多層次認證系統的實現

An Implementation of Multi-tier Authentication System

黃啟川
高雄市國稅局
e0101500@ntak.gov.tw

王明習
國立成功大學工程科學系
mswang@mail.ncku.edu.tw

摘要

線上申辦作業為政府重大資訊建設計劃之一，其中涉及個人隱私、敏感或有權責關係資料部份，可透過憑證機制達到網路線上申辦應用系統所需之身分認證、加解密及數位簽章等安全功能。

本文為研究如何在現行網頁上加入認證機制，此認證機制是以 Java 2 JSSE API 加上 Java RMI 為實作基礎，以實現一個多層次架構之線上申辦系統。Java 之技術為屬於開放源(Open Source)，具有易於掌握及學習之方便性。多層次架構則有效能佳、易於維護、彈性及擴充性之優點。Java RMI 為 Java 版的 RPC，能將分散式物件整合進以 Java 程式語言所撰寫之程式中，並使分散式物件在使用上盡量接近一般的物件。本研究中完成一個實驗性質之多層次認證系統，並經由實際之測試，其結果令人相當滿意，證明本研究所建立之系統構想其實用性是可行的，唯如何和政府機構正推動之 GCA 系統結合，則有待於進一步的探討。

關鍵詞：RMI、SSL、認證、線上申辦系統、JSSE

一 概論

由於近年網際網路之普遍與發達，政府積極配合推動電子化政府，以提升作業效率及政府形象;政府機關對於各項業務之受理方式，亦開始朝向提供上網申辦之途徑考量，使便民服務層面又跨入新的領域;稅務機關因受限於稅務資料之機密性，在未有適當之安全機制之前，僅能就無須身分確認及免附附件之服務項目開放線上申請服務[12}。

一般申辦案件如不涉及個人隱私、敏感或有權責關係資料，不一定需要使用憑證，以具線上加密功能（如 SSL）方式處理即可。反之若涉及個人隱私、敏感或有權責關係資料就需使用憑證，此方面之工作可透過憑證管理中心[7-11]及 IC（RSA）卡所提供之安全保密函式庫（API）或其他提供相同功能之技術(如 Java 之 JSSE)，達到網路申辦應用系統所需之身分認證、加解密及數位簽章等安全功能[5,6,13,14,21]。

線上加密 SSL 是由 Netscape 公司為制定資料安全性所發展的通訊協定，在 OSI(Open System Interconnection)模型中，SSL 介於應用層(Application Layer)與傳輸層(Transportation Layer)之間，提供 TCP/IP 通訊協定資料加密(Data Encryption)、用戶端與伺服端認證(Authentication)等功能。一般電子商務網站[2,3,4]所使用之線上加密 SSL 均為單向認證（伺服端認證），若需要更高的安全性則可採用雙向認證之做法。

本研究選擇以 Java 所提供之 JSSE API，來實現 SSL 伺服器或 SSL 客戶端做雙向溝通之加密機制的功能，採用 JSSE API 的原因之一為 Java 之技術為開放源，較易於掌握及學習。

在 Web Server 之選擇方面，為易於實現及顧及普遍性，故選擇由 Java 技術發展出來的 Tomcat Web Server ，Tomcat 是 Sun 和

Apache 合作所做出來的 JSP Server，它可支援到 Servlet 2.2 及 JSP 1.1 以上。Tomcat 是由 Java 所組成之 Java Program，所以只要有 JDK 就可以執行，並可跨平台[22,23]。

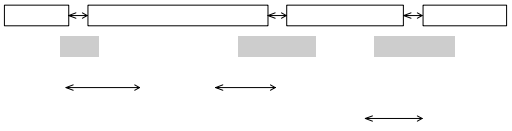
本研究中為採用多層次架構方式，由於多層次架構比 Client/Server 之雙層架構具有效能佳、易於維護、彈性及擴充性之優點。

RMI 是一種通訊機制，讓位於某個 Java 虛擬機器(Virtual Machine，VM) 內的物件，可以呼叫位於另一個 Java 虛擬機器內物件的成員函式。呼叫遠端物件的成員函式，與呼叫同一個 Java 虛擬機器內物件的成員函式，兩者方法是相同的。這兩個 Java 虛擬機器可以在同一部主機上，或是在不同的主機上。可以說 Java 版的 RPC，RMI 將分散式物件整合進以 Java 程式語言所開發的程式中，並使分散式物件在使用上盡量接近一般的物件，因此可運用來發展多層次架構。

在多層次架構中要如何提供安全性機制？吾人可探討 RMI 與 SSL 結合之可能性，若可行，那就不是一般 Client/Server SSL 機制所能比擬的，換句話說，就是可多層次又可雙向認證。

二 系統實作及執行

本文中所要實現的線上申辦多層次架構系統之整個系統關係如圖一所示。



圖一：應用範例架構圖

圖一之系統架構的流程大略說明如下：

1. 使用者端（Client 端）

以 HTML 結合 JSP 來處理前端使用者界面，當使用者輸入申辦畫面資料後，透過 HTTPS 與 Tomcat Web Server 伺服端（Servlet Engine）建立連結，使用者端使用 JSP、伺服端使用 Servlet Engine，此作法與微軟網站使用者端使用 VBScript、伺服端使用 ASP 是類似的。伺服端 Servlet 程式用 RMI 呼叫應用程式伺服端之物件（此物件為線上申辦之應用程式物件），應用程式伺服端之線上申辦物件用 RMI 呼叫資料庫伺服端之資料庫，其結果由後端傳回前端後再用 JSP 程式將結果在使用者端呈現出來。

2. Tomcat Web Server 端

使用者端與 Web Server 端間之通訊是透過 SSL 以保障其安全性，因此前端瀏覽器以 HTTPS 呼叫 Web Server 端之 Servlet 程式後再間接呼叫應用程式伺服端之線上申辦物件（此呼叫方式使用 RMI 技術，若要達到雙向認證則 RMI 要與 SSL 配合一起使用），避免瀏覽器直接呼叫應用程式伺服端之線上申辦物件，如此以達到保密的效果。

3. 應用程式伺服端

應用程式伺服端之線上申辦物件使用 JDBC 呼叫 Database 端之資料庫（若要達到雙向認證則要加上 SSL 技術一起使用），執行結果則回傳至前端呈現。

4. Database 端

本研究目前為使用 Mysql 資料庫，對於重要欄位可加上加解密動作。

以上之架構中的關鍵部份說明如下：

1. Socket、SSLSocket 及 SSL--雙向認證
2. RMI--分散式物件
3. RMI 結合 SSL--分散式物件結合雙向認證
4. 如何使用 Tomcat Web Server 及如何

使用 Servelet & JSP

5. Java 連資料庫的 API JDBC

在此僅進一步說明本研究對上述之第 1 點及第 3 點之實現方式，其餘純為使用之技術問題，可參考相關使用技術文件：

1. Java 所提供之安全 Internet 連線--SSL

隨著網際網路的普遍，資訊在網路上傳遞的安全也越來越受重視，而 Sun 所提供的 Java Secure Socket Extension（JSSE）以支援安全的 Internet 連線、資料保密（Encryption）以及認證（Authentication）。在連線上之 Server 端與 Client 端之寫法分別如下所示：

(1)Client 端寫法

```
SSLSocketFactory factory = (SSLSocketFactory)
   SSLSocketFactory.getDefault();
Socket socket = factory.createSocket
   ("localhost",8080);
```

(2)Server 端寫法

```
SSLServerSocketFactory factory =
   (SSLServerSocketFactory)
   SSLServerSocketFactory.getDefault();
SSLServerSocket server =
   (SSLServerSocket)factory.
   createServerSocket(8080);
Socket socket = server.accept();
```

(3)雙向認證之 Server 端寫法

```
SSLServerSocketFactory factory =
   (SSLServerSocketFactory)
   SSLServerSocketFactory.getDefault();
SSLServerSocket server =
   (SSLServerSocket)factory.createServerSocket(
   8080);
s.setNeedClientAuth(true);
Socket socket = server.accept();
```

2.RMI 結合 SSL 的實作

RMI 是建立在 IP 之上的 TCP 通訊協定，如果吾人需要使用非 TCP 或自訂的通訊協定，則需要自行產生一個 socketfactory，但可使用自訂的 socket 或 SSL socket。

(1)對於 Server 端的 Socket, 需實作 interface RMIServerSocketFactory,這是用來負責接受 Client 的呼叫，需要取代的 Method 為

ServerSocket createServerSocket(int port)

(2)對於 Client 端的 Socket, 則需實作 Interface RMIClientSocketfactory,這是用來提供給 Client 以產生 RMI 呼叫，需要取代的 Method 為

Socket createSocket(String host,int port)

(3)為了使用這些自訂的 Socketfactory，在 ServerSocket（如 RMIEchoServerImpl2.java）中對於被繼承的 UnicastRemoteObject class 需要被通知 RMI 所要使用的新 Socketfactory，因此是使用有三個參數的建構子：

protect UnicastRemoteObject(int port, RMIClientSocketFactory csf,RMIServerSocketFactory ssf)

三 應用範例及其結果

3.1 應用範例架構說明

本研究之應用範例的架構為將遠端物件、SSL 雙向認證與資料庫加密三者結合在一起，此應用範例之架構主要是由下列想法構成：

1. 一般民眾只能透過瀏覽器上網，所以只能用 Web 架構。
2. 為了顧及彈性及速度，所以採用多層次架構，可分成：
   前端：使用者端（以 Browser 表示）。
   後端（再細分成三層）：
   Tomcat Web Server 端（以 Web Server 表示）、
   應用程式伺服端（以 AP Server 表及資料庫伺服端(以 DB Server 表示)。

註：1. 此多層次架構可表示成如下架構：Browser<->Web Server<->AP Server<->DB Server
2. 所用之<->符號代表左右兩者間之通訊。

3. 為了考慮安全性，而將後端三層(Tomcat Web Server 端、應用程式伺服端及資料庫伺服端；Web Server<->AP Server<->DB Server）採用 Java 遠端分散式物件結合 Java SSL 雙向認證的做法，為的是讓各自信任的憑證可連上應用程式伺服端及資料庫伺服端（AP Server<->DB Server），各自信任的憑證可不相同。
4. 資料庫加密是為了多一層保護，若有人可進資料庫（如系統技術人員），在沒有應用系統把關之 Session Key 所存取的重要欄位（如應用系統加解密之 Session Key）也只是一堆亂碼罷了。
5. 為了使前端（使用者端；Browser）與後端三(Tomcat Web Server 端、應用程式伺服端及資料庫伺服端；Web Server<->AP Server<->DB Server）之 Java 物件能貫通，Web Server 採用 Tomcat Web Server，使用者端瀏覽器與 Web Server 間採用 JSP 及 Servlet 技術（類似微軟之 ASP 技術），為了全程要採用雙向認證，因此 Web Server 提昇為更安全的 Secure Tomcat Web Server，此時使用者端（Browser）要提供憑證以辨認使用者的身分，另應用程式伺服端與資料庫伺服端間（AP Server<-> DB Server）用 JDBC 呼叫資料庫，資料庫採用流行的 MySQL 資料庫。

以上述想法為基礎，並為了充分運用 Java 及多層次架構的優點，本研究中所擬訂之應用範例為一多層次線上申辦架構(如圖 3)。

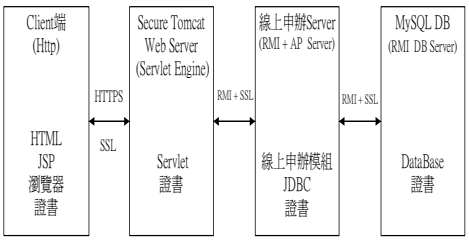


圖3：修正的應用範例的架構圖

3.2 應用範例結果說明

本節中將說明上節所述之應用範例執行之部份結果，如下：

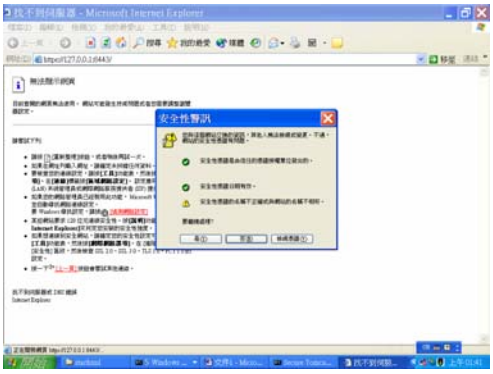
1. 執行線上申辦系統

當 AP Server、DB Server 之環境設定完成後，以下的畫面為一系列執行線上申辦之畫面，說明如下。

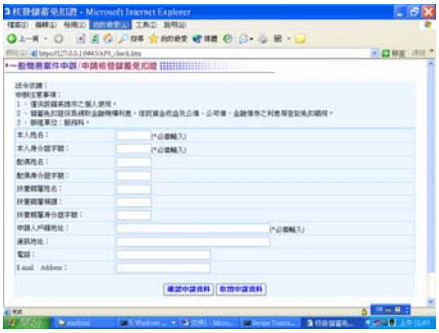
(1)透過瀏覽器執行http://127.0.0.1/ 時因無法正確連結Web Server，所以畫面顯示無法顯示網頁（圖 4(a)所示），正確方式為透過瀏覽器執行 https://127.0.0.1:8443/以Https呼叫 Secure Tomcat Web Server 之Servlet 程式執行線上申辦作業模組，Secure Tomcat Web Server內定連接Port 為 8443，執行時會顯示兩個安全性警訊畫面（圖 4(a)、圖 4(b)所示）。

(2)通過安全性警訊畫面後就連上 Secure Tomcat Web Server 之首頁，來進行輸入資料（圖 4(c)所示）。

(a)：以 Https 呼叫 Tomcat Web Server 所出現的安全性警告訊息 1



(b)：以 Https 呼叫 Tomcat Web Server 所出現的安全性警告訊息 2



(c)：所選擇之申請核發儲蓄免扣證的輸入畫面

圖 4：執行線上申辦系統系列畫面

2. 對所選的線上申辦業務輸入資料

當 Browser 端經認證連上 Web Server 端，再選擇所需要的畫面就可開始輸入資料（圖 5 所示）。

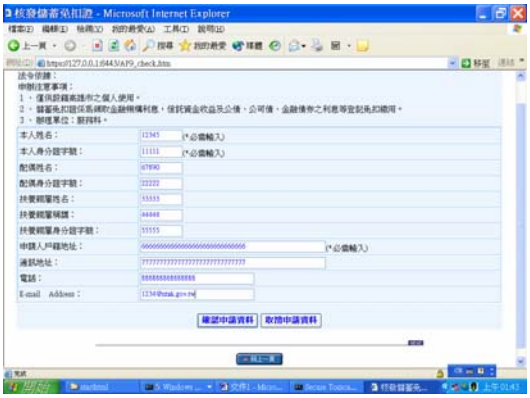


圖 5：線上申辦業務輸入資料的畫面

3. 經過處理後系統顯示已完成申辦資料申報的回應畫面

使用者輸入申辦畫面資料後透過 HTTPS（避免瀏覽器直接呼叫 AP Server 之 ONLine AP 物件，如此有保密效果）與 Tomcat Web Server 伺服端(MiddleServer.iava)建立連結，伺服端 Servlet 程式用 RMI 呼叫 AP Server 之 ONLine AP 物件，AP Server 之 ONLine AP 物件用 RMI 呼叫 DB Server 之資料庫(所有進出資料庫的資料或重要欄位均經加解密處理，如應用系統加解密之 Session Key)，最後再將儲存於資料庫的欄位資料經必要的加解密動作，將結果由後端傳回前端後再用 JSP 程式呈現回應畫面。圖六為經過處理後系統顯示已完成申辦資料申報的回應畫面

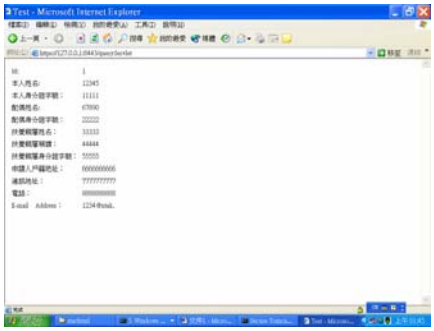


圖 6：完成申辦資料申報的回應畫面

3.3 實作時所遇到之問題

對本文之作者而言，再實作的過程中遇到以下的幾點問題,在此提出以供讀者參考。

1. 雖然 Java 速度已經有所改善,但感覺起來還是慢一拍，還有改善的空間。
2. Java 手冊的說明並不容易了解且範例不夠多,而 Java 其他相關書籍深淺不一且手法不盡相同，對書籍所敘述的範例最好實際演練一遍，才會知到問題所在。
3. 撰寫 Java 程式一定要對所使用的相關類別作深入研究，不然程式執行後會不知道問題出在哪裡。
4. Java 為 Open Source 所以可以了解整體結構，但對原始程式的追蹤也非易事(因缺乏相關說明文件),一切只能靠經驗了，但這還是較原始程式不公開來得好太多了（如微軟的系統)。
5. Java 執行時期環境設定的部份也相當重要，往往問題就出在初期對 JCA、JCE、存取控制及 Policy 的設定。
6. JSSE 及 RMI 等 API 的熟悉要花較多的時間。
7. RMI 與 SSL 的組合為最困難的部分，程式稍稍改不正確，就執行不起來。多層次架構就是依靠此作法來達到的,遠端物件結合 SSL 雙向認證 若再將進資料庫的 Data 作加密處理應該更安全，唯一缺點是執行上要花多一點時間。

Java 在 Security 上的目前功能完整的且持續不斷的改進中，因此進入的門檻也越來越高，當然更安全了。

四 結論與後續研究

4.1 結論

電腦與網路越進步，資訊安全就越顯得重要。如何確保網路的資訊安全，安全無慮的認證系統應被加以考慮。於網路傳輸過程中採行認證機制，可鑑別交易雙方的身分，可保障您進行網路交易時，

（1）避免資料被竄改或網站被冒用。

（2）避免被冒名傳送假資料。

（3）避免資料被竊取偷看。

（4）避免對方不承認收到交易訊息。

為了展現完整的多層次認證系統,遂設計如下之應用範例架構（如圖 7）

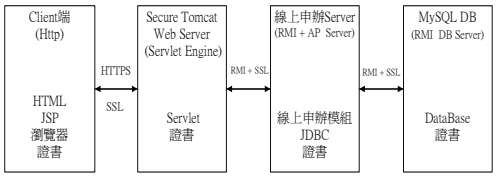


圖 7：修正的應用範例的詳細架構圖

此多層次架構的做法如下：

Browser<->Web Server<->AP Server<->DB Server

(1) 為了顧及彈性及速度，採用多層次架構。

(2) 對 Browser 端辨認身分所以用更安全的 Secure Tomcat Web Server 及 Servlet 技術。

(3) 為了考慮安全性，而將 Web Server<->AP Server<->DB Server 這三層採用 Java 遠端分散式物件結合 Java SSL 雙向認證的做法。

(4) 為了使資料庫多一層保護，重要的欄位均經過加密處理。

實作上為了易於掌握，全程採用 Java 語言。

(1)Secure Tomcat Web Server 是由 Java 所組成之 Java Program。

(2)Java RMI 為 Java 版的 RPC,能將分散式物件整合進 Java 程式語言中，並

使分散式物件在使用上盡量接近一般的物件。

(3)JSSE 提供 SSL 加密機制，透過 JSSE API,用戶端與伺服端間的交易處理將享有資料加密、雙方認證等功能。

如此將以上三者結合就可實現線上申辦的多層次架構。

雖然應用範例所表現出來畫面並不多，但為了達到雙向認證的功能，實驗上卻花了很多的時間才得到初步的成果。

4.2 後續研究

此系統目前僅為認證機制的雛型版本，尚未將完整系統全部應用上，但也證實多層次認證系統的可行性，有此次發展經驗，相信系統能調整更為精緻且功能更強大,另外,對不同平台或其他 Web Server 之認證領域相信也將可輕鬆探討,例如 IIS Server 之認證做法或採用 Linux、Apache、Open SSL、PHP、MySQL 互相結合在一起之認證做法…等。Open SSL 為 C 語言之 Open Source Code 在 Linux 平台應用很廣，至於現行政府認證機制採 GCA 憑證是否可與 Java 結合,經研究發現政府認證 GCA 機制之函數庫是 C 語言型態,若要將兩者結合可以依下列說明來考慮之。

因為較嚴格的執行時間需求、需要執行和系統有關的低階運作或是要呼叫其他語言之函數庫，Java 的 JNI(Java Native Interface；關於 JNI 規格的細節，可參考 http://java.sun.com/J2se/1.3/docs/guide/jni/)提供一個標準的命名和呼叫的方式來使用這些由其它程式語言所產生的程式庫。

一般而言，產生 Native Method 的步可如下所述：

1. 產生 Java 程式

產生 native method 的第一個步驟是產生使用 native method 的 Java 程式，其中，最重要的兩個部分是：使用 Systern.loadLibrary()載入包含所要使用的 native method 的程式庫，

2. 編譯 Java 程式來產生 class 檔案。
3. 使用 J2SE 的工其程式 javah 來產生前面 class 的 header 檔案(一個可用於 C/C++的 header 檔案)，這個 header 檔案會包含該 class 所宣告的 native method,以提供 C/C++程式所需函數的 signature。
4. 產生包含 native method 的程式如同產生 C/C++程式的方式，需記得包含前面所產生的 header 檔案且函數的 signature 需吻合。
5. 產生共享程式庫，此種做法在實際應用政府 GCA 認證機制憑證時可思考採用。

參考文獻

[1] 洪淑芬，「公開金鑰認識中心（CA）主管機關權責範圍探討」，資訊法務透析，25~31 頁，民國 86 年 3 月。

[2] 樊國楨，電子商務高階安全防護：公開金鑰密碼資訊系統安全原理，資策會資訊與電腦，民國 86 年。

[3] 樊國楨，電子文件交換與公開金鑰基磐建設(二)(三)(四)(五)，叡揚資訊，民國 86 年 12 月~民國 87 年 10 月。

[4] 樊國楨，電子商務資訊安全認證機制初探(上)(下)，叡揚資訊，民國 89 年 4 月、民國 89 年 7 月。

[5] 賴溪松、韓亮、張真誠，近代密碼學及其應用，松崗電腦圖書資料股份有限公司，民國 88 年 4 月。

[6] 張真誠、林祝興、江季翰，電子商務安全，松崗電腦圖書資料股份有限公司，民國 89 年。

[7] 中華電信股份有限公司數據通信分公司，GCA 簡介，民國 90 年。

[8] 中華電信股份有限公司數據通信分公司，政府憑證管理中心 SSL 安全保密程式介面使用說明書 V1.0 ，民國 91 年 4 月。

[9] 中華電信股份有限公司數據通信分公司，政府憑證管理中心憑證實作準則--第一版，政府憑證管理中心，民國 91 年 8 月。

[10] 中華電信股份有限公司數據通信分公司，政府憑證管理中心通用界面服務草案，中華電信數據通信分公司，民國 88 年 9 月。

[11] 中華電信股份有限公司數據通信分公司，政府憑證管理中心安全保密函式庫介紹，資訊安全通訊第六卷第三期，民國 89 年 6 月。

[12] 政府憑證總管理中心及政府憑證管理中心委外服務需求說明書（草案）徵求意見書 」(民九十、七月)，行政院研究發展考核委員會。

[13]「電子簽章法」(民九十、十一月)，總統府公報，總統府，http://www.president.gov.tw/2_report/index.html。

[14]「政府憑證管理中心憑證實作準則」，http://www.pki.gov.tw/cps.htm。

[15] RSA Laboratories〔2001〕，PKCS #1：RSA Cryptography Standard，RSA Security。

[16] RSA Laboratories〔1999〕，PKCS #5：Password-Based Cryptography Standard，RSA Security。

[17] RSA Laboratories〔1999〕，PKCS #6：Password-Based Cryptography Standard，RSA Security.

[18]RSA Laboratories〔1993〕，PKCS #7：Cryptographic Message Syntax Standard，RSA Security。

[19]RSA Laboratories〔1993〕，PKCS #8：Private-Key Information Syntax Standard，RSA Security。

[20]RSA Laboratories〔2000〕，PKCS #11：Cryptographic Token Interface Standard，RSA Security。

[21]Burton S. Kaliski Jr.，Internet X.509 Public Key Infrastructur Certificate and CRL Profile，January 1999。

[22] IBM，Java Network Security，November 1997。

[23] IBM，Java 2 Network Security，June 1999。