

聯宏科技股份有限公司  
PAYSECURE TECHNOLOGY CO., LTD.

# PKI 電子商務實務技術

## 實習教材



台北市內湖路一段91巷17號10樓之1 [WWW.PAYSECURE.COM.TW](http://WWW.PAYSECURE.COM.TW)

TEL : (02)2657-1187 FAX : (02)2657-1205

---

**PaySecure Technology**

所有內容受到 中華民國著作權法 及國際著作權法律的保障，著作權為聯宏科技股份有限公司所有。閱讀者不得變更、發行、播送、轉賣、重製、改作、散布、展示或利用這些文章的全部或局部內容。使用者必須遵守著作權法的所有相關規定，如構成著作權侵害，須負民事及刑事責任。

# PKI 電子商務實務技術 實習教材

版本: 0.1

日期: 2007/6/5

♥ Copyright 2007. PaySecure Technology Co., Ltd.

This document, which contains confidential material, is private and confidential and is the property and copyright of PaySecure Technology Co., Ltd. No part of this document may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, chemical, photocopy, recording or otherwise without the prior written permission of PaySecure Technology Co., Ltd.

---

**PaySecure Technology**

所有內容受到 中華民國著作權法 及國際著作權法律的保障，著作權為聯宏科技股份有限公司所有。閱讀者不得變更、發行、播送、轉賣、重製、改作、散布、展示或利用這些文章的全部或局部內容。使用者必須遵守著作權法的所有相關規定，如構成著作權侵害，須負民事及刑事責任。

### 版本更新紀錄

版 本	日 期	修 訂 章 節	說 明	備 註
0.1	2007/6/5		Initial	

## 目 錄

實驗 1.	硬體及軟體環境設定 .....	7
1.1	實驗目的 .....	7
1.2	相關原理說明 .....	7
1.2.1	PSE 操作模式 .....	7
1.2.2	Cryptoki 設定 .....	8
1.2.3	軟體開發環境 .....	9
1.3	實驗範例 .....	9
1.3.1	設定Admin Token PIN 範例 .....	9
1.3.2	新增user slot 範例 .....	10
1.3.3	Token 初始化範例 .....	12
1.3.4	軟體環境設定範例 .....	13
1.4	習題 .....	18
實驗 2.	建立連線及初始化TOKEN .....	19
2.1	實驗目的 .....	19
2.2	相關原理說明 .....	19
2.2.1	程式流程 .....	19
2.2.2	Token 初始化函式 .....	20
2.3	實驗範例 .....	20
2.4	習題 .....	22
實驗 3.	建立資料及憑證物件 .....	23
3.1	實驗目的 .....	23
3.2	相關原理說明 .....	23
3.3	實驗範例 .....	25
3.3.1	建立資料物件 .....	25
3.3.2	建立憑證物件 .....	26
3.3.3	習題 .....	30
實驗 4.	建立金鑰物件 .....	31
4.1	實驗目的 .....	31
4.2	相關原理說明 .....	31
4.3	實驗範例 .....	32

4.3.1	建立秘密金鑰物件 .....	32
4.3.2	建立公開金鑰及私密金鑰物件 .....	34
4.3.3	習題 .....	39
實驗 5.	金鑰產生 .....	<b>40</b>
5.1	實驗目的 .....	40
5.2	相關原理說明 .....	40
5.3	實驗範例 .....	41
5.3.1	產生秘密金鑰 .....	41
5.3.2	產生金鑰對 .....	44
5.4	習題 .....	47
實驗 6.	資料加解密 .....	<b>48</b>
6.1	實驗目的 .....	48
6.2	相關原理說明 .....	48
6.3	實驗範例 .....	51
6.3.1	對稱式加解密 .....	51
6.3.2	非對稱式加解密 .....	54
6.4	習題 .....	59
實驗 7.	簽章及驗證 .....	<b>61</b>
7.1	實驗目的 .....	61
7.2	相關原理說明 .....	61
7.3	實驗範例 .....	65
7.4	習題 .....	69
實驗 8.	金鑰管理 .....	<b>70</b>
8.1	實驗目的 .....	70
8.2	相關原理說明 .....	70
8.3	實驗範例 .....	74
8.3.1	Wrap 及Unwrap 金鑰 .....	74
8.3.2	產生金鑰分持值(key component) .....	83
8.3.3	輸入分持值回復金鑰 .....	87
8.4	習題 .....	90
實驗 9.	單向雜湊函數 .....	<b>92</b>

9.1	實驗目的.....	92
9.2	相關原理說明.....	92
9.3	實驗範例.....	93
9.4	習題.....	94
<b>實驗 10.</b>	<b>憑證管理.....</b>	<b>95</b>
10.1	實驗目的.....	95
10.2	相關原理說明.....	95
10.3	實驗範例.....	99
10.3.1	產生X509 憑證.....	99
10.3.2	產生PKCS#10 憑證申請檔.....	104
10.4	習題.....	107
<b>附錄1:PKCS#11 簡介 .....</b>		<b>108</b>



ProtectServer External

---

## PaySecure Technology

所有內容受到 中華民國著作權法 及國際著作權法律的保障，著作權為聯宏科技股份有限公司所有。閱讀者不得變更、發行、播送、轉賣、重製、改作、散布、展示或利用這些文章的全部或局部內容。使用者必須遵守著作權法的所有相關規定，如構成著作權侵害，須負民事及刑事責任。

## 實驗1. 硬體及軟體環境設定

### 1.1 實驗目的

本實驗主要在於了解開發 PSE 應用程式時，PSE 所要做的設定及初始化。並建立軟體開發環境以供開發 PSE 應用程式。

### 1.2 相關原理說明

#### 1.2.1 PSE 操作模式

PSE 運行在網路操作模式下，其透過 TCP/IP 和主機端通訊，如圖 1-1 所示。在此模式下，主機端需安裝 Network HSM Access Provider(Etpcihsm.exe)，主要功能為網路連線的建立及資料的傳送及接收。安裝步驟請參考 SafeNet 文件 *HSM Access Provider Install & Configuration Guide*。

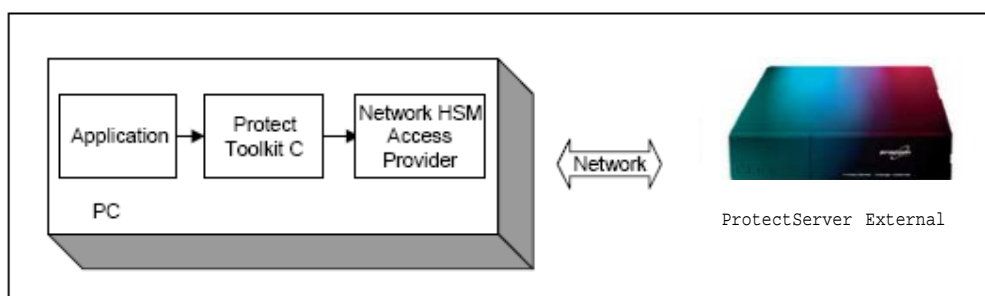


圖 1-1:PSE 操作模式

為了讓主機端應用程式可以操作或使用 HSM 的各項功能，HSM 需要提供一介面或稱為 API(Application Programmer Interfaces)供應用程式呼叫使用。因此使用者便可以透過這些應用程式介面來設計所需的應用程式。在密碼服務 API 標準上，RSA 公司針對密碼裝置定義了 PKCS#11 應用程式介面標準，希望可以解決不同密碼裝置間程式開發的問題。Protect Toolkit C 是 SafeNet 公司針對 PKCS#11 標準所實作出來的密碼服務函式庫(cryptographic service provider)。因此主機端的應用程式可以透過 Protect Toolkit C PKCS#11 來操作 PSE HSM，如圖 1-1 所示。

Protect Toolkit C(安裝檔案名稱為 ETcpsdk.exe)在安裝時可選擇是否採用 software-only，以方便使用者在沒有 HSM 的情形下依然可以開發及測試主機端應用程式。若之後想更改模式為網路操作模式或 software-only 模式，可至 windows 控制台->系統->進階->環境變數->系統變數，更改 Path 環境變數。若是網路操作模式，改為 C:\Program Files\Safenet\ProtectToolkit C SDK\bin\hsm。若採用 software-only，則改為 C:\Program Files\Safenet\ProtectToolkit C SDK\bin\sw 即可。Protect Toolkit C 即依不同的路徑去使用不同的 Cryptoki 函式庫。

## 1.2.2 Cryptoki 設定

PKCS#11 又可稱為 Cryptoki，其簡介請參考附錄 1。在 PKCS#11 中，slot 為一密碼裝置的介面。Protect Toolkit C 又將 slot 分成 3 種型式，user slot、smart card slot 及 admin slot。

- User slot 是由管理者(administrator)針對HSM 的每一個使用者所建立並專供應用程式使用，每台HSM 可以有多個user slot。
- Smart card slot 主要是用來做金鑰備分及金鑰回復。
- Admin slot 供管理者使用來管理及設定 HSM。每台 HSM 只有一個 admin slot。admin slot 裏面的 token 即為 admin token。而 admin token 裏面所含的物件即為 admin object。admin object 即為用來表示 HSM 硬體及操作上的設定。並可供應用程式查詢及管理者修改其值。

由以上可知每台HSM 會具有一個以上且不同型態的 slot。

Protect Toolkit C 定義四種使用者角色(role)外，分別是：

- Security Officer(SO)
- User
- Administration Security Officer (ASO)
- Administrator

其中 SO 及 user 為 PKCS#11 中所定義，SO 是用來初始化 token 及設定 user 的 PIN，而 user 則可以進行物件的存取，應用程式在執行時一般為 user。而 ASO 及 Administrator 則為 Protect Toolkit C 所額外定義。ASO 及 Administrator 是用來設定及管理 HSM 的使用者角色。管理者所需負責的設定請參考 SafeNet 文件 *Protect Toolkit C Administration Manual*。



### 1.2.3 軟體開發環境

在應用程式開發前必須先成功安裝 Protect Toolkit C。本教材之程式開發平台為 Windows，開發環境及編譯器採用 Microsoft Visual C++ 6.0。在每一個 Microsoft Visual C++ 6.0 程式專案裏，必須在 Project -> Settings -> C/C++ tab 中選擇 Category 為 General，並在 Preprocessor definitions 欄位中加上“\_WINDOWS”定義。以及在 Project -> Settings -> Link tab 中選擇 Category 為 General，並在其 Object/library modules 欄位中加上 cryptoki2.lib。cryptoki2.lib 是一個 import library，指向 cryptoki.dll 中各個 cryptoki 函式的位址。因此我們是採用隱式連結 (Implicitly Link) 的方式來載入動態連結函式庫。另外在程式專案中必須載入“cryptoki.h”，此 header 檔放在 Protect Toolkit C 的安裝目錄 C:\Program Files\Safenet\ProtectToolkitC SDK\include 中，可將此路徑新增到 Microsoft Visual C++ 6.0 程式專案裏。加入方法為選擇 Tools->Options->Directories，將 C:\Program Files\Safenet\ProtectToolkit C SDK\include 新增到 include 路徑中。

## 1.3 實驗範例

### 1.3.1 設定 Admin Token PIN 範例

在安裝完 Protect Toolkit C 之後，接下來第一步就是要設定 ASO 及 Administrator 的 PIN 碼，以便將來設定 HSM 所用。此設定可由 *ctconf* 工具程式達成。詳細指令說明請參考 SafeNet 文件 *Protect Toolkit C Administration Manual*

第 7 章。

1. 開啟命令提示字元視窗，輸入 *ctconf* 後按下 enter。
2. 輸入 ASO 的 PIN 碼，並依照提示再輸入一次。
3. 輸入 Administrator 的 PIN 碼，並依照提示再輸入一次。

程序及結果如表 1-1 所示。

表 1-1: 設定 Admin Token PIN

Microsoft Windows XP [版本 5.1.2600] (C) Copyright 1985-2001 Microsoft Corp.  C:\Documents and Settings\allen>ctconf ProtectToolkit C Configuration Utility \$Revision: 5624 \$
---

Copyright (c) Safenet, Inc. 2006

Initializing Admin Token SO and Administrator PINs (Device 0, S/N: 0000):

Please enter new Admin SO pin:

Please confirm new Admin SO pin:

Please enter new Administrator's pin:

Please confirm new Administrator's pin:

Current Adapter Configuration for Device 0:

Model : 0000

Serial Number : 0000

Adapter Clock : 24/03/2009 01:13:23 (+8:00)

Battery Status : GOOD

Security Mode : Default (No flags set)

Transport Mode : None

FM Support : Disabled

Open Session Count: 0

Number of Slots : 1

RTC Adjustment Access Control: Not Supported

C:\Documents and Settings\allen>

### 1.3.2 新增user slot 範例

PSE 在初始狀態時只有 3 個 slot，分別為一個 user slot，一個 smart card slot 及一個 Admin slot。其中 smart card slot 只有在讀卡機接上 PSE 時才會出現。而 Protect Toolkit C 提供新增 user slot 功能，以供不同使用者使用。新增 slot 功能可由 *ctconf* 工具程式加上 -c 參數來達成。設定過程中要提供 Administrator 的 PIN，驗證過後才會執行新增。執行結果如表 1-2 所示。其中 -c 後緊連的 1 表示要新增 1 個 user slot。 *Ctstat* 可查詢目前 PSE 所具有的 slot 之資訊。

表 1-2: 新增 user slot

```
C:\Documents and Settings\allen>ctconf -c l
ProtectToolkit C Configuration Utility $Revision: 5624
Copyright (c) Safenet, Inc. 2006

Please enter Administrator's pin (Device 0, S/N: 000)

C:\Documents and Settings\allen>ctstat ProtectToolkit
C Status Utility $Revision: 28 $ Copyright (c)
Safenet, Inc. 2006
ShowAllSlots:3 slots, 3 with tokens
Slot ID 0
    Description      : SAFENET Software Only.:28654
    Manufacturer     : SAFENET Pty. Ltd.
    Hardware Version : 0.00
    Firmware Version : 0.00
Token for Slot ID 0
    Label            :
    Manufacturer     : SAFENET Pty. Ltd.
    Model            : 0000
    Serial Number    : 0000:28654
    Hardware Version : 0.00
    Firmware Version : 0.00
Slot ID 1
    Description      : SAFENET Software Only.:54142
    Manufacturer     : SAFENET Pty. Ltd.
    Hardware Version : 0.00
    Firmware Version : 0.00
Token for Slot ID 1
    Label            :
    Manufacturer     : SAFENET Pty. Ltd.
    Model            : 0000
    Serial Number    : 0000:54142
    Hardware Version : 0.00
    Firmware Version : 0.00
Slot ID 2
```

```

Description      : SAFENET Software Only.
Manufacturer     : SAFENET Pty. Ltd.
Hardware Version : 0.00
Firmware Version : 0.00
Token for Slot ID 2
Label           : AdminToken (0000)
Manufacturer    : SAFENET Pty. Ltd.
Model           : 0000
Serial Number   : 0000
Hardware Version : 0.00
Firmware Version : 0.00
C:\Documents and Settings\allen>

```

### 1.3.3 Token 初始化範例

Token 在使用前必須先初始化。Token 的初始化含設定 SO 的 PIN 碼及 token 的 label，此設定必須由 Administrator 來執行。另外 Token 的初始化也包含設定 user 的 PIN 碼。此設定由誰執行要視安全政策中的“**No clear PINs**”旗號的設定值而定，若“**No clear PINs**”為 true，則由 Administrator 來執行；若“**No clear PINs**”為 false，則由 SO 來執行。**No clear PINs** 是 Protect Toolkit C 中安全政策中的一個旗號，如果此旗號設為 true，則所有傳輸於主機和 PSE 之間的敏感性資料皆會加密，以確保傳輸通道的安全。

設定 SO 的 PIN 碼及 token 的 label 請在命令提示字元視窗輸入 `ctconf -n0`，此例即會初始化位於 slot 0 的 token 0。如表 1-3 所示，先設定 token 的 label，接著設定 SO 的 PIN 碼。Token 可以重覆的初始化，只要使用 -r 參數，但是每一次的初始化都會將 token 上所有的物件刪除。

表1-3:初始化token

```

C:\Documents and Settings\allen>ctconf -n0
ProtectToolkit C Configuration Utility $Revision: 686 $
Copyright (c) Safenet Technologies 2003

```

```
Please enter the new token label: token0
```

```
Please enter Security Officer's pin:
```

```
Please confirm Security Officer's pin:
```

```
C:\Documents and Settings\allen >
```

設定user 的PIN 則使用 *ctkmu* 工具程式。例如 *ctkmu p -s0* 即會設定 token 0 的user PIN。如表1-4 所示，設定user PIN 前需先提供SO 的PIN。

表1-4:設定user PIN

```
C:\Documents and Settings\LILUN.LIN>ctkmu p -s0
```

```
ProtectToolkit C Key Management Utility $Revision: 445 $
```

```
Copyright (c) Safenet Technologies 2002-2005
```

```
The User PIN is currently uninitialised. Only Security Officer may  
initialise the User PIN.
```

```
Please enter Security Officer PIN for token in Slot 0:
```

```
Please enter the new user PIN for token in Slot 0:
```

```
Please confirm the new user PIN for token in Slot 0:
```

```
C:\Documents and Settings\allen >
```

### 1.3.4 軟體環境設定範例

程式開發及開發環境設定步驟如下：

1. 開啟Microsoft Visual C++ 6.0。
2. 選擇 File->New..後出現新增專案視窗，專案類型選擇 Win32 Console Application，並在Project name 輸入專案名稱，此處為PSE\_TEST，輸入完成後按下OK 鍵。如圖1-2。

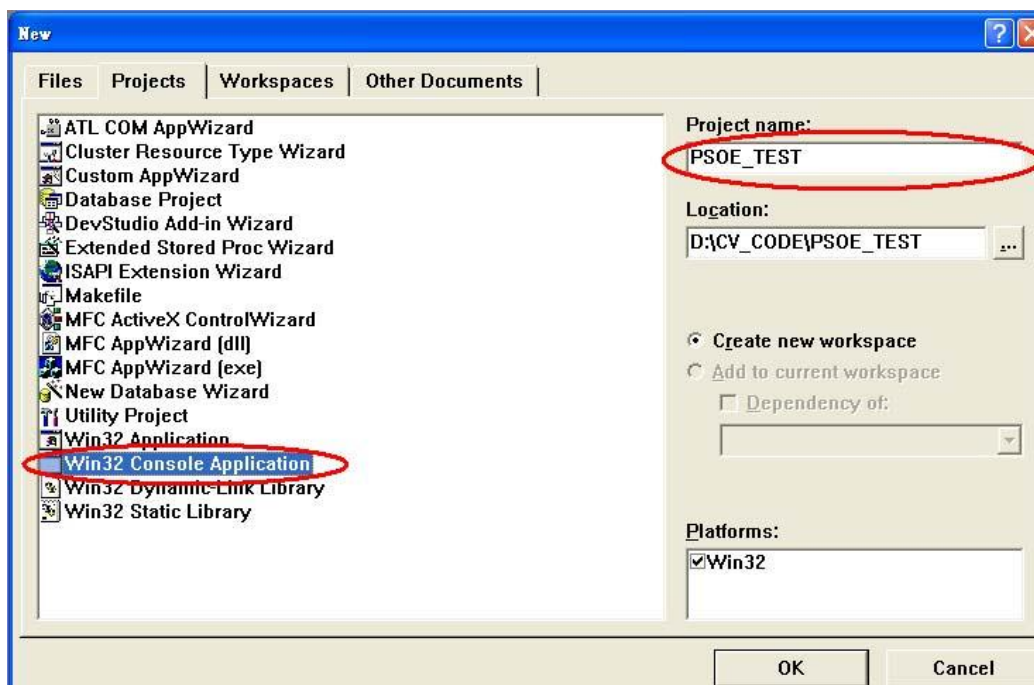


圖 1-2:新增專案

3. 接下來選擇 A simple application 後按下 Finish。如圖 1-3
4. 進入專案後即可在 .cpp 檔案中輸入程式碼。如圖 1-4



ProtectServer External

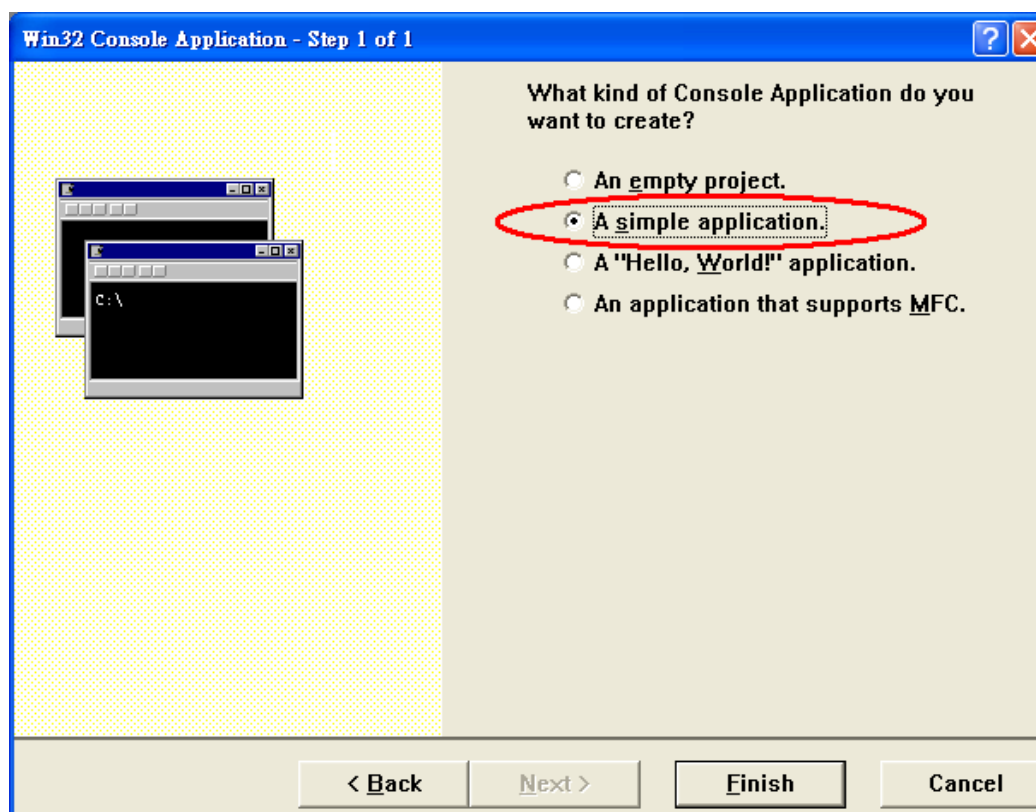


圖 1-3:選擇A simple application

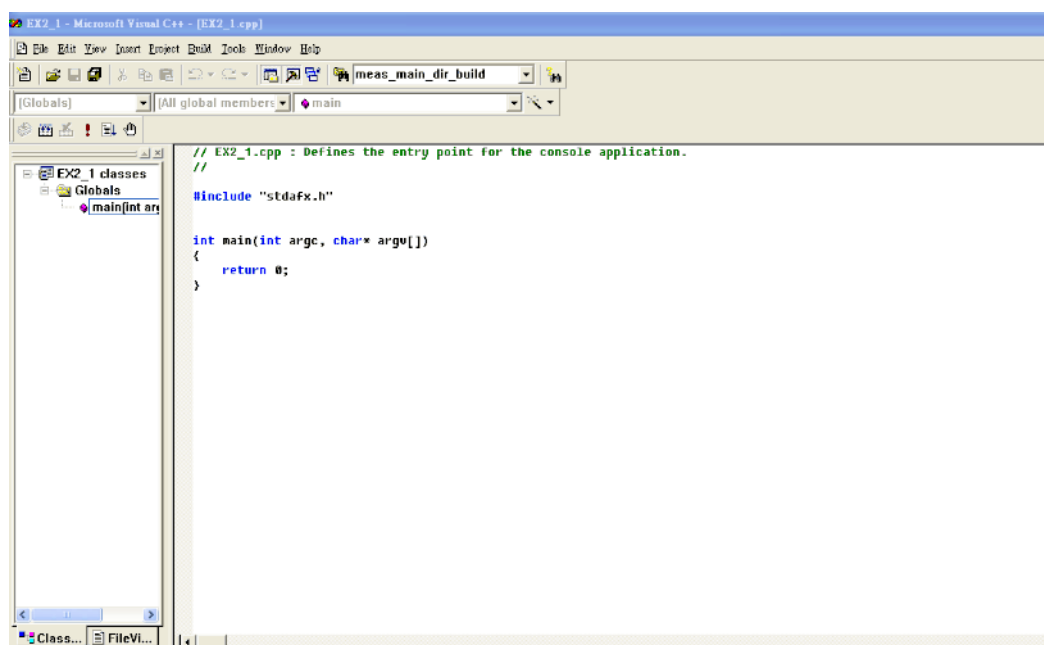


圖1-4: 開發專案環境

- 選擇 Project->Settings->C/C++ tab 中選擇 Category 為 General，並在 Preprocessor definitions 欄位中加上“\_WINDOWS”定義。如圖1-5 所示。

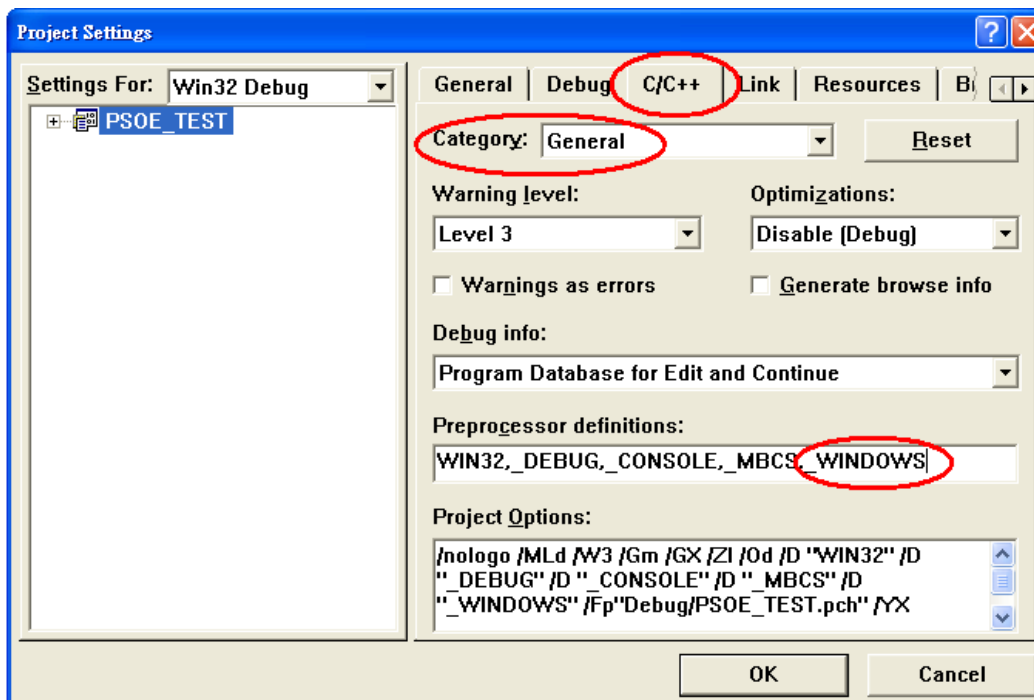


圖1-5:加入\_WINDOWS 定義

- 在 Link tab 中選擇 Category 為 General，並在其 Object/library modules 欄位中加上 cryptoki2.lib，完成後請按 OK 鍵。如圖1-6。
- 將 C:\Program Files\Safenet\ProtectToolkit C SDK\lib 下的檔案 cryptoki2.lib 複製至專案所在的目錄內。如圖1-7。



ProtectServer External



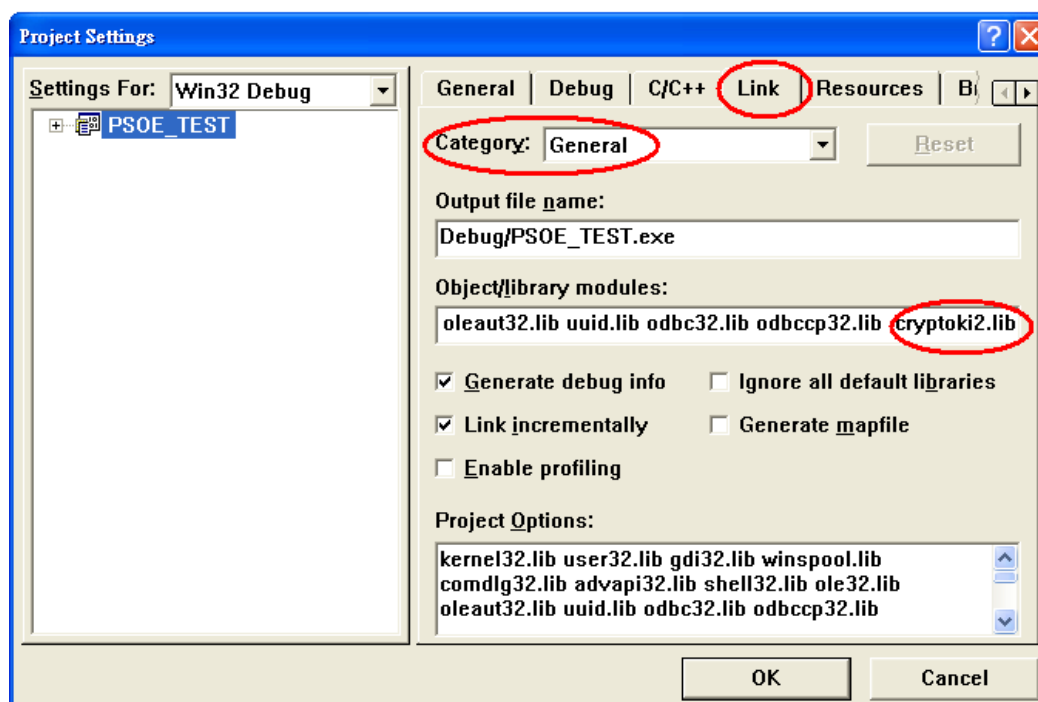


圖 1-6: 加入 cryptoki2.lib

名稱	大小	類型	修改日期
Debug		檔案資料夾	2007/4/24 下午 02:13
main.c	0 KB	C 檔案	2007/4/24 下午 02:24
PSOE_TEST.dsp	5 KB	Project File	2007/4/24 下午 02:13
PSOE_TEST.dsw	1 KB	Project Workspace	2007/4/24 下午 02:13
PSOE_TEST.ncb	25 KB	NCB 檔案	2007/4/24 下午 02:13
cryptoki2.lib	17 KB	LIB 檔案	2005/10/19 下午 04:...

圖 1-7: 複製 cryptoki2.lib

8. 輸入程式 EX1\_1，或從光碟載入程式碼。輸入完成後按下工具列上的 Go 或是 F5 編譯程式。如果設定及輸入都正確的話，就可編譯及執行成功。

### EX1\_1 開發環境測試程式

```
#include "cryptoki.h"
```

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    rv = C_Initialize(NULL);

    rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

    rv = C_CloseSession(hSession);

    rv = C_Finalize(NULL);

    return 0;
}
```

## 1.4 習題

1. 如何更改PSE 的操作模式。分別說明從software-only 模式轉換成網路操作 模式及從網路操作模式轉換成software-only 模式？
2. 在PSE 中可能會出現那三種型式的slot？
3. 請使用ctconf 工具程式新增1 個user slot，並初始化其token。
4. 參考Safenet 文件*Protect Toolkit C Administration Manual* 第7 章的ctconf 指令說明，將 No Clear PIN 致能。接下來再新增一個 slot，並初始化其 token。並比較和習題三的有何不同。
5. 如何使用ctconf 工具程式刪除slot。



ProtectServer Gold

## 實驗2. 建立連線及初始化token

### 2.1 實驗目的

本實驗主要了解Cryptoki 應用程式的撰寫架構，以及在PKCS#11 有關連線建立及初始化token 的函式。

### 2.2 相關原理說明

#### 2.2.1 程式流程

程式在開發時固定會使用到一些函式，如下所述：

1. C\_Initialize: 用來初始化Cryptoki 函式庫，只要應用程式一呼叫此函式即變成Cryptoki 應用程式。
2. C\_OpenSession: 用來建立唯讀連線或是可讀可寫連線，並取得 session 的handler。
3. C\_CloseSession: 關閉連線。
4. C\_Finalize: 釋放Cryptoki 函式庫。

以上僅列出函式名稱，詳細的函式原型請參考 PKCS#11v2.10。程式流程如圖 2-1 所示。程式最先呼叫 C\_Initialize，再呼叫 C\_OpenSession 建立和 token 之間的連線，接下來就可以執行其它的函式，如加解密等。最後再呼叫 C\_CloseSession 及C\_Finalize 結束程式。



ProtectServer Gold

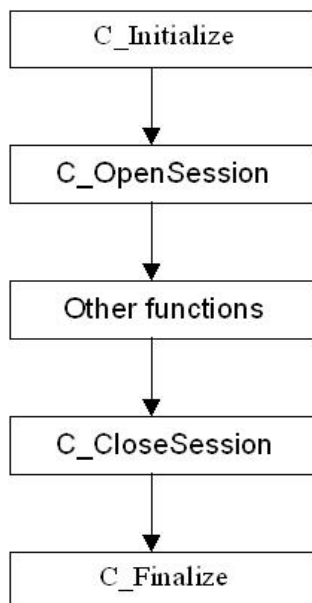


圖2-1:應用程式流程

### 2.2.2 Token 初始化函式

Token 除了可以用ctconf 工具程式執行初始化，也可以使用Cryptoki 的函式來做初始化。執行初始化所需的函式如下：

1. C\_InitToken: 初始化token。
2. C\_InitPIN: 初始化user PIN。
3. C\_SetPIN: 修改user PIN。
4. C\_Login: 登入到user 或是SO。

## 2.3 實驗範例

程式 EX2\_1 執行 token 的初始化，首先決定 user 及 SO 的 PIN 碼 [EX2\_1(1)(2)]，如果此 token 之前已經被初始化過，則SO 的PIN 要給定上一次初始化所設定的PIN。如果此token 為新建立還未初始化過的，則SO 的PIN 可自行設定。PIN 的長度為 1 到 32 個字元。呼叫C\_Initialize 初始化cryptoki[EX2\_1(3)]。接下來設定 token 的 label 值，label 的值限定為 32 位元組長，不足的部分要補空白字元 (ASCII 值 0x20)，從[EX2\_1(4)]開始連續 3 行即為設定合法的 label 值。此時呼叫 C\_InitToken 初始化token，此處指定初始化 slot 0 並給定 SO 的 PIN 及 label[EX2\_1(5)]。接下來要進行 user PIN 的設定，設定 user PIN 是由 SO 來執行。因此要先開啟連線，再登入成 SO 即可設定 user PIN。程式

[EX2\_1(6)]即為開啟連線，由於C\_InitPIN 只能在R/W session 下執行，所以開啟 session 時要設定成 R/W，即使用 CKF\_RW\_SESSION 參數。開啟連線完即可呼叫C\_Login 以登入成 SO，[EX2\_1(7)]。由於要登入成 SO 所以要給定參數 CKU\_SO，並給定前面初始化 token 時所設定 SO 的 PIN，如驗證成功即可成功登入。這時就可呼叫 C\_InitPIN 來設定 user PIN，並且須要提供所要設定的 PIN 碼[EX2\_1(8)]。經過此範例程式即可初始化 token 0，並設定 token 0 之 SO 及 user 的 PIN 碼。

### EX2\_1 Token 初始化

```
#include "cryptoki.h"

#include <stdio.h>

#include <string.h>

#define BLANK_CHAR 0x20

int main(int argc, char* argv[])
{
    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    CK_CHAR label[32];

    CK_CHAR userPin[] = "0000"; (1)

    CK_CHAR soPin[] = "1234"; (2)

    printf("start of initialing token\n");

    rv = C_Initialize(NULL); (3)

    memset(label,BLANK_CHAR,sizeof(label)); (4)

    strcpy(label,"token0");

    label[strlen(label)] = BLANK_CHAR;

    rv = C_InitToken(slotId, soPin, strlen((char*)soPin), label); (5)

    if (rv != CKR_OK) {

        printf("Could not initialize token %d,error code = %x\n",slotId,rv);

    }
}
```

```
rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession); (6)

if (rv != CKR_OK) {

    printf("Could not open session on token %d,error code =
        %x\n",slotId,rv);

}

rv = C_Login(hSession,CKU_SO,soPin,strlen((char*)soPin)); (7)

if (rv != CKR_OK) {

    printf("Could not login SO on token %d,error code = %x\n",slotId,rv);

}

rv = C_InitPIN(hSession, userPin, strlen((char*)userPin)); (8)

if (rv != CKR_OK) {

    printf("Could not initialize user pin on token %d,error code =
        %x\n",slotId,rv);

}

rv = C_CloseSession(hSession);

rv = C_Finalize(NULL);

printf("end of initialing token\n");

return 0;

}
```

## 2.4 習題

1. 如果將在1.3.3 節中所提到的No Clear PINs 旗號設定及清除,則程式EX2\_1 結果為何。
2. 在程式EX2\_1 中,為何C\_InitToken 函式可以在C\_OpenSession 函式之前。
3. 為何設定user PIN 的時候,session 要開啟成R/W 狀態。
4. 請撰寫程式使用Cryptoki 中的slot 管理函式C\_GetSlotInfo 取得slot 0 的資訊(函式原型請參考PKCS#11v2.10)。
5. 請撰寫程式使用Cryptoki 中的token 管理函式C\_GetTokenInfo 取得token 0 的資訊(函式原型請參考PKCS#11v2.10)。

## 實驗3. 建立資料及憑證物件

### 3.1 實驗目的

本實驗主要了解物件(object)的建立及刪除，包含了資料物件(data object)及憑證物件(certificate object)。

### 3.2 相關原理說明

在 PKCS#11 中，token 就是存有物件的裝置。PKCS#11 定義了三種物件的類別，分別是資料物件、憑證物件及金鑰物件。而物件其實就是一些屬性(attribute)的集合，以說明此物件所具有的特性。物件的建立使用下列的 Cryptoki 函式

```
CK_RV  
C_CreateObject( CK_SESSION_HA  
NDLE hSession, CK_ATTRIBUTE_PTR  
pTemplate, CK_ULONG ulCount,  
CK_OBJECT_HANDLE_PTR phObject  
);
```

hSession: 所建立session 的handle。

pTemplate: 所要建立物件的屬性樣板。  
ulCount: 屬性的個數。

phObject: 建立物件後所傳回的物件handle。

物件的刪除使用下列函式：

```
CK_RV C_DestroyObject(  
CK_SESSION_HANDLE hSession,  
CK_OBJECT_HANDLE hObject  
);
```

hSession: 所建立session 的handle。

hObject: 所要刪除物件的handle。

而在建立物件的時候要給定物件的樣板(Template)，Cryptoki 會依此樣板建立物件。因此要先宣告一個資料型態為CK\_ATTRIBUTE的樣板變數。

CK\_ATTRIBUTE 定義如下：

```
typedef struct CK_ATTRIBUTE
{ CK_ATTRIBUTE_TYPE type;
  CK_VOID_PTR pValue;
  CK_ULONG ulValueLen;
} CK_ATTRIBUTE;
```

第一個參數 `type` 為物件屬性的型態，因為物件是一組屬性的集合，所以此為其中一個屬性。第二個參數為 `pValue`，為此屬性的值。第三個參數為 `ulValueLen`，為屬性的值的長度。

三種物件型態皆可用上述的函式建立及刪除，而金鑰物件除了可用 `C_CreateObject` 建立外，`Cryptoki` 函式也提供 `C_GenerateKey` 函式以建立金鑰物件。

而所有的物件皆有下列的共同屬性：

Attribute	Data Type	Meaning
CKA_TOKEN	CK_BBOOL	CK_TRUE : token 物件 CK_FALSE : session 物件
CKA_PRIVATE	CK_BBOOL	CK_TRUE : 私密物件 CK_FALSE : 公開物件
CKA_MODIFIABLE	CK_BBOOL	CK_TRUE : 物件可以被更改
CKA_LABEL	RFC2279 string	用來描述物件



### 3.3 實驗範例

#### 3.3.1 建立資料物件

程式 EX3\_1 為建立資料物件的程式碼，資料物件可以讓使用者儲存自行定義的資料。在程式中要決定所以建立物件的型態為資料物件 [EX3\_1(1)]，以及要儲存的資料[EX3\_1(2)]及 label[EX3\_1(3)]。建立連線之後就可以建立資料物件的樣板[EX3\_1(4)]，在這個樣板裏面共有四個屬性。第一個屬性的型態為 CKA\_CLASS，決定物件的種類，給定值為 objClass，其值已於前面定義為資料物件， [EX3\_1(1)]，再來給定此屬性值的長度為 sizeof(objClass)，sizeof 為計算變數的長度的函式。第二個屬性為 CKA\_VALUE，為物件的值，其值已於前面定義[EX3\_1(2)]。第三個屬性為 CKA\_LABEL，決定物件的標籤，在給定其值的長度的時候要注意需不包含字串的停止字元，因此在用 sizeof 計算長度後要減 1，其值定義於 [EX3\_1(3)]。第四個屬性為 CKA\_TOKEN，決定物件為 token 物件或是 session 物件，在此設為TRUE，即為token 物件。接下來計算此樣板裏面的屬性個數 [EX3\_1(5)]，在此程式計算出來的結果為 4。此時就可以呼叫 C\_CreateObject 函式建立物件[EX3\_1(6)]。

#### EX3\_1 建立資料物件

```
#include <stdio.h>

#include <string.h>

#include "cryptoki.h"

int main(int argc, char* argv[])
{
    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hObject = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    CK_OBJECT_CLASS objClass = CKO_DATA;                                (1)

    CK_CHAR objValue[] = "your data of the object";                    (2)

    CK_CHAR objLabel[] = "DataObject";                                (3)
```

```

CK_BBOOL boolTrue = TRUE;

rv = C_Initialize(NULL);

rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

if (rv != CKR_OK) {

    printf("Could not open session on token %d,error code =
           %x\n",slotId,rv);

    return 0;

}

CK_ATTRIBUTE dataTemplate[] = {                                     (4)

    {CKA_CLASS, &objClass, sizeof(objClass)},

    {CKA_VALUE, objValue, sizeof(objValue)},

    {CKA_LABEL, objLabel, sizeof(objLabel)-1},

    {CKA_TOKEN, &boolTrue, sizeof(boolTrue)}

};

CK_SIZE objectSize = sizeof(dataTemplate) / sizeof(CK_ATTRIBUTE);    (5)

rv = C_CreateObject(hSession, dataTemplate, objectSize, &hObject); (6)

if (rv != CKR_OK) {

    printf("Could not create data boject on token %d,error code =
           %x\n",slotId,rv);

    return 0;

}

rv = C_CloseSession(hSession);

rv = C_Finalize(NULL);

return 0;

}

```

### 3.3.2 建立憑證物件

程式 EX3\_2 為建立憑證物件的程式碼。在執行本程式之前要先準備一憑證檔，以供 PSE 建立，也可以使用在 EX3\_2 程式目錄裏面所提供的 cert1.cer 憑證檔案。首先決定所要建立物件的型態為憑證物件[EX3\_2(1)]。憑證的格式為

X.509 憑證[EX3\_2(2)]、label[EX3\_2(3)]、憑證的主旨[EX3\_2(4)]及憑證的ID[EX3\_2(5)]。並宣告一個陣列存放憑證檔，大小為 2K，因此提供的憑證檔要小於 2K，如果有較大的憑證檔請更改所需的大小[EX3\_2(6)]。接下來要讀取憑證檔案的資料到程式裏面，因此要給定憑證檔的路徑，如果不是存放在程式EX3\_2的目錄裏面，請更改其路徑[EX3\_2(7)]。[EX3\_2(8)]為開啟憑證檔。[EX3\_2(9)]為計算憑證檔的長度，並將其讀取到 certificate 陣列中。建立連線之後就可以建立資料物件的樣板[EX3\_2(10)]，在這個樣板裏面共有七個屬性，如下：

CKA\_CLASS: 決定物件的種類為憑證物件。  
 CKA\_CERTIFICATE\_TYPE: 決定憑證的格式為X.509。  
 CKA\_TOKEN: 決定此物件為token物件。  
 CKA\_LABEL: 物件的標籤，定義於[EX3\_2(3)]。  
 CKA\_SUBJECT: 物件的主旨，定義於[EX3\_2(4)]。  
 CKA\_ID: 物件的ID，定義於[EX3\_2(5)]。  
 CKA\_VALUE: 憑證的值，存放於certificate 陣列中。

接下來計算此樣板裏面的屬性個數後，就可以呼叫 C\_CreateObject 函式建立憑證物件[EX3\_2(11)]。

### EX3\_2 建立憑證物件

```
#include <stdio.h>

#include <string.h>

#include "cryptoki.h"

int main(int argc, char* argv[])
{
    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hObject = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    CK_OBJECT_CLASS objClass = CKO_CERTIFICATE;           (1)

    CK_CERTIFICATE_TYPE certType = CKC_X_509;             (2)
```

```
CK_CHAR label[] = "A certificate object"; (3)

CK_BYTE subject[] = "testing certificate"; (4)

CK_BYTE id[] = "123"; (5)

CK_BYTE certificate[2048]; (6)

CK_BBOOL bTrue = TRUE;

FILE *CertFile; (7)

char cert_path[] = ".\\cert1.cer";

int cert_len=0;

int data_start;

int data_end;

int i;

if ((CertFile = fopen(cert_path,"rb")) == NULL) { (8)

    printf("open cert file fail!\n");

    return 0;

}

data_start = ftell(CertFile); (9)

fseek(CertFile,0L,SEEK_END);

data_end = ftell(CertFile);

cert_len = data_end - data_start;

rewind(CertFile);

for (i = 0;i<cert_len;i++) {

    certificate[i] = fgetc(CertFile);

}

rv = C_Initialize(NULL);

rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

if (rv != CKR_OK) {

    printf("Could not open session on token %d,error code =

        %x\n",slotId,rv);

    return 0;

}
```

```
CK_ATTRIBUTE certTemplate [] =  
    {                                                                    (10)  
        {CKA_CLASS, &ObjClass, sizeof(ObjClass)},  
        {CKA_CERTIFICATE_TYPE, &certType, sizeof(certType)},  
        {CKA_TOKEN, &bTrue, sizeof(CK_BBOOL)},  
        {CKA_LABEL, label, sizeof(label)-1},  
        {CKA_SUBJECT, subject, sizeof(subject)},  
        {CKA_ID, id, sizeof(id)},  
        {CKA_VALUE, certificate, cert_len}  
    };  
  
CK_SIZE objectSize = sizeof(certTemplate) / sizeof(CK_ATTRIBUTE);  
rv = C_CreateObject(hSession, certTemplate, objectSize, &hObject); (11)  
  
if (rv != CKR_OK) {  
    printf("Could not create certificate object on token %d,error code =  
        %x\n",slotId,rv);  
    return 0;  
}  
  
FILE *pfx_File;  
  
char pfx_path[] = ".\\test.pfx";  
  
if ((pfx_File = fopen(pfx_path,"wb")) == NULL) {  
    printf("create pfx file fail!\n");  
    return 0;  
}  
  
for (int i=0;i<pfxfile_len;i++) {  
    fputc((char)pfxfile[i],pfx_File);  
}  
  
fclose(pfx_File);  
  
rv = C_CloseSession(hSession);  
  
rv = C_Finalize(NULL);
```

```
    return 0;  
}
```

### 3.3.3 習題

1. 請說明於程式範例中所建立的資料物件及憑證物件為公開物件或是私密物件。
2. 請用 Safenet 所提供的 KMU 軟體或是 Browser 軟體觀察是否有成功建立物件，並確認屬性值是否和在程式中所設定的相同。
3. 請自行建立一私密資料物件或是私密憑證物件。
4. 請使用 C\_DestroyObject 函式刪除所建立的物件。



ProtectServer Gold

## 實驗4. 建立金鑰物件

### 4.1 實驗目的

本實驗在於了解如何在已知金鑰值的情形下建立金鑰物件，包含對稱式及非對稱式金鑰。

### 4.2 相關原理說明

加密元件一般分對稱式(symmetric)與非對稱式(asymmetric)二類。非對稱式也可稱為公開金鑰(public key)密碼系統。對稱式加密系統是指用相同之秘密金鑰  $K_s$  (Secret Key) 來做加解密處理，如圖 4-1。對稱式加密系統繁多，常見的有 DES、AES 等。

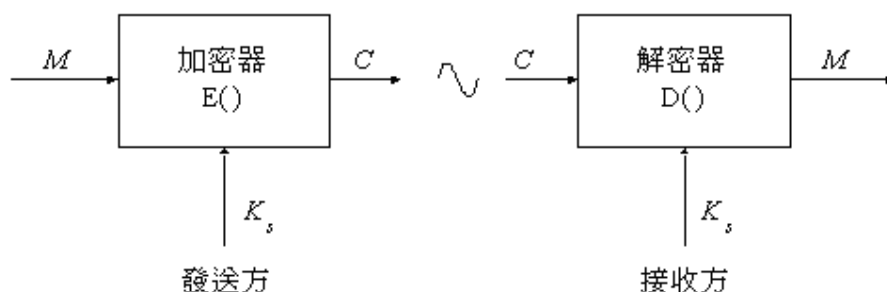


圖4-1:對稱式秘密金鑰加密系統

公開金鑰密碼系統如圖 4-2 所示，加密與解密用不同之金鑰  $K_1$ 、 $K_2$ ，故稱為非對稱密碼系統。非對稱式密碼系統的使用者，擁有由特定秘密參數所推導出來之兩把金鑰。若未知此秘密參數，則無法由其中一把金鑰推導出另一把金鑰。此二把金鑰可將其一設為可公開者，即稱為公鑰(Public Key)，另一設為不可公開者，即稱為私鑰(Private Key)。若希望做公鑰加密時，則於圖 1-5 中  $K_1$  為接收者的公鑰、 $K_2$  則為接收方的私鑰。若需要做數位簽章時，則  $K_1$  為簽署者的私鑰， $K_2$  為簽署者的公鑰。目前較常用的為 RSA 公開金鑰密碼系統。

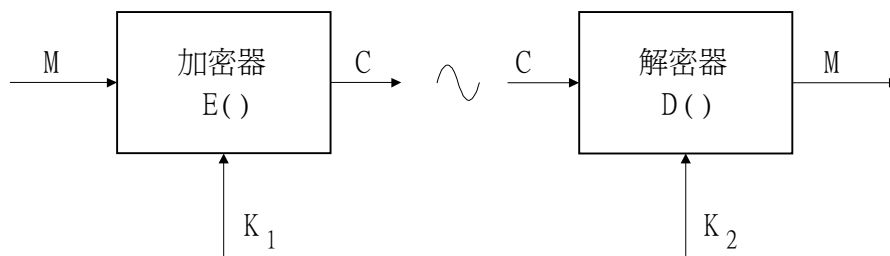


圖4-2:非對稱式公鑰加密系統

在 PKCS#11 中，金鑰物件又可分為秘密金鑰、公開金鑰及私密金鑰物件。每種金鑰物件，也會依不同的密碼演算法而有不同的屬性。一般而言如果在已知金鑰值的情況下，想建立金鑰物件，可以使用 `C_CreateObject` 函式。如果想由亂數產生金鑰值的話，則可以使用 `C_GenerateKey` 來產生金鑰物件，此部分將在實驗 5 中介紹。

## 4.3 實驗範例

### 4.3.1 建立秘密金鑰物件

程式 EX4\_1 為建立秘密金鑰物件的程式碼。首先決定所要建立物件的型態為秘密金鑰物件[EX4\_1(1)]。秘密金鑰的種類為 Triple DES 所需的 2 keys[EX4\_1(2)]及金鑰 label[EX4\_1(3)]。由於 Triple DES with 2 keys 的長度含同位元檢查碼為 16 位元組，因此需宣告一個陣列存放金鑰值，大小為 16 位元組[EX4\_1(4)]。如果前面所採用的秘密金鑰種類為 Triple DES with 3 keys，則需要 24 位元組。若為 DES 金鑰則需 8 位元組。一般來說 DES 的金鑰應該要填入正確的同位元檢查位元，但在 Protect Toolkit C 中，錯誤的檢查碼並不會回傳錯誤訊息。接下來建立連線之後就可以建立秘密金鑰物件的樣板[EX4\_1(5)]，在這個樣板裏面共有六個屬性，如下：

CKA\_CLASS: 決定物件的種類為秘密金鑰物件。

CKA\_KEY\_TYPE: 決定秘密金鑰種類為 Triple DES with 2 keys。

CKA\_TOKEN: 決定此物件為 token 物件。

CKA\_LABEL: 物件的標籤，定義於[EX4\_1(3)]。

CKA\_ENCRYPT: 決定此金鑰是否可以拿來加密資料，此處為 TRUE。

CKA\_VALUE: 金鑰的值，定義於[EX4\_1(4)]。



接下來計算此樣板裏面的屬性個數後，就可以呼叫C\_CreateObject 函式建立秘密金鑰物件[EX4\_1(6)]。

#### EX4\_1 建立秘密金鑰物件

```
#include <stdio.h>

#include <string.h>

#include "cryptoki.h"

int main(int argc, char* argv[])
{
    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hObject = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    CK_OBJECT_CLASS objClass = CKO_SECRET_KEY;                (1)

    CK_KEY_TYPE keyType = CKK_DES2;                            (2)

    CK_CHAR label[] = " Triple DES with 2 keys";              (3)

    CK_BYTE value[16] = {0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,
                        0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02}; (4)

    CK_BBOOL bTrue = TRUE;

    rv = C_Initialize(NULL);

    rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

    if (rv != CKR_OK) {

        printf("Could not open session on token %d,error code =
                %x\n",slotId,rv);

        return 0;

    }

    CK_ATTRIBUTE secretkeytemplate[] = {                      (5)

        {CKA_CLASS, &ObjClass, sizeof(ObjClass)},

        {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
```

```

        {CKA_TOKEN, &bTrue, sizeof(bTrue)},
        {CKA_LABEL, label, sizeof(label)-1},
        {CKA_ENCRYPT, &bTrue, sizeof(bTrue)},
        {CKA_VALUE, value, sizeof(value)}
    };

    CK_SIZE objectSize = sizeof(secretkeytemplate) / sizeof(CK_ATTRIBUTE);
    rv = C_CreateObject(hSession, secretkeytemplate, objectSize, &hObject);
                                                                    (6)

    if (rv != CKR_OK) {
        printf("Could not create secret key object on token %d,error code =
            %x\n",slotId,rv);

        return 0;
    }

    rv = C_CloseSession(hSession);

    rv = C_Finalize(NULL);

    return 0;
}

```

### 4.3.2 建立公開金鑰及私密金鑰物件

程式 EX4\_2 可建立 RSA 公私鑰對。首先決定所要建立物件的型態為公開及私密金鑰物件[EX4\_2(1)]。金鑰對的種類為RSA[EX4\_2 (2)]及公私鑰的label[EX4\_2 (3)]。接下來要定義RSA 金鑰對的金鑰值[EX4\_2 (4)]，在此定義的為 512 位元(64 位元組)長度的金鑰對。由於定義金鑰值的資料型態為字串陣列，因此要將其轉成位元組陣列[EX4\_2 (5)]，Protect Toolkit C有提供一轉換的函式hex2bin，包在 util.lib 中，使用前要先將此檔名放在連結路徑中方可使用。接下來建立連線之後就可以建立秘密金鑰物件的樣板[EX4\_2(6)]，如下：

CKA\_CLASS: 決定物件的種類為公開金鑰物件。

CKA\_KEY\_TYPE:決定金鑰種類為RSA。

CKA\_TOKEN: 決定此物件為token物件。

CKA\_LABEL: 公鑰物件的標籤。

CKA\_WRAP: 決定此金鑰是否可以拿來加密其它的金鑰，此處為TRUE。

CKA\_ENCRYPT: 決定此金鑰是否可以拿來加密資料，此處為TRUE。

CKA\_MODULUS: 模數 $n$ 。

CKA\_PUBLIC\_EXPONENT: public exponent  $p$ 。

再來計算樣板裏面的屬性個數後，就可以呼叫 C\_CreateObject 函式建立公鑰物件[EX4\_2(7)]。接下來定義私鑰物件的樣板[EX4\_2 (8)]，如下：

CKA\_CLASS: 決定物件的種類為私密金鑰物件。

CKA\_KEY\_TYPE: 決定金鑰種類為RSA。

CKA\_TOKEN: 決定此物件為token物件。

CKA\_LABEL: 私鑰物件的標籤。

CKA\_SENSITIVE: 決定是否為敏感性資料，此處為TRUE。若設為TRUE，建立物件後將無法觀測其值。

CKA\_DECRYPT: 決定此金鑰是否可以拿來解密資料，此處為TRUE。

CKA\_SIGN: 決定此金鑰是否可以拿來簽章，此處為TRUE。

CKA\_MODULUS: 模數。

CKA\_PUBLIC\_EXPONENT: public exponent  $p$ 。

CKA\_PRIVATE\_EXPONENT: private exponent  $d$ 。

CKA\_PRIME\_1: prime  $p$ 。

CKA\_PRIME\_2: prime  $q$ 。

CKA\_EXPONENT\_1: private exponent  $d$  modulo  $p-1$

CKA\_EXPONENT\_2: private exponent  $d$  modulo  $q-1$

CKA\_COEFFICIENT: CRT coefficient  $q^{-1} \bmod p$

再來計算樣板裏面的屬性個數後，就可以呼叫C\_CreateObject 函式建立私鑰物件[EX4\_2(9)]。

#### EX4\_2 建立公開及私密金鑰物件

```
#include <stdio.h>

#include <string.h>

#include "cryptoki.h"
```

```

#include "hex2bin.h"

int main(int argc, char* argv[])
{
    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hObject = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    CK_OBJECT_CLASS pubObjClass = CKO_PUBLIC_KEY;                (1)

    CK_OBJECT_CLASS priObjClass = CKO_PRIVATE_KEY;

    CK_KEY_TYPE keyType = CKK_RSA;                                (2)

    CK_CHAR publabel[] = "An RSA public key object";              (3)

    CK_CHAR prilabel[] = "An RSA private key object";

    char modulus_str[] = "D83A4A2B07CC609769755EE44E2200CCCCFE71    (4)
    CADEFB5DB4740BAB3BC70B7AFD1FBAF288E4525A24DC80F09EA1763C5189FC5A9E64C
    83CC0064DEEEFF";

    CK_BYTE modulus[64] = {0};

    CK_BYTE pub_exponent[2] = {0xB6, 0x09};

    char pri_exponent_str[] = "B87B02BFB19E807884C31712BE75A723960
    CBA58403B2E68B7DAABB2B5AB81FAE80744862A1FBEDFB8ECD0A7E738D9DCA76477C9
    53B53A50628023ED307E10F9";

    CK_BYTE pri_exponent[64] = {0};

    char p_str[] = "FAA71BB8108EB78A311878FACCC8F80BDCF
    2E8163A48885AFDD6A1D9818CB6B5";

    char q_str[] = "CC7FCB6D602AE8115DCE8143C9A85AA43A
    536321736C93171885DC48BB408B63";

    CK_BYTE p[32] = {0};

    CK_BYTE q[32] = {0};

    char dp_str[] = "0D7AA1160E212CDE63CC15A54C5B6294A46
    F436609D8E306C4328E726C878801";

    char dq_str[] = "5619349234B78C8F4EAD775DE73D3F544C8B9

```

```

        60F0EEB53674F97110824176C1";

char qinv_str[] = "AB7AA68B6BB1DD6540D59B08D319C4058F4750
        ABEF209A2E0C5DE507E241E6D8";

CK_BYTE dp[32]={0};
CK_BYTE dq[32]={0};
CK_BYTE qinv[32]={0};
CK_BBOOL bTrue = TRUE;

rv = C_Initialize(NULL);

rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

if (rv != CKR_OK) {
    printf("Could not open session on token %d,error code =
        %x\n",slotId,rv);

    return 0;
}

hex2bin(modulus, modulus_str, 64);
hex2bin(pri_exponent, pri_exponent_str, 64);
hex2bin(p, p_str, 32);
hex2bin(q, q_str, 32);
hex2bin(dp, dp_str, 32);
hex2bin(dq, dq_str, 32);
hex2bin(qinv, qinv_str, 32);

CK_ATTRIBUTE publickeytemplate [] = {
    {CKA_CLASS, &pubObjClass, sizeof(pubObjClass)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &bTrue, sizeof(bTrue)},
    {CKA_LABEL, publabel, sizeof(publabel)-1},
    {CKA_WRAP, &bTrue, sizeof(bTrue)},
    {CKA_ENCRYPT, &bTrue, sizeof(bTrue)},
    {CKA_MODULUS, modulus, sizeof(modulus)},
    {CKA_PUBLIC_EXPONENT, pub_exponent, sizeof(pub_exponent)}
}

```

(5)

(6)

```

};

CK_SIZE objectSize = sizeof(publickeytemplate) / sizeof(CK_ATTRIBUTE);

rv = C_CreateObject(hSession, publickeytemplate, objectSize, &hObject);

(7)

if (rv != CKR_OK) {

    printf("Could not create public key object on token %d,error code =
        %x\n",slotId,rv);

    return 0;

}

CK_ATTRIBUTE privatekeytemplate[] = {

(8)

    {CKA_CLASS, &priObjClass, sizeof(priObjClass)},

    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},

    {CKA_TOKEN, &bTrue, sizeof(bTrue)},

    {CKA_LABEL, prilabel, sizeof(prilabel)-1},

    {CKA_SENSITIVE, &bTrue, sizeof(bTrue)},

    {CKA_DECRYPT, &bTrue, sizeof(bTrue)},

    {CKA_SIGN, &bTrue, sizeof(bTrue)},

    {CKA_MODULUS, modulus, sizeof(modulus)},

    {CKA_PUBLIC_EXPONENT, pub_exponent, sizeof(pub_exponent)},

    {CKA_PRIVATE_EXPONENT, pri_exponent, sizeof(pri_exponent)},

    {CKA_PRIME_1, p, sizeof(p)},

    {CKA_PRIME_2, q, sizeof(q)},

    {CKA_EXPONENT_1, dp, sizeof(dp)},

    {CKA_EXPONENT_2, dq, sizeof(dq)},

    {CKA_COEFFICIENT, qinv, sizeof(qinv)}

objectSize = sizeof(privatekeytemplate) / sizeof(CK_ATTRIBUTE);

rv = C_CreateObject(hSession, privatekeytemplate, objectSize, &hObject);

(9)

if (rv != CKR_OK) {

    printf("Could not create private key object on token %d,error code

```

```
        = %x\n",slotId,rv);

    return 0;

}

rv = C_CloseSession(hSession);

rv = C_Finalize(NULL);

return 0;

}
```

### 4.3.3 習題

1. 金鑰屬性CKA\_SENSITIVE 的作用為何。
2. 如果允許某一金鑰可以用來加密其它的金鑰，則要設定那一屬性。
3. 請建立一把AES 的秘密金鑰，長度為128 位元，金鑰值請自行定義。建立後 將無法觀得其金鑰值，且此金鑰不可用來加密資料及無法再更改任何屬性。
4. 請建立一把 Triple DES with 3 keys 的秘密金鑰。金鑰值請自行定義，並須符合奇同位檢查。建立後將可以觀得其金鑰值，且此金鑰可用來加密資料及更改任何屬性。
5. 請建立一RSA 金鑰對，長度為768 位元。此金鑰對的主要功能為簽章及驗證，金鑰值及各金鑰屬性請自行定義及規畫。



ProtectServer Gold

## 實驗5. 金鑰產生

### 5.1 實驗目的

本實驗在於了解使用 Cryptoki 中產生金鑰物件的函式來產生對稱及非對稱金鑰。

### 5.2 相關原理說明

在Cryptoki 中，用來產生秘密金鑰的函式為 C\_GenerateKey，此函式可亂數產生金鑰值。函式原型如下：

```
CK_RV C_GenerateKey( CK_SESSION_HANDLE hSession,
                     CK_MECHANISM_PTR pMechanism,
                     CK_ATTRIBUTE_PTR pTemplate,
                     CK_ULONG ulCount,
                     CK_OBJECT_HANDLE_PTR phKey)
```

hSession: 所建立session 的handle。

pMechanism: 金鑰產生機制。

pTemplate: 所要建立物件的屬性樣板。

ulCount: 屬性的個數。

phKey: 建立物件後所傳回的物件handle。

其中金鑰產生機制的資料型態為CK\_MECHANISM\_PTR，其定義如下：

```
typedef struct CK_MECHANISM
{ CK_MECHANISM_TYPE mechanism;
  CK_VOID_PTR pParameter;
  CK_ULONG ulParameterLen;
} CK_MECHANISM;
```

mechanism: 產生機制的種類。

pParameter: 產生機制所需的參數之指標。

ulParameterLen: 參數的長度。



用來產生公私鑰對的函式為C\_GenerateKeyPair，其原型如下：

```
CK_RV C_GenerateKeyPair(CK_SESSION_HANDLE hSession,  
                        CK_MECHANISM_PTR pMechanism,  
                        CK_ATTRIBUTE_PTR pPublicKeyTemplate, CK_ULONG  
                        ulPublicKeyAttributeCount, CK_ATTRIBUTE_PTR  
                        pPrivateKeyTemplate, CK_ULONG  
                        ulPrivateKeyAttributeCount, CK_OBJECT_HANDLE_PTR  
                        phPublicKey, CK_OBJECT_HANDLE_PTR phPrivateKey)
```

hSession: 所建立session 的handle。

pMechanism: 金鑰產生機制。

pPublicKeyTemplate: 公鑰的屬性樣板。

ulPublicKeyAttributeCount: 公鑰的屬性個數。

pPrivateKeyTemplate: 私鑰的屬性樣板。

ulPrivateKeyAttributeCount: 私鑰的屬性個數。

phPublicKey: 建立後所傳回的公鑰物件handle。

phPrivateKey: 建立後所傳回的私鑰物件handle。

## 5.3 實驗範例

### 5.3.1 產生秘密金鑰

程式 EX5\_1 的功能為產生秘密金鑰。秘密金鑰產生機制的種類為 Triple DES with 3 keys[EX5\_1 (1)]。接下來建立連線之後就可以建立秘密金鑰物件的樣板[EX5\_1 (2)]，在這個樣板裏共有如下屬性：

CKA\_LABEL: 物件的標籤。

CKA\_TOKEN: 決定此物件為token物件。

CKA\_SENSITIVE: 決定金鑰是否為敏感性，如為敏感性則無法取出其值，此處為FALSE。

CKA\_PRIVATE: 決定是否為私密物件，此處為FALSE。

CKA\_ENCRYPT: 決定此金鑰是否可以拿來加密資料，此處為TRUE。  
CKA\_DECRYPT: 決定此金鑰是否可以拿來加密資料，此處為TRUE。  
CKA\_SIGN: 決定此金鑰是否可以拿來簽章，此處為TRUE。  
CKA\_VERIFY: 決定此金鑰是否可以拿來驗證，此處為FALSE。  
CKA\_EXPORT: 決定是否可以拿來加密其它的金鑰，只有SO可以設為TRUE。  
CKA\_IMPORT: 和CKA\_UNWRAP類似，如果此設為TRUE而CKA\_UNWRAP設為FALSE，則解密演算法只可以使用CKM\_WRAPKEY\_DES3\_CBC。  
CKA\_WRAP: 決定是否可以用來加密其它的金鑰。  
CKA\_UNWRAP: 決定是否可以用來解密其它的金鑰。  
CKA\_EXPORTABLE: 如果此設為TRUE，則只有CKA\_EXPORT為TRUE的金鑰可以加密此金鑰。  
CKA\_EXTRACTABLE: 決定是否可以被其它金鑰加密。另外如果此屬性設為TRUE則無法看到金鑰值。  
CKA\_MODIFIABLE: 決定金鑰建立後，是否還可以再修改屬性值。  
CKA\_DERIVE: 決定是否可以拿來產生其它的金鑰。

接下來計算此樣板裏面的屬性個數後，就可以呼叫 C\_CreateKey 函式建立秘密金鑰物件[EX5\_1(3)]。

### EX5\_1 產生秘密金鑰物件

```
#include <stdio.h>

#include <string.h>

#include "cryptoki.h"

#include "ctvdef.h"

int main(int argc, char* argv[])

{

    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hKey= CK_INVALID_HANDLE;
```

```
CK_SLOT_ID slotId = 0;

CK_CHAR label[] = "Generate secret key";

CK_BBOOL bTrue = TRUE;

CK_BBOOL bFalse = FALSE;

CK_MECHANISM mechanism = {CKM_DES3_KEY_GEN, NULL_PTR, 0};           (1)

rv = C_Initialize(NULL);

rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

if (rv != CKR_OK) {

    printf("Could not open session on token %d,error code =
           %x\n",slotId,rv);

    return 0;

}

CK_ATTRIBUTE secretkeyTemplate [] = {                               (2)

    {CKA_LABEL,      label,      sizeof(label)-1},

    {CKA_TOKEN,      &bTrue,     sizeof(CK_BBOOL)},

    {CKA_SENSITIVE,  &bFalse,    sizeof(CK_BBOOL)},

    {CKA_PRIVATE,    &bFalse,    sizeof(CK_BBOOL)},

    {CKA_ENCRYPT,     &bTrue,     sizeof(CK_BBOOL)},

    {CKA_DECRYPT,     &bTrue,     sizeof(CK_BBOOL)},

    {CKA_SIGN,       &bTrue,     sizeof(CK_BBOOL)},

    {CKA_VERIFY,     &bTrue,     sizeof(CK_BBOOL)},

    {CKA_EXPORT,     &bFalse,    sizeof(CK_BBOOL)},

    {CKA_IMPORT,     &bFalse,    sizeof(CK_BBOOL)},

    {CKA_WRAP,       &bFalse,    sizeof(CK_BBOOL)},

    {CKA_UNWRAP,     &bFalse,    sizeof(CK_BBOOL)},

    {CKA_EXPORTABLE, &bFalse,    sizeof(CK_BBOOL)},

    {CKA_EXTRACTABLE, &bTrue,    sizeof(CK_BBOOL)},

    {CKA_MODIFIABLE, &bTrue,    sizeof(CK_BBOOL)},

    {CKA_DERIVE,     &bFalse,    sizeof(CK_BBOOL)}

};
```

```

    CK_SIZE objectSize = sizeof(secretkeytemplate) / sizeof(CK_ATTRIBUTE);

    rv = C_GenerateKey(hSession, &mechanism, secretkeyTemplate,
        objectSize,&hKey);                                     (3)

    if (rv != CKR_OK) {

        printf("Could not generate secret key object on token %d,error code
            = %x\n",slotId,rv);

        return 0;

    }

    rv = C_CloseSession(hSession);

    rv = C_Finalize(NULL);

    return 0;

}

```

### 5.3.2 產生金鑰對

程式 EX5\_2 的功能為產生 RSA 金鑰對。金鑰對產生機制的種類為 RSA 且依照 PKCS 標準[EX5\_2 (1)]。建立連線之後就可以建立公開金鑰及秘密金鑰物件的樣板[EX5\_2 (2)(3)]。在這兩個樣板裏面的屬性大致與 5.3.1 節裏面設定的相同。其中 CKA\_PRIME\_BITS 為設定金鑰對的長度，此處設為 1024 位元。而在私密金鑰樣板中的 CKA\_SIGN\_LOCAL\_CERT 屬性設為 TRUE，表示此私密金鑰只可用來簽章在 PSE 裏面產生的 public key，以產生憑證。設完樣板後，即可呼叫 C\_GenerateKeyPair[EX5\_2 (4)]。

#### EX5\_2 產生金鑰對

```

#include <stdio.h>

#include <string.h>

#include "cryptoki.h"

#include "ctvdef.h"

int main(int argc, char* argv[])

{

    CK_RV rv;

```

```

CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

CK_OBJECT_HANDLE hPublicKey = CK_INVALID_HANDLE;

CK_OBJECT_HANDLE hPrivateKey = CK_INVALID_HANDLE;

CK_SLOT_ID slotId = 0;

CK_CHAR publabel[] = "Generate public key";

CK_CHAR prilabel[] = "Generate private key";

CK_ULONG keySize = 1024;

CK_BBOOL bTrue = TRUE;

CK_BBOOL bFalse = FALSE;

CK_MECHANISM mechanism = {CKM_RSA_PKCS_KEY_PAIR_GEN, NULL, 0};           (1)

rv = C_Initialize(NULL);

rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

if (rv != CKR_OK) {

    printf("Could not open session on token %d,error code =
           %x\n",slotId,rv);

    return 0;

}

CK_ATTRIBUTE publicKeyTpl [] = {                                         (2)

    {CKA_TOKEN,          &bTrue,    sizeof(CK_BBOOL)},

    {CKA_PRIVATE,        &bFalse,    sizeof(CK_BBOOL)},

    {CKA_LABEL,          publabel,   sizeof(publabel)-1},

    {CKA_ENCRYPT,         &bTrue,     sizeof(CK_BBOOL)},

    {CKA_VERIFY,         &bTrue,     sizeof(CK_BBOOL)},

    {CKA_WRAP,           &bTrue,     sizeof(CK_BBOOL)},

    {CKA_MODIFIABLE,     &bTrue,     sizeof(CK_BBOOL)},

    {CKA_DERIVE,         &bTrue,     sizeof(CK_BBOOL)},

    {CKA_PRIME_BITS,     &keySize,   sizeof(CK_ULONG)}

};

CK_SIZE pubSize = sizeof(publicKeyTpl) / sizeof(CK_ATTRIBUTE);

```

```

CK_ATTRIBUTE privateKeyTpl[] = {                                     (3)

    {CKA_TOKEN,          &bTrue,    sizeof(CK_BBOOL)},
    {CKA_PRIVATE,        &bFalse,    sizeof(CK_BBOOL)},
    {CKA_LABEL,          prilabel,   sizeof(prilabel)-1},
    {CKA_SENSITIVE,       &bTrue,    sizeof(CK_BBOOL)},
    {CKA_DECRYPT,         &bTrue,    sizeof(CK_BBOOL)},
    {CKA_SIGN,           &bTrue,    sizeof(CK_BBOOL)},
    {CKA_IMPORT,         &bFalse,    sizeof(CK_BBOOL)},
    {CKA_UNWRAP,         &bTrue,    sizeof(CK_BBOOL)},
    {CKA_MODIFIABLE,      &bTrue,    sizeof(CK_BBOOL)},
    {CKA_EXTRACTABLE,     &bTrue,    sizeof(CK_BBOOL)},
    {CKA_DERIVE,          &bTrue,    sizeof(CK_BBOOL)},
    {CKA_EXPORTABLE,      &bTrue,    sizeof(CK_BBOOL)},
    {CKA_SIGN_LOCAL_CERT, &bTrue,    sizeof(CK_BBOOL)},

};

CK_SIZE priSize = sizeof(privateKeyTpl)/sizeof(CK_ATTRIBUTE);

rv = C_GenerateKeyPair(hSession,                                     (4)

    &mech,
    publicKeyTpl,
    pubSize,
    privateKeyTpl,
    priSize,
    &hPublicKey,
    &hPrivateKey);

if (rv != CKR_OK) {

    printf("Could not generate key pair on token %d,error code =
        %x\n",slotId,rv);

    return 0;

}

rv = C_CloseSession(hSession);

```

```
rv = C_Finalize(NULL);  
  
return 0;  
  
}
```

## 5.4 習題

1. 金鑰屬性CKA\_WRAP 及CKA\_UNWRAP 的功能為何。
2. 金鑰屬性CKA\_EXPORT 及CKA\_WARP 的差別在那裏。
3. 金鑰屬性CKA\_EXTRACTABLE 的功能為何。
4. 金鑰屬性CKA\_EXPORTABLE 的功能為何。
5. 使用C\_GenerateKey 產生256 位元長度的AES 金鑰。
6. 使用C\_GenerateKey 產生1024 位元長度的DSA 金鑰對。



ProtectServer External

## 實驗6. 資料加解密

### 6.1 實驗目的

本實驗在於如何使用秘密金鑰及公私鑰對來做對稱式及非對稱式資料加解密。

### 6.2 相關原理說明

Cryptoki 提供下列函式來做資料加密的運算：

1. C\_EncryptInit: 主要功能為初始化加密運算，也就是提供金鑰。函式原型定義如下：

CK\_RV

```
C_EncryptInit( CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hKey)
```

hSession: 所建立session 的handle。

pMechanism: 加密演算法。

hKey: 用來執行加密金鑰的handle。

2. C\_Encrypt: 加密運算，函式原型如下：

```
CK_RV C_Encrypt(CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)
```

hSession: 所建立session 的handle。

pData: 要加密的資料。

ulDataLen: 要加密的資料長度。



pEncryptedData: 加密結果。

pulEncryptedDataLen: 加密結果的長度。

3. C\_EncryptUpdate: 繼續一個multi-part 加密運算，函式原型如下:

```
CK_RV, C_EncryptUpdate(  
    CK_SESSION_HANDLE hSession,  
    CK_BYTE_PTR pPart,  
    CK_ULONG ulPartLen,  
    CK_BYTE_PTR pEncryptedPart,  
    CK_ULONG_PTR pulEncryptedPartLen)
```

hSession: 所建立session 的handle。

pPart: 要加密的某一部分資料。

ulPartLen: 要加密的某一部分資料的長度。

pEncryptedPart: 要加密的某一部分資料的加密結果。

pulEncryptedPartLen: 要加密的某一部分資料的加密結果的長度。

4. C\_EncryptFinal: 完成multi-part 加密運算，函式原型如下:

```
CK_RV C_EncryptFinal(  
    CK_SESSION_HANDLE hSession,  
    CK_BYTE_PTR pLastEncryptedPart,  
    CK_ULONG_PTR pulLastEncryptedPartLen)
```

hSession: 所建立session 的handle。

pLastEncryptedPart: 最後一部份的加密結果。

pulLastEncryptedPartLen: 最後一部份的加密結果長度。

Cryptoki 提供下列函式來做資料解密的運算:

1. C\_DecryptInit: 主要功能為初始化解密運算，也就是提供金鑰。函式原型定義如下:

```
CK_RV C_DecryptInit (
```

```
CK_SESSION_HANDLE hSession,  
CK_MECHANISM_PTR pMechanism,  
CK_OBJECT_HANDLE hKey)
```

hSession: 所建立session 的handle。

pMechanism: 解密演算法。

hKey: 用來執行解密金鑰的handle。

2. C\_Decrypt: 解密運算，函式原型如下：

```
CK_RV C_Decrypt (  
    CK_SESSION_HANDLE hSession,  
    CK_BYTE_PTR pEncryptedData,  
    CK_ULONG_PTR pulEncryptedDataLen)  
    CK_BYTE_PTR pData,  
    CK_ULONG ulDataLen,
```

hSession: 所建立session 的handle。

pEncryptedData: 要解密的資料。

pulEncryptedDataLen: 要解密的資料長度。

pData: 解密結果。

ulDataLen: 解密結果的長度。

3. C\_DecryptUpdate: 繼續一個multi-part 解密運算，函式原型如下：

```
CK_RV, C_DecryptUpdate (  
    CK_SESSION_HANDLE hSession,  
    CK_BYTE_PTR pEncryptedPart,  
    CK_ULONG_PTR pulEncryptedPartLen)  
    CK_BYTE_PTR pPart,  
    CK_ULONG ulPartLen,
```

hSession: 所建立session 的handle。

pEncryptedPart: 要解密的某一部分資料。

pulEncryptedPartLen: 要解密的某一部分資料的長度。

pPart: 要解密的某一部分資料的解密結果。  
ulPartLen: 要解密的某一部分資料的解密結果的長度。

4. C\_DecryptFinal: 完成multi-part 解密運算，函式原型如下:

```
CK_RV C_DecryptFinal (  
    CK_SESSION_HANDLE hSession,  
    CK_BYTE_PTR pLastPart,  
    CK_ULONG_PTR pulLastPartLen)
```

hSession: 所建立session 的handle。  
pLastEncryptedPart: 最後一部份的解密結果。  
pulLastEncryptedPartLen: 最後一部份的解密結果長度。

其中C\_Encrypt可以用C\_EncryptUpdate加上C\_EncryptFinal來達成。而C\_Decrypt可以用C\_DecryptUpdate加上C\_DecryptFinal來取代。其目的是用在連續的資料加解密。

## 6.3 實驗範例

### 6.3.1 對稱式加解密

程式 EX6\_1 的功能為使用 Triple DES 來加解密資料。在加解密資料之前，PSE 要先建立一把秘密金鑰，在此將採用 EX5\_1 所建立的秘密金鑰。因此要提供這把秘密金鑰的資訊[EX6\_1(1)(2)]以供待會尋找金鑰之用。為了要從 PSE 中尋找程式 EX5\_1所產生的金鑰，要定義提供來尋找金鑰的樣板[EX6\_1(3)]。在此表示要從PSE 中尋找一把金鑰其 label 為 Generate secret key，以及其物件種類為CKO\_SECRET\_KEY。

接下來使用C\_FindObjectsInit 來初始化物件的尋找，也就是提供尋找的條件[EX6\_1(4)]。numObjectsToFind 變數為 1，表示只要尋找到一個符合條件的物件即可[EX6\_1(5)]。numObjectsFound 是待會要存放實際找到物件的個數。設定好之後即可開始尋找物件[EX6\_1(6)]，找到的物件會被指向 hKey handle。再來就可

以結束物件的尋找[EX6\_1(7)]。接下來判斷是否有找到物件，numObjectsFound 如果為0，表示沒找到符合條件的物件[EX6\_1(8)]。

找到金鑰物件後就可以進行加密。先設定加解密機制為 CKM\_DES3\_ECB\_PAD[EX6\_1(9)]。再來初始化加密運算[EX6\_1(10)]。再宣告資料的陣列及存放結果的陣列 [EX6\_1(11)]，在此是宣告資料為長度20 個位元組的0，但是加解密機制為 CKM\_DES3\_ECB\_PAD，表示會 padding 為 8 的倍數，因此存放結果的陣列的長度必須為 24。接下來即可進行加密運算[EX6\_1(12)]，結果會存放於 EncData 陣列中。再來進行解密運算，先進行初始化[EX6\_1(13)]。接下來進行解密運算[EX6\_1(14)]，結果會存放於Data 陣列中，解密的結果應該是 20 個位元組的0。資料及加解密後的結果都會顯示在命令提示字元的視窗中。

### EX6\_1 對稱式加解密

```
#include <stdio.h>

#include <string.h>

#include "cryptoki.h"

#include "ctvdef.h"

int main(int argc, char* argv[])
{
    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hKey= CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    CK_CHAR objLabel[] = "Generate secret key";           (1)

    CK_OBJECT_CLASS objClass = CKO_SECRET_KEY;           (2)

    rv = C_Initialize(NULL);

    rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

    if (rv != CKR_OK) {

        printf("Could not open session on token %d,error code =
                %x\n",slotId,rv);

        return 0;
    }
}
```

```
}

CK_ATTRIBUTE objectTemplate[] = {                                     (3)
    {CKA_CLASS, &objClass, sizeof(objClass)},
    {CKA_LABEL, objLabel, sizeof(objLabel)-1}
};

CK_SIZE templateSize = sizeof(objectTemplate) / sizeof(CK_ATTRIBUTE);

rv = C_FindObjectsInit(hSession, objectTemplate, templateSize);      (4)

if (rv != CKR_OK) {
    printf("C_FindObjectsInit failed on token %d,error code =
        %x\n",slotId,rv);
    return 0;
}

CK_ULONG numObjectsToFind = 1;                                       (5)

CK_ULONG numObjectsFound = 0;

rv = C_FindObjects(hSession,&hKey,                                   (6)
    numObjectsToFind,&numObjectsFound);(5)

if (rv != CKR_OK) {
    printf("C_FindObjects failed on token %d,error code =
        %x\n",slotId,rv);
    return 0;
}

rv = C_FindObjectsFinal(hSession);                                    (7)

if (rv != CKR_OK) {
    printf("C_FindObjectsFinal failed on token %d,error code =
        %x\n",slotId,rv);
    return 0;
}

if (numObjectsFound == 0){                                           (8)
    printf("Object not found.\n");
    return 0;
}
```

```

    }

    CK_MECHANISM mech = {CKM_DES3_ECB_PAD, NULL, 0};           (9)

    rv = C_EncryptInit(hSession, &mech, hKey);                 (10)

    CK_CHAR Data[20] = {0};                                     (11)

    CK_CHAR EncData[24] = {0};

    CK_ULONG EncDataLen = 24;

    CK_ULONG DataLen = 20;

    char Data_str[50] = {0};

    char EncData_str[50] = {0};

    bin2hex(Data_str, Data, 20);

    Data_str[40] = '\0';

    printf("data = %s\n", Data_str);

    rv = C_Encrypt(hSession, Data, sizeof(Data), EncData, &EncDataLen); (12)

    bin2hex(EncData_str, EncData, EncDataLen);

    EncData_str[EncDataLen*2] = '\0';

    printf("encrypted data = %s\n", EncData_str);

    rv = C_DecryptInit(hSession, &mech, hKey);                 (13)

    rv = C_Decrypt(hSession, EncData, EncDataLen, Data, &DataLen); (14)

    bin2hex(Data_str, Data, 20);

    Data_str[40] = '\0';

    printf("data = %s\n", Data_str);

    rv = C_CloseSession(hSession);

    rv = C_Finalize(NULL);

    return 0;
}

```

### 6.3.2 非對稱式加解密

程式EX6\_2 的功能為使用RSA public key 來加密資料，而用RSA private key 來解密資料。在加解密資料之前，PSE 要先建立一對公私鑰，在此將採用EX5\_2

所建立的金鑰對。因此要提供公私鑰的資訊 [EX6\_2(1)(2)] 以供待會尋找金鑰之用。在此表示要從 PSE 中尋找公鑰其 label 為 Generate public key，私鑰 label 為 Generate private key。以及其物件種類為 CKO\_PUBLIC\_KEY 及 CKO\_PRIVATE\_KEY。

首先尋找公鑰物件 [EX6\_2(4)(5)(6)(7)(8)]，找到的物件會被指向 hPubKey handle。找到公鑰物件後就可以進行加密。先設定加解密機制為 CKM\_RSA\_PKCS [EX6\_2(9)]。再來初始化加密運算 [EX6\_2(10)]。再宣告資料的陣列及存放結果的指標 [EX6\_2(11)]，在此是宣告資料為長度 20 個位元組的 0。接下來先呼叫 C\_Encrypt 一次，在存放結果的地方給 NUUL 值以取得加密結果的長度 [EX6\_2(12)]。得到結果長度後就可分配足夠的記憶體給 EncData 指標 [EX6\_2(13)]。這時就可進行加密運算 [EX6\_2(14)]。再來進行解密運算，先尋找私鑰物件 [EX6\_2(15)(16)(17)(18)]。找到後就可進行解密，先呼叫 C\_Decrypt 取得結果的長度 [EX6\_2(19)]，如此便可分配足夠的記憶體。這時在呼叫一次 C\_Decrypt 進行解密運算，結果會存放於 DecData 中 [EX6\_2(20)]，解密的結果應該是 20 個位元組的 0。資料及加解密後的結果都會顯示在命令提示字元的視窗中。

## EX6\_2 非對稱式加解密

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "cryptoki.h"

#include "ctvdef.h"

int main(int argc, char* argv[])
{
    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hPubKey = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hPriKey = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    CK_CHAR pubLabel[] = "Generate public key";
```

(1)

```
CK_CHAR priLabel[] = "Generate private key";

CK_OBJECT_CLASS pubobjClass = CKO_PUBLIC_KEY;                                (2)

CK_OBJECT_CLASS priobjClass = CKO_PRIVATE_KEY;

rv = C_Initialize(NULL);

rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

if (rv != CKR_OK) {

    printf("Could not open session on token %d,error code =
           %x\n",slotId,rv);

    return 0;

}

CK_ATTRIBUTE pubobjectTemplate [] = {                                         (3)

    {CKA_CLASS, &pubobjClass, sizeof(pubobjClass)},

    {CKA_LABEL, pubLabel, sizeof(pubLabel)-1}

};

CK_SIZE templateSize = sizeof(pubobjectTemplate) / sizeof(CK_ATTRIBUTE);

rv = C_FindObjectsInit(hSession, pubobjectTemplate, templateSize);            (4)

if (rv != CKR_OK) {

    printf("C_FindObjectsInit failed on token %d,error code =
           %x\n",slotId,rv);

    return 0;

}

CK_ULONG numObjectsToFind = 1;                                                (5)

CK_ULONG numObjectsFound = 0;

rv = C_FindObjects(hSession,& hPubKey,                                       (6)

    numObjectsToFind,&numObjectsFound);

if (rv != CKR_OK) {

    printf("C_FindObjects failed on token %d,error code =
           %x\n",slotId,rv);

    return 0;

}
```



```
rv = C_FindObjectsFinal(hSession); (7)

if (rv != CKR_OK) {
    printf("C_FindObjectsFinal failed on token %d,error code =
        %x\n",slotId,rv);
    return 0;
}

if (numObjectsFound == 0){ (8)
    printf("Object not found.\n");
    return 0;
}

CK_MECHANISM mech = {CKM_RSA_PKCS,NULL,0}; (9)

rv = C_EncryptInit(hSession, &mech, hPubKey); (10)

CK_CHAR Data[20] = {0}; (11)

CK_CHAR *EncData;

CK_ULONG EncDataLen = 0;

CK_ULONG DataLen = 0;

char Data_str[50] = {0};

char *EncData_str;

memset(Data,0,sizeof(Data));

bin2hex(Data_str,Data,20);

Data_str[40] = '\0';

printf("data = %s\n",Data_str);

rv = C_Encrypt(hSession, Data, sizeof(Data), NULL, &EncDataLen); (12)

EncData = (CK_CHAR *) malloc(EncDataLen); (13)

if (EncData == NULL) {
    printf("malloc memory failed\n");
    return 0;
}

EncData_str = (char *) malloc(EncDataLen*2+1);

if (EncData_str == NULL) {
```

```

    printf("malloc memory failed\n");
    return 0;
}

rv = C_Encrypt(hSession, Data, sizeof(Data), EncData, &EncDataLen); (14)

bin2hex(EncData_str, EncData, EncDataLen);

EncData_str[EncDataLen*2] = '\0';

printf("encrypted data = %s\n", EncData_str);

CK_ATTRIBUTE priobjectTemplate[] = {
    {CKA_CLASS, &priobjClass, sizeof(priobjClass)},
    {CKA_LABEL, priLabel, sizeof(priLabel)-1}
};

templateSize = sizeof(priobjectTemplate) / sizeof(CK_ATTRIBUTE);

rv = C_FindObjectsInit(hSession, priobjectTemplate, templateSize); (16)

if (rv != CKR_OK) {
    printf("C_FindObjectsInit failed on token %d,error code =
        %x\n", slotId, rv);
    return 0;
}

numObjectsToFind = 1;

numObjectsFound = 0;

rv = C_FindObjects(hSession, &hPriKey,
    numObjectsToFind, &numObjectsFound); (17)

if (rv != CKR_OK) {
    printf("C_FindObjects failed on token %d,error code =
        %x\n", slotId, rv);
    return 0;
}

rv = C_FindObjectsFinal(hSession); (18)

if (rv != CKR_OK) {
    printf("C_FindObjectsFinal failed on token %d,error code =

```

```

        %x\n",slotId,rv);

                                                                    return 0;

    }

    if (numObjectsFound == 0) {
        printf("Object not found.\n");
        return 0;
    }

    rv = C_DecryptInit(hSession, &mech, hPriKey);                (19)

    CK_CHAR *DecData = NULL;                                     (20)

    CK_ULONG DecDataLen = 0;

    char *DecData_str= NULL;

    rv = C_Decrypt(hSession, EncData, EncDataLen, NULL, &DecDataLen);    (21)

    DecData = (CK_CHAR *) malloc(DecDataLen);

    DecData_str = (char *) malloc(DecDataLen*2+1);

    rv = C_Decrypt(hSession, EncData, EncDataLen, DecData, &DecDataLen);(22)

    bin2hex(DecData_str,DecData,DecDataLen);

    DecData_str[DecDataLen*2] = '\0';

    printf("Decrypted data = %s\n",DecData_str);

    rv = C_CloseSession(hSession);

    rv = C_Finalize(NULL);

    return 0;

}

```

## 6.4 習題

1. 在程式EX6\_2 中為何要各呼叫C\_Encrypt 及C\_Decrypt 兩次。
2. 如果要進行加解密運算，則那些金鑰的屬性值一定要設為TRUE。
3. 先建立一把AES 金鑰，再用此金鑰來加解密資料。
4. 將程式EX6\_1 中的C\_Encrypt 用C\_EncryptUpdate 加上C\_EnryptFinal 來進行加密運算。

5. 將程式EX6\_2 中的加解密機制改成使用PKCS#1 RSA OAEP 機制，來執行資料的加解密。



ProtectServer Gold

## 實驗7. 簽章及驗證

### 7.1 實驗目的

本實驗之主要目的在於了解如何執行資料的簽章及驗證。

### 7.2 相關原理說明

數位簽章(Digital Signature)提供了整合「身分鑑別」與「資料完整性」而達成不可否認安全性服務，可以確認訊息的來源與確保訊息不會遭受到篡改。完整的程序如圖 7-1 所示。簽章者將要簽章的訊息經雜湊函數得到一訊息摘要 (Message Digest)，再使用簽章者本身的私密金鑰將訊息摘要加密而得到簽章。再將訊息及簽章傳送給驗證者，此時驗證者使用簽章者的公開金鑰將簽章解密得到一雜湊值，而再將接收到的訊息經雜湊函數得另一雜湊值，此時驗證者比較兩雜湊值，如果相等則表示驗證(Verify)成功，反之則驗證失敗。由於任何人都可利 用公開金鑰進行驗證，且只有簽章者知道其本身的私密金鑰來產生簽章。因此當 發生爭論時，公正的第三者可以做出公正的判斷，如此便具有「身分鑑別」性質。由於沒有任何人知道簽章者的私密金鑰，也就沒有辦法假造簽章，再經最後的比較雜湊值，如此便可確定訊息的完整性(Integrity)。數位簽章一般有 RSA、DSA(Digital Signature Algorithm)、ECC(Elliptic Curve Cryptosystem)等方法。本實驗僅介紹常用之RSA 數位簽章方法。

我們可以使用Cryptoki 所提供的下列函式來進行資料的數位簽章：

1. C\_SignInit: 初始化簽章運算，其函式原型定義如下。

```
CK_RV C_SignInit(  
    CK_SESSION_HANDLE hSession,  
    CK_MECHANISM_PTR pMechanism,  
    CK_OBJECT_HANDLE hKey)
```

hSession: 所建立session 的handle。

pMechanism: 簽章演算法。

hKey: 用來執行簽章金鑰的handle。

2. C\_Sign: 簽章資料，其函式原型定義如下。

```
CK_RV C_Sign(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pData,
    CK_ULONG ulDataLen,
    CK_BYTE_PTR pSignature,
    CK_ULONG_PTR pulSignatureLen)
```

hSession: 所建立session 的handle。

pData: 要簽章的資料。

ulDataLen: 要簽章資料的長度。

pSignature: 簽章結果。

pulSignatureLen: 簽章結果的長度。

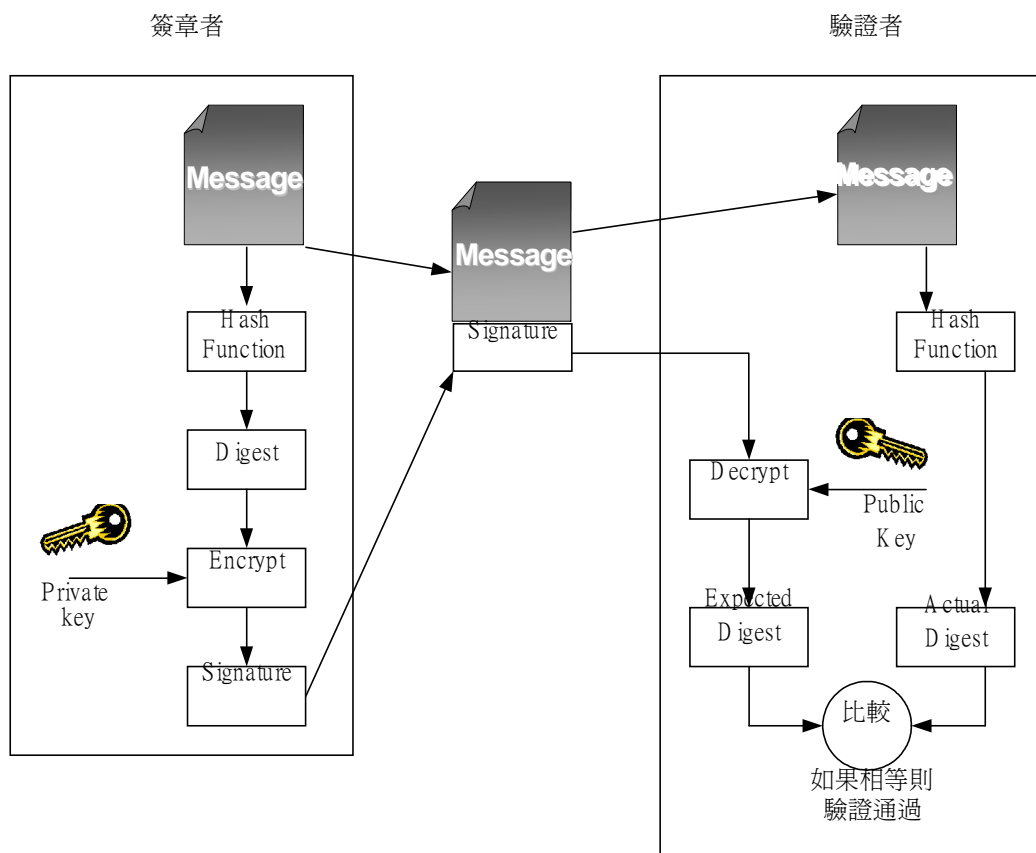


圖 7-1:數位簽章

3. C\_SignRecoverInit: 初始化簽章運算，此簽章運算可以將資料從簽章回復出來。

CK\_RV

```
C_SignRecoverInit( CK_SESSION_HANDLE hSession,  
                  CK_MECHANISM_PTR pMechanism,  
                  CK_OBJECT_HANDLE hKey)
```

hSession: 所建立session 的handle。

pMechanism: 簽章演算法。

hKey: 用來執行簽章金鑰的handle。

4. C\_SignRecover: 簽章資料，此簽章運算可以將資料從簽章回復出來。

CK\_RV

```
C_SignRecover( CK_SESSION_HANDLE hSession, CK_BYTE_PTR  
              pData,  
              CK_ULONG ulDataLen,  
              CK_BYTE_PTR pSignature,  
              CK_ULONG_PTR pulSignatureLen)
```

hSession: 所建立session 的handle。

pData: 要簽章的資料。

ulDataLen: 要簽章資料的長度。

pSignature: 簽章結果。

pulSignatureLen: 簽章結果的長度。

我們可以使用Cryptoki 所提供的下列函式來進行數位簽章的驗證：

1. C\_VerifyInit: 初始化簽章驗證運算。

CK\_RV

```
C_VerifyInit( CK_SESSION_HANDLE hSession,  
              CK_MECHANISM_PTR pMechanism,  
              CK_OBJECT_HANDLE hKey)
```

hSession: 所建立session 的handle。

pMechanism: 簽章演算法。

hKey: 用來執行簽章金鑰的handle。

2. C\_Verify: 驗證簽章，其函式原型定義如下。

```
CK_RV C_Verify
( CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pData,
  CK_ULONG ulDataLen,
  CK_BYTE_PTR pSignature,
  CK_ULONG_PTR pulSignatureLen)
```

hSession: 所建立session 的handle。

pData: 被簽章的資料。

ulDataLen: 被簽章資料的長度。

pSignature: 簽章。

pulSignatureLen: 簽章的長度。

3. C\_VerifyRecoverInit: 初始化簽章驗證運算，此簽章運算可以將資料從簽章回復出來。

```
CK_RV C_VerifyRecoverInit
( CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hKey)
```

hSession: 所建立session 的handle。

pMechanism: 簽章演算法。

hKey: 用來執行簽章金鑰的handle。

4. C\_VerifyRecover: 驗證簽章，此簽章運算可以將資料從簽章回復出來。

```
CK_RV
C_SignRecover)( CK_SESSION_HA
NDLE hSession, CK_BYTE_PTR
pSignature,
```



```
CK_ULONG_PTR pulSignatureLen,  
CK_BYTE_PTR pData,  
CK_ULONG ulDataLen)
```

hSession: 所建立session 的handle。

pSignature: 簽章。

pulSignatureLen: 簽章的長度。

pData: 所回復的資料。

ulDataLen: 所回復資料的長度。

在使用C\_SignRecover及C\_VerifyRecover做簽章驗證運算時，金鑰的CKA\_SIGN\_RECOVER屬性要設為TRUE，才可進行此種運算。

## 7.3 實驗範例

程式EX7\_1 的功能為使用RSA private key 來簽章資料，而用RSA public key 來驗證簽章。在簽章資料之前，PSE 要先建立一對公私鑰，在此將採用 EX5\_2 所建立的金鑰對。因此要提供公私鑰的資訊[EX7\_1(1)(2)]以供待會尋找金鑰之用。在此表示要從 PSE 中尋找公鑰其 label 為 Generate public key，私鑰 label 為 Generate public key。以及其物件種類為 CKO\_PUBLIC\_KEY 及 CKO\_PRIVATE\_KEY。

首先尋找私鑰物件[EX7\_1(4)(5)(6)(7)(8)]，找到的物件會被指向 hPriKey handle。找到私鑰物件後就可以進行簽章。先設定簽章機制為 CKM\_SHA1\_RSA\_PKCS [EX7\_1(9)]。再來初始化簽章運算[EX7\_1(10)]。再宣告資料的陣列及存放結果的指標[EX7\_1(11)]，在此是宣告資料為長度 20 個位元組的 0。接下來先呼叫 C\_Sign 一次，在存放結果的地方給 NUUL 值以取得加密結果的長度[EX7\_1(12)]。得到結果長度後就可分配足夠的記憶體給 EncData 指標[EX7\_1(13)]。這時就可進行簽章運算[EX7\_1(14)]。再來進行驗證運算，先尋找公鑰物件[EX7\_1(15) (16) (17) (18)]。找到後就可進行驗證，呼叫 C\_Verify 進行驗證運算[EX7\_1(20)]，如果回傳值為 CKR\_OK 就表示簽章驗證成功。

### EX7\_1 資料簽章及驗證

```
#include <stdio.h>
```

```
#include <string.h>

#include <stdlib.h>

#include "cryptoki.h"

#include "ctvdef.h"

int main(int argc, char* argv[])
{
    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hPubKey = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hPriKey = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    CK_CHAR pubLabel[] = "Generate public key";           (1)
    CK_CHAR priLabel[] = "Generate private key";

    CK_OBJECT_CLASS pubobjClass = CKO_PUBLIC_KEY;        (2)
    CK_OBJECT_CLASS priobjClass = CKO_PRIVATE_KEY;

    rv = C_Initialize(NULL);

    rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

    if (rv != CKR_OK) {

        printf("Could not open session on token %d,error code =
                %x\n",slotId,rv);

        return 0;

    }

    CK_ATTRIBUTE priobjectTemplate [] = {                 (3)

        {CKA_CLASS, &priobjClass, sizeof(priobjClass)},

        {CKA_LABEL, priLabel, sizeof(priLabel)-1}

    };

    CK_SIZE templateSize = sizeof(priobjectTemplate) / sizeof(CK_ATTRIBUTE);

    rv = C_FindObjectsInit(hSession, priobjectTemplate, templateSize);   (4)

    if (rv != CKR_OK) {
```

```

        printf("C_FindObjectsInit failed on token %d,error code =
                %x\n",slotId,rv);

        return 0;
    }

    CK_ULONG numObjectsToFind = 1; (5)

    CK_ULONG numObjectsFound = 0;

    rv = C_FindObjects(hSession,& hPriKey, (6)
        numObjectsToFind,&numObjectsFound);

    if (rv != CKR_OK) {

        printf("C_FindObjects failed on token %d,error code =
                %x\n",slotId,rv);

        return 0;
    }

    rv = C_FindObjectsFinal(hSession); (7)

    if (rv != CKR_OK) {

        printf("C_FindObjectsFinal failed on token %d,error code =
                %x\n",slotId,rv);

        return 0;
    }

    if (numObjectsFound == 0){ (8)

        printf("Object not found.\n");

        return 0;
    }

    CK_MECHANISM mech = { CKM_SHA1_RSA_PKCS,NULL,0}; (9)

    rv = C_SignInit(hSession, &mech, hPriKey); (10)

    CK_CHAR Data[20] = {0}; (11)

    CK_CHAR *pSignature;

    CK_ULONG SignatureLen = 0;

    memset(Data,0,sizeof(Data));

    rv = C_Sign(hSession, Data, sizeof(Data), NULL, &SignatureLen); (12)

```

```

pSignature = (CK_CHAR*)malloc(SignatureLen); (13)

if (pSignature == NULL) {
    printf("malloc memory failed\n");
    return 0;
}

rv = C_Sign(hSession, Data, sizeof(Data), pSignature,
            &SignatureLen); (14)

CK_ATTRIBUTE pubobjectTemplate[] = { (15)
    {CKA_CLASS, &pubobjClass, sizeof(pubobjClass)},
    {CKA_LABEL, pubLabel, sizeof(pubLabel)-1}
};

templateSize = sizeof(pubobjectTemplate) / sizeof(CK_ATTRIBUTE);

rv = C_FindObjectsInit(hSession, pubobjectTemplate, templateSize); (16)

if (rv != CKR_OK) {
    printf("C_FindObjectsInit failed on token %d,error code =
        %x\n",slotId,rv);
    return 0;
}

numObjectsToFind = 1;
numObjectsFound = 0;

rv = C_FindObjects(hSession,&hPubKey, (17)
    numObjectsToFind,&numObjectsFound);

if (rv != CKR_OK) {
    printf("C_FindObjects failed on token %d,error code =
        %x\n",slotId,rv);
    return 0;
}

rv = C_FindObjectsFinal(hSession); (18)

if (rv != CKR_OK) {
    printf("C_FindObjectsFinal failed on token %d,error code =

```

```
        %x\n",slotId,rv);

                                                                    return 0;

    }

    if (numObjectsFound == 0) {
        printf("Object not found.\n");
        return 0;
    }

    rv = C_VerifyInit (hSession, &mech, hPubKey);                      (19)

    rv = C_Verify(hSession,Data,sizeof(Data),pSignature,SignatureLen); (20)

    if (rv != CKR_OK) {
        printf("Verify signature failed on token %d,error code =
            %x\n",slotId,rv);

        return 0;
    }

    else {
        printf("Verify signature success!\n");
    }

    rv = C_CloseSession(hSession);

    rv = C_Finalize(NULL);

    return 0;
}
```

## 7.4 習題

1. 使用C\_Sign 及C\_SignRecover 函式前各要將那一個金鑰屬性設為TRUE。
2. 建立一把 DES 金鑰，運算機制設為 CKM\_DES\_MAC，計算資料的 MAC(message authentication code)值。
3. 將範例程式的簽章機制設為CKM\_RSA\_PKCS，重新簽章及驗證。
4. 使用C\_SignRecover 及C\_VerifyRecover 簽章資料及驗證。
5. 使用DSA 演算法來簽章及驗證資料。

## 實驗8. 金鑰管理

### 8.1 實驗目的

本實驗主要在於了解 Cryptoki 中所提供有關金鑰管理的函式功能，以及 Safenet 所提供額外的金鑰管理函式庫(kmllib)。

### 8.2 相關原理說明

在密碼機制的背後，往往需要一個安全的金鑰管理(Key Management)機制來管理各式各樣的密碼金鑰。金鑰管理可參考 ANSI X9.17 標準，而一般管理規劃項目如下：

1. 為不同的密碼系統及不同的應用產生金鑰。
2. 產生及獲得公開金鑰憑證。
3. 金鑰分送給預期的使用者。
4. 金鑰儲存，包含如何授權使用者存取金鑰。
5. 更換及更新金鑰。
6. 處理洩露的金鑰。
7. 金鑰註銷。
8. 金鑰封存。
9. 金鑰銷毀。
10. 記錄及稽核金鑰管理的相關行為。

Cryptoki 函式庫提供下列的金鑰管理相關函式：

1. C\_GenerateKey：產生對稱式金鑰。
2. C\_GenerateKeyPair：產生金鑰對。
3. C\_WrapKey：用來加密保護金鑰，用於金鑰分送的金鑰輸出(key export)。
4. C\_UnwrapKey：用來解密金鑰，用於金鑰分送的金鑰輸入(key import)。
5. C\_DeriveKey：衍生金鑰。

其中 C\_GenerateKey 及 C\_GenerateKeyPair 已經於實驗五金鑰產生中介紹。鑑於 Cryptoki 中相關金鑰管理函式之不足，Safenet 於 PKCS#11 之上額外開發了金鑰管理相關函式。在知道 Safenet 提供那些金鑰管理函式之前，我們先了解目前業界有關金鑰管理的做法。我們將針對金鑰分送做特別介紹，因為在上述 10

項的金鑰管理項目中，除了金鑰分送外，每一組織或公司的做法不盡相同。但由於金鑰分送機制必須有一套相同的做法，兩個單位之間才可以成功的交換共享金鑰。在交換金鑰的時候，金鑰一定要經過一把稱為 transport key 或是 KEK(key encryption key) 的金鑰加密保護再傳送，此方法可使用 C\_WrapKey 及 C\_UnwrapKey 來達成而雙方建立共同 transport key 的方法通常為將 transport key 分成 2 或 3 個分持值(part)，每一個分持值由一個人保管。當要至對方處建立 transport key 時，所有的持有人要全員到齊(視分享機制而定，也有一定人數以上 即可的分享機制)，並各輸入其分持值後重建此金鑰，常見的組合方式為互斥或。此方法即為秘密分享(secret sharing)。而分持值的保管方式最常見的為印在紙上，也可以使用 smart card 來儲存。兩種方式各有其優缺點，紙張容易污損，但是在建立的時候較簡便，只要系統提供一輸入介面即可。Smart card 較安全，但是建立系統較複雜，雙方都要有同一套的讀卡系統。而秘密分享也可以用在金鑰的回復上，通常重要的金鑰都要備份儲存，而做法也是分成許多分持值。當金鑰需重新回復出來時，則須將這些分持值集合起來以回復金鑰。分持值的方式在 PKCS#11 中並沒有提供此功能，但是可由 Safenet 額外提供的金鑰管理函式來達成。

關於Safenet 實作的金鑰管理函式詳細定義可參考文件 *Management Library Programmers Manual*。本實驗將介紹如下常用函式：

1. KM\_GenerateSecretKey: 建立一把secret key 且產生分持值。
2. KM\_ImportFromScreen: 從螢幕上輸入各分持值後建立金鑰。

在產生分持值的時候，也要為每一個分持值產生 KCV(Key Check Value，金鑰檢查碼)，以及這把金鑰的 KCV。比方說產生一把金鑰並將其分成 3 個分持值，則每個分持值會有 1 個 KCV，而金鑰本身也會有 KCV，共 4 個 KCV。KCV 的產生方式為使用分持值或金鑰加密 8 byte 的 0 值。再從 8byte 的加密結果中取 MSB 的3 個byte 當做KCV，也就是6 位16 進制數字。KCV 的主要用意是防止 持有者在使用分持值重建金鑰時，輸入錯誤的分持值時檢查用的。

本實驗所會用到的cyptoki 函式及kmlib 函式其宣告定義如下：

1. CK\_RV C\_WrapKey( CK\_SESSION\_HANDLE  
hSession, CK\_MECHANISM\_PTR  
pMechanism, CK\_OBJECT\_HANDLE  
hWrappingKey,

```
    CK_OBJECT_HANDLE hKey,  
    CK_BYTE_PTR pWrappedKey,  
    CK_ULONG_PTR pulWrappedKeyLen  
}
```

hSession: 所建立session 的handle。

pMechanism: 加密演算法。

hWrappingKey: 用來執行加密的金鑰。

HKey: 被加密的金鑰。

PWrappedKey: 存放加密結果。

PulWrappedKeyLen: 加密結果的長度。

```
2. CK_RV C_UnwrapKey( CK_SESSION_HANDLE  
    hSession, CK_MECHANISM_PTR  
    pMechanism, CK_OBJECT_HANDLE  
    hUnwrappingKey, CK_BYTE_PTR  
    pWrappedKey, CK_ULONG  
    ulWrappedKeyLen, CK_ATTRIBUTE_PTR  
    pTemplate, CK_ULONG ulAttributeCount,  
    CK_OBJECT_HANDLE_PTR phKey  
)
```

hSession: 所建立session 的handle。

pMechanism: 解密演算法。

hUnwrappingKey: 用來執行解密的金鑰。

pWrappedKey: 加密結果。

ulWrappedKeyLen: 加密結果的長度。

pTemplate: 要建立金鑰的樣板。

ulAttributeCount: 樣板屬性個數。

phKey: 產生金鑰的handle

```
3. CK_RV KM_GenerateSecretKey  
    ( CK_SESSION_HANDLE hSession,  
    CK_KEY_TYPE keyType,
```



```
    CK_SIZE keySizeInBits,  
    CK_ATTRIBUTE * pTpl,  
    CK_COUNT tplSize,  
    CK_COUNT numComps,  
    CK_OBJECT_HANDLE * phKey  
)
```

hSession: 所建立session 的handle。

keyType: 金鑰種類。

keySizeInBits: 金鑰長度位元。

pTpl: 要建立金鑰的樣板。

tplSize: 樣板屬性個數。

NumComps: 分持值個數。

phKey: 產生金鑰的handle。

#### 4. CK\_RV KM\_ImportFromScreen

```
( CK_SESSION_HANDLE hSession,  
  CK_CHAR * pszLabel,  
  CK_KEY_TYPE keyType,  
  CK_SIZE keySizeInBits,  
  CK_ATTRIBUTE * pTpl,  
  CK_COUNT tplSize,  
  CK_COUNT numComps,  
  CK_OBJECT_HANDLE hUnwrapKey,  
  CK_BBOOL isEncMultiPart  
)
```

hSession: 所建立session 的handle。

pszLabel: 金鑰的label。

keyType: 金鑰種類。

keySizeInBits: 金鑰長度位元。

pTpl: 要建立金鑰的樣板。

tplSize: 樣板屬性個數。

numComps: 分持值個數。

hUnwrapKey: 用來解密的金鑰，如果分持值為明文則填入

CK\_INVALID\_HANDLE。

IsEncMultiPart: 分持值是否有加密。

## 8.3 實驗範例

### 8.3.1 Wrap 及Unwrap 金鑰

在實驗之前必須先於 token 中建立 2 把秘密金鑰一把為加密金鑰(wrapping key)，另一把為被加密金鑰(wrapped key)。建立方式可參考實驗 5 並撰寫程式碼產生，產生時請將加密金鑰的label 設為wrapping key，而被加密金鑰的label 設為wrapped key。或是可以使用Safenet 所提供的KMU(key management utility)軟體來產生，產生步驟如下：

1. 於程式集裏的Safenet 中開啟KMU 軟體，如圖8-1。
2. 選擇 token0 或是其它可供操作的 token，並輸入 user 的 PIN 碼登入，如圖8-2。



ProtectServer External

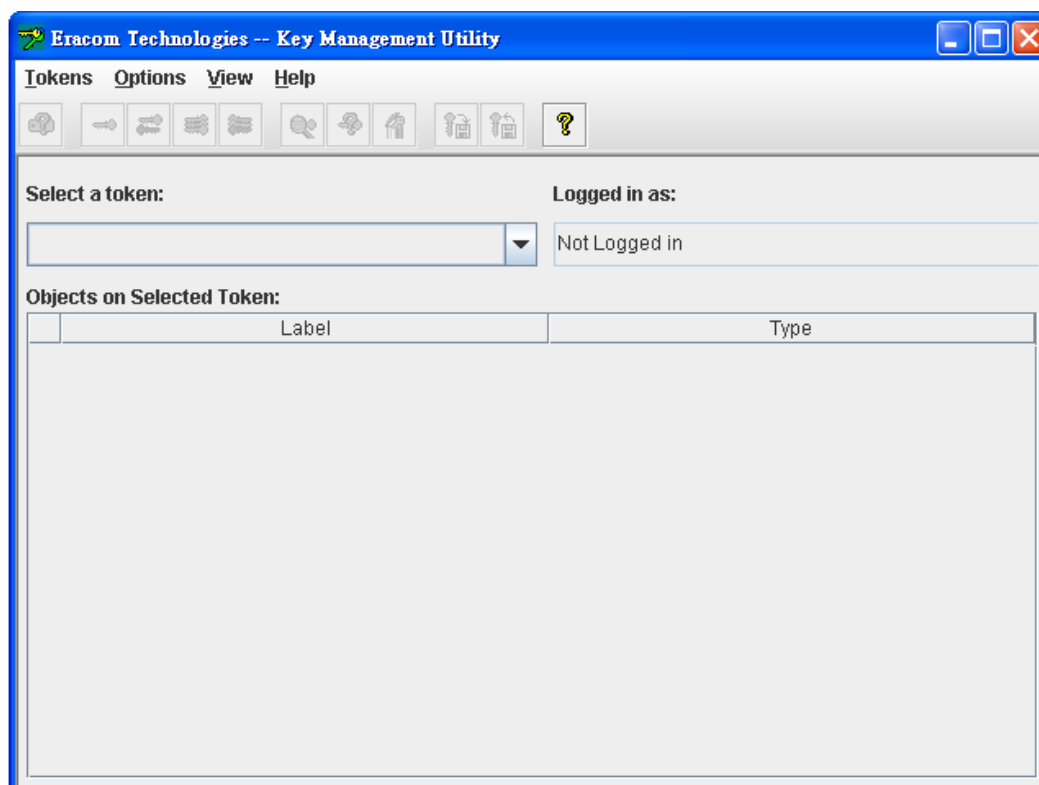


圖8-1:KMU 軟體

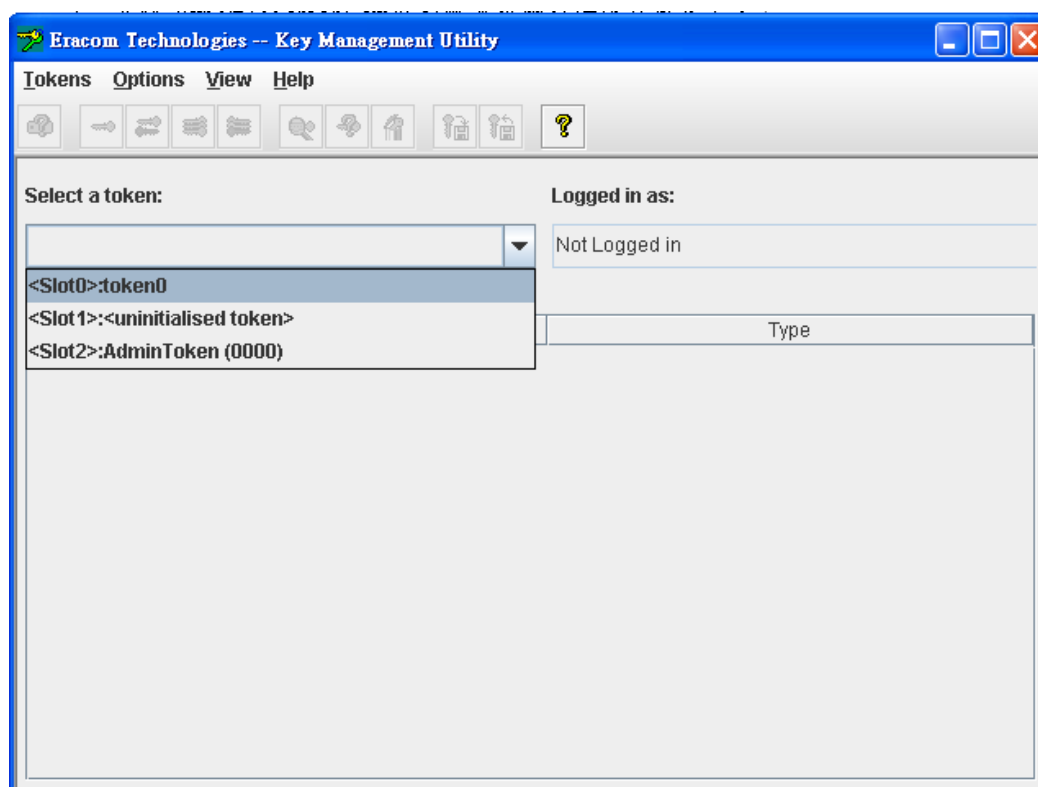


圖8-2: 登入token

3. 按下產生秘密金鑰的按鍵，如圖8-3。
4. 在機制(mechanism)欄位中選擇Double DES 以建立Triple DES with 2 key 的金鑰並在 label 欄位中填入 Wrapping key 接下來選擇屬性可取消 Sensitive 屬性，如此便可以看到其所產生的金鑰值。也可取消 Private 屬性，這樣不用登入也可以看到此把金鑰。設定完成後按下OK 便可產生秘密金鑰。如圖8-4。
5. 接下來重覆步驟 3 及 4 建立 label 為 Wrapped key 的秘密金鑰。要確認 Wrapping key 的 Wrap 屬性及 Unwrap 屬性都要勾選。而 Wrapped key 的 Extractable 屬性要勾選。
6. 產生成功的話會顯示此兩把金鑰，如圖8-5。

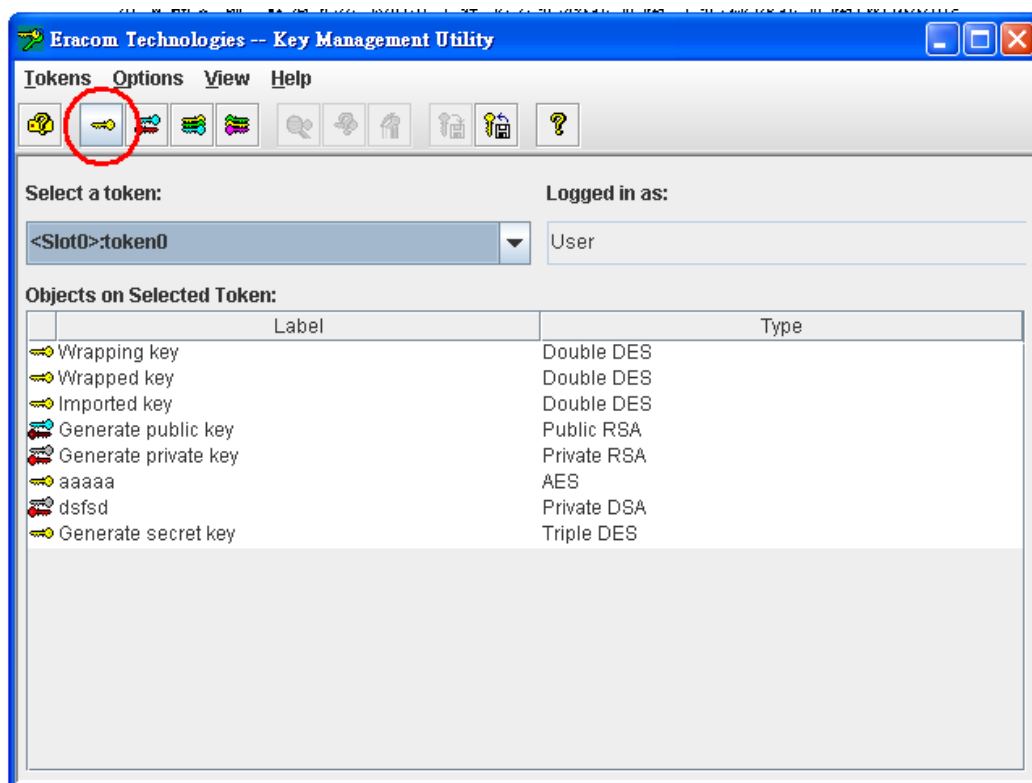


圖8-3:產生秘密金鑰



圖8-4:金鑰設定

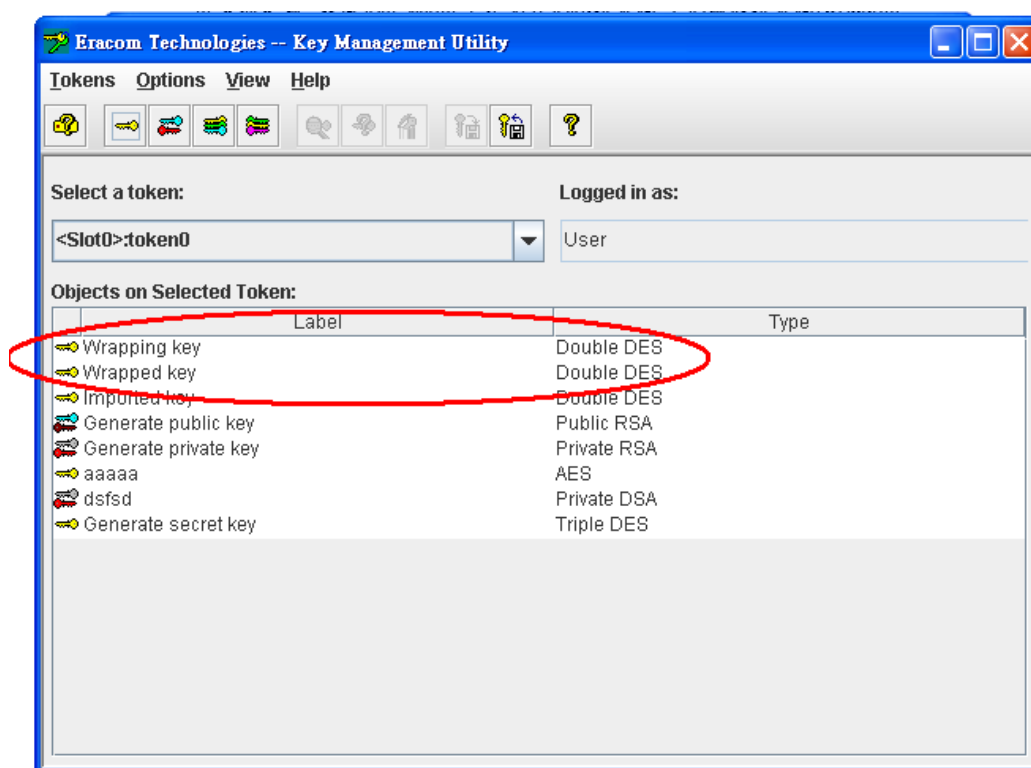


圖8-5:加密及被加密金鑰

程式 EX8\_1 之主要功能為 Wrap 及 Unwrap 金鑰。本程式有使用到 util.lib 所以要將此函式庫放在連結路徑上。由於要先尋找前所建立的兩把金鑰，因此定義金鑰的label[EX8\_1(1)]及物件的種類為CKO\_SECRET\_KEY[EX8\_1(2)]。首先尋找wrapping key，先定義其樣板[EX8\_1(3)]後開始尋找[EX8\_1(4)(5)(6)(7)(8)]，找到的 wrapping key 會指向 hWrappingKey。接下來尋找 wrapped key[EX8\_1(9)(10)(11)(12)(13)]，找到的 wrapped key 會指向 hWrappedKey。定義金鑰加密機制為 CKM\_DES3\_ECB[EX8\_1(14)]。再來即可呼叫 C\_WrapKey 來進行金鑰加密，也就是使用 wrapping Key 來加密 wrapped key[EX8\_1(15)]。加密結果放在 wrappedkey 陣列中，長度為 16byte，結果也會被印至螢幕上。此加密過後的金鑰值即可用於金鑰分送。

下一步驟進行金鑰解密，也就是把剛才加密過後的值解密並將解密過後的金鑰建立至PSE 中。首先定義待會建立至PSE 中的金鑰種類為CKK\_DES2 及

label 為 Imported key[EX8\_1(16)]，並定義樣板[EX8\_1(17)]。接下來即可呼叫 C\_UnwrapKey 進行解密，在此我們將剛才存放加密結果的 wrappedkey 陣列傳入[EX8\_1(18)]。而解密金鑰同加密金鑰，解密機制同加密機制為 CKM\_DES3\_ECB。解密成功後即可在PSE 建立label 為Imported key 的triple DES with 2 key 的金鑰。

### EX8\_1 Wrap及Unwrap金鑰

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "cryptoki.h"

#include "hex2bin.h"

int main(int argc, char* argv[])
{
    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hWrappingKey = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hWrappedKey = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hImportedKey = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    CK_CHAR wrappingkeyLabel[] = "Wrapping key";           (1)

    CK_CHAR wrappedkeyLabel[] = "Wrapped key";

    CK_OBJECT_CLASS objClass = CKO_SECRET_KEY;             (2)

    CK_BBOOL bTrue = TRUE;

    CK_BBOOL bFalse = FALSE;

    rv = C_Initialize(NULL);

    rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

    if (rv != CKR_OK) {

        printf("Could not open session on token %d,error code =

                %x\n",slotId,rv);
```

```
        return 0;

    }

    CK_ATTRIBUTE wrappingkeyTemplate[] = {                                (3)
        {CKA_CLASS, &objClass, sizeof(objClass)},
        {CKA_LABEL, wrappingkeyLabel, sizeof(wrappingkeyLabel)-1}
    };

    CK_SIZE templateSize = sizeof(priobjectTemplate) / sizeof(CK_ATTRIBUTE);
    rv = C_FindObjectsInit(hSession, wrappingkeyTemplate, templateSize); (4)
    if (rv != CKR_OK) {
        printf("C_FindObjectsInit failed on token %d,error code =
            %x\n",slotId,rv);
        return 0;
    }

    CK_ULONG numObjectsToFind = 1;                                       (5)
    CK_ULONG numObjectsFound = 0;

    rv = C_FindObjects(hSession,& hWrappingKey,                          (6)
        numObjectsToFind,&numObjectsFound);

    if (rv != CKR_OK) {
        printf("C_FindObjects failed on token %d,error code =
            %x\n",slotId,rv);
        return 0;
    }

    rv = C_FindObjectsFinal(hSession);                                    (7)
    if (rv != CKR_OK) {
        printf("C_FindObjectsFinal failed on token %d,error code =
            %x\n",slotId,rv);
        return 0;
    }

    if (numObjectsFound == 0){                                           (8)
        printf("Object not found.\n");
    }
```



```
        return 0;
    }

    CK_ATTRIBUTE wrappedkeyTemplate[] = {
        {CKA_CLASS, &objClass, sizeof(objClass)},
        {CKA_LABEL, wrappedkeyLabel, sizeof(wrappedkeyLabel)-1}
    };

    templateSize = sizeof(wrappedkeyTemplate) / sizeof(CK_ATTRIBUTE);
    rv = C_FindObjectsInit(hSession, wrappedkeyTemplate, templateSize); (10)
    if (rv != CKR_OK) {
        printf("C_FindObjectsInit failed on token %d,error code =
            %x\n",slotId,rv);

        return 0;
    }

    numObjectsToFind = 1;
    numObjectsFound = 0;

    rv = C_FindObjects(hSession,&hWrappedKey,
        numObjectsToFind,&numObjectsFound); (11)

    if (rv != CKR_OK) {
        printf("C_FindObjects failed on token %d,error code =
            %x\n",slotId,rv);

        return 0;
    }

    rv = C_FindObjectsFinal(hSession); (12)

    if (rv != CKR_OK) {
        printf("C_FindObjectsFinal failed on token %d,error code =
            %x\n",slotId,rv);

        return 0;
    }

    if (numObjectsFound == 0){ (13)
        printf("Object not found.\n");
    }
```

```

        return 0;
    }

    CK_MECHANISM mech = {CKM_DES3_ECB, NULL, 0};
    CK_ULONG wrappedKeyLen = 0;
    CK_BYTE wrappedKey[16] = {0};
    char Data_str[60] = {0};
    wrappedKeyLen = sizeof(wrappedKey);

    rv = C_WrapKey(hSession, &mech, hWrappingKey, hWrappedKey, wrappedKey,
                  &wrappedKeyLen);
    if (rv != CKR_OK) {
        printf("Wrap key failed on token %d,error code = %x\n",slotId,rv);
        return 0;
    }

    bin2hex(Data_str,wrappedKey,wrappedKeyLen);
    Data_str[wrappedKeyLen] = '\0';
    printf("wrapped(encrypted) key value = %s\n",Data_str);

    CK_KEY_TYPE keyType = CKK_DES2;

    CK_CHAR importedkeyLabel[] = "Imported key";

    CK_ATTRIBUTE importedkeytpl[] = {
        {CKA_LABEL, importedkeyLabel, sizeof(importedkeyLabel)-1},
        {CKA_TOKEN,      &bTrue,      sizeof(CK_BBOOL)},
        {CKA_WRAP,       &bFalse,     sizeof(CK_BBOOL)},
        {CKA_UNWRAP,     &bFalse,     sizeof(CK_BBOOL)},
        {CKA_EXTRACTABLE, &bFalse,    sizeof(CK_BBOOL)},
        {CKA_SENSITIVE,  &bFalse,    sizeof(CK_BBOOL)},
        {CKA_KEY_TYPE,   &keyType,    sizeof(CK_KEY_TYPE)}
    };

    templateSize = sizeof(importedkeytpl)/sizeof(CK_ATTRIBUTE);

    rv = C_UnwrapKey(hSession,&mech, hWrappingKey, wrappedKey,

```

```

        wrappedKeyLen,importedkeytpl, templateSize,&hImportedKey);          (18)

    if (rv != CKR_OK) {

        printf("Unwrap key failed on token %d,error code = %x\n",slotId,rv);

        return 0;

    }

    rv = C_CloseSession(hSession);

    rv = C_Finalize(NULL);

    return 0;

}

```

### 8.3.2 產生金鑰分持值(key component)

程式EX8\_2 會產生一把 triple DES with 2 key 的金鑰，在建立的同時會將其拆成 3 個分持值，並將各分持值及 KCV 輸出在螢幕上。此程式會使用到 kmllib.lib、cryptoki2.lib、ctextra2.lib、ctutil2.lib 及 util.lib。所以這些函式庫都要加入連結路徑中。首先我們要實作 3 個callback 函式，因為kmllib 在使用時會呼叫一些 callback 函式來印出訊息及取得使用者所輸入的字串。在此我們實作 3 個 callback 函式。第一個為 PromptString，用來取得使用者輸入的字元。第二個為 ShowKeyComponent，功能為將產生的分持值印在螢幕上。第三為 ShoMsg，用來將訊息印在螢幕上[EX8\_2(1)(2)(3)]。

產生金鑰分持值時必須登入成SO 或USER。在此登入成SO，因此定義SO 的 PIN，此處需改成符合你 token 的 SO PIN[EX8\_2(4)]。接下來定義一個存放 callback 函式的結構變數km\_callbacks[EX8\_2(5)]，並將其初始化為0。接下來給定此結構變數其值，分別是剛才所定義的 3 個實作的函式[EX8\_2(6)]。設定完成後即可呼叫 KM\_SetCallbacks[EX8\_2(7)] 設定 callback 函式。再來先登入至 SO[EX8\_2(8)]。之後定義此金鑰的樣板[EX8\_2(9)]。最後即可呼叫 KM\_GenerateSecretKey 來產生金鑰分持值，在此產生 triple DES 2 key 的金鑰，也就是128 位元，分持值數量為3。其結果都會印在螢幕上。這些分持值即可由不同的持有者保管，當需重建金鑰時，再由這些持有者集合重建。KCV 是確保持有者在輸入時是否不小心輸入錯誤分持值的檢查碼。執行結果如圖8-6 所示。

**EX8\_2 產生金鑰分持值**

```

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "kmlib.h"

#include "ctextra.h"

static CK_RV PromptString(const char*, char*,CK_ULONG* );

static void ShowMsg( UICB_MsgType_t, const char*);

static CK_RV ShowKeyComponent(int compNum,int numComps,const char*
pKeyComp,const char* pKCV);

static CK_RV PromptString(const char* pszMessage,char* pBuf,CK_ULONG* pBufLen)
{
    (1)

    CK_RV rv = CKR_OK;

    printf("%s", pszMessage);

    ReadLine(pBuf, *pBufLen);

    return rv;
}

static CK_RV ShowKeyComponent(int compNum,int numComps,const char*
pKeyComp,const char* pKCV)
{
    (2)

    CK_RV rv = CKR_OK;

    printf("key component %d of %d: %s KCV: %s",
        compNum,numComps,pKeyComp,pKCV);

    return rv;
}

static void ShowMsg( UICB_MsgType_t msgType, const char* pszMessage)
{
    (3)

    printf("%s", pszMessage);
}

int main(int argc, char* argv[])

```

```

{

    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hSecKey = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    CK_CHAR so_pin[] = "allen"; (4)

    CK_CHAR secLabel[] = "Secret key";

    CK_BBOOL bTrue = TRUE;

    CK_BBOOL bFalse = FALSE;

    KM_Callbacks_t km_callbacks; (5)

    memset(&km_callbacks, 0, sizeof(KM_Callbacks_t));

    km_callbacks.promptString = PromptString; (6)

    km_callbacks.showMsg = ShowMsg;

    km_callbacks.showKeyComponent = ShowKeyComponent;

    rv = KM_SetCallbacks(&km_callbacks); (7)

    if (rv != CKR_OK){

        printf("set callbacks function failed\n");

        return 0;

    }

    rv = C_Initialize(NULL);

    rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

    rv = C_Login(hSession, CKU_SO, so_pin, strlen((char*)so_pin)); (8)

    CK_ATTRIBUTE secKeyTpl [] = { (9)

        {CKA_LABEL, secLabel, sizeof(secLabel)-1},

        {CKA_TOKEN, &bTrue, sizeof(CK_BBOOL)},

        {CKA_WRAP, &bTrue, sizeof(CK_BBOOL)},

        {CKA_UNWRAP, &bTrue, sizeof(CK_BBOOL)},

        {CKA_SENSITIVE, &bTrue, sizeof(CK_BBOOL)}

    };

};

```

```

CK_SIZE templateSize = sizeof(secKeyTpl) / sizeof(CK_ATTRIBUTE);

rv = KM_GenerateSecretKey( hSession, CKK_DES3, 128, secKeyTpl,
                           templateSize, 3, &hSecKey);           (10)

if (rv != CKR_OK) {
    printf("Generate Secret Key failed on token %d,error code =
           %x\n",slotId,rv);

    return 0;
}

rv = C_Logout(hSession);

if (rv != CKR_OK) {
    printf("log out failed on token %d,error code = %x\n",slotId,rv);

    return 0;
}

rv = C_CloseSession(hSession);

rv = C_Finalize(NULL);

return 0;
}

```

```

C:\WINDOWS\system32\cmd.exe
D:\PSOE_code\EX8_2\Release>ex8_2
About to generate Key Component 1
Press <ENTER> to continue ...

key component 1 of 3: 48C3B4286FF421A4A328E68AD9E542A4   KCU: 692E4B
About to generate Key Component 2
Press <ENTER> to continue ...

key component 2 of 3: 5D07291B0B73E5EB2FBDDA934EC4346C   KCU: 53AAF1
About to generate Key Component 3
Press <ENTER> to continue ...

key component 3 of 3: 16F963073105EB8F0900091796CFCE6B   KCU: F5A2FA

Key 'Secret key' KCU : C92BC8
Key "Secret key" was created
D:\PSOE_code\EX8_2\Release>

```

圖8-6:執行結果

### 8.3.3 輸入分持值回復金鑰

程式 EX8\_3 會讓使用者在螢幕上輸入分持值後建立金鑰。此程式會使用到 kmllib.lib、cryptoki2.lib、ctextra2.lib、ctutil2.lib 及 util.lib。所以這些函式庫都要加入連結路徑中。首先我們要實作 3 個 callback 函式，第一個為 PromptString，用來取得使用者輸入的字元。第二個為 PromptTokenPin，功能為將取得使用者輸入的分持值。第三為 ShoMsg，用來將訊息印在螢幕上[EX8\_3(1)(2)(3)]。

產生金鑰分持值時必須登入成 SO 或 user。在此登入成 user，因此定義 user 的 PIN，此處需改成符合你 token 的 user PIN[EX8\_3(4)]。接下來定義一個存放 callback 函式的結構變數 km\_callbacks[EX8\_3(5)]，並將其初始化為 0。接下來給定此結構變數其值，分別是剛才所定義的 3 個實作的函式[EX8\_3(6)]。設定完成後即可呼叫 KM\_SetCallbacks[EX8\_3(7)]設定 callback 函式。再來先登入至 user [EX8\_3(8)]。之後定義此金鑰的樣板 [EX8\_3(9)]。最後即可呼叫 KM\_ImportFromScreen 來輸入分持值回復金鑰，此處回復的金鑰為 triple DES 2 key，也就是 128 位元，分持值數量為 3。在此採用程式 EX8\_2 所產生的 3 個分持值來輸入，驗證 KCV 成功後即可在 PSE 中回復金鑰。執行結果如圖 8-7 所示。

#### EX8\_3 金鑰回復

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "kmllib.h"

#include "ctextra.h"

static CK_RV PromptString(const char*, char*,CK_ULONG* );

static CK_RV PromptTokenPin(const char* pszMessage,CK_CHAR *pPIN,CK_ULONG*
                             pPinLen);

static void ShowMsg( UICB_MsgType_t, const char*);

static CK_RV PromptString(const char* pszMessage,char* pBuf,CK_ULONG* pBufLen)
{
    CK_RV rv = CKR_OK;

    printf("%s", pszMessage);
    (1)
```

```
ReadLine(pBuf, *pBufLen);

return rv;
}

static CK_RV PromptTokenPin(const char* pszMessage, CK_CHAR *pPIN, CK_ULONG*
pPinLen) (2)
{
    printf("%s", pszMessage);

    ReadLine((char*)pPIN, *pPinLen);

    *pPinLen = *pPinLen - 1;

    return CKR_OK;
}

static void ShowMsg( UICB_MsgType_t msgType, const char* pszMessage)
{ (3)
    printf("%s", pszMessage);
}

int main(int argc, char* argv[])
{
    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    CK_CHAR so_user[] = "0000"; (4)

    CK_BBOOL bTrue = TRUE;

    CK_BBOOL bFalse = FALSE;

    KM_Callbacks_t km_callbacks; (5)

    memset(&km_callbacks, 0, sizeof(KM_Callbacks_t));

    km_callbacks.promptString = PromptString; (6)

    km_callbacks.PromptTokenPin = PromptTokenPin;

    km_callbacks.showKeyComponent = ShowKeyComponent;

    rv = KM_SetCallbacks(&km_callbacks); (7)
```



```

    if (rv != CKR_OK){
        printf("set callbacks function failed\n");
        return 0;
    }

    rv = C_Initialize(NULL);

    rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

    rv = C_Login(hSession, CKU_USER, so_user, strlen((char*)so_user));
(8)

    CK_ATTRIBUTE secKeyTpl [] = {
(9)
        {CKA_LABEL,      Label,      sizeof(Label)-1},
        {CKA_TOKEN,      &bTrue,      sizeof(CK_BBOOL)},
        {CKA_PRIVATE,    &bTrue,      sizeof(CK_BBOOL)},
        {CKA_ENCRYPT,     &bTrue,      sizeof(CK_BBOOL)},
        {CKA_DECRYPT,     &bTrue,      sizeof(CK_BBOOL)},
        {CKA_SIGN,       &bTrue,      sizeof(CK_BBOOL)},
        {CKA_VERIFY,     &bTrue,      sizeof(CK_BBOOL)},
        {CKA_WRAP,       &bTrue,      sizeof(CK_BBOOL)},
        {CKA_UNWRAP,     &bTrue,      sizeof(CK_BBOOL)},
        {CKA_EXPORTABLE, &bTrue,      sizeof(CK_BBOOL)},
        {CKA_EXTRACTABLE, &bTrue,      sizeof(CK_BBOOL)},
        {CKA_MODIFIABLE, &bTrue,      sizeof(CK_BBOOL)},
        {CKA_SENSITIVE,  &bTrue,      sizeof(CK_BBOOL)},
        {CKA_DERIVE,     &bTrue,      sizeof(CK_BBOOL)}};

    }

    CK_SIZE templateSize = sizeof(secKeyTpl) / sizeof(CK_ATTRIBUTE);

    rv = KM_ImportFromScreen(hSession, Label, CKK_DES2, 128, secKeyTpl,
        templateSize, 3, CK_INVALID_HANDLE, bFalse);
(10)

    if (rv != CKR_OK) {
        printf("Import Secret Key failed on token %d,error code =
            %x\n",slotId,rv);
    }

```

```
        return 0;

    }

    rv = C_Logout(hSession);

    if (rv != CKR_OK) {

        printf("log out failed on token %d,error code = %x\n",slotId,rv);

        return 0;

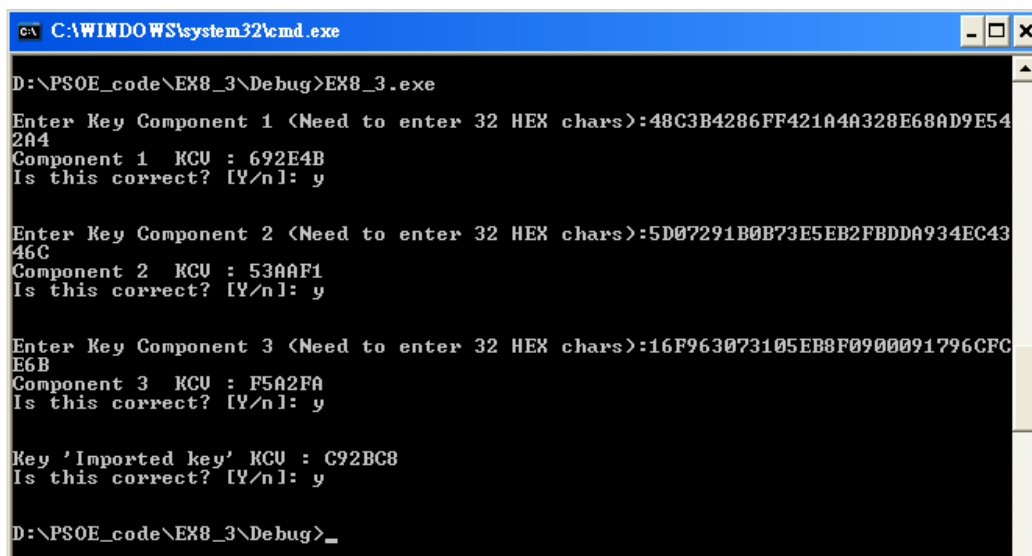
    }

    rv = C_CloseSession(hSession);

    rv = C_Finalize(NULL);

    return 0;

}
```



```
C:\WINDOWS\system32\cmd.exe
D:\PSOE_code\EX8_3\Debug>EX8_3.exe
Enter Key Component 1 <Need to enter 32 HEX chars>:48C3B4286FF421A4A328E68AD9E54
2A4
Component 1 KCU : 692E4B
Is this correct? [Y/n]: y

Enter Key Component 2 <Need to enter 32 HEX chars>:5D07291B0B73E5EB2FBDDA934EC43
46C
Component 2 KCU : 53AAF1
Is this correct? [Y/n]: y

Enter Key Component 3 <Need to enter 32 HEX chars>:16F963073105EB8F0900091796CFC
E6B
Component 3 KCU : F5A2FA
Is this correct? [Y/n]: y

Key 'Imported key' KCU : C92BC8
Is this correct? [Y/n]: y

D:\PSOE_code\EX8_3\Debug>_
```

圖8-7:金鑰回復執行結果

## 8.4 習題

1. 做為一把Wrap 及Unwrap 金鑰，必須設定那些屬性為TRUE。
2. 請建立一把RSA 金鑰對，使用其public key 來Wrap 金鑰，而使用其private key 來解密金鑰。

3. 金鑰檢查碼(KCV)的功能為何。
4. 在使用Safenet 的kmlib 時會何要設定一些callback 函式。
5. 產生AES 金鑰其分持值個數為2，並再次輸入分持值回復金鑰。
6. 參考 kmlib 文件，如果想將存在於 PSE 中秘密金鑰輸出成 3 個加密過後的分持值，要使用那一個函式。請實作程式進行輸出及輸入。



ProtectServer External

## 實驗9. 單向雜湊函數

### 9.1 實驗目的

學習如何使用cryptoki 中所提供的雜湊函式功能，進行雜湊值的運算。

### 9.2 相關原理說明

單向雜湊函數值h(Hash Value)或稱彙記(Digest)可以下列的式子表示：

$$h=H(M)$$

其中H( )為單向雜湊函數，M 是任意長度的資料，而h 則是固定長度的值。因 H( )為單向函數，故由 h 無法反求原文 M。H( )需滿足給定 M 及其雜湊函數值 H(M)=h，很難再找到另一個 N ( N ≠ M) 使得其雜湊函數值 H(N)也等於h。因此發送方在傳送訊息時，若同時送出訊息 M 與雜湊函數值 h，接收方在驗證時則必須計算訊息 M 的雜湊函數值，經過與一起傳送來的雜湊函數值 h 比較後，若無誤則可確認傳送訊息M 與其 h 為同一對資料。常見的演算法有MD5、SHA1等。

Cryptoki 中提供雜湊運算的函式如下：

1. C\_DigestInit: 初始化雜湊運算，其函式原型定義如下。

```
CK_RV C_DigestInit
( CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism)
```

hSession: 所建立session 的handle。

pMechanism: 雜湊演算法。

2. C\_Digest: 對資料進行雜湊運算，其函式原型定義如下。

```
CK_RV C_Digest
( CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pData,
  CK_ULONG ulDataLen,
  CK_BYTE_PTR pDigest,
  CK_ULONG_PTR pDigest)
```

hSession: 所建立session 的handle。  
pData: 要進行雜湊運算的資料。  
ulDataLen: 要進行雜湊運算資料的長度。  
pDigest: 雜湊值。  
pDigest: 雜湊值的長度。

### 9.3 實驗範例

程式EX9\_1 功能為執行資料的SHA1 雜湊運算,其雜湊值長度為20 位元組。首先定義雜湊演算法為SHA1[EX9\_1(1)]及要進行雜湊運算的資料[EX9\_1(2)]。再宣告存放結果的陣列[EX9\_1(3)]。接下來呼叫C\_DigestInit 初始化雜湊運算[EX9\_1(4)]。再呼叫C\_Digest 進行雜湊運算[EX9\_1(5)],雜湊值將會印在螢幕上。

#### EX9\_1 雜湊值運算

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "cryptoki.h"

int main(int argc, char* argv[])
{
    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    rv = C_Initialize(NULL);

    rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

    CK_MECHANISM mech = { CKM_SHA_1, NULL, 0 };           (1)

    CK_BYTE data[] =
        { 0x6B, 0x61, 0x72, 0x79, 0x2D, 0x76, 0x69, 0x63, 0x6B, 0x79 };   (2)

    CK_BYTE digest[20];                                           (3)

    CK_ULONG ulDigestLen;

    ulDigestLen = sizeof(digest);
```

```
rv = C_DigestInit(hSession, &mech); (4)

if (rv != CKR_OK) {

    printf("C_DigestInit failed on token %d,error code =%x\n",slotId,rv);

    return 0;

}

rv = C_Digest(hSession, data, sizeof(data), digest, &ulDigestLen); (5)

if (rv != CKR_OK) {

    printf("Digest failed on token %d,error code = %x\n",slotId,rv);

    return 0;

}

printf("hash vlaue = ");

for (int i=0;i<ulDigestLen;i++)

    printf("%02x",digest[i]);

rv = C_CloseSession(hSession);

rv = C_Finalize(NULL);

return 0;

}
```

## 9.4 習題

1. 請使用MD5 及RIPEMD-160 演算法進行雜湊運算。
2. 參考pkcs#11 文件雜湊運算部分，如何進行multi-part 資料的雜湊運算。



ProtectServer Gold

## 實驗10. 憑證管理

### 10.1 實驗目的

學習如何產生X.509 憑證及產生PKCS#10 憑證申請檔。

### 10.2 相關原理說明

在公開金鑰系統中，須存在一值得信賴的第三者來簽署用戶的公鑰，即所謂的憑證，而此公正的第三者我們稱為憑證管理中心(Certification Authority,CA)。國際標準組織 ITU-T建議使用 X.509 做為 X.500 目錄檢索服務(Directory Service)的一部分[1]。X.500是對於分散式網路中儲存用戶訊息的資料庫所提供的目錄檢索服務的標準。而X.509 是利用公開密碼技術對X.500 的服務所提供的 鑑別服務的協議標準。X.509 的最初版本發表於1988 年經過修訂的版本於1993 年發佈。第三版則於1995 年發表。X.509 是基於使用公開密碼系統及數位簽章，而且標準並沒有指定特定的演算法，但是建議採用 RSA。X.509 中使用的數位簽名也同時需要雜湊函與之同時配合使用。

X.509 計畫是將公鑰憑證與每一使用者產生關聯。這些使用者憑證是由一些受信賴的憑證管理中心發行，並且由 CA 或由使用者將其放至目錄。圖 10-1 為 X.509 憑證格式。



ProtectServer External

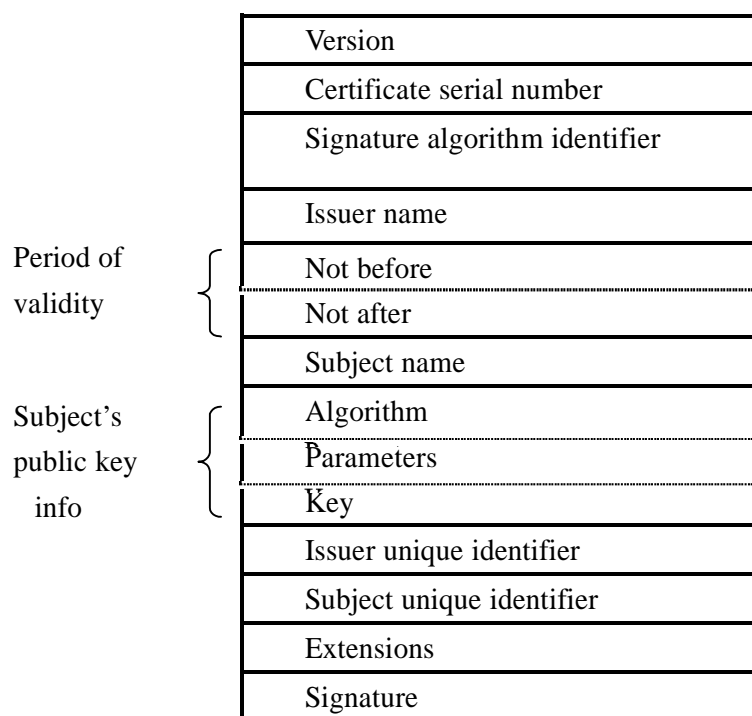


圖10-1 為X.509 憑證格式

- 版本(Version): 憑證製作所依據的X.509 版序。
- 序號(Serial Number): 由憑證中心所給定唯一憑證序號。
- 簽章演算法識別符(Signature Algorithm Identifier): CA 簽署本憑證所使用的演算法。
- 簽發單位名稱(Issuer name): 簽署本憑證之CA 名稱,依據X.500 方式命名。
- 憑證有效期限(Period of Validity): 本憑證生效日期與截止日期。
- 使用者名稱(Subject Name): 依據X.500 命名方式所命名的使用者名稱。
- 公開金鑰資料(Public Key Info): 公開金鑰的值與演算法名稱。
- 簽發者識別符(Issuer Unique Identifier): 簽署本憑證之認證中心獨有的識別號碼。
- 使用者識別符(Subject Unique Identifier): 使用者獨有唯一識別碼。
- X.509 擴充欄位(Extension): 主要有三種型態, 金鑰及政策資訊, 使用者及發行者屬性, 以及憑證路徑限制。在第三版之後才具有此欄位。詳細內容將



在下文做說明。

- 簽章(Signature): CA 對以上資料使用其私密金鑰做數位簽章運算所得到的簽章值，此部分並包含簽章演算法識別符。

在第三版中，包含了一些選擇性的擴充欄位。每一個擴充欄位是由擴充欄位識別符(Extension Identifier)，必要性指示(Criticality Indicator)以及擴充欄位值(Extension Value)所組成。若必要性值被設為真(True)，則代表此欄位非常重要。若應用程式或系統無法處理該欄位的相關設定，則必須將該憑證視同無效。

憑證擴充欄位被分成三種種類，其詳細內容如下：

#### 1. 金鑰及政策資訊(Key and Policy Information)

- 簽發者金鑰識別符(Authority Key Identifier): 憑證簽發者均會定期或不定期更新金鑰，為了在金鑰更換之後，還能找到正確的公鑰憑證來驗證簽章，必須可以識別簽發者所使用的上一把金鑰，及其相對的公鑰憑證。其目的也就是為了日後識別金鑰所用。
- 使用者金鑰識別符(Subject Key Identifier): 同上，使用在使用者的金鑰更新。由於使用者可能針對不同的用途而具有多個金鑰對及對應的憑證，因此也可以藉此達到識別的目的。
- 金鑰用途(Key Usage): 由於金鑰具有不同功能用途，為了維持正確使用金鑰，有必要強制在公鑰憑證中附加用途限制的欄位。
- 私密金鑰的使用時間限制(Private-Key Usage Period): 限制簽章用金鑰的期限，一般來說，簽章用的金鑰，期限較短於相對驗證用的公鑰。
- 憑證簽發原則與政策(Certificate Policies): 包含了一序列之政策選項，本項訊息指示憑證使用之政策訊息，允許以最簡單明瞭方式顯現。
- 政策對應方式(Policy Mapping): 使用在一 CA 的憑證是由其它 CA 所簽發時。不同的 CA 其簽發憑證的策略必定不盡相同，因此本欄位讓使用者對於不同 CA 簽發的憑證策略可以視為是自己所屬 CA 的對應策略，以接受 CA 互相簽發的憑證。

#### 2. 使用者及發行者屬性(Certificate Subject and Issuer Attributes)

此欄位可支援憑證使用者或發行者使用替代名稱 (Alternative Name) 來命名，以達到保護使用者或發行者隱私的目的。

- 使用者替代名稱 (Subject Alternative Name): 此欄位可包含使用不同格式的一或多個替代名稱，可以支援電子郵件，EDI及IPSec等命名方式。
- 簽發者替代名稱 (Issuer Alternative Name): 如使用者替代名稱所述。
- 使用者目錄屬性 (Subject Directory Attributes): 傳達任何要求的 X.500 目錄屬性值給憑證使用者。

### 3. 憑證路徑限制 (Certificate Path Constraints)

此欄位允許將限制規格放入 CA 互相簽發的憑證當中。

- 基本限制 (Basic Constraints): 限制所發出憑證的用途為個人用或 CA 用。
- 名稱限制 (Name Constraints): 此欄位用以限制 CA 名稱，目的在於避免簽發憑證給不相關的個體。
- 策略限制 (Policy Constraints): 具體說明憑證策略。

在 PKCS#11 中並沒有定義產生憑證及憑證申請檔的功能。但在 Protect Toolkit C 中有額外提供此機制。可透過 C\_DeriveKey 函式搭配 CKM\_ENCODE\_X\_509 機制可產生 X509 憑證。如搭配 CKM\_ENCODE\_PKCS\_10 可產生憑證申請檔。

由於 PKCS#11 沒有產生憑證的函式，因此 Protect Toolkit C 使用 C\_DeriveKey 來產生憑證。此函式如下所定義：

CK\_RV

```
C_DeriveKey( CK_SESSION_HANDLE  
hSession, CK_MECHANISM_PTR  
pMechanism, CK_OBJECT_HANDLE  
hBaseKey, CK_ATTRIBUTE_PTR  
pTemplate, CK_ULONG  
ulAttributeCount,  
CK_OBJECT_HANDLE_PTR phKey  
);
```

hSession: 所建立session 的handle。

pMechanism: 原為衍生金鑰的機制，用於憑證產生的話為憑證格式。

hBaseKey: 用於產生憑證的話為要產生憑證的公鑰。

pTemplate: 用於產生憑證的話為憑證樣板。

ulAttributeCount: 樣板個數。

phKey: 用於產生憑證的話為產生憑證的handle。

## 10.3 實驗範例

### 10.3.1 產生X509 憑證

程式 EX10\_1 可以產生公鑰憑證。首先產生 CA 的金鑰對[EX10\_1(4)]以便用來發行憑證，產生的演算法為 CKM\_RSA\_PKCS\_KEY\_PAIR\_GEN[EX10\_1(3)]。接下來產生使用者的金鑰對[EX10\_1(5) (6) (7)]。我們將對使用者的公鑰產生憑證。首先定義憑證的序號[EX10\_1(8)]、主旨[EX10\_1(9)]以及發行者[EX10\_1(10)]以及憑證的起始及截止日期[EX10\_1(11)(12)]。再定義憑證樣板[EX10\_1(13)]。接著定義簽章演算法為 CKM\_SHA1\_RSA\_PKCS[EX10\_1(14)]。以及衍生金鑰的演算法為 CKM\_ENCODE\_X\_509[EX10\_1(15)]，在這裏是指產生X509 憑證。再呼叫 C\_SignInit 初始化簽章運算[EX10\_1(16)]及 C\_DeriveKey 產生憑證[EX10\_1(17)]。

#### EX10\_1 產生X509憑證

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "cryptoki.h"

#include "ctvdef.h"

int main(int argc, char* argv[])

{

    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;
```

```
CK_OBJECT_HANDLE hCAPublicKey = CK_INVALID_HANDLE;

CK_OBJECT_HANDLE hCAPrivateKey = CK_INVALID_HANDLE;

CK_OBJECT_HANDLE hUserPublicKey = CK_INVALID_HANDLE;

CK_OBJECT_HANDLE hUserPrivateKey = CK_INVALID_HANDLE;

CK_OBJECT_HANDLE hCert = CK_INVALID_HANDLE;

CK_CHAR CApublabel[] = "CApub";

CK_CHAR CAprilabel[] = "CApri";

CK_CHAR Userpublabel[] = "Userpub";

CK_CHAR Userprilabel[] = "Userpri";

CK_CHAR so_user[] = "0000";

CK_ULONG keySize = 1024;

CK_BBOOL bTrue = TRUE;

CK_BBOOL bFalse = FALSE;

rv = C_Initialize(NULL);

rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

if (rv != CKR_OK) {

    printf("Could not open session on token %d,error code =
           %x\n",slotId,rv);

    return 0;

}

rv = C_Login(hSession, CKU_USER, so_user, strlen((char*)so_user));

CK_ATTRIBUTE CApublicKeyTpl [] = {                                     (1)

    {CKA_TOKEN,          &bTrue,    sizeof(CK_BBOOL)},

    {CKA_PRIVATE,        &bFalse,    sizeof(CK_BBOOL)},

    {CKA_LABEL,          CApublabel, sizeof(CApublabel)-1},

    {CKA_VERIFY,         &bTrue,    sizeof(CK_BBOOL)},

    {CKA_WRAP,           &bTrue,    sizeof(CK_BBOOL)},

    {CKA_MODIFIABLE,      &bTrue,    sizeof(CK_BBOOL)},

    {CKA_DERIVE,          &bTrue,    sizeof(CK_BBOOL)},
```

```

        {CKA_PRIME_BITS,      &keySize, sizeof(CK_ULONG)}
    };

    CK_SIZE pubSize = sizeof(CApublicKeyTpl) / sizeof(CK_ATTRIBUTE);

    CK_ATTRIBUTE CAprivateKeyTpl[] = {                                     (2)

        {CKA_TOKEN,          &bTrue,   sizeof(CK_BBOOL)},
        {CKA_PRIVATE,        &bFalse,  sizeof(CK_BBOOL)},
        {CKA_LABEL,          CAprilabel, sizeof(CAprilabel)-1},
        {CKA_SENSITIVE,      &bTrue,   sizeof(CK_BBOOL)},
        {CKA_SIGN,           &bTrue,   sizeof(CK_BBOOL)},
        {CKA_IMPORT,         &bFalse,  sizeof(CK_BBOOL)},
        {CKA_UNWRAP,         &bTrue,   sizeof(CK_BBOOL)},
        {CKA_MODIFIABLE,     &bTrue,   sizeof(CK_BBOOL)},
        {CKA_EXTRACTABLE,    &bTrue,   sizeof(CK_BBOOL)},
        {CKA_DERIVE,         &bTrue,   sizeof(CK_BBOOL)},
        {CKA_EXPORTABLE,     &bTrue,   sizeof(CK_BBOOL)},
        {CKA_SIGN_LOCAL_CERT, &bFalse,  sizeof(CK_BBOOL)}

    };

    CK_SIZE priSize = sizeof(CAprivateKeyTpl)/sizeof(CK_ATTRIBUTE);

    CK_MECHANISM gen_mech = {CKM_RSA_PKCS_KEY_PAIR_GEN, NULL, 0};      (3)

    rv = C_GenerateKeyPair(hSession, &gen_mech, CApublicKeyTpl, pubSize,
        CAprivateKeyTpl, priSize, &hCAPublicKey, &hCAPrivateKey);      (4)

    if (rv != CKR_OK) {

        printf("Could not generate CA key pair on token %d,error code =
            %x\n",slotId,rv);

        return 0;

    }

    CK_ATTRIBUTE UserpublicKeyTpl [] = {                                  (5)

        {CKA_TOKEN,          &bTrue,   sizeof(CK_BBOOL)},
        {CKA_PRIVATE,        &bFalse,  sizeof(CK_BBOOL)},
        {CKA_LABEL,          Userpublabel, sizeof(Userpublabel)-1},
        {CKA_DERIVE,         &bTrue,   sizeof(CK_BBOOL)},
    };

```

```

        {CKA_PRIME_BITS,      &keySize, sizeof(CK_ULONG)}
    };

    pubSize = sizeof(UserpublicKeyTpl) / sizeof(CK_ATTRIBUTE);

    CK_ATTRIBUTE UserprivateKeyTpl[] = {                                     (6)

        {CKA_TOKEN,           &bTrue,   sizeof(CK_BBOOL)},
        {CKA_PRIVATE,         &bFalse,   sizeof(CK_BBOOL)},
        {CKA_LABEL,           Userprilabel, sizeof(Userprilabel)-1},
        {CKA_SENSITIVE,       &bTrue,   sizeof(CK_BBOOL)},
        {CKA_DECRYPT,          &bTrue,   sizeof(CK_BBOOL)},
        {CKA_SIGN,            &bTrue,   sizeof(CK_BBOOL)},
        {CKA_IMPORT,          &bFalse,   sizeof(CK_BBOOL)},
        {CKA_UNWRAP,          &bTrue,   sizeof(CK_BBOOL)},
        {CKA_MODIFIABLE,      &bTrue,   sizeof(CK_BBOOL)},
        {CKA_EXTRACTABLE,     &bTrue,   sizeof(CK_BBOOL)},
        {CKA_DERIVE,          &bTrue,   sizeof(CK_BBOOL)},
        {CKA_EXPORTABLE,      &bTrue,   sizeof(CK_BBOOL)}

    };

    priSize = sizeof(UserprivateKeyTpl)/sizeof(CK_ATTRIBUTE);

    rv = C_GenerateKeyPair(hSession, &gen_mech, UserpublicKeyTpl, pubSize,
        UserprivateKeyTpl, priSize, &hUserPublicKey, &hUserPrivateKey);

                                                                    (7)

    if (rv != CKR_OK) {

        printf("Could not generate CA key pair on token %d,error code =
            %x\n",slotId,rv);

        return 0;

    }

    CK_BYTE serial_num_str[50];                                          (8)

    strcpy((char*)serial_num_str,"1234");

    CK_BYTE subject_str[100] = {0};

    strcpy((char*)subject_str,"CN=ary,O=whybe,OU=marketing,L=tpe,ST=tw,
        C=tw");                                                         (9)

```

```

CK_BYTE issuer_str[100] = {0};

strcpy((char*)issuer_str,"CN=allan,O=dhc,OU=sw,L=tpe,ST=tw,C=tw");
(10)

CK_DATE start_date;
(11)

memcpy(&start_date,"20070605",8);

CK_DATE end_date;
(12)

memcpy(&end_date,"20090605",8);

CK_CHAR Certlabel[] = "UserCert";

CK_ATTRIBUTE CCertTpl[] = {
(13)
    {CKA_TOKEN,          &bTrue,      sizeof(CK_BBOOL)},
    {CKA_PRIVATE,        &bFalse,     sizeof(CK_BBOOL)},
    {CKA_SERIAL_NUMBER_INT, serial_num_str,
        strlen((char*)serial_num_str)},
    {CKA_ISSUER_STR,      issuer_str,  strlen((char*)issuer_str)},
    {CKA_SUBJECT_STR,    subject_str,  strlen((char*)subject_str)},
    {CKA_DERIVE,          &bTrue,      sizeof(CK_BBOOL)},
    {CKA_END_DATE,        &end_date,   sizeof(CK_DATE)},
    {CKA_DERIVE,          &bTrue,      sizeof(CK_BBOOL)}
};

priSize = sizeof(CCertTpl)/sizeof(CK_ATTRIBUTE);

CK_MECHANISM sign_mech = {CKM_SHA1_RSA_PKCS,NULL,0};
(14)

CK_MECHANISM derive_mech = {CKM_ENCODE_X_509,NULL,0};
(15)

rv = C_SignInit(hSession, &sign_mech, hCAPrivateKey);
(16)

rv =
C_DeriveKey(hSession,&derive_mech,hUserPublicKey,CCertTpl,priSize,&hC
ert);
(17)

if (rv != CKR_OK) {
    printf("gen cert failed on token %d,error code = %x\n",slotId,rv);
    return 0;
}

```

```

    rv = C_CloseSession(hSession);

    rv = C_Finalize(NULL);

    return 0;

}

```

### 10.3.2 產生PKCS#10 憑證申請檔

程式 EX10\_2 可以產生憑證申請檔。首先產生使用者的金鑰對，產生的演算法為 CKM\_RSA\_PKCS\_KEY\_PAIR\_GEN。我們將對使用者的公鑰產生憑證申請檔。接下來定義憑證申請檔的樣板 [EX10\_2(1)]，此樣板裏有一主旨屬性。接著定義簽章演算法為 CKM\_SHA1\_RSA\_PKCS[EX10\_2(2)]。以及衍生金鑰的演算法為 CKM\_ENCODE\_PKCS\_10[EX10\_2(3)]，在這裏是指產生 PKCS#10 憑證申請檔。再呼叫 C\_SignInit 初始化簽章運算，在此要給定使用者的私鑰[EX10\_2(4)]。並呼叫 C\_DeriveKey 產生憑證申請檔[EX10\_2(5)]。憑證申請檔的值會在 hCertReq 所指向物件裏面的 CKA\_VALUE 屬性裏。此值即可送到CA 進行簽章以產生憑證。

#### EX10\_2 產生PKCS#10憑證申請檔

```

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "cryptoki.h"

#include "ctvdef.h"

int main(int argc, char* argv[])
{

    CK_RV rv;

    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;

    CK_SLOT_ID slotId = 0;

    CK_OBJECT_HANDLE hUserPublicKey = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hUserPrivateKey = CK_INVALID_HANDLE;

    CK_OBJECT_HANDLE hCert = CK_INVALID_HANDLE;

```



```
CK_CHAR Userpublabel[] = "Userpub";

CK_CHAR Userprilabel[] = "Userpri";

CK_CHAR so_user[] = "0000";

CK_ULONG keySize = 1024;

CK_BBOOL bTrue = TRUE;

CK_BBOOL bFalse = FALSE;

rv = C_Initialize(NULL);

rv = C_OpenSession(slotId, CKF_RW_SESSION, NULL, NULL, &hSession);

if (rv != CKR_OK) {

    printf("Could not open session on token %d,error code =

           %x\n",slotId,rv);

    return 0;

}

rv = C_Login(hSession, CKU_USER, so_user, strlen((char*)so_user));

CK_ATTRIBUTE UserpublicKeyTpl [] = {

    {CKA_TOKEN,          &bTrue,    sizeof(CK_BBOOL)},

    {CKA_PRIVATE,        &bFalse,   sizeof(CK_BBOOL)},

    {CKA_LABEL,          Userpublabel,sizeof(Userpublabel)-1},

    {CKA_DERIVE,          &bTrue,    sizeof(CK_BBOOL)},

    {CKA_PRIME_BITS,      &keySize,  sizeof(CK_ULONG)}

};

CK_SIZE pubSize = sizeof(UserpublicKeyTpl) / sizeof(CK_ATTRIBUTE);

CK_ATTRIBUTE UserprivateKeyTpl[] = {

    {CKA_TOKEN,          &bTrue,    sizeof(CK_BBOOL)},

    {CKA_PRIVATE,        &bFalse,   sizeof(CK_BBOOL)},

    {CKA_LABEL,          Userprilabel,sizeof(Userprilabel)-1},

    {CKA_SENSITIVE,      &bTrue,    sizeof(CK_BBOOL)},

    {CKA_DECRYPT,         &bTrue,    sizeof(CK_BBOOL)},

    {CKA_SIGN,           &bTrue,    sizeof(CK_BBOOL)},

    {CKA_IMPORT,         &bFalse,   sizeof(CK_BBOOL)},
```

```

        {CKA_UNWRAP,          &bTrue,    sizeof(CK_BBOOL)},
        {CKA_MODIFIABLE,     &bTrue,    sizeof(CK_BBOOL)},
        {CKA_EXTRACTABLE,    &bTrue,    sizeof(CK_BBOOL)},
        {CKA_DERIVE,         &bTrue,    sizeof(CK_BBOOL)},
        {CKA_EXPORTABLE,     &bTrue,    sizeof(CK_BBOOL)}

    };

    CK_SIZE priSize = sizeof(UserprivateKeyTpl)/sizeof(CK_ATTRIBUTE);

    CK_MECHANISM gen_mech = {CKM_RSA_PKCS_KEY_PAIR_GEN, NULL, 0};

    rv = C_GenerateKeyPair(hSession, &gen_mech, UserpublicKeyTpl, pubSize,
        UserprivateKeyTpl, priSize, &hUserPublicKey, &hUserPrivateKey);

    if (rv != CKR_OK) {
        printf("Could not generate CA key pair on token %d,error code =
            %x\n",slotId,rv);

        return 0;
    }

    CK_BYTE subject_str[100] = {0};

    strcpy((char*)subject_str,"CN=ary,O=whybe,OU=marketing,L=tpe,ST=tw,
        C=tw");

    CK_ATTRIBUTE CertTpl [] = {
        {CKA_TOKEN,          &bTrue,    sizeof(CK_BBOOL)},
        {CKA_PRIVATE,        &bFalse,   sizeof(CK_BBOOL)},
        {CKA_SUBJECT_STR,    subject_str, strlen((char*)subject_str)}
    };

    priSize = sizeof(CertTpl)/sizeof(CK_ATTRIBUTE);

    CK_MECHANISM sign_mech = {CKM_SHA1_RSA_PKCS,NULL,0};
    CK_MECHANISM derive_mech = { CKM_ENCODE_PKCS_10,NULL,0};

    rv = C_SignInit(hSession, &sign_mech, hUserPrivateKey);

    rv =
    C_DeriveKey(hSession,&derive_mech,hUserPublicKey,CertTpl,priSize,&hCe
rtReq);

```

```
if (rv != CKR_OK) {  
    printf("gen cert req failed on token %d,error code = %x\n",slotId,rv);  
    return 0;  
}  
  
rv = C_CloseSession(hSession);  
  
rv = C_Finalize(NULL);  
  
return 0;  
}
```

## 10.4 習題

1. 參考文件及程式EX10\_1，請由憑證申請檔來產生X509 憑證。
2. 參考文件，使用CKM\_DECODE\_X\_509 機制從憑證中取出公鑰。
3. 在程式EX10\_1 中建立了CA 的公私鑰對，請為CA 建立其自簽憑證。



ProtectServer Gold

## 附錄1:PKCS#11 簡介

PKCS#11標準的名稱為 Cryptographic Token Interface Standard，是由RSA Laboratories 所訂定的一系列PKCS(Public-Key Cryptography Standards) 標準之一 [3]。此標準主要是定義可提供密碼功能或持有密碼資訊的裝置(device)之 API(Application Programming Interface)，稱作“Cryptoki”，是“Cryptographic Token Interface”的簡寫，讀作“Crypto-key”。其採用基於物件(Object-Based)的方法，希望可以解決不同裝置間應用程式開發的問題，以及達成資源共享(多個應用程式 可以存取多個裝置)的目標。在此標準中，所謂的裝置是指具有移動性(Portable)之密碼裝置，例如: smart cards, PCMCIA cards 及 smart diskettes。“Cryptoki”提供應用程式一邏輯的觀點，稱各種裝置為“Cryptographic token”。

此標準採用 ANSI C 程式語言來定義各種資料型態及函式。在 C 語言中，通常資料型態及函式會定義在標頭檔(Header File)中。因此Cryptoki 函式庫的提供者須提供一標頭檔，包含各種資料型態及函式的宣告。一般性的 Cryptoki 標頭檔可以在RSA 公司的PKCS 網頁上下載。

Cryptoki 將應用程式及密碼裝置做一隔離。對於不同型式的裝置或在不同的平台執行，應用程式並不用更改介面。目前最新的版本為 2.2，支援許多的密碼機制，往後仍會增加許多機制，但是增加機制並不會改變其一般介面。

### 1. Cryptoki 一般模型

如圖 1 所示，其為一般的 Cryptoki 模型。最上層為應用程式，最底層為 Token，可以共同有多個應用程式及多個裝置，使用者可以連結到一或多個應用程式。Cryptoki 提供系統和密碼裝置間的介面。Token 是透過 Slot 和 Cryptoki 做連結，所謂的Slot 是對應到一實體讀卡裝置或其它的裝置介面。



ProtectServer External

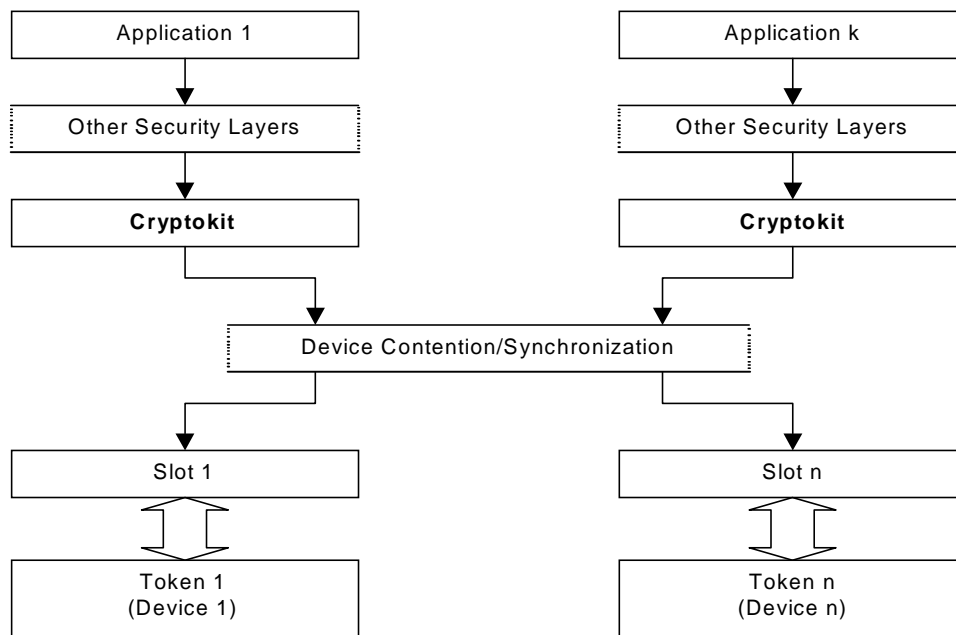


圖 1:General Cryptoki Model (取自 RSA PKCS#11 標準)

## 2. Token 之邏輯觀點

對 Cryptoki 來說，Token 其邏輯觀點為存有物件的裝置，而且可以執行密碼功能。Cryptoki 定義三種種類的物件，分別為 data、certificate 及 key。Data 物件可以用來存使用者所定義的資料。certificate 物件被使用來存憑證，key 物件使用來存密碼金鑰，金鑰可以是public key、private key 及secret key。其階層架構如圖2 所示。

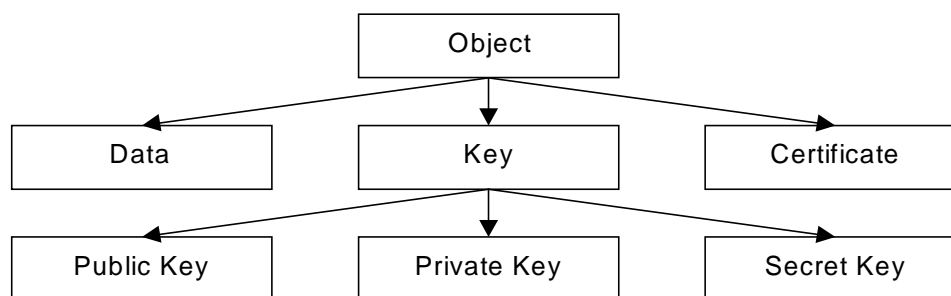


圖2:Object Hierarchy(取自 RSA PKCS#11 標準)

### 3. Object

物件種類如以生命週期及可視性來區分的話可以分成 token object 及 session object。Token object 在 session 消失後仍然存在，而 session object 在建立此物件的 session 關閉後也會跟著消失。物件種類若是以存取權限來區分的話，可以分成 public object 及 private object。在使用者未經過鑑別之前，只能讀取 public object，而使用者經過鑑別之後便可讀取 private object。

### 4. User

PKCS#11 將使用者角色(role)分成兩類，一為 normal user，另一為 security officer(簡稱 SO)。Normal user 在經過鑑別後可讀取 private object。而 SO 可以初始化 token 及設定 normal user 的 PIN。無論如何，SO 無法讀取 private object。

### 5. Session

為了可以存取 token 上的物件及函式，Cryptoki 需要在應用程式和 token 之間開啟一或多個 session。也就是說 session 在應用程式和 token 之間提供一邏輯的連線。Session 的開啟有兩種狀態，可以是 read/write(R/W) session 或是 read-only(R/O) session，指的是存取 token 上的物件。

只要一開啟 session，應用程式就可以存取 token 上的公開物件，若要存取私密物件(private object)，則使用者須事先登入並驗證通過(如驗證 PIN 碼)。Cryptoki 支援在多個 token 上有多個 session。應用程式可以在一或多個 token 上有一或多個 session。一般來說，一個 token 可以在一或多個應用程式上有多個 session。

Session 開啟的狀態決定是否可以存取物件及函式。如圖 3 及圖 4 所示。在 R/O 狀態中，應用程式只可以讀取 public token object 及讀寫 public session object。在經過使用者認證後可讀取 private token object 及讀寫 private session object。在 R/O 狀態中是無法登入成 SO 的。在 R/W 狀態中，一般只可以讀取寫

入public token object 及public session object。而使用者經鑑別通過後即可讀取或寫入private token object 及private session object。而當使用者登入R/W SO 狀態 時，只可讀取寫入公開物件，而只有在登入成SO 狀態時才可設定一般使用者的PIN 碼。存取權限如表1。

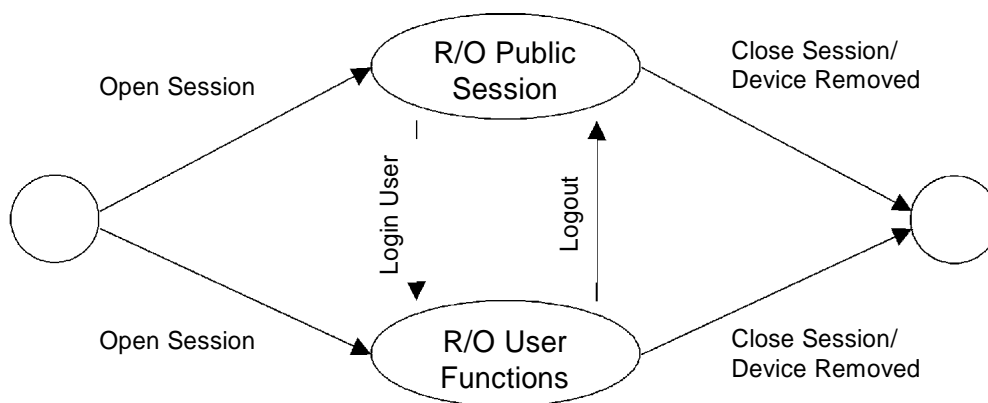


圖3:Read-Only Session States(取自 RSA PKCS#11 標準)

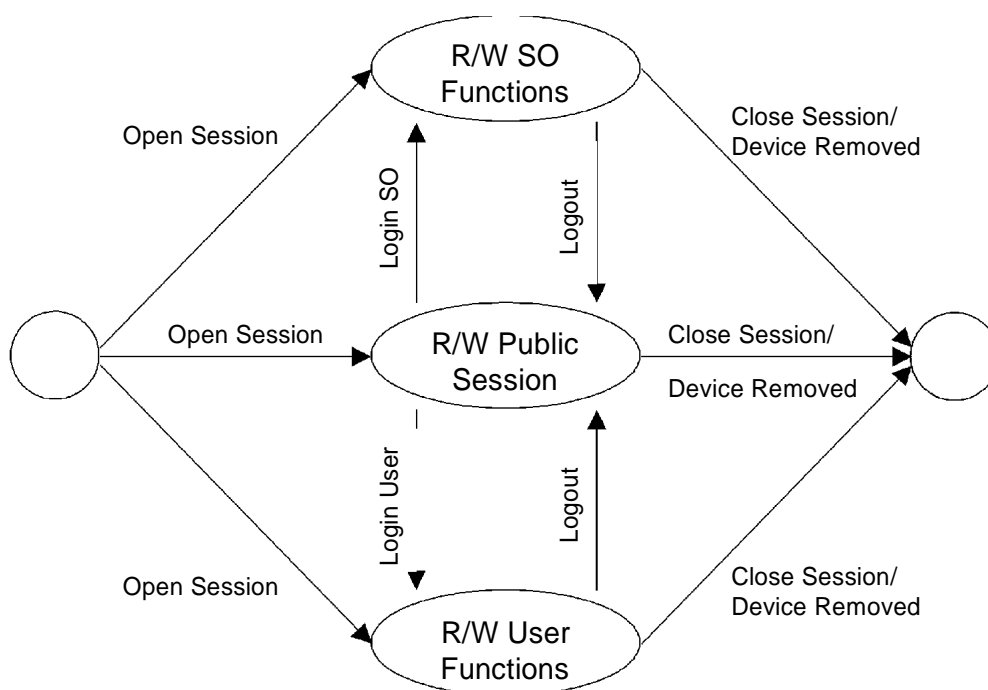


圖4:Read/Write Session States(取自 RSA PKCS#11 標準)

Type of object	Type of session				
	R/O Public	R/W Public	R/O User	R/W User	R/W SO
Public session object	R/W	R/W	R/W	R/W	R/W
Private session object			R/W	R/W	
Public token object	R/O	R/W	R/O	R/W	R/W
Private token object			R/O	R/W	

表1:物件存取權限

## 6. Cryptoki 函式

Cryptoki 所定義的函式約可分成下述幾類：

(1) 一般性函式：

用於一般性目的的函式，如C\_Initialize 用於初始化Cryptoki 函式庫。

(2) slot 及token 管理函式：

用於管理slot 及token 函式，如C\_InitToken 用來初始化token。

(3) session 管理函式：

用於管理session，如C\_OpenSession 用於開啟程式及token 之間的session。

(4) Encryption/Decryption 函式

用於加解密的函式。

(5) Message digest 函式：

用來做hash 的函式

(6) Signing/MACing 函式：

用來做簽章及訊息鑑別碼的函式

(7) 驗證Signing/MACing 函式：

用來做驗證簽章及訊息鑑別碼的函式

(8) 金鑰管理函式：

用來做金鑰的產生以及輸入輸出的函式。

(9) 亂數產生函式：

用來產生亂數的函式。



## 7. 物件的儲存

物件其實是由一組屬性所組成並儲存於token 上。雖然物件有分種類，但是 各種物件卻有共通的屬性如下：

CKA\_TOKEN:用來決定物件是 token object 或是session object。

CKA\_PRIVATE:用來決定物件是 private object 或是public object。

CKA\_MODIFIABLE:用來決定物件在建立後是否可修改。

CKA\_LABEL:用來描述物件。

**7.1 資料物件(data object)** 資料物件可用來儲存使用者所定義的資料。其具有之屬性如下: CKA\_APPLICATION:可用來描述應用程式。 CKA\_OBJECT\_ID:物件的識別碼。

CKA\_VALUE:物件實際的值。

## 7.2 金鑰物件(key object)

無論是secert key、private key 或是public key 其皆具有下列共同之屬性：

CKA\_KEY\_TYPE:金鑰的型態。

CKA\_ID:金鑰的識別碼。

CKA\_START\_DATE:金鑰的啟始日期。

CKA\_END\_DATE:金鑰的截止日期。

CKA\_DERIVE:決定是否可以用來衍生其它的金鑰。

## 7.3 公開金鑰物件(public key object) 公開

金鑰所具有的屬性如下: CKA\_ENCRYPT:決定是否可用來加密。 CKA\_SUBJECT:金鑰的主旨或名稱。 CKA\_VERIFY:決定是否可用來驗證簽章。

CKA\_VERIFY\_RECOVER:決定是否可用來驗證資料在簽章裏面的簽章種類。

CKA\_WRAP:決定是否可以用來加密其它的金鑰，用於輸出金鑰。

CKA\_TRUSTED:決定是否可以用來加密其它的金鑰，此屬性只能由 SO 來設定。

CKA\_WRAP\_TEMPLATE:只有符合此屬性所定義的金鑰屬性的金鑰可以被加密輸出。

#### 7.4 私密金鑰物件(private key object)

私密金鑰所具有的屬性如下：

CKA\_SUBJECT:金鑰的主旨或名稱。

CKA\_SENSITIVE:決定金鑰是否為敏感性，如為敏感性則無法取出其值。

CKA\_DECRYPT:決定是否可用來解密。

CKA\_SIGN:決定是否可用來簽章。 CKA\_SIGN\_RECOVER:決定是否可用來簽章資料在簽章裏面的簽章種類。 CKA\_UNWRAP:決定是否可用來解密其他的金鑰，用於輸入金鑰。 CKA\_EXTRACTABLE:決定是否可以被其它金鑰加密。

CKA\_ALWAYS\_SENSITIVE:決定是否永遠是敏感性的。

CKA\_NEVER\_EXTRACTABLE:決定是否永遠無法被其它的金鑰加密輸出。

CKA\_WRAP\_WITH\_TRUSTED:決定是否可以被 CKA\_TRUSTED 為 TRUE 的金鑰加密輸出。

CKA\_UNWRAP\_TEMPLATE:只有符合此屬性所定義的金鑰屬性的金鑰可以被解密輸入。

CKA\_ALWAYS\_AUTHENTICATE:決定是否在每次使用此物件的時候，使用者都要提供PIN。

#### 7.5 秘密金鑰物件(secret key object)

秘密金鑰所具有的屬性如下：CKA\_SENSITIVE:決定金鑰是否為敏感性，如為敏感性則無法取出其值。 CKA\_ENCRYPT:決定是否可用來加密。

CKA\_DECRYPT:決定是否可用來解密。

CKA\_SIGN:決定是否可用來簽章。 CKA\_VERIFY:決定是否可用來驗證簽章。 CKA\_WRAP:決定是否可以用來加密其它的金鑰，用於輸出金鑰。 CKA\_UNWRAP: 決定是否可用來解密其他的金鑰，用於輸入金鑰。 CKA\_EXTRACTABLE:決定是否可以被其它金鑰加密。

CKA\_ALWAYS\_SENSITIVE:決定是否永遠是敏感性的。

CKA\_NEVER\_EXTRACTABLE:決定是否永遠無法被其它的金鑰加密輸出。

CKA\_CHECK\_VALUE:金鑰的檢查碼。 CKA\_WRAP\_WITH\_TRUSTED:決定是否可以被 CKA\_TRUSTED 為 TRUE 的金鑰加密輸出。

CKA\_TRUSTED:決定是否可以用來加密其它的金鑰，此屬性只能由 SO 設

定。

CKA\_WRAP\_TEMPLATE: 只有符合此屬性所定義的金鑰屬性的金鑰可以被加密輸出。

CKA\_UNWRAP: 決定是否可用來解密其他的金鑰，用於輸入金鑰。

CKA\_UNWRAP\_TEMPLATE: 只有符合此屬性所定義的金鑰屬性的金鑰可以被解密輸入。



ProtectServer External