者(Holder)：包含公鑰憑證發行者或公鑰憑證之序號、主體名稱與物件摘要。**

- **簽章演算法(Signature Algorithm Identifier)：屬性憑證中心簽發此屬性憑證之演算法。**
- **屬性憑證序號(Serial Number)：屬性憑證中心發行此屬性憑證的唯一序號。**
- **有效期限(Period of validity)：包含起始日期及終止日期。**
- **發行者識別(Issuer Identifier)：屬性憑證中心之識別名稱。**
- **屬性(Attributes)：主體的屬性值，可包含一個以上的屬性值。**
- **屬性憑證擴充(Extensions)**

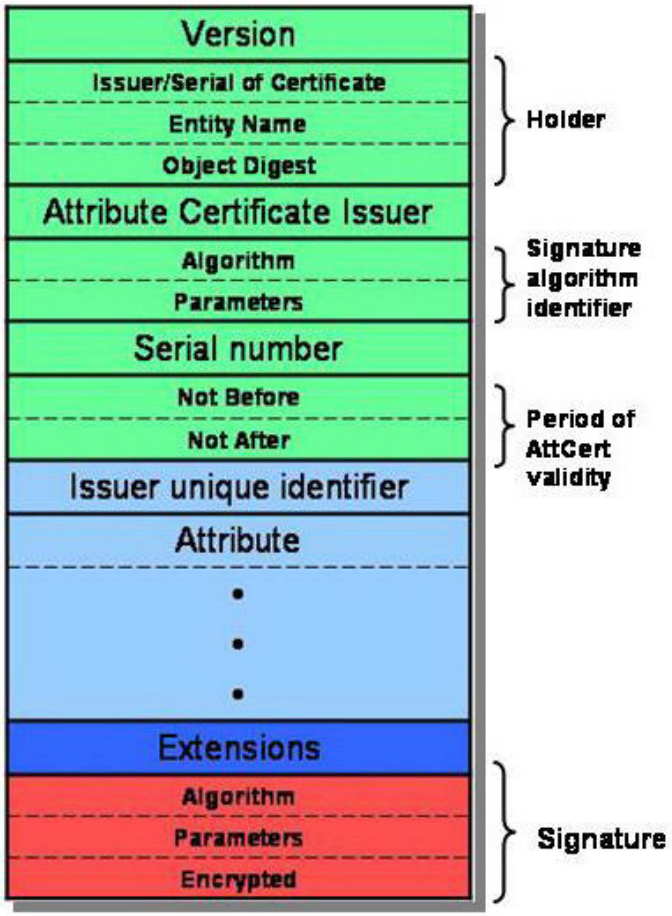

圖 2-7　屬性憑證格式

第三節 SSL/TLS 協定

在網際網路日益發達的今日，網路使用者透過網際網路來從事各種便利的活動，例如電子郵件、電子購物、資料傳輸等，在開放性網路架構下，數位化資料傳輸的安全性，成為不可或缺的重要問題；在毫無安全機制下的網路中，使用者可能面臨資料遭到冒名傳送、竄改、偽造、不法擷取等問題。為保護資料於網際網路傳送時具安全性及可信賴性，Netscape Communications 公司首先設計 SSL (Secure Socket Layer) Protocol[14]，其介於應用層(Application Layer)及傳輸層(Transport Layer)之間，並將 SSL Protocol v3.0 之網路草案文件(Internet-draft)公佈於網站，在 1999 年被 IETF(Internet Engineering Task Force)接受後，更名為 TLS (Transport Layer Security) 1.0 版，是為 RFC 2246[15]，並被廣泛應用在電子商務的安全機制。SSL Protocol v3.0 主要具備以下特性：

- 連線具隱密性，於交握協定時產生秘密金鑰(secret key)，以此秘密金鑰來對傳送之訊息進行加解密。
- 點對點(peer to peer)之間的身分驗證採非對稱式加密法(Asymmetric cryptographic)。
- 連線具可信賴性，訊息傳送時包含訊息完整性之檢查，使用具金鑰之訊息鑑別碼(keyed Message Authentication Code, MAC)。

一、SSL 協定架構

SSL 協定架構於開放性系統互連(Open System Interconnection)，介於應用層(Application Layer)與傳輸層(Transport Layer)之間(圖 2-8)，主要提供網際網路中相互連結的兩個應用層之間相互通訊時的私密性(privacy)與可靠性(reliability)。SSL 協定是層次式的協定(layered protocol)，由兩層子協定所組成，分別為紀錄協定(SSL Record Protocol) 與交握協定(SSL Handshake Protocol)。

- 紀錄協定(SSL Record Protocol)：

  SSL 協定的較低層，主要目的是藉由可靠的傳輸協定(TCP)，將各種不同之較高層協定包裝後再傳送。

- 交握協定(SSL Handshake Protocol)：

  在兩個應用層之間接收或傳送資料前，提供伺服器與用戶端之間相互認證(authenticate)之機制，並協議雙方通訊使用之加密演算法(cryptographic algorithm)及產生加解密金鑰(cryptographic key)。

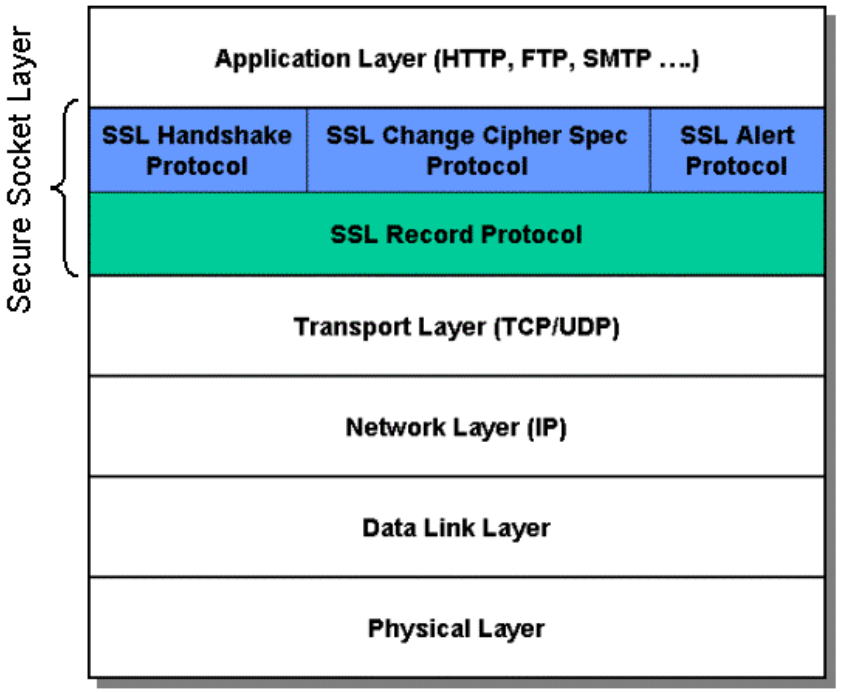

圖 2-8　SSL **安全協定**

另外，在 SSL **協定中，**SSL **的狀態依其參數可分為兩種，分別為交談狀態**(session state)**及連線狀態**(connection state)**，此兩個狀態的定義如下：**

1. **交談狀態**(session state)

交談是指用戶端與伺服器之間的聯合(association)**，交談於交握協定時建立，並定義一組安全參數**(cryptographic security parameter)**，其中包含此交談使用之加解密演算法與交談金鑰**(session key)**，在多個連線中之任一連線皆可分享使用該參數。建立一個交談所需要的參數如下：**

- **交談識別碼(session identifier)：伺服器端會任意挑選一個位元**

組序列用以辨識一個正在運作(active)或可重新開始(resumable)的交談狀態。

- 節點認證(peer certificate)：此參數為 X509 v3 之憑證，交談狀態中的此資料欄位可以為空白。
- 壓縮方法(compression method)：在資料區塊加密前使用的壓縮演算法。
- 密文規格(cipher spec)：定義密碼規格，包含資料的加密演算法、訊息驗證碼演算法。此參數並定義了一些密碼相關的屬性，例如雜湊值的大小。
- 主要秘密(master secret)：一個 48 位元組值，由伺服器端及用戶端所共同使用。
- 是否可回復(is resumable)：旗標，用來表示此交談狀態可否開始一個新的連線。

2. 連線狀態(connection state)

一個連線就是一次傳輸程序(根據 OSI 層級模型的定義)，它提供了一個適當的傳輸方式。對於 SSL 而言，這種連線是點對點(peer to peer)的關係，並且此連線是暫時的(transient)。每一連線對應一個交談(session)。建立連線所需要的相關參數如下：

- 伺服器與客戶端亂數(server and client random)：在每次連線建

立時，由伺服器端與用戶端選擇之位元序列亂數。

- 由伺服器寫入 MAC 秘密(server write MAC secret)：伺服器端針對送出的資料來產生 MAC 時所需的秘密金鑰。
- 由用戶端寫入 MAC 秘密(client write MAC secret)：用戶端針對送出的資料來產生 MAC 時所需的秘密金鑰。
- 由伺服器寫入金鑰(server write key)：伺服器端使用於加密資料之金鑰，同時也是用戶端用來解密之金鑰。
- 由用戶端寫入金鑰(client write key)：用戶端使用於加密資料之金鑰，同時也是伺服器端用來解密之金鑰。
- 起始向量(initialization vectors)：當我們使用 CBC 模式的區塊加密法時，每一把金鑰都會有一個起始向量(IV)。此欄位的初始值是由 SSL 的交握協定來設定。之後，每個紀錄所產生的密文區塊就會成為下個紀錄的起始向量。
- 序號(sequence numbers)：每個節點會針對不同的連線所發出/接收的訊息來維護一組獨立的序號。當雙方收到 change cipher spec 訊息之後，此序號值將被設定成為零;序號不得超過 2^{64}-1。

二、交握協定層(Handshake Protocol)

交握協定在整個 SSL 中是最複雜的一部份，主要目的在讓用戶端與伺服器端之間能相互認證(authentication)，並協議彼此加密及產生

訊息鑑別碼的演算法以產生用來保護欲傳送之資料安全的金鑰。在進行任何 SSL 的連結之前，一定要先經過交握協定協議這些安全參數後，才能開始傳輸資料。建立一個新的安全通道連線的交握協定的流程(圖 2-9)分為下列四個主要步驟：

1. **建立安全環境** (Establish Security Capabilities)

此一步驟主要為用戶端與伺服器端進行初始化的邏輯連結，協議彼此使用的安全環境參數。首先由用戶端送出 Client_hello 的訊息給伺服器端，伺服器端收到後送回 Server_hello 的訊息，進行初始參數值的協議。其中這兩個訊息包含下列欲進行協議的初始參數：

- **Version**：所使用 SSL 的最高的版本。
- **Random**：亂數，包含 32 位元的時間戳記(timesamp)及 28 位元的亂數產生器產生之亂數。
- **Session ID**：欲建立交談之識別碼。用戶端送出時，此值若為 0，則為欲建立一個新的交談；非零值代表欲接續之前的交談。伺服器端送出則為可以建立之交談識別碼。
- **CipherSuite**：加密套件列表(附錄 A)。
- **Compression Method**：壓縮方式列表。

2. **伺服器端認證及金鑰交換** (Server Authentication and Key Exchange)

在此步驟中，首先由伺服器端傳送 certificate 訊息，並將伺服器

端之憑證送出，此憑證為符合 X.509 規格的單一憑證或憑證鏈(如果伺服器端是使用匿名的 Diffie-Hellman 方式，則不需要傳送此訊息)。接著送出 server_key_exchange 的訊息，但如果在以下兩種情況下，就可以不需送出這個訊息：(1)伺服器端已經利用固定 Diffie-Hellman 的各項參數來傳送憑證，(2)即將進行 RSA 金鑰交換。以下為 server_key_exchange 訊息內容：

- RSA：伺服器端產生一組臨時的 RSA 金鑰對，並在 server_key_exchange 訊息中放入公開金鑰參數(指數跟模數)，目的是稍後讓用戶端用以加密前置主秘密(pre-master secret)用。
- Anonymouse Diffie-Hellman：訊息內容為伺服器端的公開 Diffie-Hellman 參數(質數與原根)，再加上伺服器端的 Diffie-Hellman 公開金鑰所組成。
- Fixed Diffie-Hellman：訊息內容是由三個提供給匿名的 Diffie-Hellman 所使用的參數，以及這三個參數的簽章所組成。
- Fortezza：送出用戶端的 Fortezza 參數。

接著，如果伺服器端是一個非匿名的伺服器(不是使用匿名 Diffie_Hellman 的伺服器)可能需要從用戶端獲得一個憑證，便會送出 certificate_request 訊息，此訊息包含兩個參數：憑證類型、憑證中心，憑證類型參數指定了所使用的公開金鑰演算法與憑證的用途，憑證中

心參數則是一個列表，用來記錄可接受的不同憑證中心。最後，伺服器端收到 certificate 訊息，再送出 server_hello_done 的訊息，表示伺服器端的這個階段已完成。

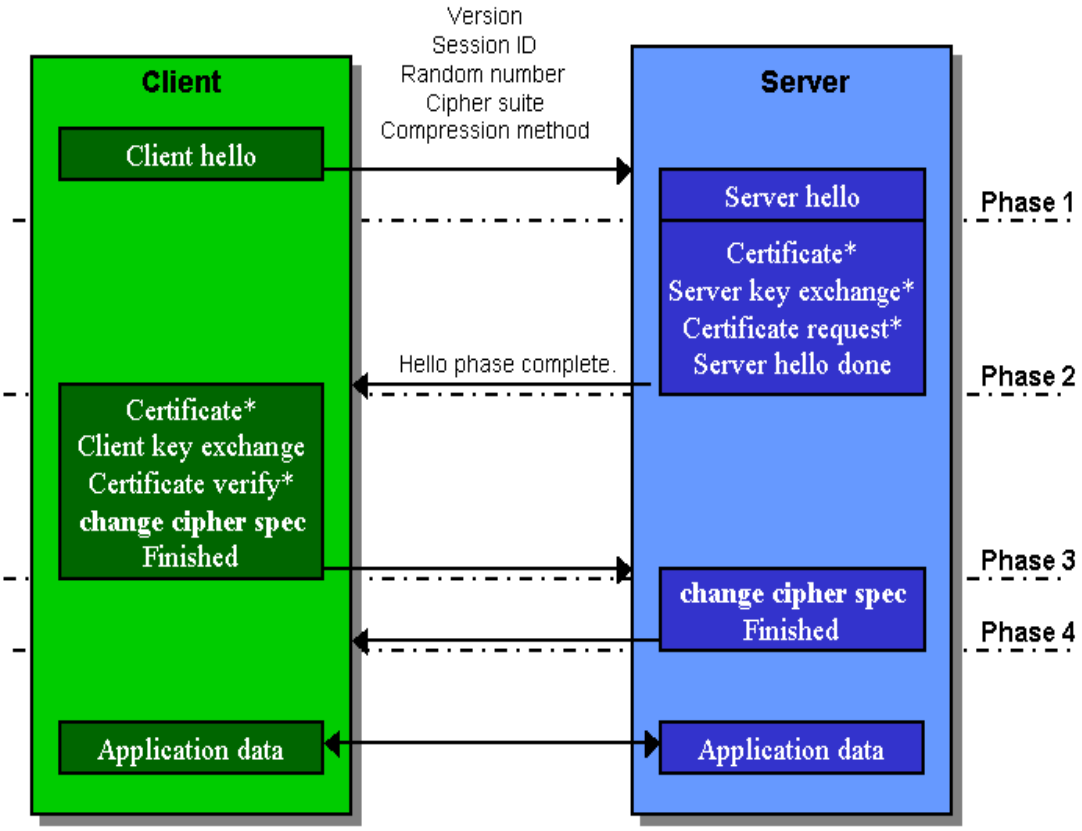

圖 2-9　SSL 交握協定流程圖

3. **用戶端認證及金鑰交換** (Client Authentication and Key Exchange)

在此步驟中，若用戶端接收到伺服器端傳送之 certificate_request **訊息，則傳送** certificate **訊息，並將用戶端的憑證送出，同樣的，此憑證為符合** X.509 **規格的單一憑證或憑證鏈。接著送出** client_key_exchange **的訊息，這個訊息是在此步驟用戶端必定要傳送的訊息，訊息內容則依金鑰交換的形式而定：**

- RSA：用戶端產生一個 48 位元組的前置主秘密(pre-master secret)，並且以伺服器端的公開金鑰(由伺服器端的憑證中取出)來加密，或是利用 server key exchange 訊息中臨時的 RSA 金鑰對來加密。
- Anonymouse Diffie-Hellman：送出用戶端的公開 Diffie-Hellman 參數。
- Fixed Diffie-Hellman：用戶端的公開 Diffie-Hellman 參數被包在送出的訊息中，所以這個訊息並未含有任何的資訊。
- Fortezza：送出用戶端的 Fortezza 參數。

最後，用戶端會送出一個 certificate_verfity 訊息，讓伺服器端藉此來驗證用戶端的憑證。這個訊息必須在用戶端已經送出任何一個具有簽章的憑證的情況下才會送出。這個訊息會根據之前所送出的訊息來簽署一個雜湊值。

```
CertificateVerfi.signature.md5_hash=
MD5(master_secret||pad_2||MD5(handshake_messages||
        master_secret||pad_1))
        Certificate.signature.sha_hash=
SHA(master_secret||pad_2||SHA(handshake_messages||
        master_secret||pad_1))
```

其中的 pad_1 跟 pad_2 是先前 MAC 中所定義的數值。

Handshake_messages 泛指自 client_hello 訊息之後所傳送的所有交握

訊息，但不包含 client_hello **這個訊息。**master **則是算出的數值，在之後會討論。如果使用者的私密金鑰採用** DSS **的話，那麼就用來加密** SHA-1 **的雜湊值。如果使用者的私密金鑰採用** RSA **的話，那麼就用來加密** MD5 **與** SHA-1 **雜湊值得到合併值。此一目的在驗證用戶端憑證中的私密金鑰的所有權，就算有人誤用或盜用用戶端的憑證資訊，也無法傳送這個訊息。**

4. **完成**(finished)

此步驟在建立一個安全連結。用戶端送出 change_cipher_spec **的訊息後，以前面步驟協議完成的加密演算法、金鑰及秘密資訊來對** finished **訊息做訊息封裝的動作，傳送至伺服器端，由伺服器端驗證此訊息，如果驗證成功，表示前述的金鑰交換及認證過程正確無誤。驗證無誤後，由伺服器端同樣送出** finished **訊息給用戶端做驗證。**finished **訊息內容包含兩個雜湊值：**

```
MD5(master_secret||pad_2||
MD5(handshake_messages||Sender||master_secret||pad_1))
SHA(master_secret||pad_2||
SHA(handshake_messages||Sender||master_secret||pad_1))
```

在交握協定的四個步驟中，以雙方協議的安全資訊，來產生 pre_master_secret(**如採取** RSA **做金鑰交換，此值為用戶端產生；如採取** Diffie-Hellman **做金鑰交換，此值為用戶端與伺服器端共同產生**)**，**

再由用戶端與伺服器端雙方各別計算出主秘密(master_secret)，再藉由計算出之主秘密導出此 SSL 連結所用來保護資料安全的金鑰。

```
master_secret=MD5(pre_master_secret||SHA(‘ A’ ||pre_master_secret||
                    ClientHello.random||ServerHello.random))||
              MD5(pre_master_secret||SHA(‘ BB’ ||pre_master_secret||
                    ClientHello.random||ServerHello.random))||
              MD5(pre_master_secret||SHA(‘ CCC’ ||pre_master_secret||
                    ClientHello.random||ServerHello.random))
```

此處的ClientHello.random及ServerHello.random是在一開始雙方的 hello 訊息交換中所採用的臨時亂數。主秘密產生後，雙方便可利用主秘密來產生交談金鑰(session key)。

```
key_block=MD5(master_secret||SHA(‘ A’ ||master_secret||
                    ServerHello.random||ClientHello.random))||
              MD5(master_secret||SHA(‘ BB’ ||master_secret||
                    ServerHello.random||ClientHello.random))||
              MD5(master_secret||SHA(‘ CCC’ ||master_secret||
                    ServerHello.random||ClientHello.random))||…
```

這個步驟一直持續到產生足夠的輸出為止，其結果便是一個虛擬的亂數函數。我們可以將主秘密視為這個虛擬的亂數函數的種子(seed)，而用戶端與伺服器端的亂數可被視為增加破解困難度的醃製劑(salt)。

上述之步驟，便是通訊雙方在 SSL 協定下建立安全連線的交握協定，如果用戶端欲跟伺服器端繼續先前的連結(先前之連線可能因時

間過長或網路壅塞等原因造成斷線)，則此時進行要求接續交談的交握協定跟上述之步驟有所差異(圖 2-10)。用戶端欲接續 SSL 交談，在交握協定的步驟一中，由用戶端送出 Client_hello 的訊息給伺服器端，所傳送的訊息格式與前面所介紹的相同；不同的是，Session ID 的內容為欲接續之交談識別碼。伺服器端收到後送回 Server_hello 的訊息，若同意用戶端接續交談，則回送可以建立連線之交談識別碼，若不同意接續交談，則送出連線失敗的警告訊息。

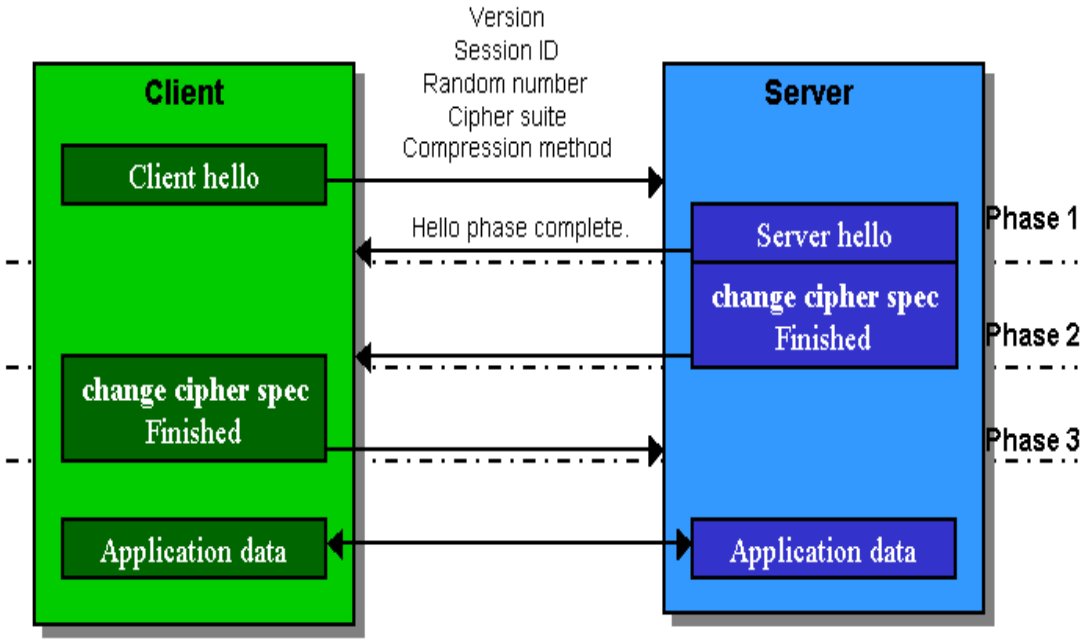

圖 2-10　SSL 交握協定流程圖(接續交談)

三、紀錄協定(Record Protocol)

紀錄協定主要負責的工作，是將欲傳輸之不固定長度的資料訊息，經過一連串的分割/組合(Fragmentation/Assembling)、壓縮/解壓縮

(Compression/ Decompression)、加/解密(Encryption/Decryption)等處理，再傳送至上層(或下層)。當使用者欲傳送資料時，資料經由應用層傳送至紀錄協定層，經過分割(fragmented)、壓縮(compressed)、加入訊息鑑別碼(MAC)、加密(encrypted)之後，加上 SSL 紀錄標頭(SSL record header)，再傳送至傳輸層(圖 2-11)；

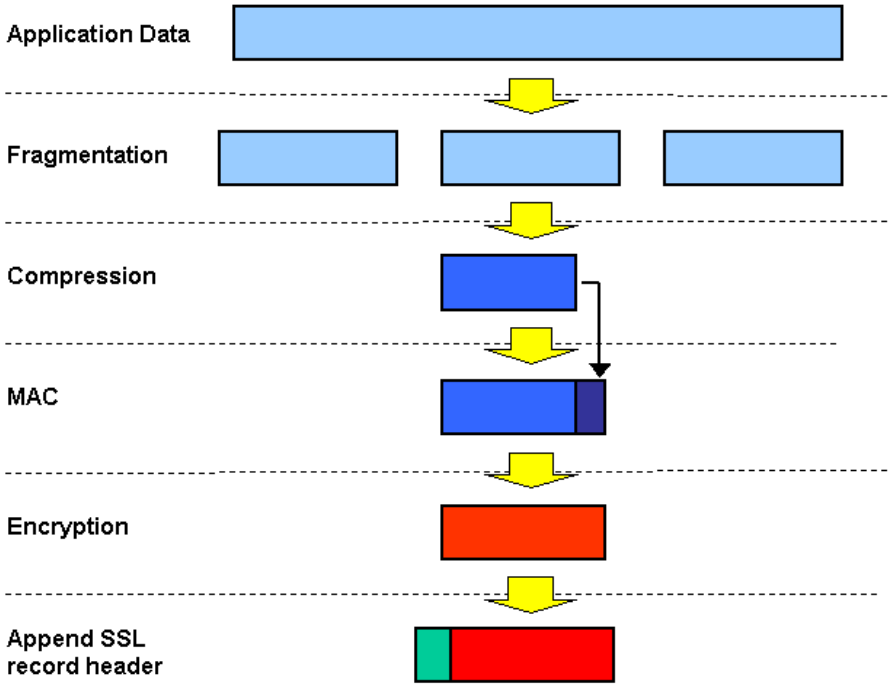

圖 2-11 SSL 紀錄協定之資料封裝步驟

反之，當使用者接收資料時，資料經由傳輸層送至紀錄協定層，經過解密(decrypted)、驗證訊息鑑別碼(verified)、解壓縮(decompressed)、組合(reassembled)後，再將資料送至應用層。紀錄協定針對 SSL 連結提供兩大服務：

- 保密性(Confidentiality)：在 SSL 交握協定中，由連結的兩端協

議出共享的秘密金鑰；以此金鑰來對傳送資料做加密，以達到保密性。

- **訊息完整性(Message Integrity)：在** SSL **交握協定中，由連結的兩端協議出共享的秘密金鑰；以此金鑰來對傳送資料加入訊息鑑別碼**(MAC)**，以達到訊息完整性。**

上述 SSL **紀錄協定中，對於欲傳送/接收的資料會經過一連串的封裝處理步驟，針對傳送之資料的處理過程如下(接收之資料則為反向動作)：**

1. **分割** (Fragmentation)

將應用資料分割成為 SSLPlaintext records**，其區塊大小不可大於** 2^{14} **位元組**(16,384 bytes)**。分割後區塊內容包括上層資料型態、**SSL **版本、分割後明文長度及分割後明文。**

2. **壓縮** (Compression)

將 SSLPlaintext **壓縮成為** SSLCompressed records**。壓縮後區塊內容包括上層資料型態、**SSL **版本、壓縮後資料長度及壓縮後資料，壓縮後長度不得超過未壓縮之區塊** 1024 **位元組，也就是壓縮後之區塊長度不得超過** 2^{14}+1024 **位元組。**

3. **訊息鑑別碼**(Message Authentication Code, MAC)

為了計算已壓縮資料的訊息鑑別碼，必須使用一把共享的秘密金

鑰，其計算過程如下：

```
hash(MAC_write_secret||pad_2||
hash(MAC_write_secret ||pad_1||seq_num||
SSLCompressed.type||SSLCompressed.length||
SSLCompressed.fragment)
```

4. **加密** (Encryption)

加密演算法包含串流加密法(stream cipher)**、區塊加密法**(block cipher)**，其中區塊加密法亦提供** CBC **操作模式。加密後區塊內容包括上層資料型態、**SSL **版本、加密後密文長度及加密後密文，加密後之區塊長度不得超過** 2^{14}+2048 **位元組。**

SSL **記錄區塊經過上述步驟的封裝後, 會在資料前端加上標頭(圖** 2-12)**，再傳送給下一層傳輸協定處理；標頭其欄位如下：**

- **內文類型(content type)：** 8 **位元值，指出用來處理被包裝區塊的較高層協定。**
- **主要版本(major version)：**8 **位元值，指出所使用的** SSL **的主要版本；對** SSL v3 **而言，其值為** 3**。**
- **次要版本(minor version)：**8 **位元值，指出所使用的** SSL **的次要版本；對** SSL v3 **而言，其值為** 0**。**

■ **壓縮後長度(compressed length)：**16 **位元值，以位元組為單位的本文區塊的長度，或是經過壓縮的區塊長度。最大值為** 2^{14}+2048。

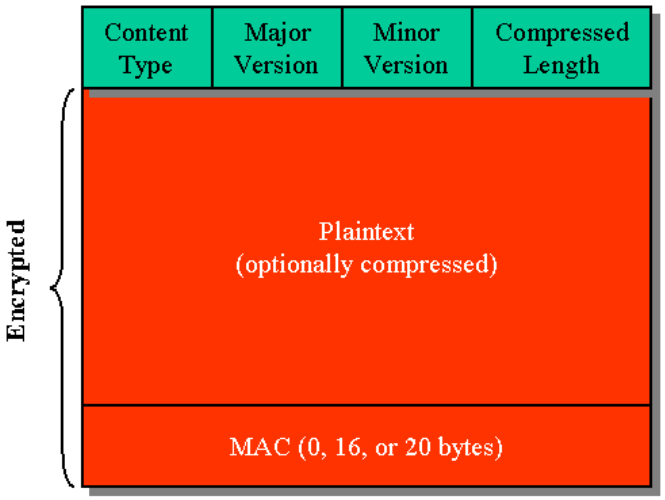

圖 2-12　SSL **紀錄協定之資料格式**

四、變更加密規格協定(Change Cipher Spec Protocol)

在 SSL **的紀錄協定層上，有三個特別的協定，分別為變更加密規格協定**(Change Cipher Spec Protocol)**、警告協定**(Alert Protocol)**與交握協定**(Handshake Protocol)。

變更加密規格協定主要在提供 SSL **紀錄協定使用，內容包含** 1 **個位元組的訊息，訊息值為** 1**，主要在提供** SSL **紀錄協定在判別同一個連結中是否變更加密套件**(cipher suite)。

五、警告協定(Alert Protocol)

警告協定是用來傳遞與 SSL **相關的警告訊息到另一連結的節**

點。此協定所傳送的訊息也經過壓縮、加密。警告協定內的每個訊息都由兩個位元所組成，第一個位元組代表警訊的類別，分為以下兩種：

- **關閉警告** (Closure alerts)
- **錯誤警告** (Error alerts)

警告協定主要負責下列工作：

- **當錯誤警告被偵測到時，偵測端將傳送訊息通知另一端。**
- **若傳送或接收到的訊息是** fatal alert message**，雙方會立即關閉此連線。**
- **若連線失敗(關閉)，發出關閉警訊，並要求伺服器跟客戶端都需要把交談識別碼(**session id**)、金鑰(**keys**)及相關的秘密資訊統統移除。**

第四節 TLS 協定

前一章節介紹之 SSL v3.0 **安全協定，目的在提供通訊雙方的資料加密與部分的認證功能，目前幾乎所有的** web **伺服器都提供了這項安全功能，應該是目前網路上最常見的安全協定。此協定在** 1999 **年為國際標準單位** IETF **所接受後，更名為** TLS(Transport Layer Security)**，成為新的標準** RFC 2246 [15]**。由於** TLS **可視為** SSL **之改版，對於** TLS **之研究則著重於其與** SSL **之差異比較，以下章節將對**

TLS 與 SSL 之差異性做詳細的探討。

一、紀錄格式

TLS 紀錄格式與 SSL 格式相同(圖 2-12)，其標頭的各項欄位也具有相同的意義，唯一不同的是版本的編號。目前 TLS 的主要版本編號為 3，次要編號為 1。

二、訊息驗證碼

TLS 與 SSL v3 在訊息驗證碼的規格上有兩點不同：所採用的演算法與訊息驗證碼計算範圍。TLS 使用的是 RFC 2104 所定義的 HMAC 演算法。HMAC 的定義如下：

$$HMAC_K = H[(K^+ \oplus opad) \| (H(K^+ \oplus ipad) \| M)]$$

其中

H=雜湊函數(對 TLS 而言，可能是 MD5 或 SHA-1)

M=*HMAC* 的輸入訊息

K=在左邊填上一些零元的秘密金鑰，使其長度符合雜湊值的區塊長度(對 MD5 與 SHA-1 而言，區塊長度為 512 位元)

ipad=00110110(十六進位的 36)重複 64 次（512 位元）

opad=01011100(十六進位的 5C) 重複 64 次（512 位元）

SSL v3 使用相同的演算法，但是填充位元組只與秘密金鑰做連結

的動作，而非直接與秘密金鑰做 XOR 運算，而 TLS 在訊息驗證碼的計算圍繞在下列欄位上：

```
HMAC_hash(MAC_write_secret,seq_num||TLSCompressed.type||
TLSCompressed.version||TLSCompressed.length||
TLSCompressed.fragment))
```

TLS 的訊息驗證碼的計算涵蓋了 SSL v3 的計算所需要的所有欄位，另外還要加上 TLSCompressed.version 這個欄位，用以表示所使用的協定版本編號。

三、虛擬亂數函數

TLS 使用一個稱為 PRF 的虛擬亂數函數將密文擴充成區塊資料，以便於金鑰的產生或確認。使用虛擬亂數函數的主要目的在利用較小的共享秘密來產生較長的資料，這種方式能防止有心人士對雜湊函數或訊息驗證碼的攻擊。PRF 是以下列的資料擴充函數為基礎：

```
P_hash(secret,seed) =HMAC_hash(secret,A(1)||seed)||
HMAC_hash(secret,A(2)||seed)||
HMAC_hash(secret,A(3)||seed)||…
```

其中

A(0)=seed
A(i)=HMAC_hash(secret,A(i-1))

資料擴充函數會用到 HMAC 演算法，可能是 MD5 或是 SHA-1，且 P_hash 可以被重複計算，直到產生足夠數量的資料為止。為了使

PRF 達到安全，它使用兩個雜湊演算法，並且只要這兩個雜湊演算法是安全的，我們就可以保證 PRF 是安全的。

PRF(secret, label, seed)=P_MD5(S1, label || seed)⊕
P_SHA-1(S2,label||seed)

PRF 有三個輸入：一個秘密的數值、一個確認用的標籤，以及一個亂數種子；並可以產生任意長度的輸出。首先，將秘密數值切成兩部分(S1、S2)，再對每一部份進行 P_hash 運算，一半用 MD5，另外一半用 SHA。最後，對這兩個結果進行 XOR 運算，便可以產生輸出。為了讓兩邊的結果長度相同，通常會重複較多次的 MD5。

四、金鑰產生

在 SSL v3 中雜湊值的計算包含了主秘密與填充值，但事實上這些額外的欄位並沒有增加任何安全性。與 SSL v3 的完成訊息相同，TLS 的完成訊息也是在建立主秘密、先前的交握訊息，以及用來辨認雙方身份的標籤的雜湊值上，TLS 的 certificate_verify 中，只針對 handshake_message 來產生 MD5 以及 SHA-1 的雜湊值，所以在計算上跟 SSL v3 不太一樣。

PRF(master_secret,finished_label,MD5(handshake_messages) ||
SHA(handshake_messages)

TLS 的前置主秘密也是利用雜湊函數以及通訊雙方在 hello 訊息交換中所採用的臨時亂數 ClientHello.random 及 ServerHello.random 所計算出來的，但計算方式與 SSL v3 不太相同：

```
Master_secret=PRF(pre_master_secret,”master secret”,
                        ClientHello.random||ServerHello.random)
```

這個演算法會持續計算出 48 個位元組的虛擬亂數為止。而金鑰(MAC 秘密金鑰、通訊加密金鑰、起始向量金鑰)則透過以下方式計算：

```
key_block=PRF(master_secret,”key expansion”,
SecurityParameters.server_random ||
SecurityParameters.client_random)
```

此程序一直持續到產生足夠的輸出為止，與 SSL v3 相同，key_block 也是一個主秘密、用戶端與伺服器端亂數所產生的函數值。

五、TLS 與 SSL v3 之其他差異

1. 加密套件

TLS 與 SSL v3 的加密套件的不同點如下：

- 金鑰交換：除了 Fortezza 之外，TLS 支援 SSL v3 所有的金鑰交換技術。
- 對稱式加密演算法：除了 Fortezza 之外，TLS 支援 SSL v3 所有

的對稱式加密演算法。

2. **警告訊息**

TLS **支援** SSL v3 **中所定義的所有警告訊息(除了** no_certificate **訊息以外)，並額外定義了一些警告訊息。**

3. **用戶端憑證**

TLS **可以利用** certificate_request **訊息來要求下列憑證：**rsa_sign、dss_sign、rsa_fixed_dh、dss_fixed_dh**，這些都定義在** SSL v3 **中。除此之外，**SSL v3 **包含** rsa_ephemeral_dh、dss_ephemeral_dh、fortezza_kea**，這些是** TLS **所不支援的。**

第參章 目錄服務

第一節 X.500

目錄服務可視為一種特別的資料庫，而此資料庫存放的資料內容主要在提供分散式系統中的其他應用程式或服務使用；目錄服務提供兩個主要的機制：儲存資料的資料庫、存取控制協定。相較於一般的資料庫系統，目錄服務主要工作是提供應用程式對資料庫內部資料快速的搜尋、讀取、瀏覽，而非著重在管理各資料間的關係，由於需要快速的搜尋及讀取資料並回應到應用程式，目錄服務採取的搜尋方式是全有全無制(all-or-nothing)，搜尋到該筆資料時便直接回應給用戶端，或回應無此資料的訊息。因此，無需要太複雜的資料庫管理系統，也不需要像 SQL 這樣完整的資料庫查詢語言。

針對目錄服務，ITU 於提出定義架構在 ISO/OSI 環境上一套完整且標準的目錄存取協定(Directory Access Protocol，DAP)，是為 X.500。X.500 將儲存在目錄資料庫內的每一個單元定義為一個物件(object)，而每一個物件皆屬於一個物件類別(object class)，這些物件類別則由一系列的屬性(attribute)所組成。例如圖 3-1，目錄資料庫中有五筆資料，每一筆資料即可視為一物件；物件會有物件名稱，例如 UserA，同時，此物件會隸屬一個物件類別，例如 Person，並包含一

些屬性：例如這個物件的名稱－CN(common name)及其他有關這個物件的相關資訊之屬性資料。用來識別這些物件的名稱的屬性，稱為 Relative Distinguished Name，RDN，也就是這個物件的名稱的屬性，在圖 3-3 的例子中，UserA 的識別名稱屬性為 CN=UserA。在整個目錄資料庫中，RDN 並不是唯一的；對於物件唯一的識別名稱為 Distinguish Name，DN，由該物件的 RDN 與該物件繼承的所有父物件的 RDN 所組成，例如，物件 UserA 的 RDN 為 CN=UserA，其上一層的父物件 CSIE 的 RDN 為 OU=CSIE，依此類推至根目錄之物件 TW，其 RDN 為 C=TW，因此，物件 UserA 的 DN 為"CN=UserA, OU=CSIE, O=THU, C=TW"。

X.500 雖定義出完整的目錄服務規格，但在實作上卻因整體過於龐大繁複而難以在現行的網路環境下實行。在 1990 年初，美國密西根大學根據 X.500 規格重新改良，提出一套可以在現行的 TCP/IP 環境下實作的版本，稱為「輕量級目錄存取協定」(Lightweight Directory Access Protocol，LDAP)，1993 年 7 月發表 LDAPv1，ITU 接受公佈為 RFC1487，成為網際網路上公開的標準；之後又於 1995 年 3 月、1997 年 12 月先後提出 LDAPv2[14]、LDAPv3[15]，發表為 RFC 1777、RFC 2551，在本章第二節中將對 LDAP 做進一步的介紹。

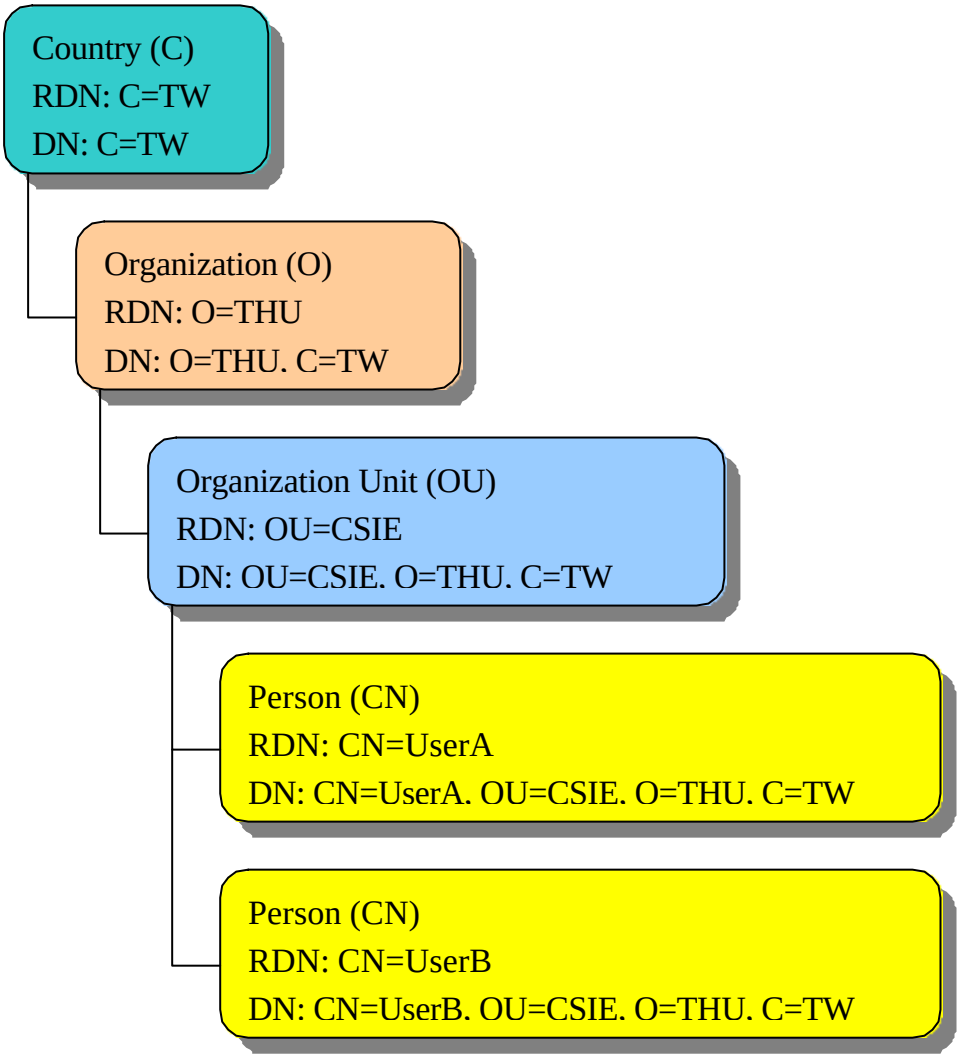

圖 3-1 目錄服務

第二節 LDAP 協定

美國密西根大學根據 X.500 **精簡後提出的較為精簡目錄存取協定，**LDAP**，主要目的是能在** TCP/IP **上實作出目錄服務系統，並簡化用戶端與** X.500 **目錄伺服器**(DSA)**之間的協定；也就是說利用** LDAP **來當做用戶端與** X.500 **目錄伺服器之間的中介閘道**(gateway)**。因此，**LDAP **伺服器提供用戶端標準的目錄服務，同時具有與** X.500 **目錄伺**

服器溝通的能力(圖 3-2)。

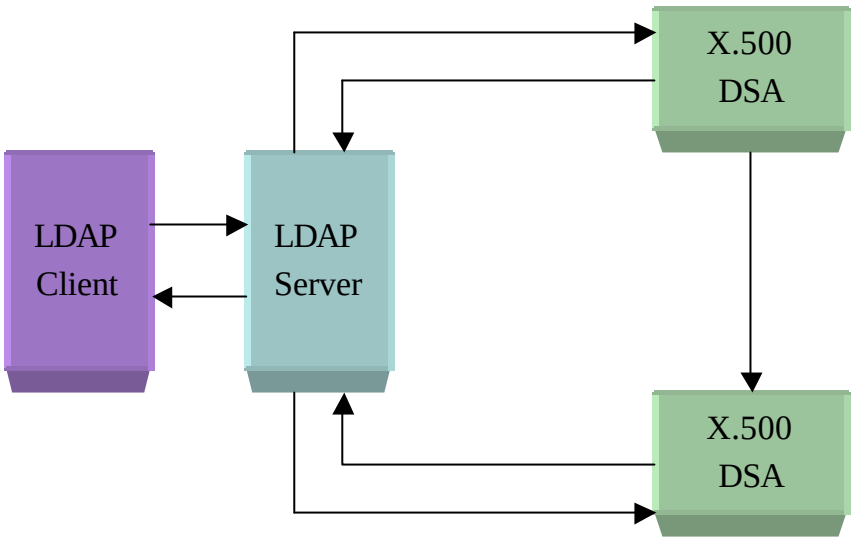

圖 3-2　LDAP 與 X.500 關係圖

一、協定模型(Protocol Model)

LDAP 的協定模型(圖 3-3)，在定義出 TCP/IP 的網路環境中，用戶端與伺服器端間，一連串的服務要求與回應之訊息的交換過程。一個標準的 LDAP 連線服務流程如下：

1. 用戶端向伺服器端發送連線要求(BindRequest)，要求建立連線。
2. 伺服器端回應給用戶端連線回應(BindResponse)，連線建立或錯誤訊息。
3. 用戶端發送操作要求(OpertionRequest)給伺服器端，
4. 伺服器端接收到操作要求後，對 LDBM(LDAP Database Management) 執行用戶端的操作服務。

5. 伺服器端將操作的結果回傳給用戶端，若是操作發生錯誤，則回傳錯誤訊息。

6. 用戶端向伺服器端發送終止連線要求(UnbindRequest)，終止 LDAP 連線。

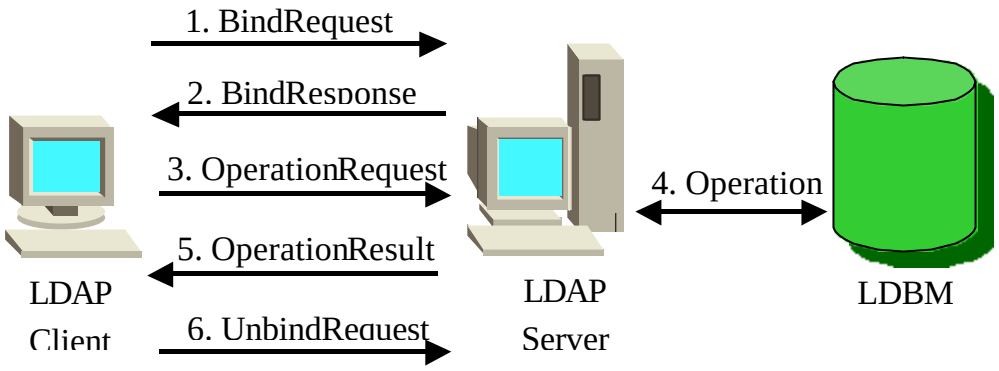

圖 3-3　LDAP 協定模型

二、資料模型(Data Model)

由於 LDAP 是根據 X.500 而制定，所定義的資料模型之資訊也相容於 X.500 定義的資料模型。LDAP 伺服器所提供的目錄服務，需要有儲存庫(repository)來存放資料，實際應用上會設計一個資料庫來負責儲存這些資料，稱為 LDBM(LDAP Database Management)。LDAP 的資料模型的組成元素可分為下列幾項：

- DIT：為了提供 LDAP 快速存取資料的特性，LDBM 的資料結構為階層性樹狀結構(hierarchical tree-like structure)，而非用一般關聯式資料庫。LDBM 的樹狀資料結構，稱為 DIT

(Directory Information Tree)。**在邏輯上，一個以上具從屬關係的** DIT**，可以接合成更大型的** DIT。

- **Entry：**LDAP **的資料項，也是組成** DIT **的基本單元；在實際應用上，一個** entry **可代表一個使用者、一個組織單位、一部電腦、一台伺服器或一個週邊裝置的識別資料。**

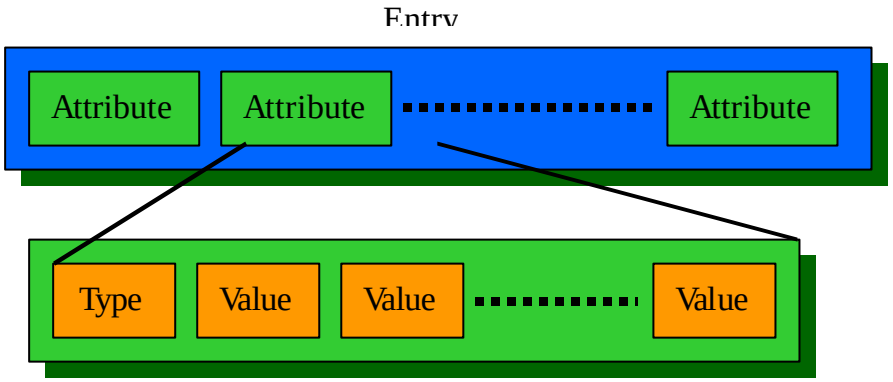

圖 3-4　LDAP **資料模型**

- **屬性(Attribute)：一個** entry **是由一個以上的屬性所組成，每一個屬性包含一個屬性型態**(attribute type)**及一個以上的屬性值**(attribute value)。**圖** 3-5 **所表示的是一個** entry**，此** entry **內的每一行即為一個屬性，例如代表此** entry **的識別名稱** DN**，其中屬性型態為** dn**，屬性值有四個：**cn=UserA、ou=CSIE、o=THU、c=TW**，此** entry **的** RDN **即為** cn=UserA。
- **物件類別(Object Class)：物件類別為一種特殊的屬性，主要目的在將一般屬性做分類歸納。物件類別屬性型態為** objectClass。**每一個** entry **必須包含一個以上的物件類別，這**

個特殊屬性的屬性值是可以更改的，但是屬性型態 objectClass 是不容許變更。

- **Schema**：schema 為屬性型態與物件類別定義的集合，主要在定義 LDAP 伺服器可允許使用的屬性及其規格資訊。LDAP 伺服器可根據 schema 的內容來做搜尋資料的過濾規則，或規範目錄資料庫內的 entry 可擁有的屬性資料。

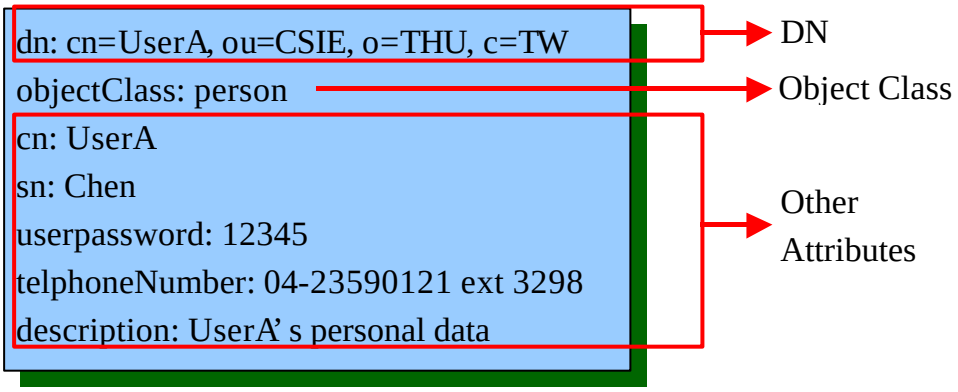

圖 3-5 LDAP 屬性資料

由於 LDAP 的目錄資料庫為階層式樹狀結構，一般而言，這樣的結構設計可以對應地理關係或國家組織這類具有從屬關係的資料。目錄資料庫中的每一個資料項都有其唯一的識別名稱 DN，DN 的命名原則有下列兩種：

- **傳統命名**(Traditional Naming)：傳統命名原則是根據 X.500 對 DN 命名原則修訂[19]，如圖 3-6 所示，是依照國家組織的從屬關係建立目錄資料項，以右邊的樹狀結構來看，最上層的資料項為國家—台灣(c=TW)，其次為省份—台灣省(st=

Taiwan)，再下一層為組織–東海大學(o=THU)，再下一層為組織內的單位 –資訊系(ou=CSIE)、數學系(ou=MATH)，最底層為人(cn=UserA)。

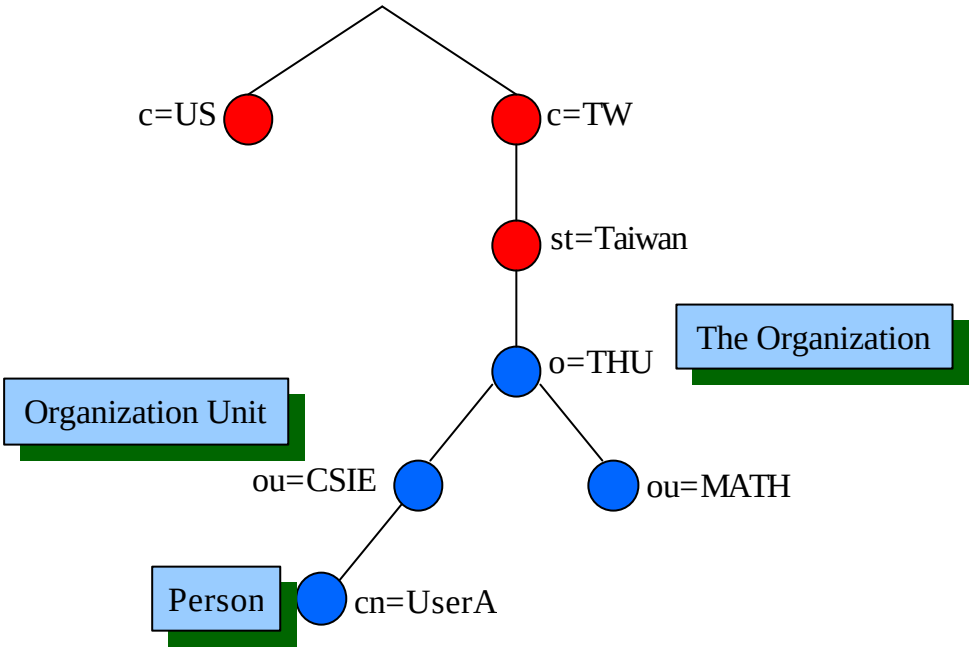

圖 3-6 LDAP 傳統命名方式

- 網際網路命名(Internet Naming)：以目前 LDAP 應用在網際網路的情況看來，傳統的命名原則不全然可以提供較適宜的命名方式，為了實際的需求及方便性，根據 TCP/IP 網域域名為基礎而訂定了新的命名原則[20][21]。如圖 3-7 所示，使用新的屬性值 dc 來代表 TCP/IP 域名，中間的子樹的根為 dc=edu.tw，再其次為 dc=thu，此一資料項的 DN 為 dc=thu, dc=edu.tw，對應域名為 thu.edu.tw，再往下層命名原則則同傳統命名原則。

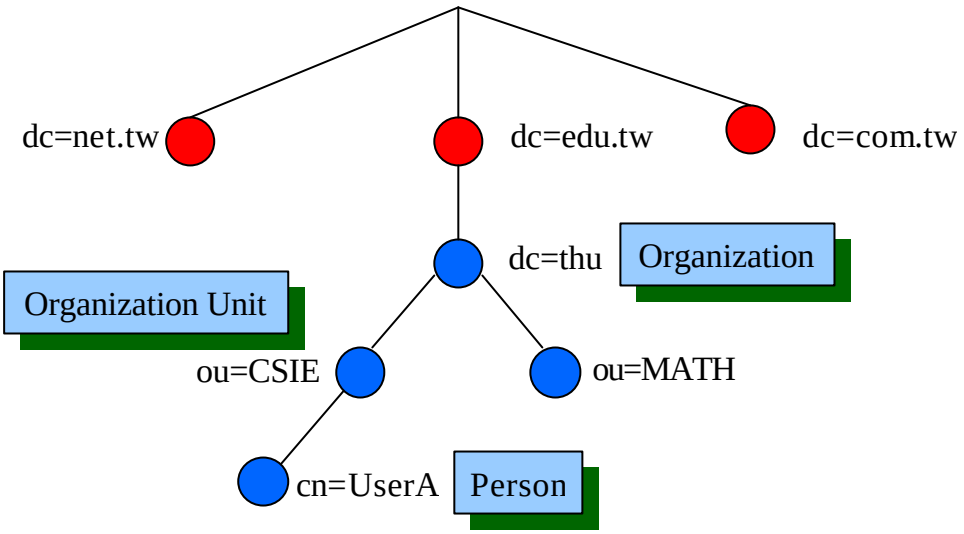

圖 3-7 LDAP 網際網路命名方式

三、協定操作功能(Operation)

LDAP 定義出前述的兩個模型：協定模型、資料模型，在 TCP/IP 的網路環境中，用戶端與 LDAP 伺服器端之間對於 LDAP 內部資料之存取操作，依 LDAPv3 所定義出來的操作功能，按功能性可分為三大類型：

- **查詢**(Interrogation)：提供用戶端對 LDAP 伺服器端的資料進行比對、搜尋。
- **異動**(Modification)：提供用戶端對 LDAP 伺服器端的資料新增、修改等管理功能。
- **鑑別**(Authentication)：提供用戶端與伺服器端之間連線的建立、終止，以及建立連線需要的鑑別機制。

1. **查詢**(Interrogation)

- **Search：允許用戶端向** LDAP **伺服器端提出搜尋資料的要求。**
- **Compare：允許用戶端對** LDAP **目錄資料庫中的一個** entry **的內容做比較。。**

2. **異動**(Modification)
   - **Add：允許用戶端向** LDAP **伺服器端要求新增一筆資料之操作服務。**
   - **Delete：允許用戶端向** LDAP **伺服器端要求刪除一筆資料之操作服務。**
   - **Modify：允許用戶端向** LDAP **伺服器端要求修改一筆資料內容之操作服務。**
   - **Modify DN：允許用戶端向** LDAP **伺服器端要求修改一筆資料的** DN**，或是將一整個子樹下所有的資料移至目錄的另一個新的位置。**
3. **鑑別**(Authentication)
   - **Bind：允許用戶端與伺服器端之間交換鑑別資訊。**
   - **UnBind：終止** LDAP **協定的連線。**
   - **Abandon：允許用戶端在操作意外時提出中斷交談的要求。**

在 LDAP **的身分辨別機制建立連線時，也就是用戶端向** LDAP **伺服器端提出** BindRequest **後，伺服器端對用戶端進行身分鑑別。鑑**

別的機制包含密碼鑑別(password authentication)、Kerberos[16]、SASL(Simple Authentication and Secure Layer)[17]。

四、LDAP 伺服器運作機制

LDAPv3 **中對於整個協定的定義，已經具備完整的協定、資料模型，以及對整個目錄服務提供操作功能，在目前實際應用上，單獨的**(stand-along)LDAP **伺服器足以提供完整的目錄服務，甚至不需要跟後端的** DSA **作資料的轉換。依照實際環境的需求，**LDAP **伺服器的運作機制，可分為下列四種：**

1. **本地目錄服務**(Local Directory Service)

此類型的伺服器只提供目錄服務給伺服器所屬的網域之用戶端，伺服器並不會與其他的目錄伺服器進行溝通。因此，如果是剛開始或只打算提供本地端的服務且不想和其他伺服器溝通，則此類型服務較適合，因為任何時候都可以容易地變更為其他類型的伺服器。

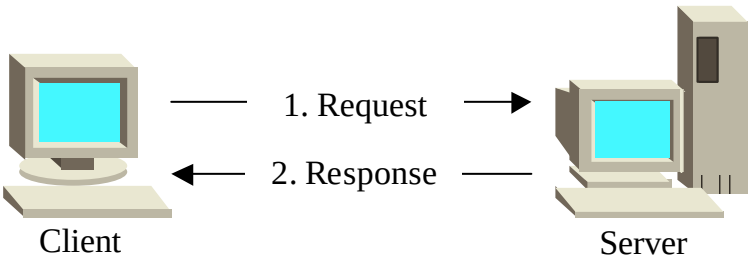

圖 3-8 **本地目錄服務**

2. **具參照功能之本地目錄服務**(Local Directory Service with

Referrals)

此類型的伺服器可以提供目錄服務給該伺服器的網域用戶以及設定並且傳送位於本地網域以外的上層伺服器的參照值。當用戶端提出服務要求，LDAP **伺服器回應結果，若** LDAP **伺服器本身無此資料，則會回應給用戶端參照的訊息，讓用戶端可向所指引的其他** LDAP **伺服器提出服務要求。**

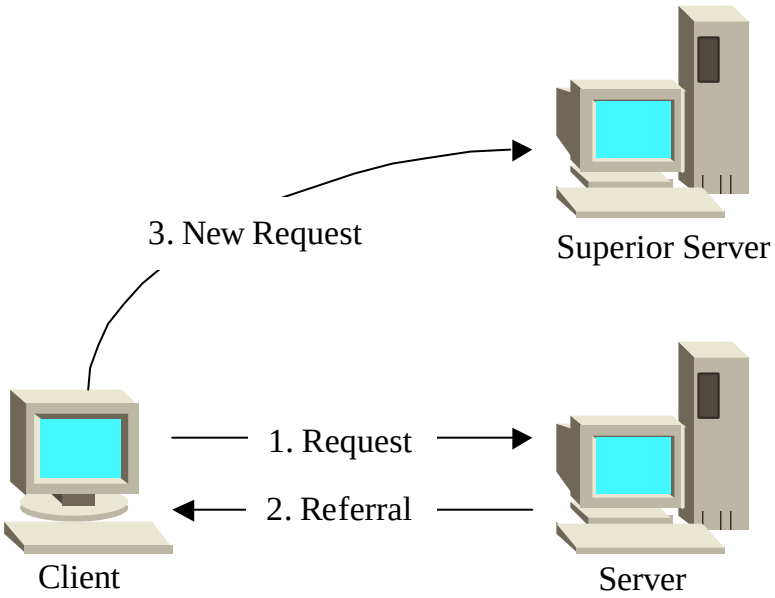

圖 3-9 具參照功能之本地目錄服務

3. **複製目錄服務**(Replicated Directory Service)

伺服器將有異動的資料從某一主要伺服器傳送到一個或多個從屬伺服器. 當單一 LDAP **伺服器無法提供服務所需的可靠性或有效性時，複製目錄服務類型的服務可以和前述兩種服務之一的設定合併使用。**

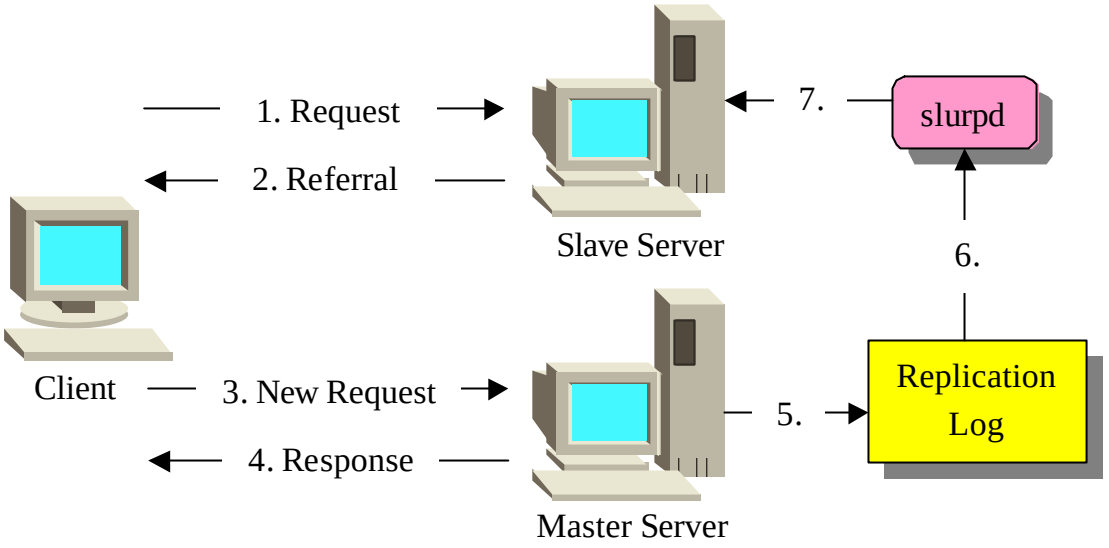

圖 3-10 複製目錄服務

4. **分散式目錄服務**(Distributed Directory Service)

本地服務將被細分為更小的多個服務，各個服務可能會重複並且與上層或下層 LDAP **伺服器結合在一起。**

第肆章 系統規劃與設計

本章將應用第二章、第三章所提出之相關技術，規劃設計一個應用於公開金鑰基礎架構下的網路存取控制機制。

第一節 系統架構及流程

一、系統架構規劃

在公開金鑰基礎架構的環境下，應用 LDAP 來實作出一個具有身分驗證及存取控制的安全環境，此一系統架構的規劃，可分為幾個主要的組成單位(圖 4-1)：

- 使用者(End Entity)：網際網路上的一般使用者。
- 註冊中心(RA, Registration Authority)：憑證中心的前端，負責處理使用者申請憑證前的註冊作業。
- 憑證中心(CA, Certification Authority)：負責發行使用者的憑證及憑證管理之公正第三者。
- LDAP 伺服器：提供目錄服務及存取控制服務之伺服器，負責發行使用者短期的屬性憑證。
- 應用程式伺服器(AS, Application Server)：提供使用者應用程式服務之伺服器，例如 FTP 伺服器、WWW 伺服器等。

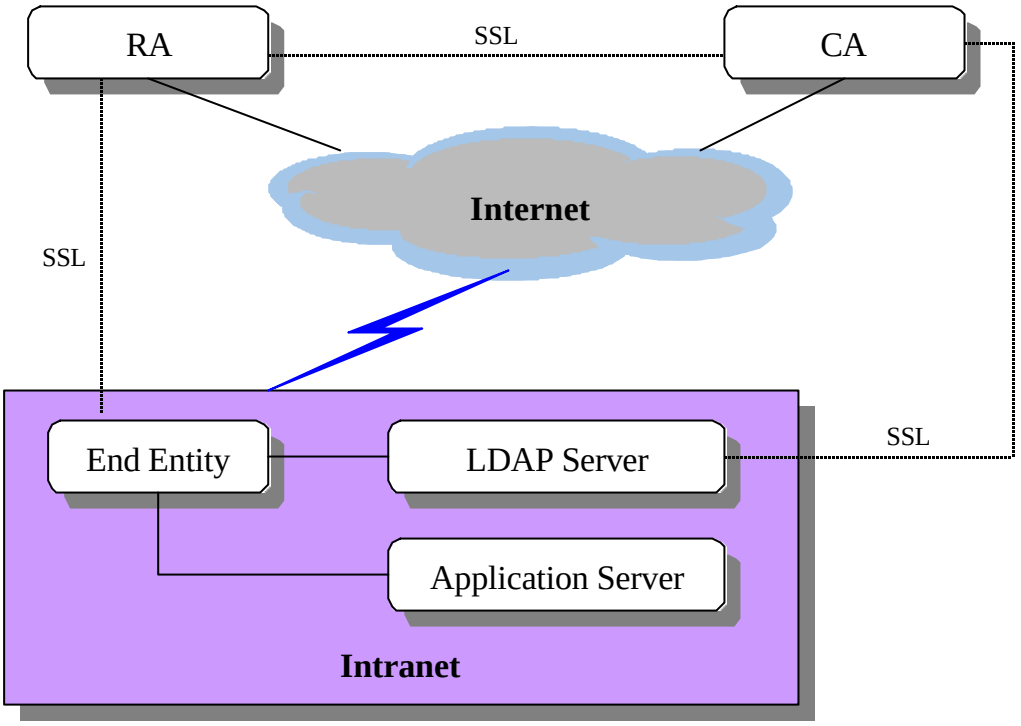

圖 4-1　系統架構圖

二、系統流程規劃

在第一小節規劃出的系統架構下，整個運作機制可分為兩大階段：註冊階段與存取控制階段，以下將針對兩個階段訂定系統主流程(圖 4-2)。

- **註冊階段**(Registration Phrase)：此階段主要在建立使用者之識別資訊、發行使用者憑證並建立使用者權限屬性。

R.1. 使用者 A 向註冊中心 RA 註冊並申請憑證。

R.2. 註冊中心 RA 處理使用者 A 的註冊作業，並向憑證中心提出憑證要求。

R.3. **憑證中心收到憑證要求，發行使用者** A **之憑證，並負責儲存及管理使用者** A **之憑證。**

R.4. **憑證中心與** LDAP **伺服器連線，將使用者** A **之識別資料加入** LDAP **資料庫中。**

R.5. LDAP **伺服器依使用者** A **之存取控制權限，在目錄資料庫中建立使用者** A **的存取權限屬性。**

■ **存取控制階段**(Access Control Phrase)**：此階段主要訂定使用者與網際網路上的應用程式伺服器連線時，所需之驗證機制與存取控制。**

AC.1. **使用者** A **向** LDAP **伺服器提出搜尋使用者** A **識別資料之要求。**

AC.2. LDAP **伺服器處理使用者** A **之服務要求，並於目錄資料庫中搜尋使用者** A **之屬性資料。**

AC.3. LDAP **根據使用者** A **的存取權限屬性，發行使用者** A **之短期的屬性憑證**(Attribute Certificate)**。**

AC.4. **使用者收到屬性憑證後，可以利用屬性憑證向應用程式伺服器登錄連線。**

AC.5. **應用程式伺服器驗證使用者** A **身分後，並以使用者** A **之屬性憑證作為存取控制依據。**

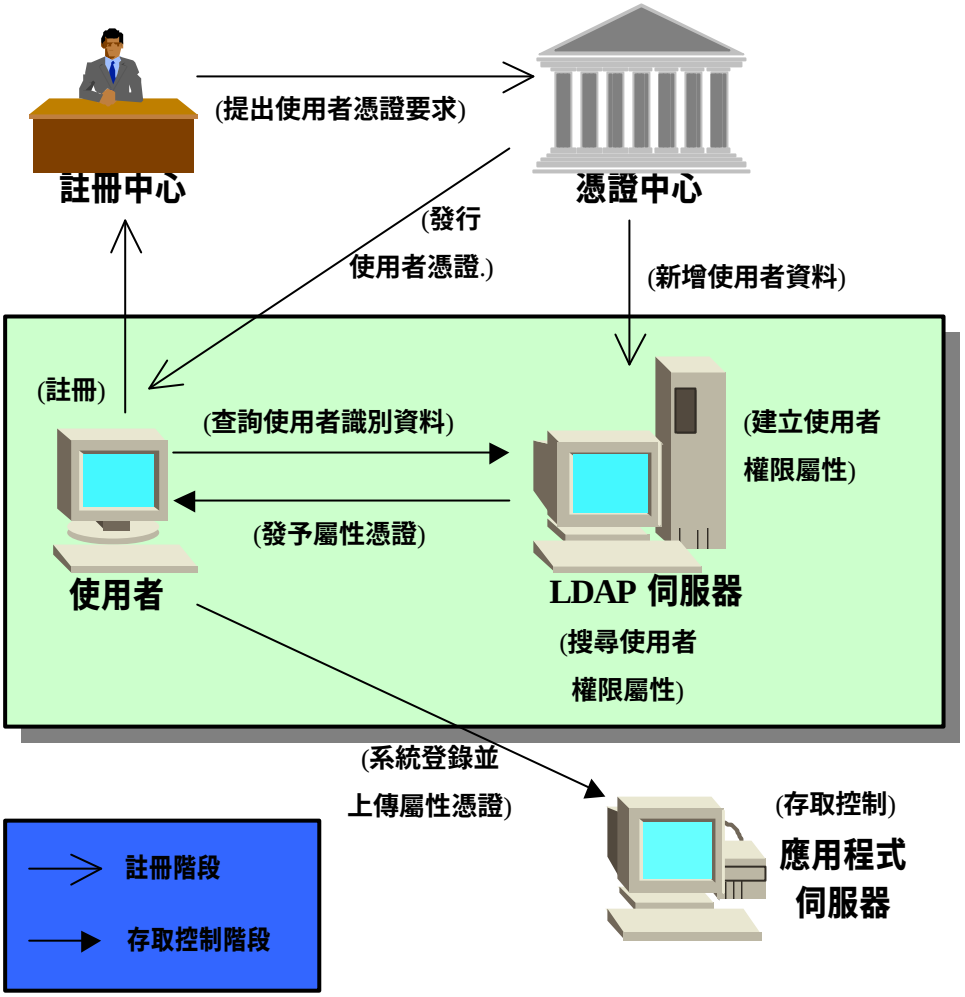

圖 4-2　系統流程

第二節 系統協定

一、註冊協定(Registration Protocol)

註冊階段主要在建立使用者之識別資訊，並發予使用者憑證。註冊協定在定義出註冊階段所需之細部流程與傳輸資料內容，註冊協定之細部規劃如下：

R.1. UserA 向註冊中心 RA 註冊並申請憑證。

R.1.1. UserA 產生金鑰對：私密金鑰 PvK、公開金鑰 PK。

R.1.2. UserA **利用** PvK **對識別碼** ID **做簽章，**SIG_{PVK}**。**

R.1.3. UserA **將** ID、PK、SIG_{PVK} **傳送給** RA**。**

R.2. **註冊中心** RA **處理** UserA **的註冊作業，並向憑證中心提出憑證要求。**

R.2.1. RA **利用** UserA **之** PK **對** SIG_{PVK} **進行驗證。**

R.2.2. RA **驗證成功後，產生** UserA **之憑證要求** CSR_{UserA}**。**

R.2.3. RA **將** ID、PK、CSR_{UserA} **傳送給憑證中心** CA**。**

R.3. **憑證中心收到憑證要求，發行** UserA **之憑證，並負責儲存及管理** UserA **之憑證。**

R.3.1. CA **依收到之** CSR_{UserA} **及** PK **產生** UserA **之憑證，**

$CERT_{UserA}$**。**

R.3.2. CA **發行、儲存及管理** $CERT_{UserA}$**。**

R.4. **憑證中心與** LDAP **伺服器連線，將** UserA **之識別資料加入** LDAP **資料庫中。**

R.4.1. CA **與** LDAP **伺服器連線, 將** UserA **之** ID **傳送給** LDAP **伺服器。**

R.4.2. CA **利用** LDAP **之** ADD **功能在** LDAP **伺服器上建立** UserA **之資料。**

R.5. LDAP 伺服器依 UserA 之存取控制權限，在目錄資料庫中建立 UserA 的存取權限屬性。

R.5.1. LDAP 伺服器將 UserA 之存取權限，利用 LDAP 之 Modify 功能建立 UserA 之權限屬性。

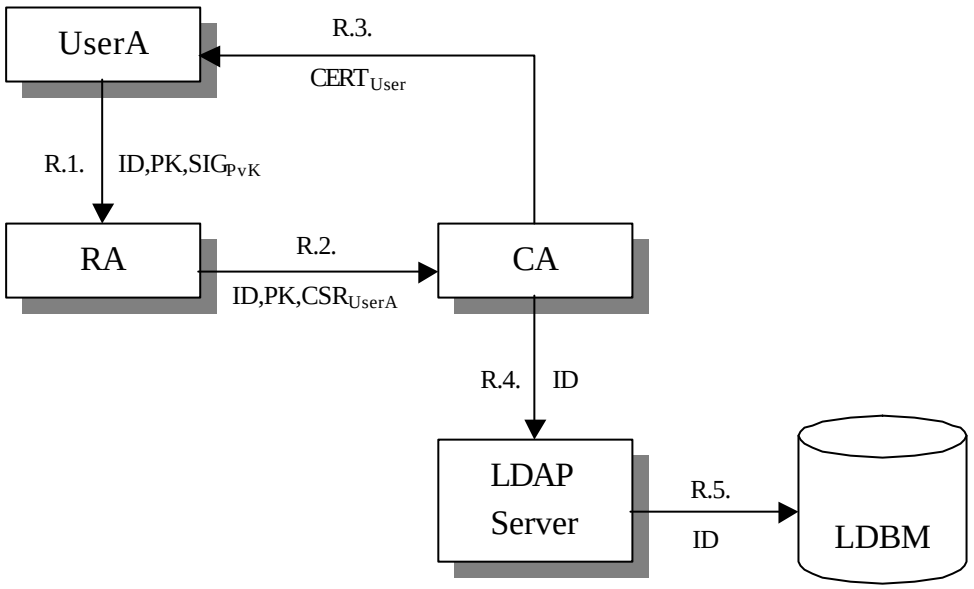

圖 4-3 註冊階段

二、存取控制協定(Access Control Protocol)

存取控制階段主要訂定使用者與網際網路上的應用程式伺服器連線時，所需之驗證機制與存取控制。存取控制協定

AC.1. UserA 向 LDAP 伺服器提出搜尋使用者 A 識別資料之要求

AC.1.1. UserA 利用識別碼 ID、密碼 PW 向 LDAP 伺服器進行連線。

AC.2. LDAP 伺服器處理 UserA 之服務要求，並於目錄資料庫中搜尋 UserA 之屬性資料。

AC.2.1. LDAP 伺服器驗證 UserA 的 ID、PW，並向目錄資料庫 LDBM 查詢 UserA 之資料。

AC.2.2. LDBM 將查詢結果回應給 LDAP 伺服器，內容包含 UserA 的屬性資料 Attributes。

AC.3. LDAP 根據使用者 A 的存取權限屬性，發行使用者 A 之短期的屬性憑證(Attribute Certificate)。

AC.3.1. LDAP 伺服器依照 UserA 之 Attributes，產生 UserA 之屬性憑證，$AttCERT_{UserA}$。

AC.3.2. LDAP 伺服器將 $AttCERT_{UserA}$ 回傳給 UserA。

AC.4. 使用者收到屬性憑證後，可以利用屬性憑證向應用程式伺服器登錄連線。

AC.4.1. UserA 將 ID、PW、$AttCERT_{UserA}$ 傳送至欲登錄之應用伺服器 AS 進行登錄作業。

AC.5. 應用程式伺服器驗證使用者 A 身分後，並以使用者 A 之屬性憑證作為存取控制依據。

AC.5.1. 應用程式伺服器 AS 驗證 ID、PW。

AC.5.2. AS **確認** UserA **身分後，再依** $AttrCERT_{UserA}$ **中所擁有之屬性值核予** UserA **存取控制權限。**

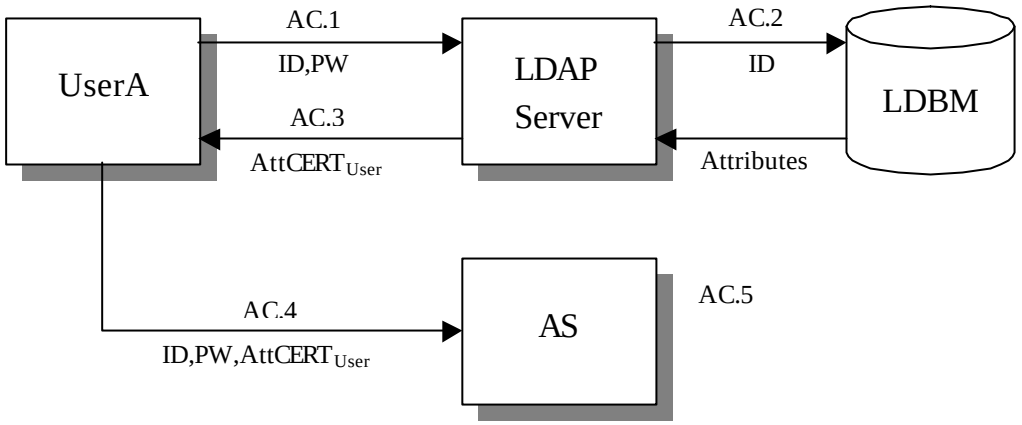

圖 4-3 註冊階段

第伍章 系統實作與應用

第一節 系統規格

1. PKI 環境：

單　　位	主要功能說明
註冊中心	提供使用者註冊、申請憑證
憑證中心	發行憑證、管理憑證

表 5-1　公開金鑰基礎架構環境

2. LDAP 伺服器環境：

	規　格	說　明
主機	AMD 700	
作業系統	Linux Redhat 7.2	
應用程式需求	LDAP 應用程式	LDAP 伺服器架設、管理
	SSL 函式庫	屬性憑證產生程式

表 5-2　LDAP 伺服器環境

3.用戶端環境：

	規　格	說　明
主機	AMD 700	
作業系統	Microsoft Windows XP Professional Edition	
應用程式需求	Web browser	使用者線上申請憑證、要求屬性憑證

表 5-3　用戶端環境

第二節 系統環境

本論文提出之系統架構，主要運用在公開金鑰基礎架構之環境，實作上可利用開放程式碼 OpenSSL 提供之函式庫開發 CA、RA 的環境，包含使用者註冊、產生金鑰對、產生憑證要求，以及憑證之發行、

管理與廢止等相關的應用程式；以上之相關技術已有許多實作的範本可供參考，如國外的 VeriSign、我國的 HiTrust 或政府憑證管理中心(GCA)等憑證中心網站。本論文實作核心部分主要在於 LDAP 之應用與利用屬性憑證來達到存取控制的機制，以下將針對 LDAP 伺服器環境及應用程式伺服器之存取控制詳細說明。

一、LDAP 伺服器

就整個系統架構而言，LDAP 伺服器扮演最核心的角色，系統中的註冊協定與存取控制協定的轉承點也在 LDAP 伺服器。註冊協定在 LDAP 伺服器上建立使用者識別資料與權限屬性，存取控制協定則讓欲使用網際網路服務的使用者，透過 LDAP 伺服器建立具存取控制機制的屬性憑證。在實作上，使用開放原始碼軟體 OpenLDAP 來架設 LDAP 伺服器；此外，OpenLDAP 也提供了用戶端的應用程式及開發應用程式使用之函式庫。本系統利用 OpenLDAP 實作 LDAP 伺服器，其中包含目錄服務資料庫、LDAP 應用程式，以下將作詳細說明。

1. 目錄服務資料庫

目錄服務資料庫之設定分為三部分 (設定檔為 *slapd.conf*)：全域原則(Global Directives)、一般後端原則(General Backend Directives)、一般資料庫原則(General Database Directives)，其主要內容資訊如下：

■ **全域原則**(Global Directives)

全域原則主要在定義 LDAP **伺服器系統環境參數與系統存取權限，**LDAP **伺服器可以同時有一個以上的目錄服務，也就是說會有一個以上的目錄資料庫，而全域原則所定義的存取權限是針對系統中所有目錄資料庫而定。**

```
#######################################################################
# Global Directives
#######################################################################
# 指定 Schema 定義檔
include         /usr/local/etc/openldap/schema/core.schema
include         /usr/local/etc/openldap/schema/cosine.schema
include         /usr/local/etc/openldap/schema/inetorgperson.schema

# 指定提供參照之 LDAP 伺服器
referral        ldap://root.openldap.org

pidfile         /usr/local/var/slapd.pid
argsfile        /usr/local/var/slapd.args

# 載入後端動態模組
# modulepath      /usr/local/libexec/openldap
# moduleload      back_ldap.la
# moduleload      back_ldbm.la
# moduleload      back_passwd.la
# moduleload      back_shell.la

# 系統存取控制
access to * by * read

# 若無設定 ACL，則預設值為
#    Allow read by all
#    rootdn can always write!
```

■ **一般後端原則**(General Backend Directives)

後端原則主要在指定後端資料庫(目錄資料庫)的之型態，後端資料庫型態的設定值可以是 ldbm、shell、passwd **或其他支援的資料庫型態。**

```
#######################################################################
# General Backend Directives
#######################################################################
# backend        ldbm                                    # 指定後端資料庫型態
```

■ **一般資料庫原則**(General Database Directives)

資料庫原則用來定義單一目錄資料庫的相關資訊，包含目錄位置、BDN**、系統管理者** DN **及密碼(明文格式，亦可使用** DES **或** MD5 **加密過後之密文格式)、資料庫存取控制等資訊。雖然全域原則中已經設定存取控制，但** LDAP **伺服器仍以資料庫原則定義之存取控制為優先。**

```
#######################################################################
# General Database Directives
#######################################################################
# ldbm database definitions for thu.edu.tw
database    ldbm                                  # 資料庫名稱
directory   /usr/local/var/openldap-ldbm          # 資料庫存放位置

suffix      "dc=thu,dc=edu.tw"                    # BDN
rootdn      "cn=root,dc=thu,dc=edu.tw"            # 管理者 DN
rootpw      secret                                # 設定管理者密碼
                                                  # (明文)
```

```
# 索引定義
index       cn,mail,uid,surname,givenname                    eq,subinitial

# 目錄資料庫存取控制
#access to attr=userPassword
#                   by  self  write
#                   by  anonymous  auth
#                    by  dn="cn=root,dc=thu,dc=edu.tw" write
#                   by  *  none
```

2. **應用程式**

本系統中，LDAP **伺服器所負責的功能，除了提供** LDAP **目錄服務供憑證中心或使用者存取、查詢外，另外一個重要的功能，就是依照使用者及其權限屬性產生屬性憑證並發給使用者，讓使用者能以屬性憑證作為登錄應用程式伺服器後，應用程式伺服器對於該使用者之存取權限控管。**LDAP **伺服器的應用程式包含了利用** OpenLDAP **函式庫撰寫之伺服器程式、用戶端應用程式，及以** OpenSSL **函式庫開發之屬性憑證產生程式，表列如下。**

LDAP 伺服程式		
名　稱	**功　能**	**說　明**
slapd	LDAP **伺服器程式**	**啟動獨立的** LDAP **伺服器**
slurpd	**複製目錄服務伺服器程式**	**啟動** LDAP **目錄複製**

表 5-4　LDAP **伺服器－**LDAP **伺服程式**

LDAP 用戶端應用程式		
名　稱	功　能	說　明
ldapadd	新增	新增一筆使用者資料項
ldapdelete	刪除	刪除一筆使用者資料項
ldapmodify	修改	修改使用者資料項內之屬性
ldapsearch	搜尋	搜尋符合條件之資料項

表 5-5　LDAP 伺服器－LDAP 用戶端應用程式

屬性憑證產生程式		
名　稱	功　能	說　明
gen_attrcert	屬性憑證產生程式	依使用者之權限屬性，產生屬性憑證。

表 5-6　LDAP 伺服器－屬性憑證產生程式

二、應用程式伺服器

1. **環境需求**

在本論文提出之系統架構中，應用程式伺服器可為網站伺服器、檔案傳輸伺服器、目錄伺服器及其他網路上提供網路服務及網路應用程式之伺服器。

2. **應用程式**

應用程式伺服器除了本身提供之身分驗證機制外，在本系統規劃中，必須有屬性憑證驗證程式，以驗證使用者之屬性憑證，再依照屬性憑證中之權限屬性，核予使用者被授權的存取控制權限。

屬性憑證驗證程式		
名　稱	功　能	說　明
verify_attrcert	屬性憑證驗證程式	驗證使用者之屬性憑證，核予存取控制權限。

表 5-7　應用程式伺服器－屬性憑證驗證程式

第三節 系統應用程式設計

一、開放程式碼研究

1. OpenSSL 程式碼研究

OpenSSL 計畫是一個實作 SSL/TLS 協定的合作計畫，目的在開發出一套商業級且具備完整 SSL/TLS 協定功能的應用程式與函式庫。OpenSSL 計畫主要根據 Eric A. Young 與 Tim J. Hudson 開發之 SSLeay 函式庫為基礎，其中包含 SSL/TLS 協定中所支援的密碼函式、X.509 規格中的憑證/憑證廢止清冊標準；所開發出之軟體皆為開放性原始碼(open source)，程式開發者可自由取得，並從事商業及非商業用途之應用程式開發。

OpenSSL 套件主要分成三大部分：OpenSSL 應用程式、SSL 函式庫、Crypto 函式庫，說明如下

- OpenSSL 應用程式

  由 OpenSSL 函式庫開發之應用程式，完整的功能包含以下幾點：

◆ **金鑰產生，包含** RSA、Diffie-Hellman、DSA。

◆ **產生訊息摘要碼。**

◆ X.509 **憑證、憑證要求、憑證廢止清冊之產生與驗證。**

◆ **文件之加密、解密。**

◆ SSL/TLS **用戶端與伺服器端程式**

◆ S/MIME **格式文件簽章、電子郵件加密。**

■ SSL **函式庫**

提供 SSL **協定之紀錄協定資料結構 交握協定函式 與** SSL **連線建立之相關函式。**

■ Crypto **函式庫**

提供訊息摘要碼(MAC)**、對稱式密碼函式、公開金鑰密碼函式等密碼元件。**

2. OpenLDAP **程式碼研究**

與 OpenSSL **相似，**OpenLDAP **計畫也是一個合作計畫，目的在研究並提供符合** LDAP **協定功能之商業級的開放原始碼套件，讓程式開發者可以自由取得以從事商業與非商業用途應用程式之開發**

OpenLDAP **程式套件包含三大部分：伺服器端伺服程式、用戶端應用程式、應用程式開發函式庫**

■ **伺服器端伺服程式**

主要在提供用戶端之連結與 LDAP 目錄資料庫之資料存取。伺服程式包含獨立的 LDAP 伺服程式 —*slpad*、LDAP 伺服器之目錄複製服務—*slurpd*。

- 用戶端應用程式

  透過用戶端應用程式可與 LDAP 伺服程式進行連線、溝通，並執行 LDAP 目錄資料庫之資料管理功能。

- 應用程式開發函式庫：提供程式開發者使用之 LDAP 函式庫。

二、系統應用程式設計

在本論文之系統中，本章第二節提出系統環境所需之應用程式，實作部份皆能以 OpenSSL、OpenLDAP 之函式庫開發，應用程式流程將在以下作說明。

1. LDAP 伺服器

- *ldapadd* / *ldapmodify* / *ldapdelete*

  憑證中心發行使用者的憑證之後，與 LDAP 伺服器連線並要求建立使用者的資料，LDAP 伺服器以 *ldapadd* 程式新增該使用者之資料。

  程式流程主要是先利用 *ldap_init*( )函式與 LDAP 伺服器程式(slapd deamon)連線，再以 *ldap_start_tls_s*( )建立 SSL 連線，建立連線之後以 *ldap_bind_s*( )函式與 LDAP 資料庫連結，再依

照不同功能呼叫 *ldap_add*( )、*ldap_modify*( )、*ldap_delete*( )等函式進行對資料的新增、修改或刪除的動作，最後以 *ldap_unbind*( ) 函式結束整個連線。

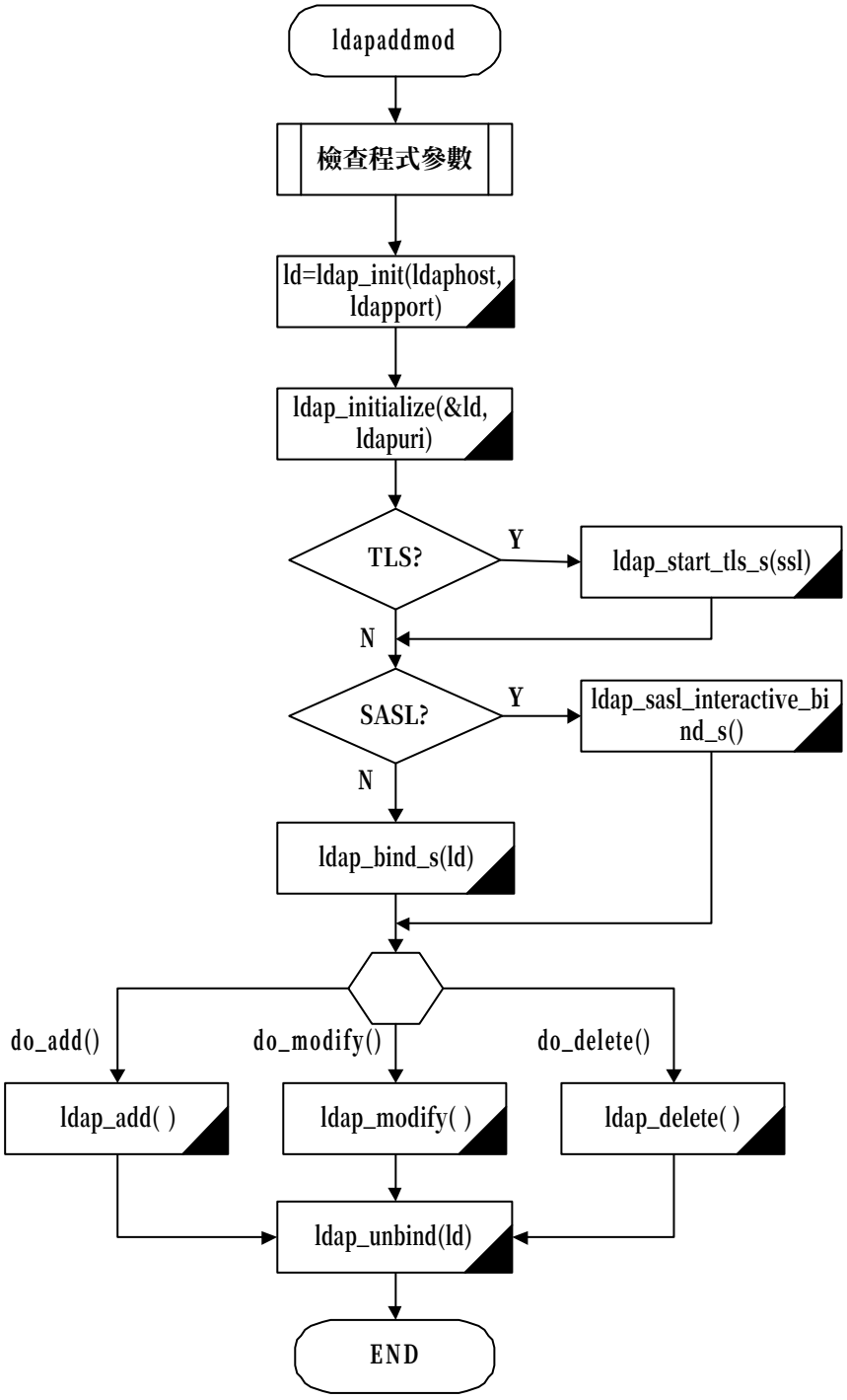

圖 5-1 *ldapadd* / *ldapmodify* / *ldapdelete* **程式流程圖**

■ *ldapsearch*

使用者向 LDAP **伺服器索取屬性憑證時，**LDAP **伺服器必須查詢使用者資料是否儲存在資料庫中，若有此使用者之資料，便依據此使用者擁有之存取控制屬性產生屬性憑證給使用者。查詢程式為** *ldapsearch***，程式的流程與** *ldapsadd* **程式相似，主要差異是搜尋資料項的函式** *do_search*( )**之流程(圖** 5-2**)，結束連線亦是利用** *ldap_ubind*( )**函式。**

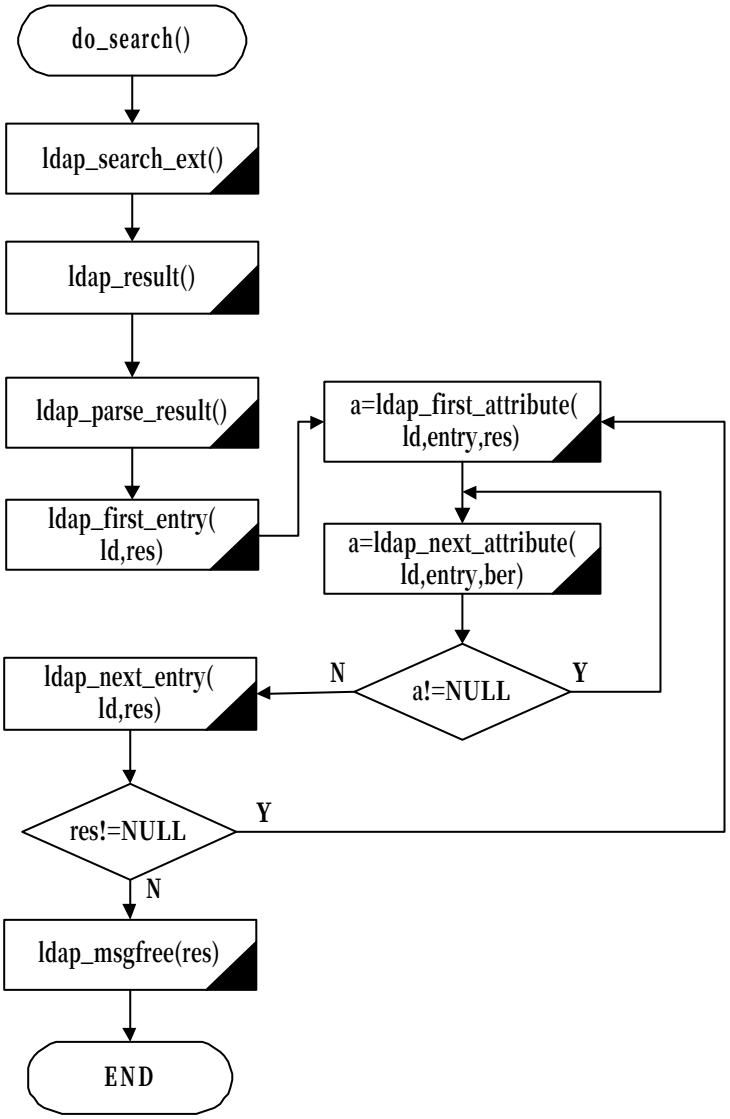

圖 5-2 *ldapsearch* **程式流程圖**

■ *gen_attrcert*

完成註冊階段的使用者，在與應用程式伺服器做連結時，必須先向 LDAP **伺服器索取使用者之屬性憑證，**LDAP **伺服器在接受使用者端的要求後，向後端的目錄資料庫查詢此使用者之資料項，並將該使用者之相關屬性值輸出，再利用** gen_attrcert **程式產生使用者之屬性憑證。在設計此程式前，我們參考憑證發行程式的主要流程(圖** 5-3)**，輸入檔案為憑證簽署要求** (Certificate Sign Request, CSR)**檔，以憑證中心憑證及密鑰將憑證資訊簽章，輸出檔案為憑證檔。**

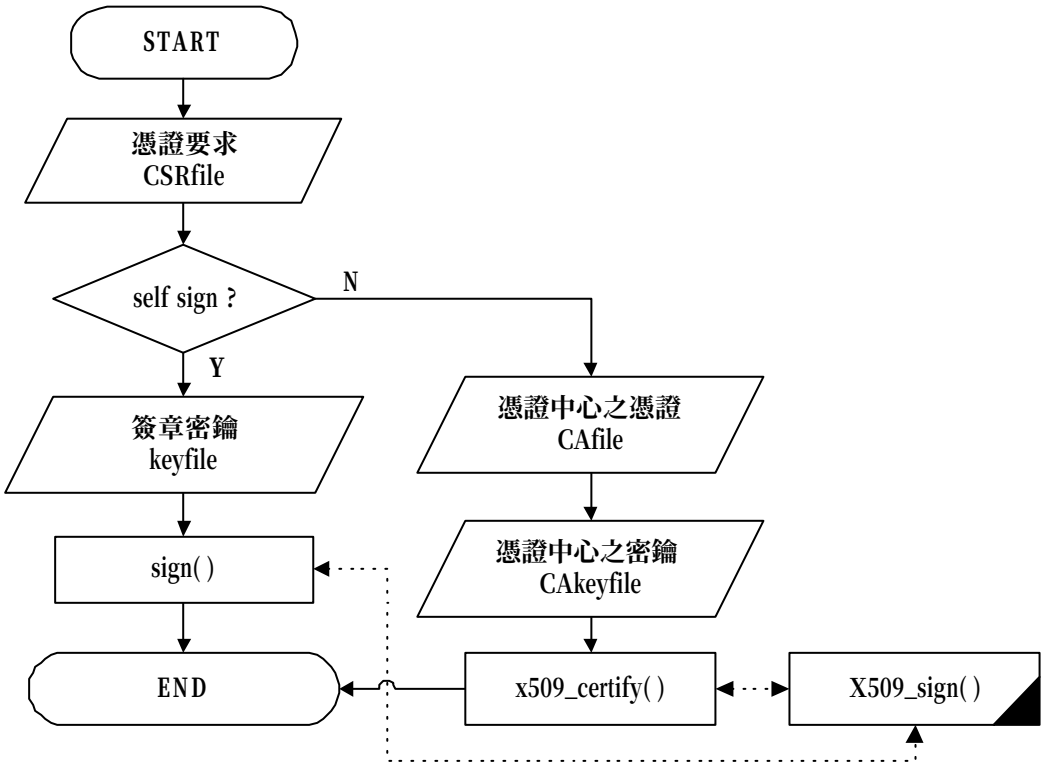

圖 5-3　**憑證發行程式主要流程**

依此流程，我們利用使用者憑證為輸入，取出憑證資訊並將使用者之權限屬性加入憑證擴充部分，再以 LDAP 伺服器之密鑰簽章(圖 5-4)，並輸出屬性憑證。上述之憑證發行程式與屬性憑證發行程式之細部流程如圖 5-5、圖 5-6。

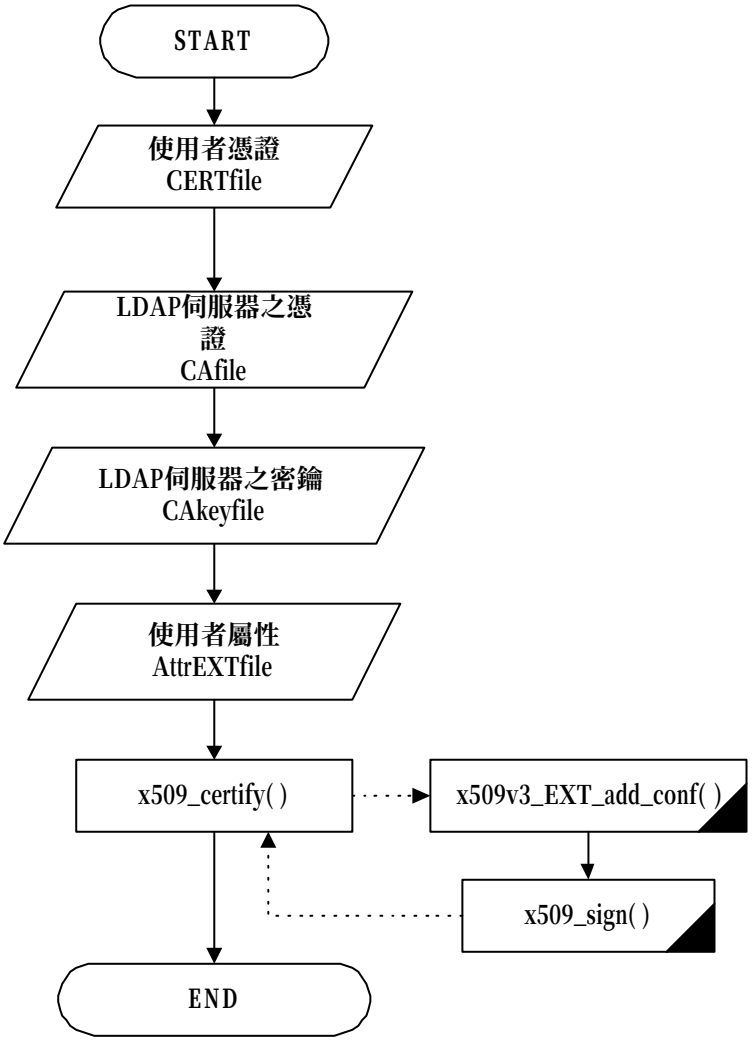

圖 5-4　屬性憑證產生程式主要流程

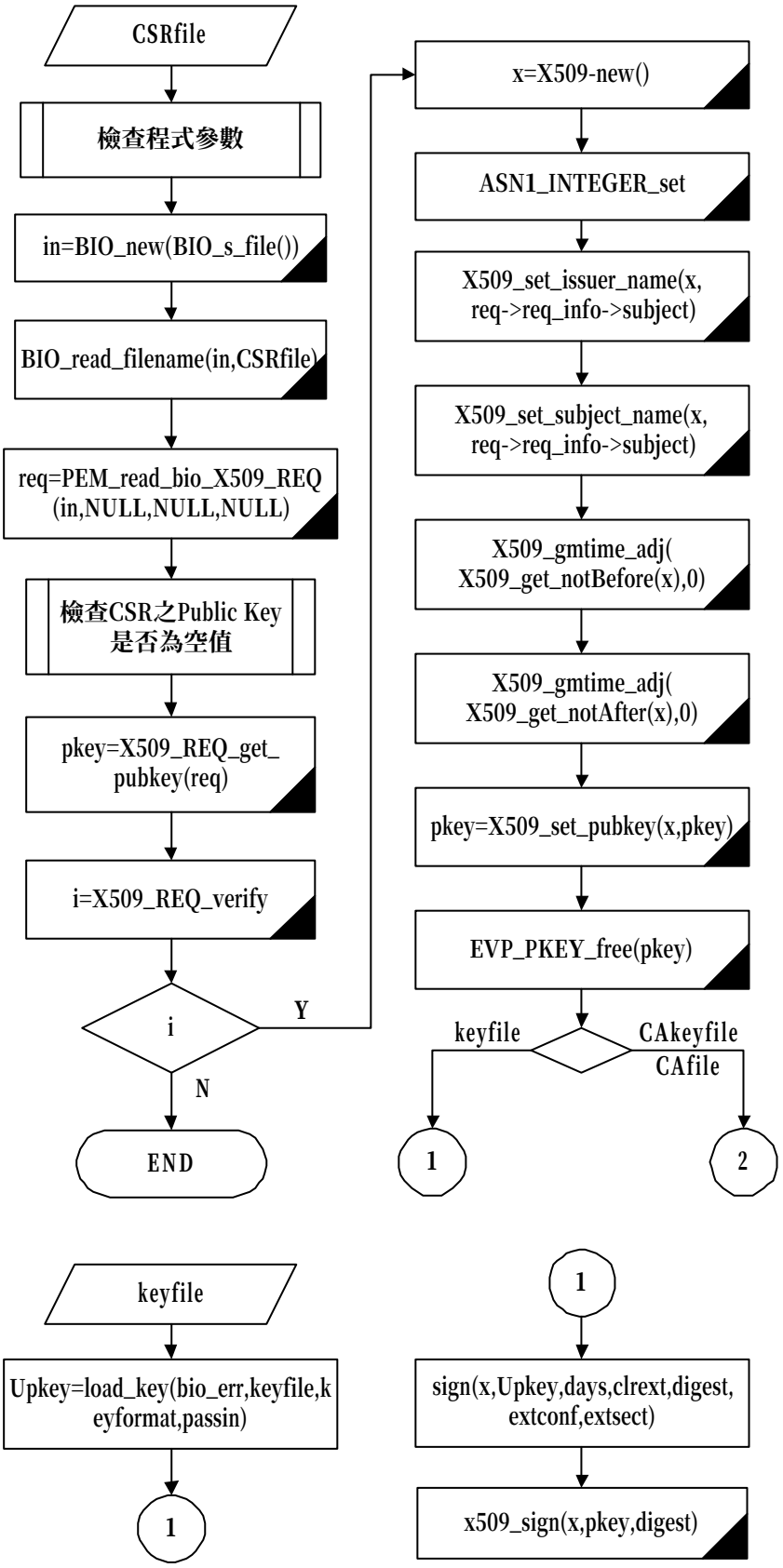

圖 5-5 憑證發行/屬性憑證產生程式細部流程(1/2)

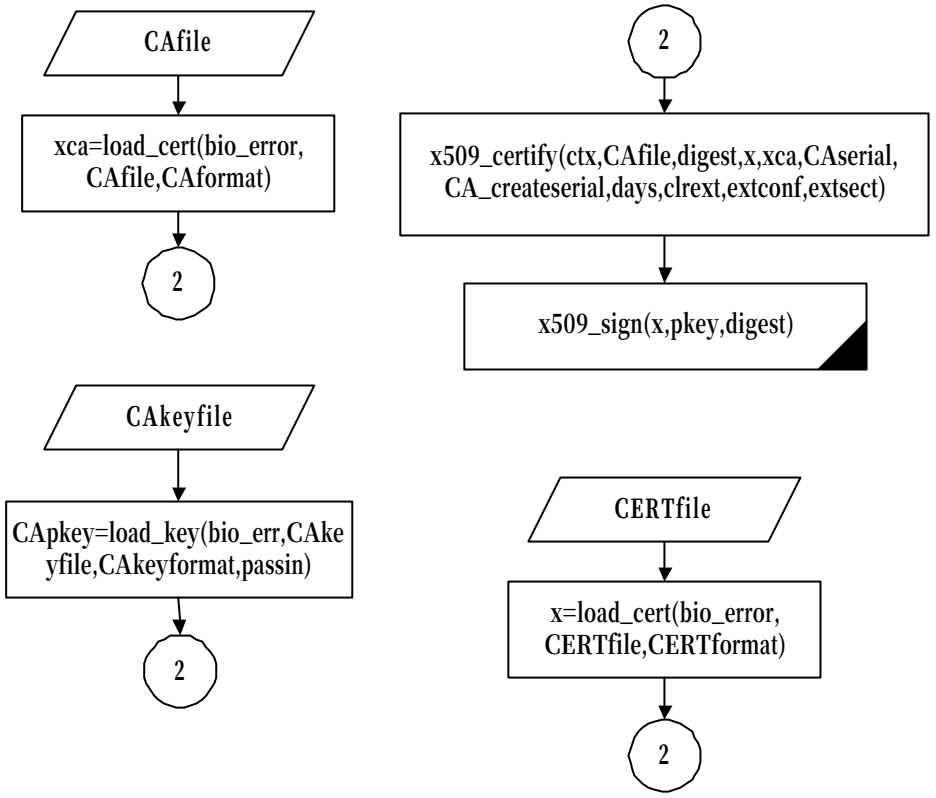

圖 5-6 憑證發行/屬性憑證產生程式細部流程(2/2)

2.**應用程式伺服器**

使用者在取得屬性憑證後，憑此與應用程式伺服器進行連結，使用者上傳識別資料及屬性憑證給應用程式伺服器，伺服器端在接收到使用者之識別資料及屬性憑證後，先進行使用者身份鑑別，若使用者身份正確無誤，再對使用者之屬性憑證做驗證。屬性憑證驗證正確後，再依照屬性憑證內之權限屬性核予使用者適當存取權限。

- *verify_attrcert*

由於本系統設計之屬性憑證由金鑰憑證延伸而來，因此，屬性憑證驗證程式也以金鑰憑證驗證程式流程為依據，驗證程序主要流程如圖 5-7。

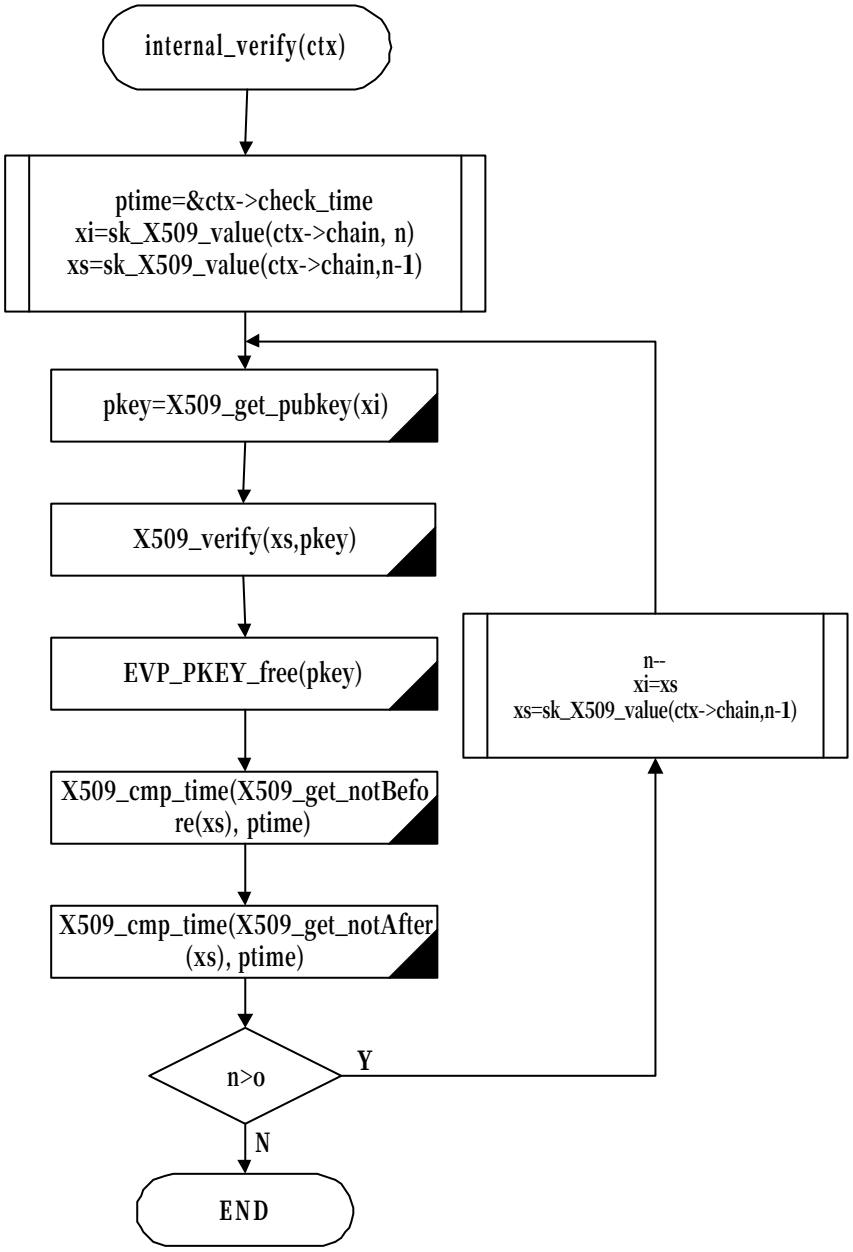

圖 5-7　屬性憑證驗證程式流程

第陸章　結論

第一節 研究成果討論

在資訊技術與網路系統發達的今日，一般民眾、企業商家、公私立機關單位等，對於網際網路上之服務的使用與需求，量與值都快速地在提升；傳統的人類行為模式，一一被搬上四通八達的網際網路殿堂。透過網路繳稅納款、商業交易、資訊互通甚至休閑娛樂都已經逐漸成為網路服務的範疇。伴隨著網路服務帶來的便利性與時效性，網路系統中的安全機制也更顯其重要性，除了對資料傳輸的保密，網路使用者的身份認證更是值得深入研究探討的議題。面對愈來愈龐雜而多樣化的網際網路服務，對於網路使用者身份識別資訊的控管，不再是單純的使用者帳號管理機制所能夠完全掌控。

因此，本論文研究並規劃出在網際網路中更具安全性及有效率的身分認證與存取控制系統，運用公開金鑰基礎架構的環境，整合目錄存取協定來提高其效率及實用性。LDAP 擁有目錄服務的基本要件，階層狀的資料庫結構以達物件分類分級的目的，除了提供快速的資料搜尋功能之外，更有著很大的彈性可在 TCP/IP 架構下，透過複製、參照的功能延伸其目錄服務的範圍。利用 LDAP 的種種優點，配合應用在公開金鑰基礎架構的環境下，更能有效地將其應用範圍，達到更寬廣、更完整的程度。

第二節 未來研究方向

本論文研究主旨在公開金鑰基礎架構環境下，規劃、設計出應用 LDAP 目錄服務在公開金鑰環境的系統，並提出利用開放原始碼 OpenSSL、OpenLDAP 實作本系統之方法。回顧第四章所提出的應用系統之規劃與實作做一探討，在整個系統架構下對於存取控制的機制，尚有一些課題可以更深入探討與研究，其中例如本系統中利用屬性憑證來實作存取控制部分，可再加入以「角色為基礎之存取控制 (Role- based Access Control)」機制，讓整個存取控制機制更臻完整。而在未來的應用範圍，可利用分散式的 LDAP 服務，進一步整合至本論文研究之系統架構。除此之外，本論文可進一步研究配合公開金鑰基礎架構，應用目錄服務來達成憑證中心管理憑證功能，並將本系統模擬真實生活的網路環境進行效能分析，以建構更完整、更有效率的公開金鑰基礎架構之網路環境。

參考文獻

[1] Y.S. Yeh, W.S. Lai, C.J. Cheng, "Applying Lightweight Directory Access Protocol Service on Session Certification Authority," *Computer Networks*, vol 38, Issue 5, April 2002, pp. 675-692.

[2] K. Gutzmann, "Access Control and Session Management in the HTTP Environment," *IEEE Internet Computing* , vol 5 Issue: 1, Jan.-Feb. 2001, pp. 26-35

[3] E. Jamhour, "Distributed Security Management Using LDAP Directories," *Computer Science Society*, 2001. SCCC '01. Proceedings. XXI Internatinal Conference of the Chilean , 2001, pp. 144 -153

[4] C.S. Yang, C.Y Liu, J.H. Chen, C.Y. Sung, "Design and Implementation of Secure Web-based LDAP Management System," *Information Networking*, 2001. Proceedings. 15th International Conference on , 2001, pp. 259 -264

[5] B. Schneier, Applied Cryptography, 2nd Edition, John Wiley & Sons, 1996.

[6] W. Stallings, Network Security Essentials: Applications and Standards, Prentice Hall, 1999.

[7] R. Housley, W. Ford, W. Polk, D. Solo, Internet X.509 Public Key Infrastructure Certificate and CRL Profile, Request for Comments 2459, January 1999.

[8] C. Adams, S. Farrell, Internet X.509 Public Key Infrastructure Certificate Management Protocols, Request for Comments 2510, March 1999.

[9] R. Housley, W. Polk, Internet X.509 Public Key Infrastructure Representation of Key Exchange Algorithm (KEA) Keys in Internet X.509 Public Key Infrastructure Certificates, Request for Comments 2528, March 1999.

[10] S. Chokhani, W. Ford, Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework, Request for Comments 2527, March 1999.

[11] A. Freier, P. Karlton, and P. Kocher. "The SSL Protocol Version 3.0, Internet Draft," March 1996, <http://home.netscape.com/eng/ssl3/ssl-toc.html>

[12] T. Dierks, C. Allen, The TLS Protocol Version 1.0, Request for Comments 2246, January 1999.

[13] R. Oppliger, "SSL and TLS Protocols: How to Address Critical Security Issues," *Computer Security Journal*, vol 16, Num. 1, 2000, pp. 15-37

[14] W. Yeong, T. Howes, S. Kille, Lightweight Directory Access Protocol, Request for Comments 1777, March 1995.

[15] M. Wahl, T.Howes, S. Kille, Lightweight Directory Access Protocol (v3), Request for Comments 2251, December 1997.

[16] B. Neuman, T. Ts' o, "Kerberos: An Authentication Service for Computer Networks," *IEEE Communications Magazine*, vol. 32,

1994, pp. 33-38.

[17] J. Myers, Simple Authentication and Security Layer (SASL), Request for Comments 2222, October 1997.

[18] M. Wahl, A. Coulbeck, T. Howes, S. Kille, Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions, Request for Comments: 2252, December 1997.

[19] M. Wahl, A Summary of the X.500(96) User Schema for use with LDAPv3, Request for Comments 2256, December 1997.

[20] M. Wahl, S. Kille, T. Howes, " Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

[21] S. Kille, M. Wahl, A. Grimstad, R. Huber, S. Sataluri, "Using Domains in LDAP Distinguished Names", RFC 2247, January 1998.

[22] A. Grimstad, R. Huber, S. Sataluri, M. Wahl, Naming Plan for Internet Directory-Enabled Applications, Request for Comments 2377, September 1998.

[23] J. Hodges, R. Morgan, M. Wahl, Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security, Request for Comments: 2830, May 2000.

[24] S. Kaliski, A Layman’ s Guide to a Subset of ASN.1, BER, and DER, RSA Laboratories Technical Note, November 1993.

[25] **陳彥學, 資訊安全理論與實務, 文魁資訊股份有限公司,** 2000.

附錄一：ASN.1 Definition of X.500/X.509

1. ASN.1 Definition of X.509 v3 Certificate

```
Certificate                 ::=     SIGNED { SEQUENCE {
   version              [0]     Version DEFAULT v1,
   serialNumber                 CertificateSerialNumber,
   signature                    AlgorithmIdentifier,
   issuer                       Name,
   validity                     Validity,
   subject                      Name,
   subjectPublicKeyInfo         SubjectPublicKeyInfo,
   issuerUniqueID       [1]     IMPLICIT UniqueIdentifier OPTIONAL,
                                -- If present, version shall be v2 or v3
   subjectUniqueID      [2]     IMPLICIT UniqueIdentifier OPTIONAL,
                                -- If present, version shall be v2 or v3
   extensions           [3]     EXPLICIT Extensions OPTIONAL
                                -- If present, version shall be v3 } }

Version                     ::=     INTEGER  { v1(0), v2(1), v3(2) }

CertificateSerialNumber     ::=     INTEGER

Validity                    ::=     SEQUENCE {
   notBefore        Time,
   notAfter         Time }

Time                        ::=     CHOICE {
   utcTime              UTCTime,
   generalTime          GeneralizedTime }

UniqueIdentifier        ::=     BIT STRING
```

```
SubjectPublicKeyInfo        ::=   SEQUENCE {
   algorithm              AlgorithmIdentifier,
   subjectPublicKey     BIT  STRING  }

Extensions              ::=   SEQUENCE  SIZE  (1..MAX) OF Extension

Extension               ::=    SEQUENCE    {
   extnID                 OBJECT IDENTIFIER,
   critical                BOOLEAN DEFAULT FALSE,
   extnValue              OCTET STRING }
```

2. ASN.1 Definition of X.509 v3 CRL

```
CertificateList                ::=        SIGNED{ SEQUENCE {
   version                     Version  OPTIONAL,
                               -- if present, shall be v2
   signature                    AlgorithmIdentifier,
   issuer                       Name,
   thisUpdate                   Time,
   nextUpdate                  Time  OPTIONAL,
   revokedCertificates          SEQUENCE  OF  SEQUENCE    {
      serialNumber                    CertificateSerialNumber,
      revocationDate                   Time,
      crlEntryExtensions                Extensions OPTIONAL } OPTIONAL,
   crlExtensions               [0]      Extensions OPTIONAL } }
```

3. ASN.1 Definition of Attribute Certificate

```
   AttributeCertificate          ::=            SIGNED{ SEQUENCE {
    version                     AttCerVersion DEFAULT v1,
    holder                      Holder,
    issuer                      AttCertissuer,
```

```
    signature                   AlgorithmIdentifier,
    serialNumber                CertificateSerialNumber,
    attrCerValidityPeriod       AttCerValidityPeriod,
    attributes                  SEQUENCE OF Attribute,
    issuerUniqueID              UniqueIdentifier OPTIONAL
    extensions                  Extensions      OPTIONAL
  }

AttCertVersion      ::= INTEGER {v1(0),v2(1)}
Holder ::=  SEQUENCE
        {
        baseCertificateID    [0] IssuerSerial           OPTIONAL,
              -- the issuer and serial nimber of the holder's Public Key Certificate
        entityName           [1] GeneralNames           OPTIONAL,
        -- the name of the entity or role
        objectDigestInfo     [2] ObjectDigestInfo       OPTIONAL
        -- if present, version must be v2
        --at least one of baseCertificateID, entityName or objectDigestInfo must be
present--}

ObjectDigestInfo          ::=          SEQUENCE{
  digestedObjectType ENUMERATED{
        publicKey            (0),
        publicKeyCert        (1),
        otherObjectType      (2) },
  otherObjectTypeID       OBJECT IDENTIFIER OPTIONAL,
  digestAlgorithm          AlgorithmIdentifier,
  objectDigest             BIT STRING }

AttCertIssuer ::= CHOICE {
   v1Form GeneralNames, -- v1 or v2
   v2Form [0] V2Form    -- v2 only
}
V2Form ::= SEQUENCE {
```

```
issuerName                  GeneralNames  OPTIONAL,
baseCertificateId     [0]   IssuerSerial  OPTIONAL,
objectDigestInfo      [1]   ObjectDigestInfo  OPTIONAL
}
-- At least one component must be present
(WITH COMPONENTS {..., issuerName PRESENT }|
 WITH COMPONENTS {..., baseCertificateId PRESENT }|
 WITH COMPONENTS {..., objectDigestInfo PRESENT })

IssuerSerial ::= SEQUENCE {
   issuer          GeneralNames,
   serial          CertificateSerialNumber,
   issuerUID       UniqueIdentifier  OPTIONAL  }

AttCertValidityPeriod ::= SEQUENCE {
   notBeforeTime        GeneralizedTime,
   notAfterTime         GeneralizedTime }
```

4. ASN.1 Definition of LDAPv3

```
LDAPMessage ::= SEQUENCE {
        messageID           MessageID,
        protocolOp          CHOICE  {
                bindRequest         BindRequest,
                bindResponse        BindResponse,
                unbindRequest       UnbindRequest,
                searchRequest       SearchRequest,
                searchResEntry      SearchResultEntry,
                searchResDone       SearchResultDone,
                searchResRef        SearchResultReference,
                modifyRequest       ModifyRequest,
                modifyResponse      ModifyResponse,
                addRequest          AddRequest,
```

```
                addResponse        AddResponse,
                delRequest         DelRequest,
                delResponse        DelResponse,
                modDNRequest       ModifyDNRequest,
                modDNResponse      ModifyDNResponse,
                compareRequest     CompareRequest,
                compareResponse    CompareResponse,
                abandonRequest     AbandonRequest,
                extendedReq        ExtendedRequest,
                extendedResp       ExtendedResponse },
        controls        [0] Controls OPTIONAL }

Attribute ::= SEQUENCE {
        type      AttributeDescription,
        vals      SET OF AttributeValue }

LDAPResult ::= SEQUENCE {
        resultCode      ENUMERATED {
                        success                      (0),
                        operationsError              (1),
                        protocolError                (2),
                        timeLimitExceeded            (3),
                        sizeLimitExceeded            (4),
                        compareFalse                 (5),
                        compareTrue                  (6),
                        authMethodNotSupported       (7),
                        strongAuthRequired           (8),
                                -- 9 reserved --
                        referral                     (10),   -- new
                        adminLimitExceeded           (11),   -- new
                        unavailableCriticalExtension (12),   -- new
                        confidentialityRequired      (13),   -- new
                        saslBindInProgress           (14),   -- new
                        noSuchAttribute              (16),
```

```
                undefinedAttributeType          (17),
                inappropriateMatching           (18),
                constraintViolation             (19),
                attributeOrValueExists          (20),
                invalidAttributeSyntax          (21),
                                -- 22-31 unused --
                noSuchObject                    (32),
                aliasProblem                    (33),
                invalidDNSyntax                 (34),
                -- 35 reserved for undefined isLeaf --
                aliasDereferencingProblem       (36),
                                -- 37-47 unused --
                inappropriateAuthentication     (48),
                invalidCredentials              (49),
                insufficientAccessRights        (50),
                busy                            (51),
                unavailable                     (52),
                unwillingToPerform              (53),
                loopDetect                      (54),
                                -- 55-63 unused --
                namingViolation                 (64),
                objectClassViolation            (65),
                notAllowedOnNonLeaf             (66),
                notAllowedOnRDN                 (67),
                entryAlreadyExists              (68),
                objectClassModsProhibited       (69),
                                -- 70 reserved for CLDAP --
                affectsMultipleDSAs             (71), -- new
                                -- 72-79 unused --
                other                           (80)  },
                -- 81-90 reserved for APIs --
        matchedDN       LDAPDN,
        errorMessage    LDAPString,
        referral        [3] Referral OPTIONAL }
```

附錄二：LDAP 伺服器環境架設

1. LDAP 伺服器架設

- **軟體來源：**OpenLDAP (http://www.openldap.org)
- **軟體安裝：以下安裝步驟，工作目錄皆在** OpenLDAP **的軟體套件置放之目錄下。**

(1).**將** OpenLDAP **軟體套件解壓縮**

$ *tar zcvf openldap-2.0.22.tgz*

(2).**設定安裝起始檔**

$ *./configure*
($ *./configure --help*)

(3).**建立安裝環境**

$ *make depend*
$ *make*

(4).**安裝軟體與測試**

$ *su root -c 'make install'*
$ *make test*

- **啟動與停止** OpenLDAP **伺服器：伺服器端程式主要** daemon **為** *slapd* **及** *slurpd*，*slapd* **為提供用戶端連結之** daemon，*slurpd* **則為提供其他** LDAP **伺服器連結之** daemon**，在有需要與其他伺服器作複製機制時便可利用次** daemon **進行溝通。**

(1). **啟動** LDAP **伺服器**

$ *su root -c /usr/local/libexec/slapd*

(2). **檢查** LDAP **伺服器是否已經啟動**

$ *ps –ef | grep slapd*

(3). **停止** LDAP **伺服器**

$ *kill –INT `cat /usr/local/var/slapd.pid`*

或

$ *pkill slapd*

2. LDAP **伺服器設定**

在 OpenLDAP **安裝完成後，在未做任何相關設定之前，**LDAP **伺服器是以預設值啟動，此時的目錄資料庫內並無任何資料項。相對於** *slpad* **這個伺服器** daemon**，其設定檔為** *slapd.conf***，目錄位置在***/usr/local/etc/openldap* **底下。下列為** *slapd.conf* **之範例：**

```
#######################################################################
# Global Directives
#######################################################################
include       /usr/local/etc/openldap/schema/core.schema
referral      ldap://root.openldap.org                    # 參照之 LDAP 伺服器

pidfile       /usr/local/var/slapd.pid
argsfile      /usr/local/var/slapd.args

# 載入後端動態模組
# modulepath      /usr/local/libexec/openldap
# moduleload      back_ldap.la
# moduleload      back_ldbm.la
# moduleload      back_passwd.la
# moduleload      back_shell.la
```

```
# 系統存取控制
access to * by * read

# 若無設定 ACL，則預設值為
#   Allow read by all
#   rootdn can always write!

#######################################################################
# General Backend Directives
#######################################################################
# backend        ldbm                                    # 指定後端資料庫型態

#######################################################################
# General Database Directives
#######################################################################
# ldbm database definitions for thu.edu.tw
database    ldbm                                          # 資料庫名稱
directory   /usr/local/var/openldap-ldbm                  # 資料庫存放位置
suffix      "dc=thu,dc=edu.tw"                            # BDN
rootdn      "cn=root,dc=thu,dc=edu.tw"                    # 管理者 DN
rootpw      secret                                        # 設定管理者密碼
                                                          # (明文)
# 索引定義
index  objectClass      eq

# 目錄資料庫存取控制
#access to attr=userPassword
#                   by  self  write
#                   by  anonymous  auth
#                    by  dn="cn=root,dc=thu,dc=edu.tw" write
#                   by  *  none
```

3. **建立** LDAP **目錄資料庫**

OpenLDAP **安裝完成後，**LDAP **的目錄資料庫中並無儲存任何資料，因此要先建立第一筆資料(建立第一個** entry**，也就是系統的**

entry)，此資料代表此 LDAP 伺服器提供服務的識別資料 BDN(Base Distinguished Name)，也就是 DIT 的根節點。當目錄資料庫尚未建立 BDN 的資料時，無論在伺服器端本身或用戶端連線，都無法對此 LDAP 伺服器做資料的查詢、新增、修改等操作。

(1).利用 vi 或其他編輯器以 LDIF 語法編輯一個 entry 的文件，將其存檔成 top.ldif，內容如下：

```
dn: dc=thu,dc=edu.tw
objectclass: top
```

(2).將 top.ldif 檔利用 OpenLDAP 用戶端程式 ldapadd 新增入 LDAP 的目錄資料庫中。

```
$ ldapadd –f top.ldif -x –D “cn=root,dc=thu,dc=edu.tw” \
  –w secret
```

(3).建立一個 LDAP 伺服器管理者的資料，依照在 *slapd.conf* 設定檔中設定的管理者(例如 root)，為其編輯一個 entry 的文件，存成 root.ldif 檔，內容如下：

```
dn: dc=thu,dc=edu.tw
objectclass: organizationalRole
cn: root
```

(4).將 root.ldif 檔利用 ldapadd 新增入 LDAP 的目錄資料庫中。

```
$ ldapadd –f root.ldif -x –D “cn=root,dc=thu,dc=edu.tw” \
  –w secret
```

(5).完成後，重新啟動 slapd，之後便可以對此 LDAP 伺服器操作

查詢、新增、修改等動作。

$ *kill –HUP `cat /usr/local/var/slapd.pid`*

附錄三：LDAP 伺服器環境架設(含 SSL 機制)

1. OpenSSL 軟體安裝

OpenLDAP 提供了 SSLv3 的，為了要能在 LDAP 伺服器上加入 SSL 之安全連線機制，安裝 OpenSSL 來。

- 軟體來源：OpenSSL(http://www.openssl.org)
- 軟體安裝：以下安裝步驟，工作目錄皆在 OpenSSL 的軟體套件置放之目錄下。

(1).將 OpenSSL 軟體套件解壓縮

$ *tar zcvf openssl-0.9.6b.tgz*

(2).設定安裝起始檔

$ *./Configure linux-elf*

(3).建立安裝環境

$ *make*

(4).測試

$ *make test*

(5).安裝

$ *su root -c 'make install'*

2. 建立具 SSL 機制之 OpenLDAP 伺服器

由於系統架構中，必須將 SSL 機制加入到 LDAP 伺服器中，OpenLDAP 利用 OpenSSL 之函式庫開發具 SSL 連結的伺服器 daemon 與用戶端應用程式，安裝要點如下：

■ **安裝軟體**

(1).**安裝** OpenSSL(**如第** 1.**點**)

(2).**安裝** OpenLDAP(**使用** OpenSSL **函式庫**)

a.**設定編譯器之環境參數**

$ *env CPPFLAGS=-I/usr/local/openssl/include *
LDFLAGS=-L/usr/local/openssl/lib

b.**安裝與測試**

安裝與測試方式大致與之一般OpenLDAP(**不具**SSL**機制**)**安裝方式相同，差異在設定安裝起始檔的步驟，以下步驟皆在** OpenLDAP **的原始檔案目錄中執行。**

$ *./configure –with-tls*
$ *make depend*
$ *make*
$ *make test*
$ *su root -c 'make install'*

■ SSL **機制設定**

在上述步驟完成後，OpenLDAP **已將** OpenSSL **之函式庫加入，並產生具有** SSL **機制的伺服器端** daemon **與用戶端應用程式，由於** SSL **的機制中，伺服器需要使用伺服器憑證，可以利用** OpenSSL **的應用程式** *x509* **產生** LDAP **伺服器所需要之私密金鑰與憑證：**

(1).**產生憑證要求**(my-server.csr)

$ *openssl req -config openssl.cnf -new -out my-server.csr*

(2).**產生私密金鑰**(my-server.key)

$ *openssl rsa -in privkey.pem -out my-server.key*

(3).**產生憑證**(my-server.cert)

$ *openssl x509 -in my-server.csr -out my-server.cert -req *
-signkey my-server.key -days 365

產生 LDAP **伺服器之私密金鑰與憑證完畢後，修改** OpenLDAP **的伺服器設定檔** *slapd.conf* (*/usr/local/etc/openldap/*)，**在其中指定私密金鑰、伺服器憑證、憑證中心憑證之存放位置，由於產生之憑證為自行簽發** (self-sign)，**憑證中心憑證的指定則指向伺服器憑證，如下所示：**

TLSCertificateFile	/path/to/my-server.cert
TLSCertificateKeyFile	/path/to/my-server.key
TLSCACertificateFile	/path/to/my-server.cert

附註: **憑證與金鑰檔案必須是** PEM **編碼**

■ **啟動具** SSL **機制之** LDAP **伺服器：**

完成上述設定後，接著便可以啟動具 SSL **機制之** LDAP **伺服器，一般** LDAP **使用的通訊** port **為** 389**, 具** SSL **機制之** LDAP **使用之通訊** port **為** 636，**啟動命令如下：**

$ *slapd –h "ldap:/// ldaps:///"*

附錄四：屬性憑證產生程式原始碼

1. 程式解釋與前置處理

```
//============================================================
// File Name: gen_attcert.c
// Comment: Generate the Attribute Certificate for the authenticated user.
// Input: 1. Public Key Certificate(or CSR)
//          2. AA's (or LDAP Server) Certificate
//          3. AA's (or LDAP Server) private key
//            4. AA's (or LDAP Server) serial_no file
//           5. Attribute file
//            6. Section in attribute file
//            7. Expired days
// Output: Attribute Certificate
//============================================================

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "apps.h"
#include <openssl/bio.h>
#include <openssl/asn1.h>
#include <openssl/err.h>
#include <openssl/bn.h>
#include <openssl/evp.h>
#include <openssl/x509.h>
#include <openssl/x509v3.h>
#include <openssl/objects.h>
#include <openssl/pem.h>

#undef POSTFIX
#define   POSTFIX       ".srl"
#define DEF_DAYS      30

static char *x509_usage[]={
"用法: gen_attcert args\n",
```

```
"-in arg            - 輸入檔案(公鑰憑證/憑證要求)\n",
"-out arg           - 輸出檔案(屬性憑證)\n",
"-AA arg            - AA 之憑證(PEM 格式)\n",
"-AAkey arg         - AA 之密鑰(PEM 格式)\n",
"-AAserial arg      - 序號檔案\n",
"-extfile arg       - 權限屬性檔案\n",
"-extensions arg    - 權限屬性檔案之指定部分\n",
"-req               - 指定輸入檔案為憑證要求\n",
"-signkey arg       - 自行簽發使用之密鑰\n",
"-days arg          - 屬性憑證有效期限(天數)\n",
NULL
};

static int MS_CALLBACK callb(int ok, X509_STORE_CTX *ctx);
static int sign (X509 *x, EVP_PKEY *pkey,int days,int clrext, const EVP_MD
*digest, LHASH *conf, char *section);
static int x509_certify (X509_STORE *ctx,char *AAfile,const EVP_MD *digest,
                X509 *x,X509 *xca,EVP_PKEY *pkey,char *serial,
                int create,int days, int clrext, LHASH *conf, char *section);
```

2. 屬性憑證簽發函式

```
static int x509_certify(X509_STORE *ctx, char *AAfile, const EVP_MD *digest,
        X509 *x, X509 *xca, EVP_PKEY *pkey, char *serialfile, int create,
        int days, int clrext, LHASH *conf, char *section)

    {
    int ret=0;
    BIO *io=NULL;
    MS_STATIC char buf2[1024];
    char *buf=NULL,*p;
    BIGNUM *serial=NULL;
    ASN1_INTEGER *bs=NULL,bs2;
    X509_STORE_CTX xsc;
    EVP_PKEY *upkey;

    // 抓取 AA 之公鑰 ==> pkey
    // xca 為 AA 之憑證
```

```
upkey = X509_get_pubkey(xca);
EVP_PKEY_copy_parameters(upkey,pkey);
EVP_PKEY_free(upkey);

// 準備一個憑證鏈空間 ctx
X509_STORE_CTX_init(&xsc,ctx,x,NULL);

// ----------由序號檔讀取序號----------//
// 配置 buffer
buf=OPENSSL_malloc(EVP_PKEY_size(pkey)*2+
      ((serialfile == NULL)
            ?(strlen(AAfile)+strlen(POSTFIX)+1)
            :(strlen(serialfile)))+1);
if (buf == NULL) { BIO_printf(bio_err,"out of mem\n"); goto end; }

if (serialfile == NULL)
      {
      strcpy(buf,AAfile);
      for (p=buf; *p; p++)
            if (*p == '.')
                  {
                  *p='\0';
                  break;
                  }
      strcat(buf,POSTFIX);
      }
else
      strcpy(buf,serialfile);

serial=BN_new();
bs=ASN1_INTEGER_new();
if ((serial == NULL) || (bs == NULL))
      {
      ERR_print_errors(bio_err);
      goto end;
      }
// 配置一個 BIO 空間
io=BIO_new(BIO_s_file());
```

```
if (io == NULL)
    {
    ERR_print_errors(bio_err);
    goto end;
    }

// 讀取序號檔
if (BIO_read_filename(io,buf) <= 0)
    {
    if (!create)
        {
        perror(buf);
        goto end;
        }
    else
        {
        ASN1_INTEGER_set(bs,1);
        BN_one(serial);
        }
    }
else
    {
    if (!a2i_ASN1_INTEGER(io,bs,buf2,1024))
        {
        BIO_printf(bio_err,"unable to load serial number from %s\n",buf);
        ERR_print_errors(bio_err);
        goto end;
        }
    else
        {
        serial=BN_bin2bn(bs->data,bs->length,serial);
        if (serial == NULL)
            {
            BIO_printf(bio_err,"error converting bin 2 bn");
            goto end;
            }
        }
    }
```

```
if (!BN_add_word(serial,1))
    { BIO_printf(bio_err,"add_word failure\n"); goto end; }
bs2.data=(unsigned char *)buf2;
bs2.length=BN_bn2bin(serial,bs2.data);

    // 將加 1 後的序號寫回序號檔
if (BIO_write_filename(io,buf) <= 0)
    {
    BIO_printf(bio_err,"error attempting to write serial number file\n");
    perror(buf);
    goto end;
    }
i2a_ASN1_INTEGER(io,&bs2);
BIO_puts(io,"\n");
BIO_free(io);
io=NULL;
// ------------完成抓取序號之工作--------------------//

if (!X509_STORE_add_cert(ctx,x)) goto end;

// -------------開始產生屬性憑證---------------------//

X509_STORE_CTX_set_cert(&xsc,x);
// 檢查私鑰是否為發行者(AA)擁有
if (!X509_check_private_key(xca,pkey))
    {
    BIO_printf(bio_err,"AA certificate and AA private key do not match\n");
    goto end;
    }

// 將憑證資訊寫入預備簽章的欄位 ==> x
// 1. 發行者名稱
if (!X509_set_issuer_name(x,X509_get_subject_name(xca))) goto end;
// 2. 憑證序號
if (!X509_set_serialNumber(x,bs)) goto end;
// 3. 起始日期
if (X509_gmtime_adj(X509_get_notBefore(x),0L) == NULL)
```

```
        goto end;
    // 4. 終止日期
    if (X509_gmtime_adj(X509_get_notAfter(x),(long)60*60*24*days) == NULL)
        goto end;
    // 5. 將屬性加入憑證擴充欄位
    if (conf)
        {
        X509V3_CTX ctx2;
        X509_set_version(x,2); /* version 3 certificate */
                X509V3_set_ctx(&ctx2, xca, x, NULL, NULL, 0);
                  X509V3_set_conf_lhash(&ctx2, conf);
                   if (!X509V3_EXT_add_conf(conf, &ctx2, section, x)) goto end;
        }
    // 使用發行者之公鑰簽章
    if (!X509_sign(x,pkey,digest)) goto end;
    ret=1;
    //-------------產生屬性憑證結束----------------//
end:
    // 將憑證鏈內之資料移除
    X509_STORE_CTX_cleanup(&xsc);
    if (!ret)
        ERR_print_errors(bio_err);
    if (buf != NULL) OPENSSL_free(buf);
    if (bs != NULL) ASN1_INTEGER_free(bs);
    if (io != NULL)    BIO_free(io);
    if (serial != NULL) BN_free(serial);
    return ret;
    }
```

3. 屬性憑證簽發函式(自行簽章)

```
static int sign(X509 *x, EVP_PKEY *pkey, int days, int clrext, const EVP_MD
*digest,
                                LHASH *conf, char *section)
    {

    EVP_PKEY *pktmp;
```

```
    pktmp = X509_get_pubkey(x);
    EVP_PKEY_copy_parameters(pktmp,pkey);
    EVP_PKEY_save_parameters(pktmp,1);
    EVP_PKEY_free(pktmp);

    // 1. 發行者名稱
    if (!X509_set_issuer_name(x,X509_get_subject_name(x))) goto err;
    // 2. 起始日期
    if (X509_gmtime_adj(X509_get_notBefore(x),0) == NULL) goto err;
    // 3. 終止日期
    if (X509_gmtime_adj(X509_get_notAfter(x),(long)60*60*24*days) == NULL)
        goto err;
    // 4. 公鑰
    if (!X509_set_pubkey(x,pkey)) goto err;
    // 5. 將屬性寫入憑證擴充欄位
    if (conf)
        {
        X509V3_CTX ctx;
        X509_set_version(x,2); /* version 3 certificate */
                X509V3_set_ctx(&ctx, x, x, NULL, NULL, 0);
                X509V3_set_conf_lhash(&ctx, conf);
            if (!X509V3_EXT_add_conf(conf, &ctx, section, x)) goto err;
    }
    // 使用輸入之私鑰作簽章
    if (!X509_sign(x,pkey,digest)) goto err;
    return 1;
err:
    ERR_print_errors(bio_err);
    return 0;
    }
```

附錄五：屬性憑證驗證程式原始碼

1. 程式解釋與前置處理

```
//============================================================
// File Name: verify_attcert.c
// Comment: Verify the Attribute Certificate of the authenticated user.
// Input: 1. Attribute Certificate(or CSR)
//        2. AA's (or LDAP Server) Certificate
// Output: ok ==> success
//         error code ==> failure
//============================================================

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "apps.h"
#include <openssl/bio.h>
#include <openssl/err.h>
#include <openssl/x509.h>
#include <openssl/x509v3.h>
#include <openssl/pem.h>

static int MS_CALLBACK cb(int ok, X509_STORE_CTX *ctx);
static int check(X509_STORE *ctx, char *file, STACK_OF(X509) *uchain,
STACK_OF(X509) *tchain, int purpose);
```

2. 屬性憑證驗證核心函式

```
static int internal_verify(X509_STORE_CTX *ctx)
{
    int i,ok=0,n;
    X509 *xs,*xi;
    EVP_PKEY *pkey=NULL;
    time_t *ptime;
    int (*cb)();

    cb=ctx->verify_cb;
```

```
if (cb == NULL) cb=null_callback;

n=sk_X509_num(ctx->chain);
ctx->error_depth=n-1;
n--;
xi=sk_X509_value(ctx->chain,n);
if (ctx->flags & X509_V_FLAG_USE_CHECK_TIME)
      ptime = &ctx->check_time;
else
      ptime = NULL;

// 驗證憑證之發行者是否確(整個憑證鏈)
if (ctx->check_issued(ctx, xi, xi))
      xs=xi;
else
{
      if (n <= 0)
            {
            ctx->error=
                X509_V_ERR_UNABLE_TO_VERIFY_LEAF_SIGNATURE;
            ctx->current_cert=xi;
            ok=cb(0,ctx);
            goto end;
            }
      else
            {
            n--;
            ctx->error_depth=n;
            xs=sk_X509_value(ctx->chain,n);
            }
      }

// ----------驗證憑證資訊-----------//
while (n >= 0)
{
   ctx->error_depth=n;
  if (!xs->valid)
   {
```

```
if ((pkey=X509_get_pubkey(xi)) == NULL)
{
    ctx->error=
      X509_V_ERR_UNABLE_TO_DECODE_ISSUER_PUBLIC_KEY;
    ctx->current_cert=xi;
    ok=(*cb)(0,ctx);
    if (!ok) goto end;
}
// 驗證公鑰
if (X509_verify(xs,pkey) <= 0)
{
    ctx->error=X509_V_ERR_CERT_SIGNATURE_FAILURE;
    ctx->current_cert=xs;
    ok=(*cb)(0,ctx);
    if (!ok)
    {
        EVP_PKEY_free(pkey);
        goto end;
    }
}
    EVP_PKEY_free(pkey);
    pkey=NULL;
// 驗證起始日期
i=X509_cmp_time(X509_get_notBefore(xs), ptime);
if (i == 0)
{
    ctx->error=
        X509_V_ERR_ERROR_IN_CERT_NOT_BEFORE_FIELD;
    ctx->current_cert=xs;
    ok=(*cb)(0,ctx);
    if (!ok) goto end;
    }
if (i > 0)
    {
    ctx->error=X509_V_ERR_CERT_NOT_YET_VALID;
    ctx->current_cert=xs;
    ok=(*cb)(0,ctx);
    if (!ok) goto end;
```

```
            }
            xs->valid=1;
            }

        // 驗證終止日期
        i=X509_cmp_time(X509_get_notAfter(xs), ptime);
        if (i == 0)
            {
            ctx->error=
                X509_V_ERR_ERROR_IN_CERT_NOT_AFTER_FIELD;
            ctx->current_cert=xs;
            ok=(*cb)(0,ctx);
            if (!ok) goto end;
            }
        if (i < 0)
            {
            ctx->error=X509_V_ERR_CERT_HAS_EXPIRED;
            ctx->current_cert=xs;
            ok=(*cb)(0,ctx);
            if (!ok) goto end;
            }
        ctx->current_cert=xs;
        ok=(*cb)(1,ctx);
        if (!ok) goto end;

        // 準備驗證下一張憑證
        n--;
        if (n >= 0)
        {
            xi=xs;
            xs=sk_X509_value(ctx->chain,n);
        }
        }
    ok=1;
end:
    return ok;
    }
```