# 國立交通大學

## 資訊科學系

## 碩 士 論 文

# 軟 體 保 護 之 研 究

Software Protection

研 究 生：莊吳祐

指導教授：曾文貴　教授

中 華 民 國　九 十 四　年 六 月

# 軟體保護之研究
# Software Protection

研 究 生：莊吳祐　　Student：Wu-You Zhuang

指導教授：曾文貴　　Advisor：Wen-Guey Tzeng

國 立 交 通 大 學
資 訊 科 學系
碩 士 論 文

A Thesis
Submitted to Institute of Computer and Information Science
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer and Information Science

June 2005

Hsinchu, Taiwan, Republic of China

# 誌　　謝

在此感謝我的指導老師曾文貴教授，在我碩士班的學習過程中，不只在學業上帶領我走進密碼學的領域，更在生活和言行舉止上孜孜不倦的教導我，使我受益良多。另外，我要感謝口試委員，交大資工蔡錫均教授、交大資管劉敦仁教授與台科大資工吳宗成教授，在論文上給我許多良好的建議與指導，讓我的論文更加完善。除此之外，我要感謝實驗室同學，孝盈、政儒、與建宇的幫忙，實驗室學長，成康的指導，以及實驗室學弟們在精神方面的鼓勵。

最後，我要感謝我的家人，不論在精神或物質上都給我極大的支持，讓我在無後顧之憂的情況下可以順利完成學業。在此，謹以此文獻給所有我想要感謝的人。

# 軟體保護之研究

學生：莊吳祐　　指導教授：曾文貴　博士

國立交通大學資訊科學系

## 摘要

目前市場上各家軟體公司的產品，除了結合網路使用 Client-Server 架構的業者，靠著持續提供伺服器服務收取費用外(例如線上遊戲)，大多面臨軟體盜版的問題。

商業軟體，依程式碼功能可分為 1.軟體保護程式 2.軟體原有功能程式。目前軟體保護程式的保護方法可說千變萬化，但其變化僅限於軟體保護程式本身。軟體保護程式與軟體原有功能程式分界十分明顯，破解者可透過執行時軟體給與的線索把軟體保護程式所在位置找出並破解之。

我們從軟體本質與軟體破解的行為模式分析，提出軟體破解複雜度上限的假設，根據這個假設在 80x86 CPU/Windows 作業系統上設計兩層式的保護架構，讓軟體保護程式與軟體原有功能程式做更緊密的結合以增加破解難度。除非破解者具有重製該軟體的能力否則無法達到破解的目地。

# Software Protection

Student: Wu-You Zhuang Advisor: Dr. Wen-Guey Tzeng

Institute of Computer and Information Science

National Chiao Tung University

## Abstract

Business software can be defined as two categories. One is software protection program and the other is software directive program. So far the software protection programs have various ways to be protected, but the difference only depends on software protection programs themselves. It is very clear and obvious to determine software protection program and software directive program. Hacker can hack the software protection program by the clues when running the program.

We can provide a more difficult and complicate way for hacking by the analysis of the software and the hacking behavior. In this case we can design a double protection structures on 80x86 CPU/Windows. This will make software protection software and software directive program to be more associated and harder to be hacked. Only the hacker who can reprogram the program can hack, otherwise it's not possible that the program can be hacked after this design.

# 目次

# 第一章 引言

## 1.1 研究動機

目前市場上各家軟體公司的產品，除了結合網路使用 Client-Server 架構的業者，靠著持續提供伺服器服務收取費用外(例如線上遊戲)，大多面臨軟體盜版的問題。

儘管軟體保護方式五花八門，從密碼，軟體狗，加殼到韌體上動手腳，無所不用其極，卻終究逃不過被破解的命運，其中最主要的原因在於軟體本身始終儲存於使用者端，使用者可以經由解析機器碼了解程式運作的方式進而破解保護。

隨著資訊的發達，坊間開始有不少研究軟體保護的相關書籍，這些書籍以實例介紹軟體保護與破解的技巧，也提供不少良好的建議給程式設計師增加程式破解的難度，卻缺少一個整體性的概念，軟體保護的極限在那裡?軟體保護最多可以做到多麼安全?這個界限是否會因為被保護的軟體不同而改變?

## 1.2 研究目標與成果

商業軟體，依程式碼功能可分為 1.軟體保護程式 2.軟體原有功能程式。目前軟體保護程式的保護方法可說千變萬化，但其變化僅限於軟體保護程式本身。軟體保護程式與軟體原有功能程式分界十分明顯，破解者可透過執行時軟體給與的線索以及靜態/動態分析技術把軟體保護程式所在位置找出，再透過分析組合語言來了解軟體保護程式的保護方法進而破解之。

我們從軟體本質與軟體破解的行為模式分析起，提出軟體破解複雜度上限的假設，我們假設軟體破解複雜度的上限為

**min(重製一份該軟體,破解軟體保護)**

我們假設重製一份該軟體為很難的問題，一旦破解者具有重製該軟體的能力，則該軟體對破解者毫無價值可言。而目前市面上的軟體，其破解軟體保護難度遠小於重製一份該軟體，因此我們在 80x86 CPU/Windows 作業系統上設計兩層式的保護架構，讓軟體保護程式與軟體原有功能程式做更緊密的結合以增加破解難度，除非破解者具有重製該軟體的能力否則無法達到破解的目地。

## 1.3 各章節介紹

第二章介紹閱讀本論文相關軟體破解背景知識，第三章介紹密碼學相關理論

與技術，第四章介紹本論文設計方法的基本理論與模型，第五章介紹實作方法，第六章對開發的工具程式做介紹，第七章分析應用此設計方法所額外付出的時間與空間，未來工作與參考文獻。

# 第二章 軟體破解相關背景

在這個章節我們會介紹 80x86 處理器及 Windows 作業系統的特性以及軟體破解相關知識，作為閱讀後面章節的基礎。

## 2.1 80x86 的 CPU 及其暫存器群

CPU 是電腦執行指令的單元，為了方便執行指令起見，CPU 內部也設有一些記憶體，用來暫存資料，叫做暫存器(Register)。

在 8088/86/286/386/486 的 V86 模式下，CPU 內部共含有 14 個暫存器，可分為 4 組:

一般用途暫存器:AX，BX，CX，DX

堆疊，基底，索引及指令指位器:SP，BP，SI，DI，IP

節區暫存器:CS，DS，ES，SS

旗標:FLAG

因後文需要必需特別介紹幾個暫存器:

**1.程式指位器 IP(Instruction Pointer)**

這是一個主掌著 CPU 動向的暫存器，當 CPU 執行完一行指令後，IP 便指向下一指令的位置址，此時 CPU 便會往 IP 所指的位置去讀取記憶體內容即所要執行的指令。所以，諸如 JMP，CALL 等控制 CPU 流程的指令，其實便是以改變 IP 的值來控制 CPU 的，圖 1 為示意圖。

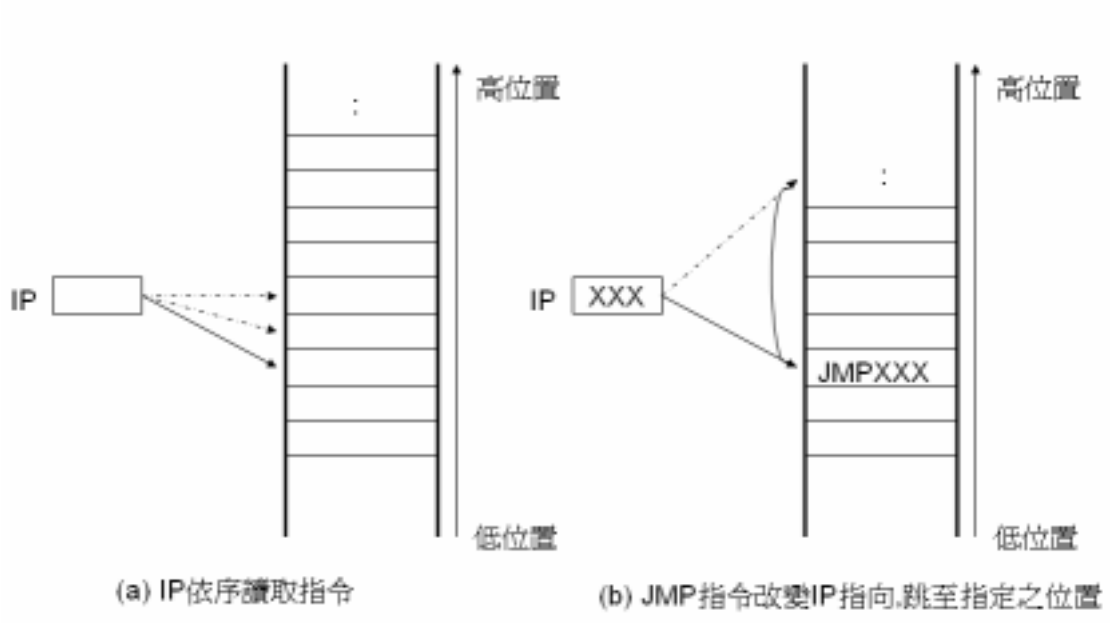

圖 1 程式指位器示意圖

## 2.堆疊指位器 SP(Stack Pointer)

SP 與 IP 相類似，不過 IP 是指向程式區，而 SP 則是指向堆疊區。堆疊(Stack)是一種重要的資料結構，所謂的資料結構指的是資料組織與管理的方法。以堆疊而言，就是規定資料元素必須依著後進先出(Last In First Out)之次序來存取。比如我們將甲，乙，丙，丁依序推入(PUSH)堆疊，則當我們想由堆疊取(POP)資料時，其次序是丁，丙，乙，甲如下圖 2 所示。而 SP 便是用來記錄堆疊最頂端位置的指位器。

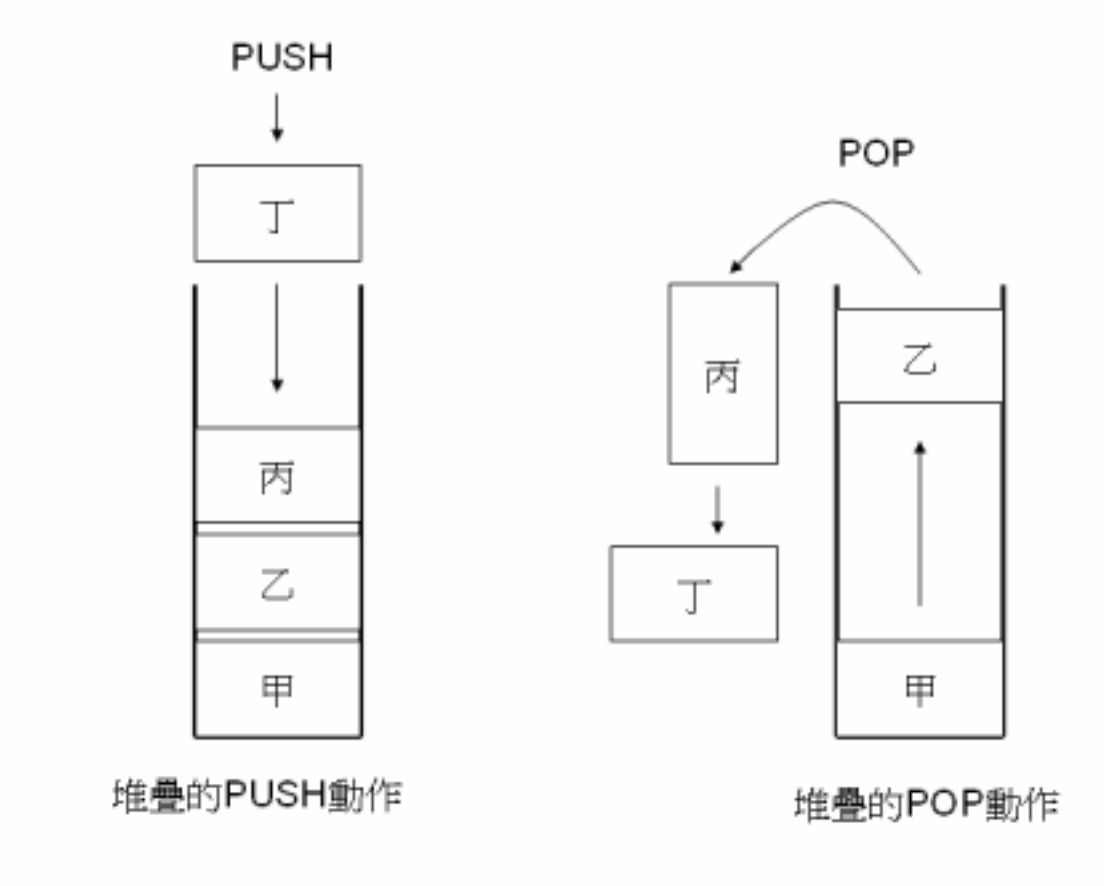


圖 2 堆疊示意圖

## 3.基底指位器 BP(Base Pointer)

BP 與 SP 一樣是堆疊的指位器，不過 SP 永遠指向堆疊的頂端，而 BP 則可以指向堆疊內的任何位置，如此可以令我們很方便的存取堆疊內的任意資料，而不在局限於 SP 所指的頂端資料了。

本節資料參考[1]。

## 2.2 局部變數與函數傳入值

局部變數主要是為了定義一些僅在單個函數裡有用的變數而提出，使用局部變數能帶來一些額外的好處，它使程式的模組化封裝變得可能，試想一下，如果要用到的變數必須定義在程式的資料段裡面，假設在一個子程式中要用到一些變數，當把這個子程式移植到別的程式時，除了把程式碼移過去以外，還必須把變數定義移過去。而即使把變數定義移過去了，由於這些變數定義在大家都可以用的資料段中，就無法對別的程式碼保持透明，別的程式碼有可能有意無意地修改它們。局部變數這個概念出現以後，兩個以上的函式都要用到的資料才被定義為總體變數統一放在資料段中，僅在單一函式內部使用的變數則放在堆疊中，這樣函式可以編成黑盒子的模樣，使程式的模組結構更加分明。

局部變數的作用範圍是單一函式，在進入函式的時後，透過修改堆疊指標 SP 來預留出需要的空間，在用 ret 指令返回主程式之前，同樣透過恢復 SP 來丟棄這些空間，這些變數就隨之無效了。它的缺點就是因為空間是臨時分配的，所以無法定義含有初始化值的變數，因此對局部變數的初始化一般在函式中以指令完成。

函式傳入值與局部變數的共通點在於皆為某一函式需要，因此兩者皆以系統堆疊實作之。不同點在於函式傳入值在被呼叫函式執行前由呼叫函式預留空間與設定其值接著才執行被呼叫函式，而局部變數則在被呼叫函式開頭預留空間執行過程中設定其值。函式呼叫的詳細作法本文略過不提只介紹函式呼叫時局部變數與傳入值的堆疊中的變化如圖 3 所示。

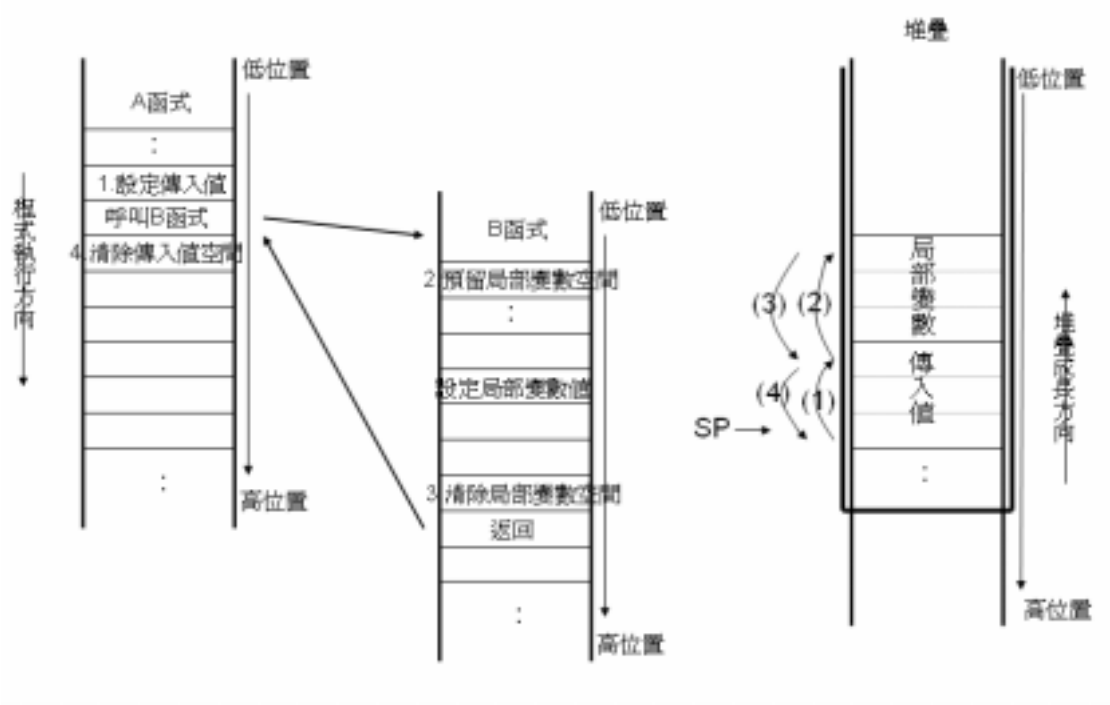


圖 3 函式呼叫與堆疊變化示意圖

## BP 與間接定址法

由記憶位址存取資料的方法，叫做記憶位址定址法(Memory Addressing mode)，記憶位址定址法又分成 1.直接定址法 2.間接定址法兩大類。

間接定址法就是不直接指明資料的位置，而把資料的位址存於暫存器，然後指示 CPU 往暫存器上取得資料所在位址以存取資料的方法。間接定址法又細分為數種，其中一種叫做基底定址法即為利用 BP 定址的方法。

相對於 SP 永遠指向堆疊頂端 BP 則隨需要而變化，在某個函式執行的過程中 BP 會保持不變(呼叫子函式不算)指向某個位置當做基底定址法的基底，此方法也是系統存取局部變數與傳入值的方法細節如圖 4 所示。

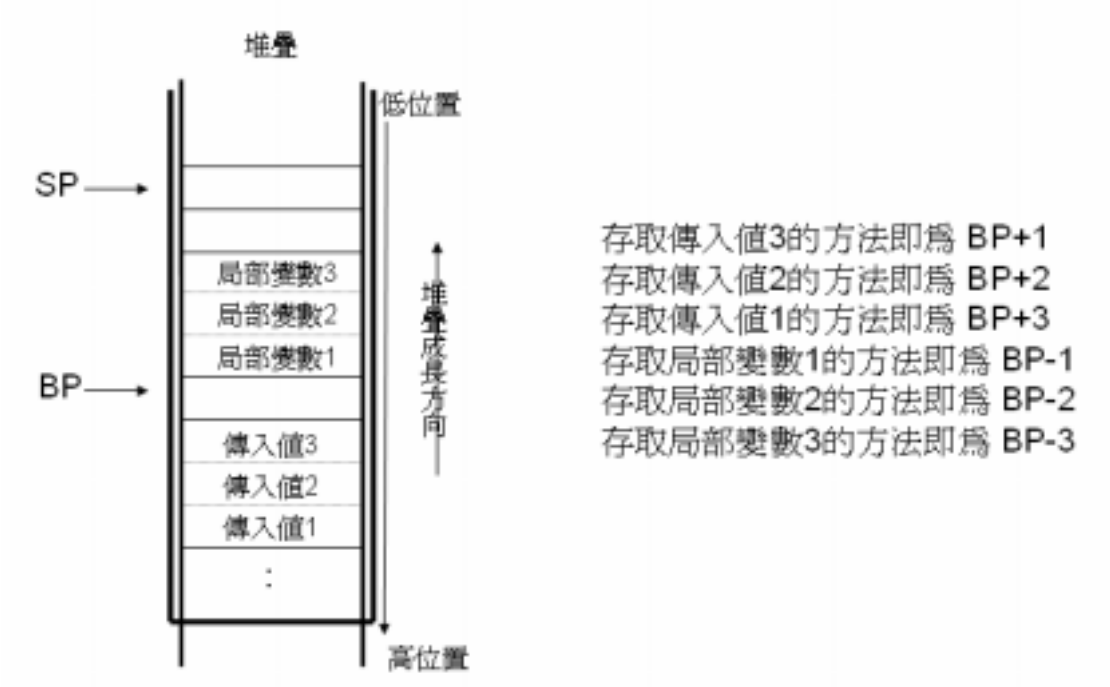


圖 4 基底定址法示意圖

上面介紹是用來存取局部變數與傳入值的間接定址法，實際上我們可以任意一個暫存器為基底來存取記憶體。本節資料參考[2]。

## 2.3 80x86 處理器的工作模式

8088/8086 的 CPU 暫存器如 AX,BX,CX,DX,SI,DI,SP 和 BP 為 16 位元暫存器，在 80386 中，這些暫存器被擴展到 32 位元，即 EAX,EBX,ECX,EDX,ESI,EDI,ESP 和 EBP。

80386 處理器有三種工作模式:真實模式，保護模式和虛擬 86 模式。

真實模式和虛擬 86 模式是為了和 8086 處理器相容而設置的。在真實模式下，80386 處理器只能使用 32 位元暫存器的前 16 位元,後面的 16 位元都浪費了，因此相當於一個快速的 8086 處理器。

保護模式是 80386 處理器的主要工作模式。在此模式下，80386 可以定址 2^32=4GB 的位置空間，同時，保護模式提供了 80386 先進的多工，記憶體分頁管理和優先順序保護等機制。

為了在保護模式下繼續提供和 8086 處理器的相容，80386 又設計了一種虛擬 86 模式，以便可以在保護模式的多工條件下，有的任務運行 32 位元程式，有的任務運行 MS-DOS 程式。在虛擬 86 模式下，同樣支援多工，記憶體分頁管理和優先順序保護等機制，但記憶體的定址方式和 8086 相同，只可以定址 1MB 的空

間。

當 80386 在保護模式下運作時，定址空間高達 2^32=4GB。雖然與 8086 可定址的 1MB 位置的差距可謂很大，但實際的微處理系統很難能安裝如此大的實體記憶體。所以，虛擬記憶體(Virtual Memory)是一種必需的技術。

虛擬記憶體(Virtual Memory)不是真正的記憶體，它通過映射(Map)的方法，使可用的虛擬位置答到 4GB，每一個應用程式可以被分配 2GB 的虛擬位置，剩下的 2GB 留給作業系統自己使用。簡單地說，虛擬記憶體的實現方法和過程是:

1. 當一個應用程式被啟動時，作業系統就創建一個新行程，並給每一個行程分配 2GB 的虛擬位置(不是記憶體，只是位置)
2. 虛擬記憶體管理器(Virtual Memory Manager)經應用程式的程式碼映射到那個應用程式的虛擬位置中的某個位置，並把當前所需要的程式碼讀取到物理位置中(注意 :虛擬位置和應用程式碼在實體記憶體中的位置沒有關連)
3. 如果使用動態連結程式庫(Dynamic-Link Library,即 DLL) ，也被映射到該行程的虛擬位置空間，在需要的時後才被讀入實體記憶體
4. 其它應用程式的空間是從實體記憶體中分配的，並被映射到虛擬記憶體中
5. 應用程式讀取它的虛擬位置空間中的資料，虛擬記憶體管理器把每次的存取的虛擬位置映射到物理位置

如果看不明白上面的步驟也不要緊，但要明白以下幾點:

1. 應用程式不會直接讀取物理位置
2. 虛擬記憶體管理器通過虛擬位置的存取要求，控制所有的物理位置存取
3. 每一個應用程式都有獨立的 4GB 定址空間，它們不能存取其他應用程式的資料，也不用擔心實體記體不足，作業系統會為應用程式處理

本節資料參考[3]。

## 2.4 作業系統如何載入並執行軟體

在 Win32 平臺上(包括 Windows 95/98/ME/NT/2000/XP/CE) ，可執行檔是 PE(Portable Executable)格式。本節介紹 PE 檔的基本知識與作業系統如何載入並執行軟體。

PE 檔使用一個線性的位置空間，所有程式碼和資料都被合併在一起，組成一個很大的結構。檔案的內容被分割為不同的區塊(Section)，塊中包含程式碼

或資料，各個塊按頁邊界來對齊，塊沒有大小限制，是一個連續結構。每一個塊都有它自己在記憶體中的一套屬性，比如:這個塊是否包含程式碼，是否唯讀或可讀/寫等。

每一個區塊都有不同的名字, 這個名字用來表示區塊的功能。例如, 一個叫.rdata的區塊表明它是唯讀區塊。常見的塊有.text，.rdata，.data，.idata，.rsrc等。這些區塊的名字只是方便人們使用，而對作業系統來說無關緊要，因此可將上面塊名改成其它毫無意義的字元，不會影想 PE 檔的執行。

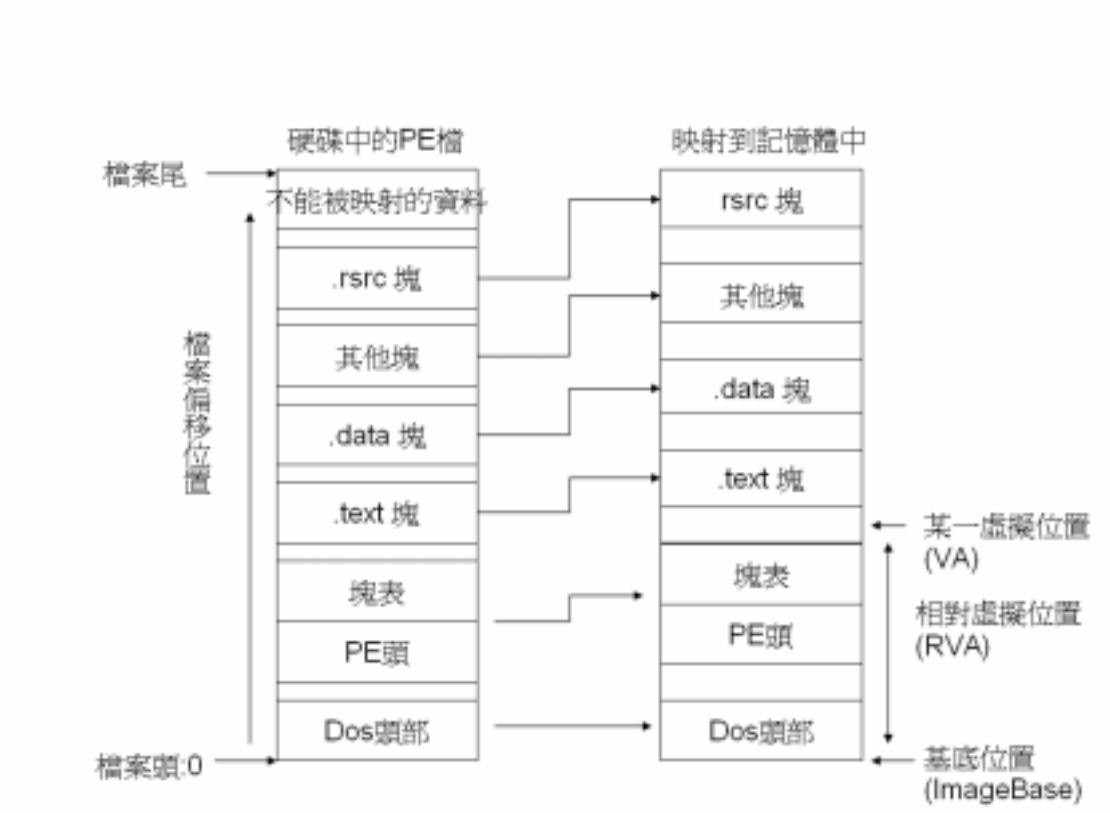


圖 5　PE 檔結構與記憶體映射圖

塊(Section)有兩種對齊值:一種在磁碟檔中的，另一種是記憶體中的。PE檔被映射到記憶體時, 塊(Section)總是以一個頁(Page)邊界為開始。也就是說，每一個塊(Section)中的第一個位元組對應於某個記憶體頁。在 x86 系統中，每個記憶體頁的大小為 4KB，即 0x1000 個位元組。所以在 x86 系統中，每個塊(Section)按照 0x1000 之倍數的記憶體偏移位置開始。而磁碟中的檔案對齊值一般取 0x200，每個塊(Section)按 0x200 之倍數的檔偏移位置開始。所以塊(Section)在檔案與記憶體中的偏移位置不一樣，其中差值可透過簡單的公式算出，最重要的是明白將程式載入記憶體的動作是以塊(Section)為單位原封不動

的搬進記憶體。當軟體載入記憶體後程式指位器(Instruction Pointer)從 PE 檔頭指定的位置開始執行依照軟體所描述的方法使用電腦資源。本節資料參考[4]。

## 2.5 軟體的本質

電腦由中央處理器，記憶單元，輔助記憶單元，輸入單元，輸出單元所組成。軟體簡而言之就是操作電腦各部份的方法。在 Windows 平台上，軟體以作業系統看的懂的機器碼形式儲存。透過簡單的轉換工具，我們可以把機器碼轉換為組合語言，因此後文將機器碼與組合語言視為一體，統一以組合語言稱之。

雖然程式語言依照人腦辨識的難易度可分為高階語言與低階語言，但把低階語言中的組合語言視為人腦可理解是合理的。因此我們把目前 Windows 平台上最後儲存在用戶電腦裡的軟體產品視為沒有加密的明文是可接受的，只是大多數人不會去研究這份明文的內容，只在乎這份明文所能達成的結果。

目前市面上有不少軟體產品將軟體本身做壓縮或加密處理，直到執行時才解壓縮或解密還原為原來的程式碼。雖然這些產品在未執行狀態大部份的程式碼是不可辨識的密文，但當程式執行時經過自我還原的步驟，最後存於記憶體裡的可執行程式碼可透過簡單的動態分析(見第四節)技術取出。簡而言之，無論程式做過多少處理，在執行的那刻必須還原為電腦可識別的機器碼，同時也是人可理解的組合語言。

## 2.6 靜態分析與動態分析

### 靜態分析技術

用高階語言編寫的程式有兩種形式，一種被編譯成機器碼在 CPU 上執行，如 Visual C++，PASCAL 等.由於機器碼與組合語言幾乎是一一對應的，因此可將機器碼轉化成組合語言，這個過程稱為反組譯(Disassembler).例如，在 x86 系統中，機器碼"0xEB"對應於組合語言語句是"jump short xx".另一種高階語言是一邊解譯一邊執行的，稱為直譯式語言，如 Visual Basic 3.0/4.0 ，Visual FoxPro 等，這類語言的編譯後程式可以被還原成高階語言的原始結構，這個過程稱為反編譯(Decompiler) 。所謂靜態分析，即從反組譯出來的組合語言上分析程式流程，瞭解程式每部分完成的功能。

```
#include "stdafx.h"
```

```
int main(int argc, char* argv[])
{
  printf("Hello World!\n");
  return 0;
}
```

圖 6　範例小程式

圖 6 的程式經過編譯後的機器碼即為圖 7 中的機器碼欄位
圖 7 中的機器碼經過反組譯後的組合語言即為圖 7 中的組合語言欄位
機器碼只能反組譯為組合語言，無法還原為原來的 C 語言程式碼

```
1:      // test.cpp : Defines the entry point for the console application.
2:      //
3:      #include "stdafx.h"
4:      int main(int argc, char* argv[])
5:      {
記憶體位置   機器碼                  組合語言
00401010     55                     push        ebp
00401011     8B EC                  mov         ebp,esp
00401013     83 EC 40               sub         esp,40h
00401016     53                     push        ebx
00401017     56                     push        esi
00401018     57                     push        edi
00401019     8D 7D C0               lea         edi,[ebp-40h]
0040101C     B9 10 00 00 00         mov         ecx,10h
00401021     B8 CC CC CC CC         mov         eax,0CCCCCCCCh
00401026     F3 AB                  rep stos    dword ptr [edi]
8:          printf("Hello World!\n");
00401028     68 1C 00 42 00         push        offset string "Hello
World!\n" (0042001c)
0040102D     E8 2E 00 00 00         call        printf (00401060)
00401032     83 C4 04               add         esp,4
6:          return 0;
00401035     33 C0                  xor         eax,eax
7:      }
00401037     5F                     pop         edi
00401038     5E                     pop         esi
00401039     5B                     pop         ebx
0040103A     83 C4 40               add         esp,40h
0040103D     3B EC                  cmp         ebp,esp
0040103F     E8 9C 00 00 00         call        __chkesp (004010e0)
00401044     8B E5                  mov         esp,ebp
00401046     5D                     pop         ebp
00401047     C3                     ret
```

圖 7　C 語言程式碼與其相對應的機器碼與組合語言

## 動態分析技術

動態分析中最重要的工具是除錯器，分為用戶模式和系統模式兩種類型.用戶模式除錯器只用來除錯用戶模式的應用程式，如 Visual C++等編譯器內建的除錯器.系統模式除錯器是能除錯作業系統核心的除錯器，它們處於 CPU 和作業系統之間，如 SoftIce TRW2000 等.動態分析就是利用除錯器在程式執行時追蹤程式執行流程以瞭解程式功能。

對於軟體破解者而言，單純用靜態分析技術尋找軟體保護的程式碼難如大海撈針，配合動態分析先找出接近的程式碼位置，再由此處分析起。

本節資料參考[5]。

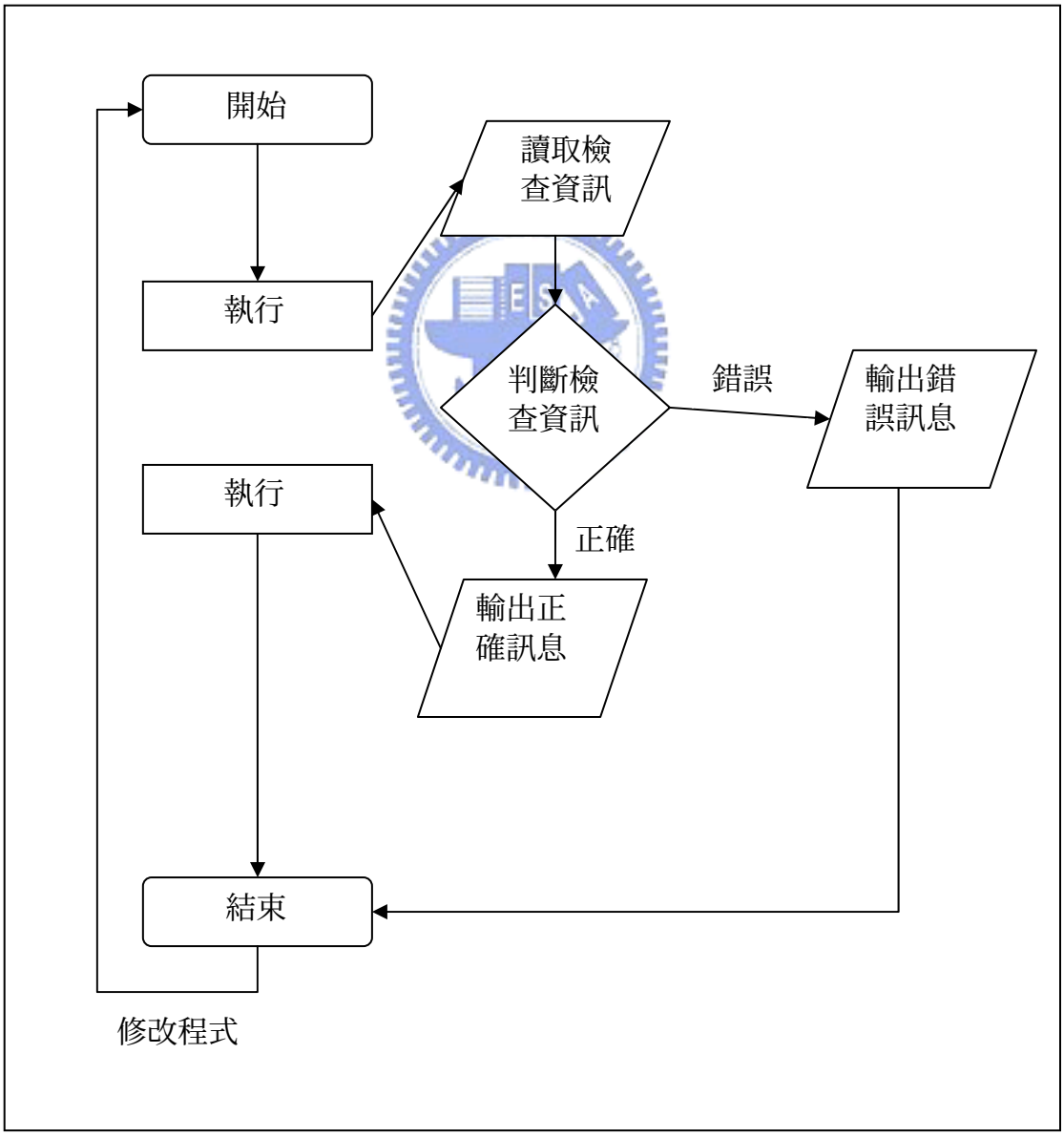


圖 8 軟體破解流程圖

## 2.7 軟體破解流程介紹

在介紹軟體破解流程前，先要建立一個觀念，雖然破解者可以理解反組譯後每個組合語言指令的意義進而了解這段程式碼的功能。但動輒數 MB 至數百 MB 的程式碼對於破解者是過於龐大的數量，再者每段程式碼與整個軟體的關係就好像一顆齒輪與機器的關係，你能了解齒輪的大小樣式卻不一定能知道這顆齒輪在整部機器裡扮演什麼角色。因此軟體破解者透過追蹤軟體執行過程所露出的線索將軟體保護程式碼可能位置範圍縮小，找到後在破解之，這個過程也就是軟體破解流程。

軟體破解流程如圖 8 所示，當軟體開始執行後會在某個時間點讀取檢查資訊，讀取檢查資訊形式可能是

1.要求使用者輸入註冊資訊。

2.從特殊的地方讀取資訊如在光碟片，硬體鎖等。

3.其它。

接著判斷檢查資訊是否符合條件，若符合條件則繼續執行，若不符合條件則顯示錯誤訊息然後結束，讀取/判斷檢查資訊可能重複數次，每次讀取的資訊與判斷的方法可能不同。

### 破解的切入點

軟體保護的程式碼通常只是整個程式的一小部份，破解者的目標是找出這段程式碼的位置分析並解除其功能。雖然破解者有辨識組合語言的能力，但面對動輒數 mb 至數百 mb 的機器碼/組合語言，要找出檢查的程式碼位置並非一件容易的事。因此破解者主要以輸出正確/錯誤訊息處作切入點從這裡開始尋找負責判斷的程式碼並設法解除其作用。

## 2.8 如何解除判斷檢查資訊

保護的方式千變萬化各有不同，像 TimeLOCK 3 保護軟體是一種時間限制軟體，對程式的加密方式就是抽掉部份程式碼，若沒過期或註冊通過，再動態地將程式碼還原回去，由於抽取出來的程式碼經過加密處理，所以別想在任何 DLL 檔中找到這段程式碼，但可透過動態分析技術在程式解密後把這段程式複製出來達到破解目地。各種保護方法在此無法一一說明，若有興趣可參考網路上各種軟體的破解方法。

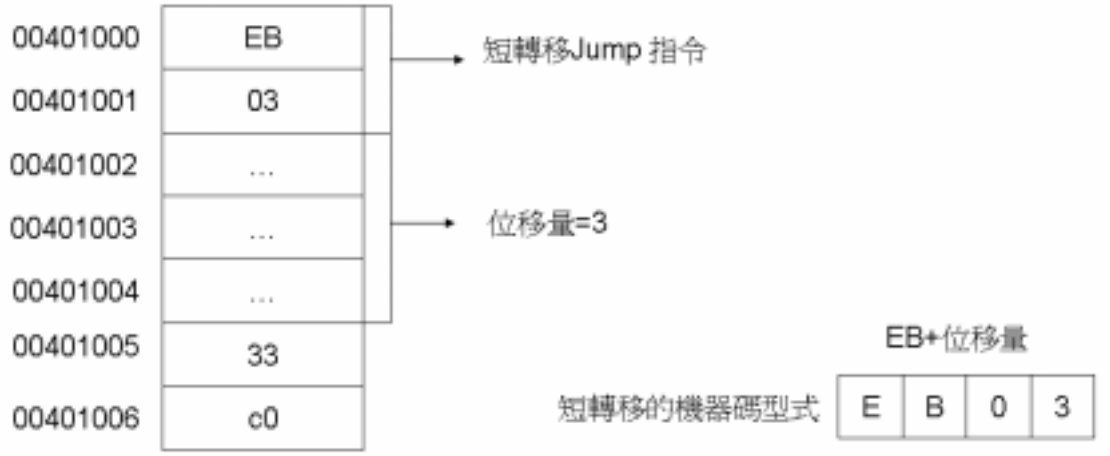


圖 9　短轉移範例




圖 10　短轉移的機器碼型式

當破解者找到保護位置後，先分析瞭解其保護方式再修改程式以破解之。接下來介紹一些的破解者修改軟體的基本方法及一個小例子，讓讀者有個概念。修改軟體的指令中 NOP 與轉移指令是常被用到的兩個修改方法。NOP 機器碼為 90，代表不做任何事。轉移指令分為長轉移與短轉移，這邊簡單介紹一下無條件短轉移，無條件短轉移的機器碼形式為 EBxx(見圖 10)，其中 EB00-EB7F 是向後轉移，EB80-EBFF 是向前轉移。圖 9 為一短轉移的範例，程式執行到 00401000 的時後讀取到短轉移指令，接著就會跳 3 bytes，從 00401005 繼續執行。

```
int main(int argc, char* argv[])
{
      bool flag=false;
      if(flag==false)
        printf("Oh My God!\n");
      else
        printf("Hello World!\n");
      return 0;
}
```

圖 11　範例小程式

```
1:     // testt.cpp : Defines the entry point for the console application.
2:     //
3:     #include "stdafx.h"
4:     int main(int argc, char* argv[])
5:     {
00401010 55                    push          ebp
```

```
00401011 8B EC                mov         ebp,esp
00401013 83 EC 44             sub         esp,44h
00401016 53                   push        ebx
00401017 56                   push        esi
00401018 57                   push        edi
00401019 8D 7D BC             lea         edi,[ebp-44h]
0040101C B9 11 00 00 00       mov         ecx,11h
00401021 B8 CC CC CC CC       mov         eax,0CCCCCCCCh
00401026 F3 AB                rep stos    dword ptr [edi]
6:        bool flag=false;
00401028 C6 45 FC 00          mov         byte ptr [ebp-4],0
7:        if(flag==false)
0040102C 8B 45 FC             mov         eax,dword ptr [ebp-4]
0040102F 25 FF 00 00 00       and         eax,0FFh
00401034 85 C0                test        eax,eax
00401036 75 0F                jne         main+37h (00401047)
8:        printf("Oh My God!\n");
00401038 68 2C 00 42 00       push        offset string "Oh My God!\n"
(0042002c)
0040103D E8 3E 00 00 00       call        printf (00401080)
00401042 83 C4 04             add         esp,4
9:       else
00401045 EB 0D                jmp         main+44h (00401054)
10:        printf("Hello World!\n");
00401047 68 1C 00 42 00       push        offset string
"Hello World!\n" (0042001c)
0040104C E8 2F 00 00 00       call        printf (00401080)
00401051 83 C4 04             add         esp,4
11:       return 0;
00401054 33 C0                xor         eax,eax
12:   }
00401056 5F                   pop         edi
00401057 5E                   pop         esi
00401058 5B                   pop         ebx
00401059 83 C4 44             add         esp,44h
0040105C 3B EC                cmp         ebp,esp
0040105E E8 9D 00 00 00       call        __chkesp (00401100)
00401063 8B E5                mov         esp,ebp
00401065 5D                   pop         ebp
00401066 C3                   ret
```

圖 12　圖 11 小程式的 C 語言程式碼與其相對應的機器碼與組合語言

接下來介紹一個修改的例子，圖 11 的程式很明顯只會印出 Oh My God!，我們將用到上面介紹的小技巧讓程式印出 Hello World!。

**方法 1:使用短轉移**

將程式位置 0040102C 的 8B 4F 改成 EB 19，即往後跳 25 bytes，程式會略過 0040102E 至 00401046 這段程式碼，從 00401047 開始執行 printf("Hello World!\n");

**方法 2:使用 NOP**

將程式從 0040102C 到 00401046 全部改成 90，即 NOP 不做任何事情，這樣一來判斷指令 if 與 printf("Oh My God!\n");的程式碼都被移除，程式自然只會印出 Hellow World!。

以上只是修改軟體的一點皮毛，最主要的目地是讓讀者對軟體修改有個概略性的瞭解，以方便後面章節的進行。本節資料參考[5]。

## 2.9 註冊碼保護方式

先介紹一種相當普遍的註冊碼保護方式，當用戶從網路上下載某個共享軟體(Shareware)後，一般都有使用時間或功能上的限制。當過了共享軟體的試用期後，必須到這個軟體的公司去註冊後方能繼續使用。註冊過程一般是用戶把自己的私人訊息(如用戶名，電子郵件位置，機器特徵等)告訴軟體公司，軟體公司根據用戶資訊利用預先寫好的一個計算註冊碼程式算出一個序號，以電子郵件或傳真等形式發給用戶。用戶在得到這個註冊碼後，按照註冊需要的步驟在軟體中輸入就會取消各種限制，如時間限制，功能限制等，而成為完全正式版。軟體每次啟動的時後，從檔案或系統登錄表中讀出註冊資訊並進行檢查。如果註冊資訊正確，則以完全正式版的模式運行，否則作為有功能限制或時間限制的版本來運行。

軟體驗證註冊碼的過程，其實就是驗證用戶資訊和註冊碼之間的數學映射關係。這個映射關係是由軟體的設計者制定的，所以各個軟體生成註冊碼的演算法是不同的。顯然，這個映射關係越複雜，註冊碼就越不容易破解。映射關係的基本與其變型如下列。

註冊碼=F(用戶資訊)

用戶資訊=$F^{-1}$(註冊碼)

$F_1$(用戶名)=$F_2$(註冊碼)

不過此保護法有幾個常見的問題。

1．軟體本身存有映射關係的函式，可能被提取出來做成註冊機

2．即使檢查註冊碼的演算法再複雜，如果可執行檔可以任意被修改，破解者還

是可以透過修改比較跳轉指令來破解保護

# 第三章 密碼學相關理論與技術

這一節介紹論文用到的密碼學背景知識。

## 3.1 非對稱式密碼系統

非對稱式密碼系統，即是所謂的雙密鑰系統。1976 年以後所發展出公開金鑰密碼系統，即是屬於這種類型。這類的密碼系統，加密金鑰可以公開，但解密金鑰卻無法由公開的加密金鑰得到。一般有加密/解密，數位簽章，鑰匙交換三種類型，接下來將介紹後文用到的數位簽章。

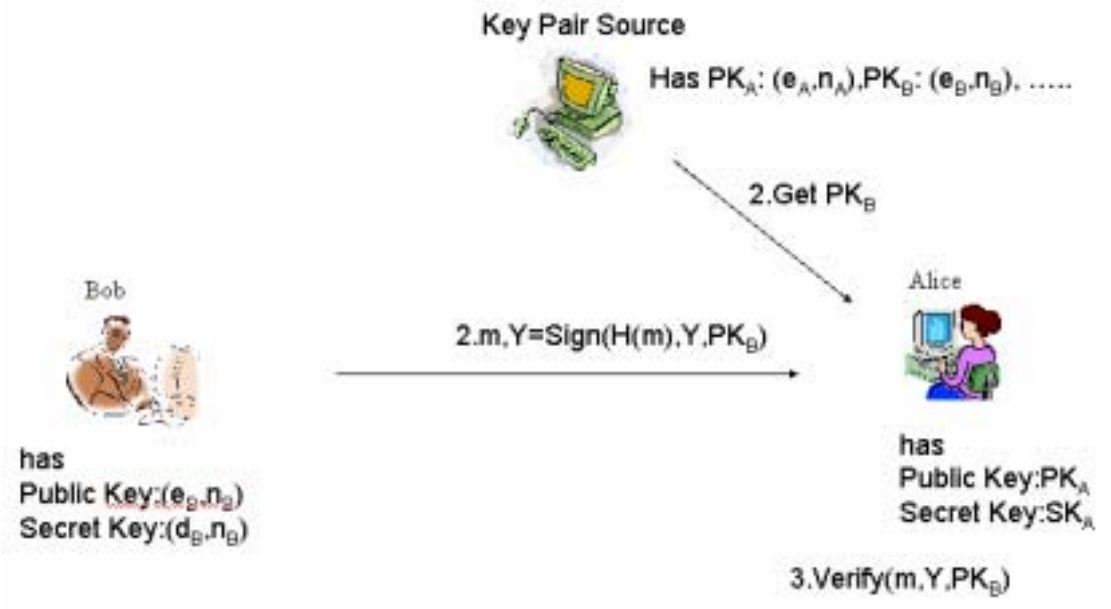


圖 13　數位簽章流程圖

### 數位簽章介紹

這個系統一共有三個成員，Bob，Alice，Key Pair Source，Bob 有一對鑰匙公開金鑰 $PK_B$ 與私密金鑰 $SK_B$，Alice 也有一對鑰匙公開金鑰 $PK_A$ 與私密金鑰 $SK_A$，Key Pair Source 擁有 Alice 與 Bob 的公開金鑰 $PK_A$,$PK_B$。

執行流程如下(圖 13)：

1.假設 Bob 對一份文件 m 簽名以示負責，則 Bob 執行簽名演算法得到簽章 $Y=Sig(m,SK_B)$ ，接著把(m,Y)送給 Alice。

2.Alice 欲驗證簽章 Y 是否為 Bob 對文件 m 做的簽名, Alice 先和 Key Pair Source

取得 Bob 的公開金鑰 $PK_A$

3.Alice 執行驗章演算法 Verify(m,Y,$PK_B$)，如果正確即證明 Y 為 Bob 對文件 m 做的簽名。

## 3.2 RSA 演算法簡介

由 Rivest，Shamir 和 Adleman[6]所發展出使用指數運算的加密方法。其演算法分為三個部份如下:

1. $GEN(1^k)=((e,n),(d,n))$

   先選兩個長約 k/2-bit long 的大質數 p 和 q，計算 k-bit 長的 n=pq，然後隨機選 e 屬於 $Z_n^*$計算 $d=e^{-1} \bmod \psi(n)$, 則 $ed \equiv 1 (\bmod \psi(n))$, 公開金鑰是(e,n), 私密金鑰(d,n)

2. $ENC((e,n),m)=m^e \bmod n$,where m 屬於 $Z_n$

3. $DEC((d,n),c)=c^d \bmod n$

下圖 14 為 RSA 數位簽章流程圖, 其背後的數學原理與相關證明請參考相關書籍。

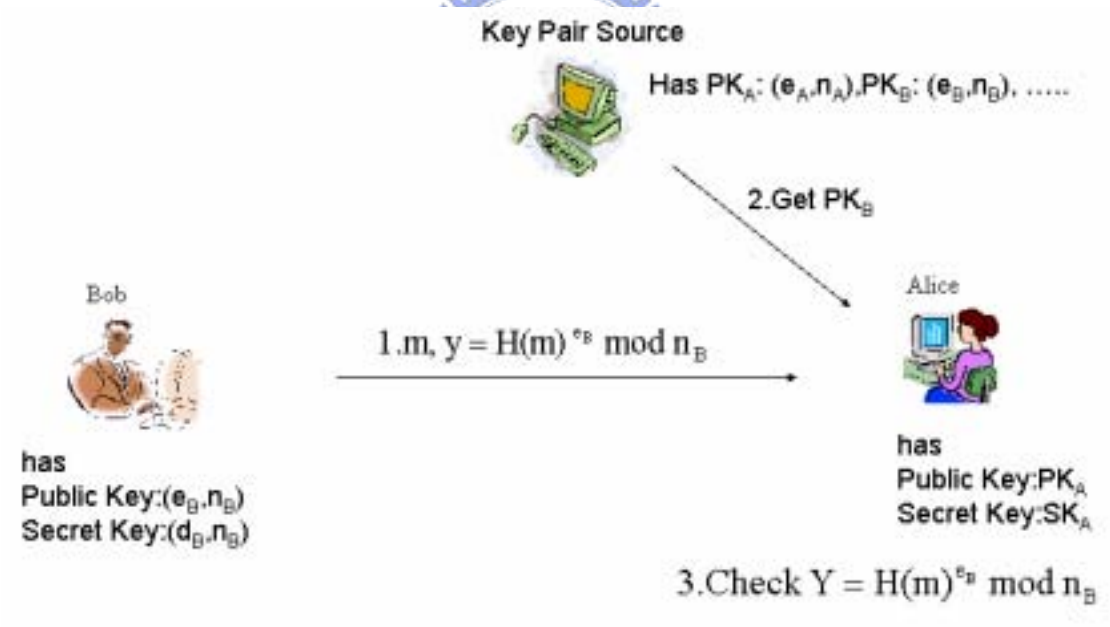


圖 14　RSA 數位簽章流程圖

# 第四章 實作原理

第一節先介紹本論文提出的軟體破解複雜度假設，第二節介紹一般市面上的軟體保護模型，第三節介紹根據第一節的假設所提出的新軟體保護模型。

## 4.1 軟體破解的複雜度

由第二章歸納出下列幾點:

1．軟體是亢長的明文，破解者經由組合語言分析整個軟體是極為費時

2．以往軟體保護只是整個軟體的一部份

對於每一位破解者而言，完整無誤的使用該軟體是破解最大的目標，無論保護做的多完美無瑕，如果破解者願意花時間分析程式裡每個函式如何運作及函式間如何配合，這就代表破解者有能力重製一份與該軟體功能相似的軟體，一旦破解者重製一份功能相近的軟體，就算對原版軟體的某些保護仍無法克服，這時也無關緊要，因為破解者的目標已經達成。因此花時間分析程式裡每個函式如何運作及函式間如何配合可說是每份軟體破解複雜度的上限。軟體保護做到超越這個難度也沒有意義，因為軟體破解複雜度的上限為

**min(重製一份該軟體,破解軟體保護)**

，軟體破解複雜度的上限與該軟體的大小是正相關的。

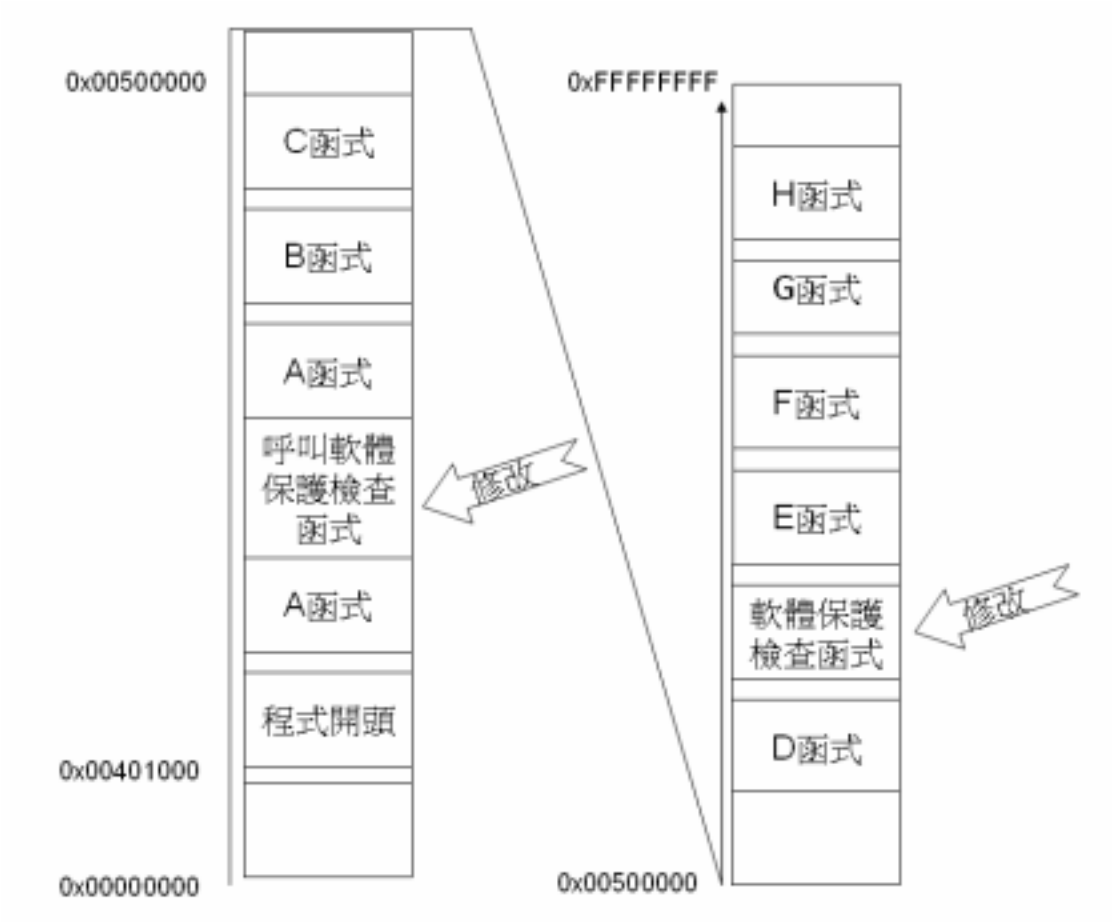


圖 15　一般軟體保護模型

## 4.2 一般軟體保護模型

圖 15 為一軟體經由作業系統載入記憶體後的狀況，每個函式依序儲存於記憶體中。軟體保護只是整個軟體的一小部份，一旦破解者找到並修改軟體保護程式碼部份，該軟體保護就被破解了。當然，軟體保護可能不止一處，但其道理是一樣的。

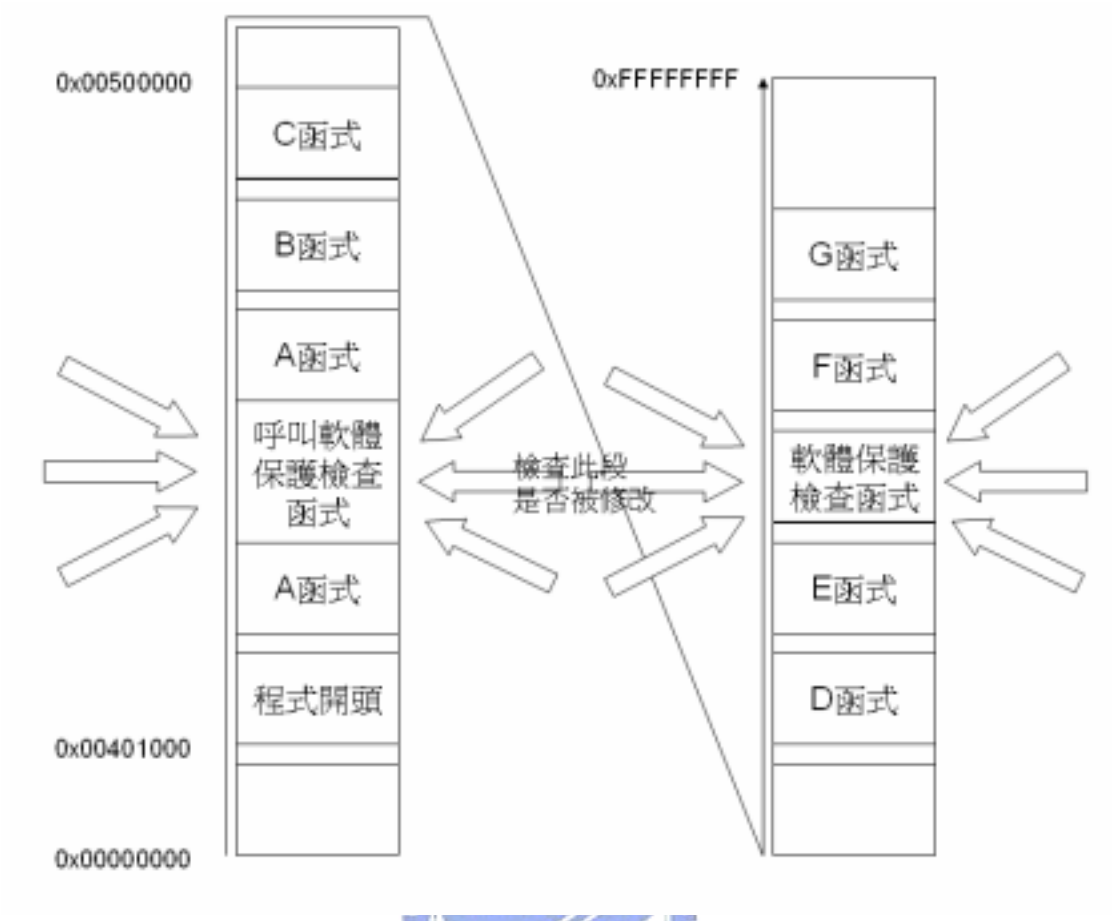


圖 16　新的軟體保護模型

## 4.3 新的軟體保護模型

本篇論文的基本模型如圖 16 所示，整個軟體有兩層保護機制，第一層如一般市面上常見的保護(本文以註冊碼保護實作此部份)，第二層則遍佈於每個函式其目地在於確保第一層保護機制沒有被修改。除非破解者破解這兩層保護才能達到破解的目地。而第二層保護分布在每個函式中，其檢查方式必需具有與原函式程式碼一定程度的不可分辨性，除非 1.人為解析分辨

2.程式按正常執行流程至此發生異常狀態(即第二層保護機制發現第一層保護機制遭到修改所採取的動作)而遭到破解者追蹤，否則破解者難以分辨之。如果破解者想花時間拆除第二層的保護，破解者必須了解每個函式的功能以分辨出何為第二層的保護程式何為原本函式的程式，即破解者了解每個函式的功能這也代表破解者具有重製該軟體的能力，如此即達到軟體保護的可能極限。雖然具有重製能力與重製完成有些微的差異，但這裡假設其為等價。也就是說整個方法的主要精神在於如何讓拆除軟體保護與重製功能類似的軟體難度一樣。

# 第五章 實作方法

第一節先介紹實作目標讓讀者瞭解我們的目地與最後成品，第二節介紹實作流程也就是對軟體加上保護的兩個步驟，第三與第四節詳細介紹每個步驟的細節。

## 5.1 實作目標介紹

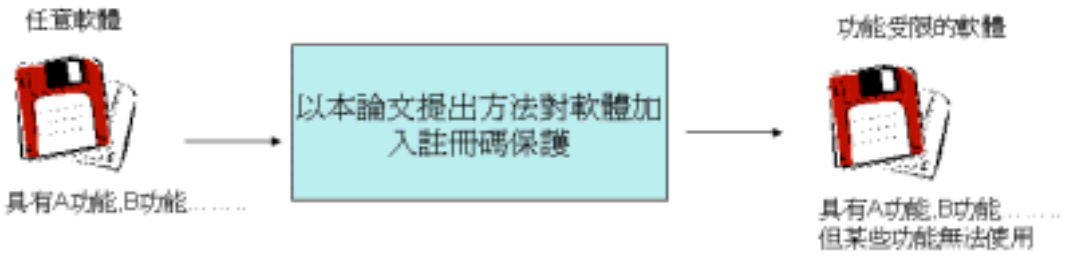


圖 17 修改流程圖

我們以 Windows 為平台，Visual C++ 這種編譯式語言為實作工具，改良第二章第七節介紹的註冊碼保護方式，我們以功能限制為基礎對任意軟體實現此保護方法(如圖 17 所示)，允許破解者在不新增塊(Section)的情況下，任意修改程式碼。

除非滿足下列任一情況，否則破解者無法正常的使用該軟體。

1. 破解者通過正常的註冊程序(圖 18)

   這符合我們的設計目標

2. 破解者經由組合語言分析程式裡每個函式如何運作，函式間如何配合

   破解者達到軟體保護的可能極限(見第四章第一節)

3. 破解者破解 RSA

   破解者破解 RSA，可以任意產生合法的用戶資訊與註冊碼配對

其中我們設計方法概分兩大步驟詳情請見第二節，註冊碼保護方式的實作方法請見第三節。本方法的主要構想是把移除保護的難度提升至軟體保護的可能極限(見第四章第一節) 。

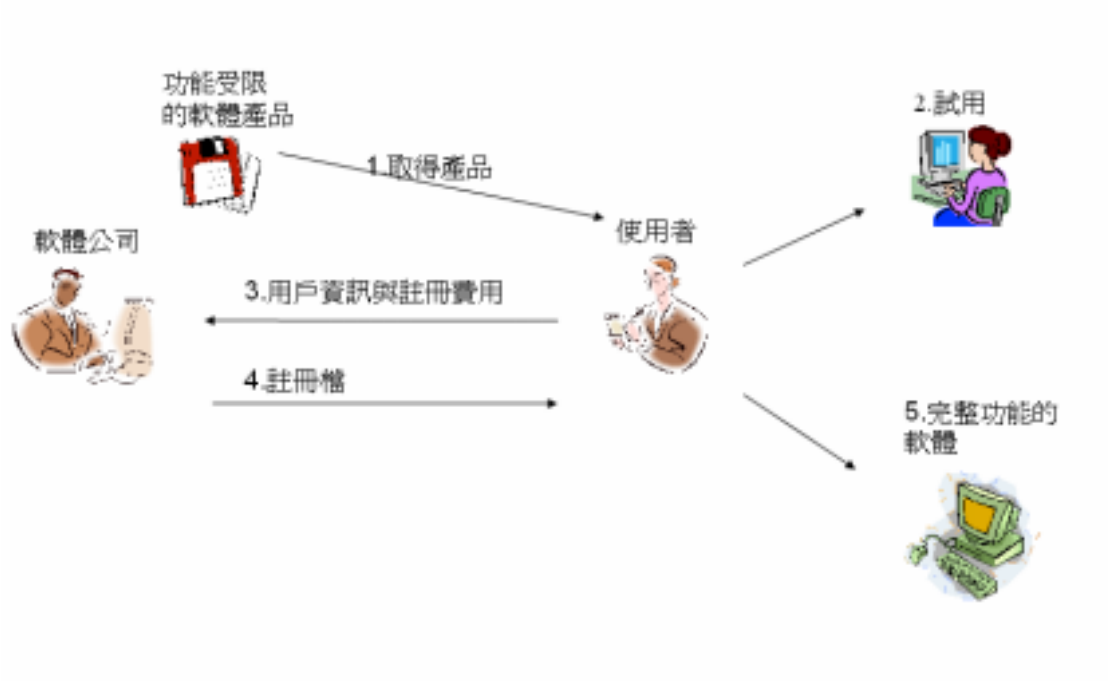


圖 18 商業模型圖

經果處理後，軟體公司與使用者互動的流程如圖 18 所示

1．使用者透過網路或任何管到取得功能受限的軟體產品

2．使用者試用該軟體

3．使用者向軟體公司註冊以取得完整功能的軟體

4．軟體公司把註冊檔給使用者

5．使用者把註冊檔輸入該軟體以成為完全版本

## 5.2 實作流程

對軟體加上保護的整個過程共分兩個步驟，這節介紹兩個步驟如何處理與配合。

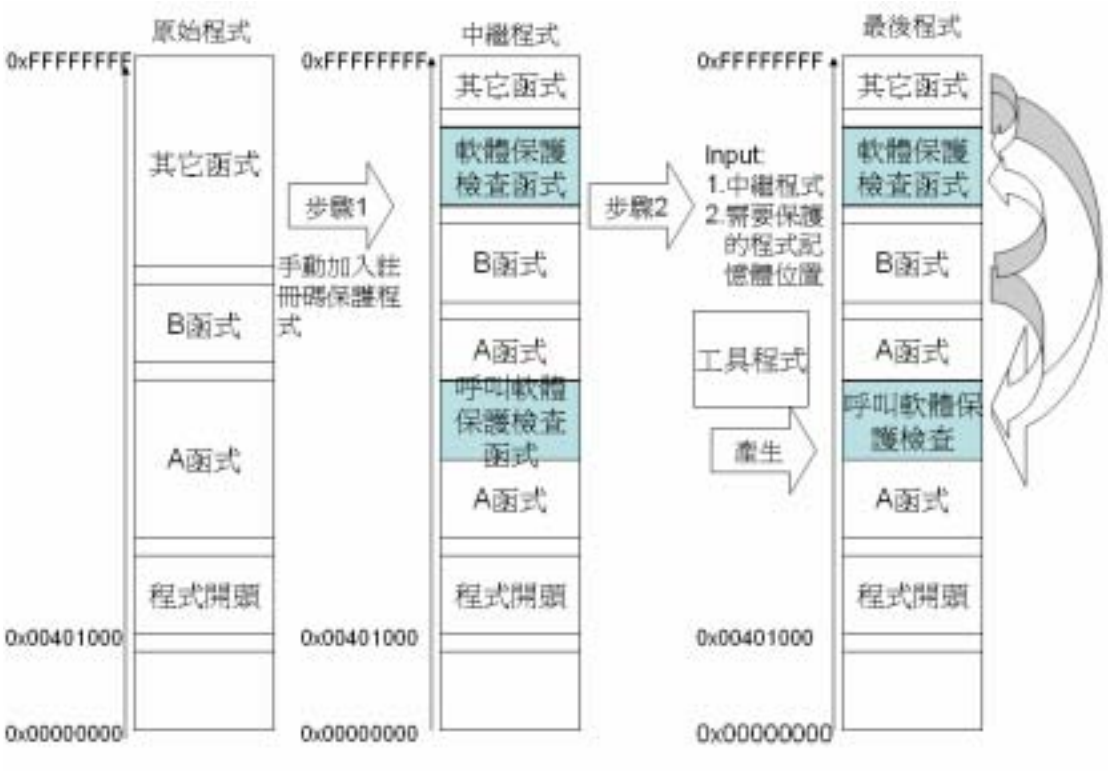


圖 19 實作流程圖

整個實作流程如圖 19，當我們拿到軟體的原始程式後，

### 步驟 1:手動加入註冊碼保護程式

分析欲保護的軟體，找出必要的重點功能，針對該功能加入註冊碼保護，除非使用者滿足 1.破解者通過正常的註冊程序，2.破解者經由組合語言分析程式裡每個函式如何運作，函式間如何配合，3.破解者破解 RSA，三個情況中的任何一個，否則使用者無法使用該功能。假設這個功能為函式 A，則此步驟就是在函式 A 中加入註冊碼保護程式得到圖 19 中的中繼程式。

### 步驟 2:在其它函式加入檢查註冊碼保護程式完整性的程式

破解者要拆除保護勢必要修改註冊碼保護程式即圖 19 中繼程式中藍色區塊位置的程式碼。我們將中繼程式及需要保護的藍色區塊記憶體位置當做輸入交給我們開發的工具程式，工具程式會對整個軟體其它所有的函式加入檢查註冊碼保護程式完整性的程式碼。如此一來，除非破解者詳細看過該軟體的每一個函式並把這些檢查程式碼逐一移除，否則無法達到破解軟體的目地。

## 5.3 如何手動加入註冊碼保護程式

這節分為兩部份，先介紹將如何將註冊碼保護程式加入軟體。再介紹註冊碼

保護程式的實作方法，也就是產生註冊碼的演算法。

### 5.3.1 如何加入註冊碼保護程式

先挑選所要限制的功能，選定功能後在此功能函式中加入 f 註冊碼的檢查，如果檢查失敗則輸出錯誤訊息並中斷該功能的執行，如果檢查通過則繼續該功能執

行。

### 5.3.2 註冊碼保護程式

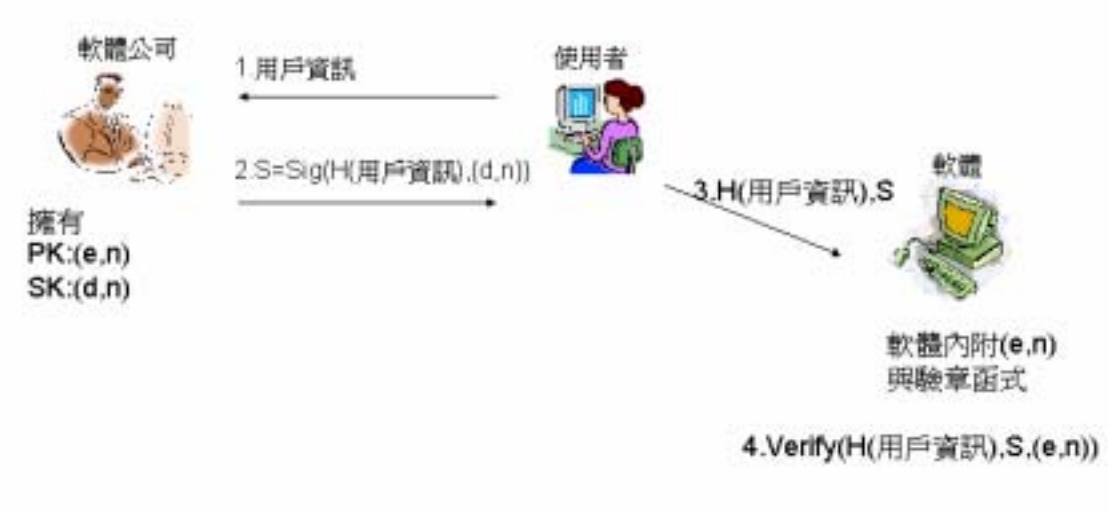


圖 20 註冊碼保護程式實作方式及其執行流程

我們採用 RSA 簽章驗章演算法實作產生註冊碼的演算法，其執行流程如圖 20 所示。

軟體公司有公開金鑰(e,n)與私密金鑰(d,n)以及不可逆雜湊函數 H

軟體使用者有該軟體(內建公開金鑰(e,n)與驗章演算法)

**整個軟體註冊流程**

1. 軟體使用者將用戶資訊及註冊費用交給軟體公司
2. 軟體公司先將用戶資訊作雜湊處理得 H(用戶資訊)，再用簽章演算法對 H(用戶資訊)簽名並交給軟體使用者
3. 軟體使用者將簽章輸入電腦
4. 軟體於執行時執行驗章演算法檢驗其正確性，如果正確則執行受限制的功能，如果不正確則不執行受限制的功能

如此一來可以避免用戶資訊和註冊碼之間的數學映射關係被解譯做成註冊

碼產生器，我們只需要保護軟體內公開金鑰(e,n)與驗章演算法完整性。除非破解者能破解 RSA 才能產生合理的用戶資訊與註冊碼配對。因此第四章第三節的圖 16 模型修正為圖 21 更貼近實作結果。

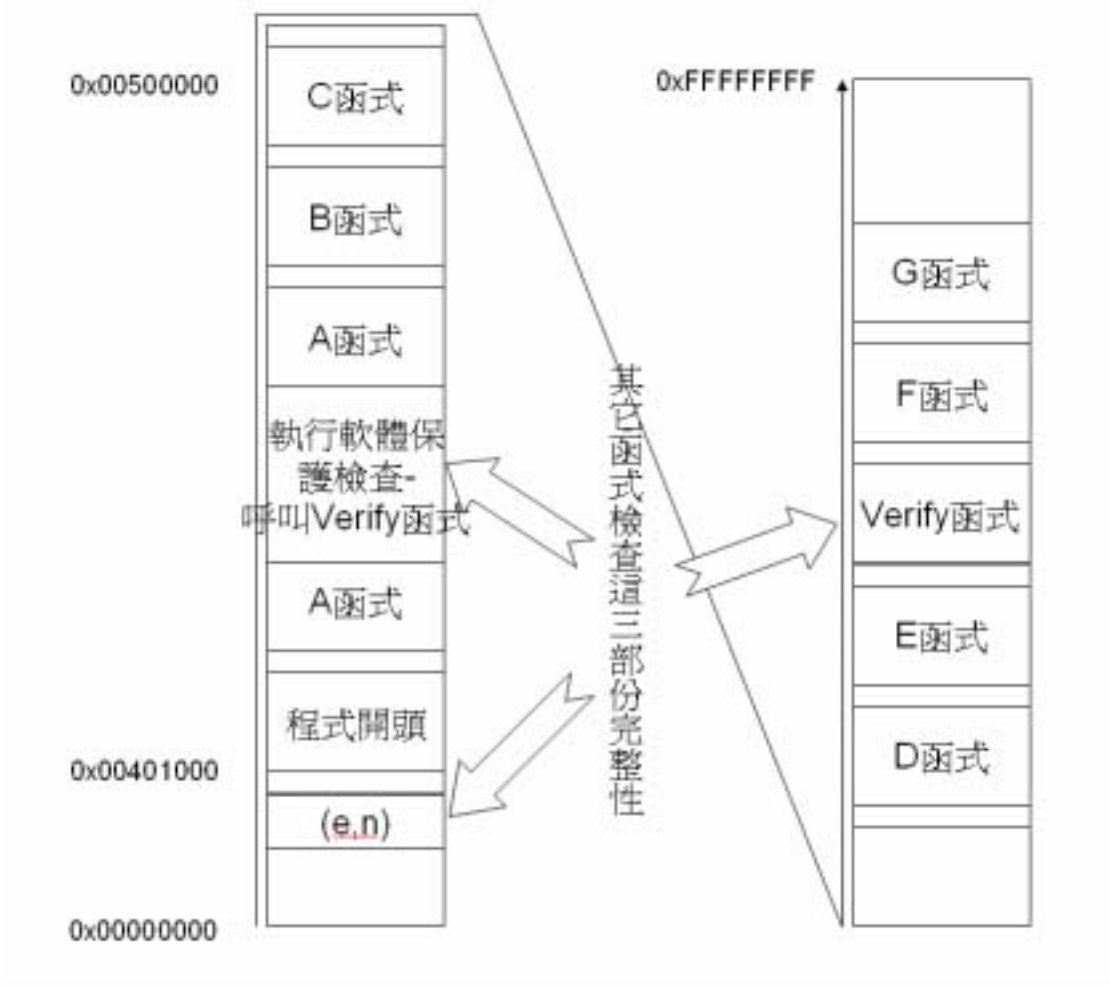


圖 21　新的軟體保護模型

## 5.4 如何在其它函式加入檢查註冊碼保護程式完整性的程式

軟體中其它函式(圖 21 中的 B,C,D,E,F,G 函式)必須檢查程式三部份 1.公開金鑰(e,n)，2.呼叫 Verify 函式的動作，3.Verify 函式本體的完整性以確定軟體保護機制正確運作。其檢查的方法必須滿足下列所有條件:

1. 不需要每個函式在每次被呼叫都執行檢查工作

假設我們在其它函式(圖 21 中的 B,C,D,E,F,G 函式)放入為數 100 次的檢查動作，而且我們在每次的函式呼叫皆執行檢查工作，則破解者最少執行此軟體 100 次即可找出所有檢查點並破解此保護方法，因此我們希望滿足某些

條件才執行檢查動作，透過設定其條件我們可以設計例如執行每 n 小時才有檢查一次的機會, 如此一來該軟體必需經過至少 n*檢查次數的時間才能破解。

即

```
if(條件滿足)
執行檢查工作
```

我們需要一個資料來源做為條件判斷依據, 函數的傳入值與局部變數(見第二章第二節)是最好的選擇，因為我們以函式為單位處理，傳入值與局部變數皆為該函式所私用，因此對傳入值與局部變數的操作是基本又不具辨識性的行為。另外因為可以經由軟體測試拿到每個函數傳入值與局部變數的分布，因此我們可以透過選擇傳入值或局部變數來調整條件滿足的機率。

2. 如果無法通過檢查，要以不明顯的方式讓軟體無法正常執行

如果以類似跳出一個警告訊息的方式，破解者很容易以訊息出現的時間為線索拆除保護。

即

```
if(條件滿足)
執行檢查工作
……
if(無法通過檢查)
執行某些動作
```

透過稍微修改傳回值，傳入值或局部變數，讓該函式執行結果有些微誤差，以達到軟體無法正常執行的目地。

3. 不可用過長或有組織的程式(如函式)進行檢查工作

我們不能用過長的程式或有組織的程式(如函式)進行檢查工作，因為如此十分容易被辨識出來, 比如說其它函式(圖 21 中的 B,C,D,E,F,G 函式)皆呼叫 H 函式進行檢查工作，破解者只需修改 H 函式即可癱瘓檢查機制。

程式即為某段記憶體裡存的資料(見第二章第一節)，檢查方法的基本精神為直接讀取該記憶體位址的資料來比對，讀取的大小為 1-4bytes，這種讀取記憶體的指令是十分基本且廣用的指令, 如此設計以避免前述的問題如圖 22 所示。

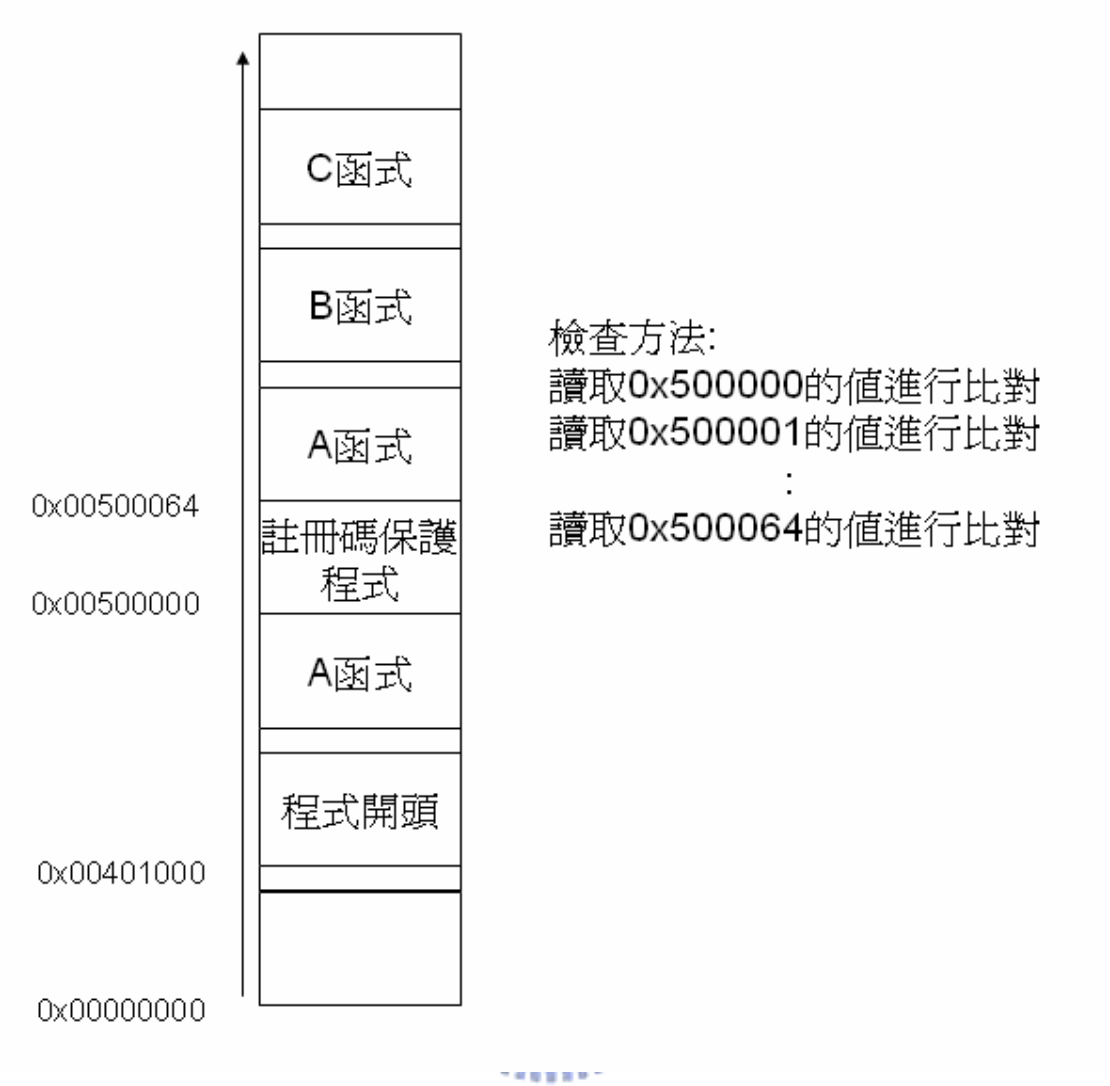


圖 22 檢查方法的基本精神

不過，這個檢查方法還有個漏洞，註冊碼保護程式所在位址 0x500000-0x500064 為一固定位址，破解者可以該位址為線索進行辨別工作。不過這只是檢查方式的基本精神，其實際作法稍有改變以避免這個缺點(見下文)。

4．檢查的方法需具有不易辨識的性質

前面提到註冊碼保護程式所在位址這個漏洞，針對這個問題我們先對破解者的能力做假設再進行設計。破解者可以動態分析查看電腦任何一刻記憶體的狀況，我們額外假設破解者有能力把每個指令執行前後記憶體的狀況儲存下來並分析之，因此檢查方法在執行中的任何一刻記憶體與暫存器都不能出現 0x500000-0x500064。我們使用第二章第二節介紹過間接定址法的觀念，解決方法即是用間接定址的觀念讓記憶體與暫存器在執行任何指令的前後接不會出現 0x500000-0x500064。例子如下:

假設我們要讀出記憶體位置 0x500000 的資料放到變數 j 中:

```
int *p=(int *)0x300000;  指令 1
int j=*(p+80000);        指令 2
```

其中因為型別 int=4 bytes 故增加一各位置等於增加 4 bytes

即 p+80000=0x300000+80000*4=0x500000

指令 2 的機器碼與組合語言如下所列:

```
   int j=*(p+0x80000);
0040B539 8B 4D EC             mov         ecx,dword ptr [ebp-14h]
0040B53C 8B 91 00 00 20 00    mov         edx,dword ptr [ecx+200000h]
0040B542 89 55 E8             mov         dword ptr [ebp-18h],edx
```

第一行把變數 p 的值取出來放入 ecx

第二行將以 ecx 為基底加上 0x200000 為位址所存的值拷貝至 edx

第三行將 edx 的值拷貝到變數 j 裡

關鍵第二行間接定址法在一個指令裡做完讀取動作，該指令執行前後記憶體與暫存器從未出現目標位址 0x500000。雖然破解者可以把 ecx 與 0x200000 相加得到 0x500000，我們可以如前文所提做個簡單的條件判斷只在真正執行比較的時才讓變數 p 存正確的基底 0x300000。如此一來，除非 1.人為解析分辨 2.程式按正常執行流程至此發生異常狀態(即檢查不通過)而遭到破解者追蹤，否則破解者難以分辨之。

```
void sort(int *a,const int n)
{
 int i,j,k,temp;
 for(i=0;i<n;i++)
 {
  j=i;
  for(k=i+1;k<n;k++)
   if(a[k]<a[j])
    j=k;

  temp=a[i];
  a[i]=a[j];
  a[j]=temp;
 }
}
```

圖 23 Selection Sort

```
01:void sort(int *a,const int n)
02:{
03:int i,j,k,temp,*m;
04: for(i=0;i<n;i++)
05: {
06:  j=i;
07:  for(k=i+1;k<n;k++)
08:   if(a[k]<a[j])
09:    j=k;
10:  temp=a[i];
11:  if(temp%2==0)
12:   m=(int *)0x300000;
13:  a[i]=a[j];
14:  a[j]=temp;
15:  if(temp%4==0)
16:  { temp=*(m+0x42d3c);
17:    if(temp == 0x83ec8b54)
18:     a[i]=a[i]+1;
19:  }
20: }
21:}
```

圖 24 修改後的 Selection Sort

實例解說:

圖 23 為 Selection Sort，圖 24 為修改後的 Selection Sort，比較兩個程式的變化來檢視是否滿足上列四點。

11-12:以原函式局部變數 temp 決定是否執行檢查，若是則放入正確基底，即第一點提到的方法

15-19:執行實際檢查動作

16:用間接定址法取出值，即第四點提到的方法

17-18:若否則修改變數，讓執行結果錯誤，即第二點提到的方法

除非破解者能了解這個函式的功能，否則無法正確移除檢查程式即達到第四章第三節提出的目標。

在實作目標裡，我們限制破解者修改軟體的能力，不允許破解者新增塊(Section)，這主要為了簡化整個模型以方便解說，其實軟體所含的塊(Section)數在 PE 檔頭裡有完整資訊，而此檔頭在執行時也載入記憶體中，我們可以用一樣的檢查方式檢查之。

在此架構下，如果破解者已取得一組合法的用戶資訊與註冊碼的確可以達到與破解相同的效果，不過只要對此架構稍做改良即可克服，例如我們要求用戶資訊必需含有電腦資料(如 cpu 型號和硬碟型號)並在軟體中檢查之，以一機一授權的模式即可克服合法註冊碼洩露的問題。

# 第六章 程式說明

第一節與第二節我們將對為本系統所撰寫的程式做功能介紹，第三節則介紹如何使用前兩節的工具對軟體加入保護。

## 6.1 註冊碼產生器

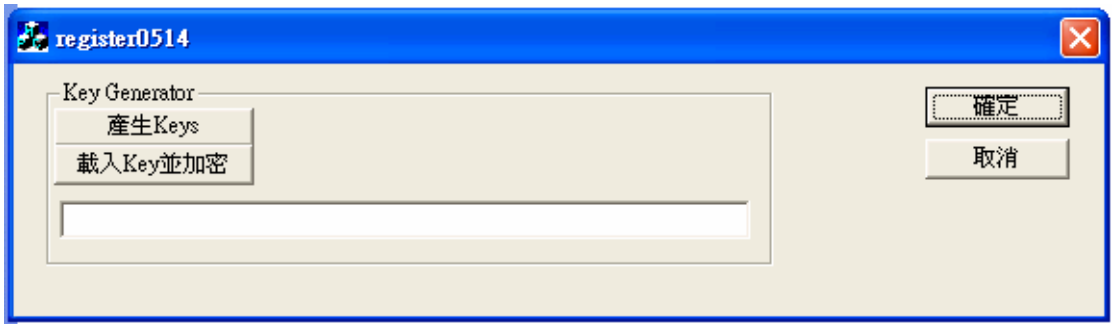


圖 25 註冊碼產生器介面

圖 25 為註冊碼產生器的介面，程式共分兩個功能:

1. 產生 Keys
2. 載入 Key 並加密

產生 Keys 使用 OpenSSL 產生長度為 2048bits 的 RSA 金鑰依照(n,e,d) 的順序儲存於檔案 key.pem 裡，其中 d=65537。載入 Key 並加密將以 key.pem 為金鑰對輸入文字長度為 20 bytes 做簽章並儲存於檔案 licence 裡，另外將金鑰的 n 以方便程式撰寫的格式儲存於檔案 keyn 中。

## 6.2 加入完整性檢查的工具程式

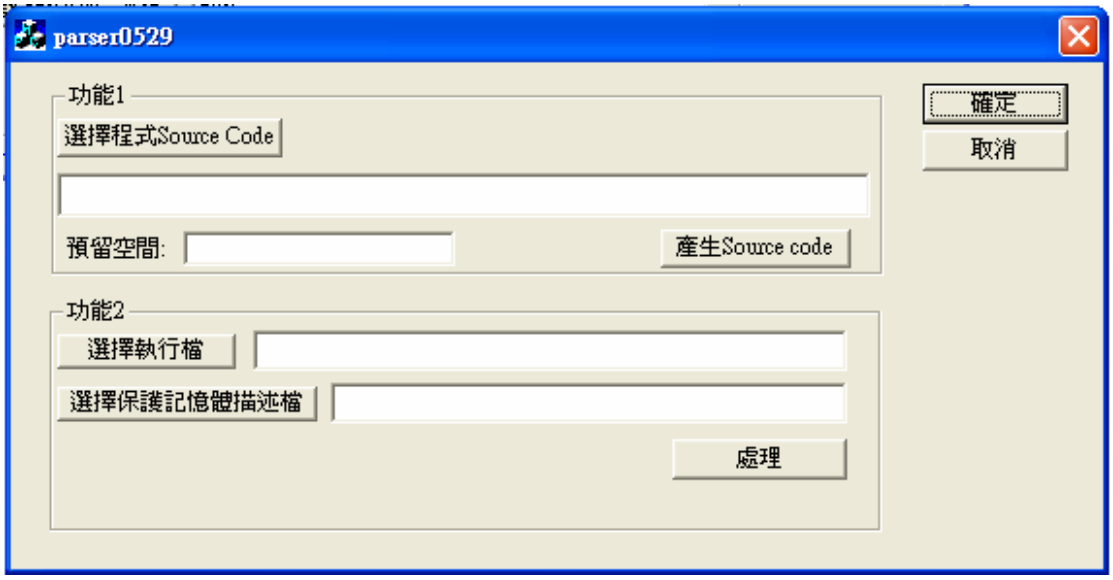


圖 26 程式介面

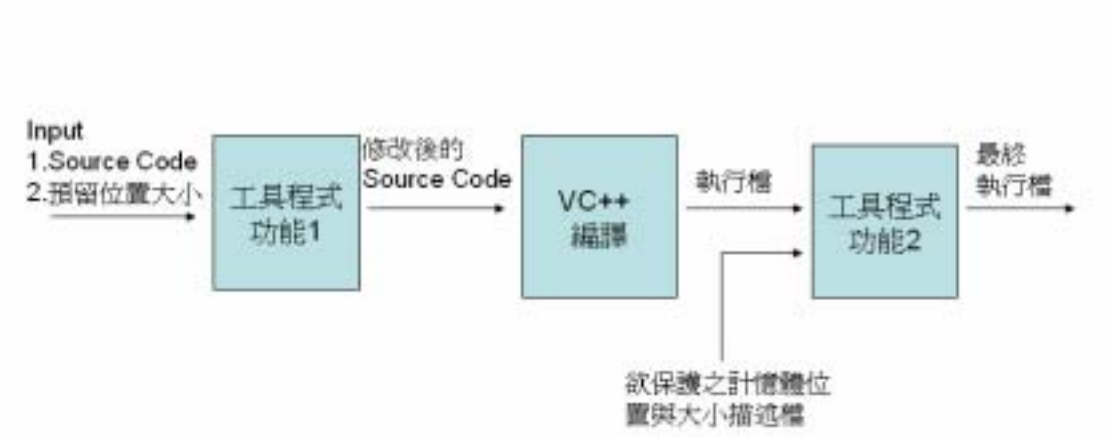


圖 27 操作流程圖

本節介紹的程式將會在所有函式加入檢查註冊碼保護程式完整性的程式。程式介面如圖 26 所示，這一節的操作流程將如圖 27 示分為三個步驟。1.將軟體原始碼及預留位置大小輸入程式功能 1，再按下產生原始碼就會將完整性檢查程式碼加入原始碼中，因為此時只有原始碼沒有執行檔，無法得知程式實際執行位置，因此這時後完整性檢查程式碼的檢查位置及值皆是錯誤的，工具程式功能 1 只是在原始碼中預留指定數量的檢查程式碼。2.將修改後的軟體原始碼編譯為執行檔，由於前面提到的檢查位置與值皆錯誤，一旦檢查將會出現非法記憶體存取錯誤如圖 28 所示。3.將執行檔及欲保護之記憶體位置與大小描述檔交給工具程式功能 2 處理，工具程式將依描述檔的內容修正前面放入錯誤的檢查位置及值，

最後得到加入保護的程式。


圖 28 錯誤訊息

欲保護之記憶體位置與大小描述檔格式:

範例:

&&4198535 ##100
&&4193024 ##200
&&後的數字是 10 進位的欲保護記憶體開始位置

##後的數字是 10 進位的欲保護記憶體大小

因此 &&4198535 ##100 代表從記憶體位置 4198535=0x401087 開始檢查 100 bytes，&&4193024 ##202 代表從記憶體位置 4193024=0x3ffb00 開始檢查 200 bytes。

## 6.3 完整流程

1.使用 6.1 介紹的註冊碼產生器產生一組 2048 bits 的 key。

2.將光碟裡自行開發的簡短 RSA 驗章函式 crypt.cpp 放入軟體中。

3.將 crypt.cpp 裡驗章 key 組的 n 置換成新產生的 n，即 crypt.cpp 第一行的

unsigned char mod[]=·····

4.分析軟體找出重要功能加入第一層保護註冊碼保護

對重要功能加入檢查程式碼

```
FILE *fp;
Unsigned char name[20],name2[512],m[512];
fp=fopen("licence","rb");
fread(name,1,20,fp);
fread(m,1,256,fp);
fclose(fp);
fexpm(m,name2);
if(strncmp((char *)name,(char *)name2,20)!=0)return;
```

這段程式讀取註冊檔 licence 中前 20 bytes 的明文與後 256 bytes 的簽章做比對，如果驗章成功則繼續執行，若失敗則中斷。

3.使用 6.2 的程式完成加入步驟。

4.使用註冊碼產生器則可產生合法註冊碼。

# 第七章 實驗結果

## 7.1 空間複雜度分析

這邊比較沒有保護的軟體大小與實作此方法後的軟體大小:

1.圖 14 中三部份 1.公開金鑰(e,n)，2.呼叫 Verify 函式的動作，3.Verify 函式本體所佔額外空間，如下表 1。

| 種類 / 大小 | 公開金鑰 (e,n) | 呼叫 Verify 函式的動作 | Verify 函式本體 | 總共 |
|---|---|---|---|---|
| 以 Byte 為單位 | 2048bits | 267 bytes | 924 bytes | 1447 bytes |

表 1 空間複雜度分析表

2.圖 14 中其它函式檢查上述三部份所佔空間大小分析

最多需要 22 bytes 的空間來檢查 1 byte 的正確性，因此檢查一遍的最差代價為 1447 bytes *22 =31834 bytes

假設檢查 n 次，總共需要空間為 1447 bytes + 1447 bytes *22*n=1447 bytes*(22n+1)

## 7.2 時間複雜度分析

由第五章第四節的介紹可以瞭解沒有保護的軟體和加上保護的軟體在時間複雜度上會是同一個 order，因此以第五章第四節的例子做了些測試讓大家知道能夠看到一些實際數據。我們以第五章第四節的 Selection sort 進行實驗，所有需要排序的資料皆以亂數產生，對不同的排序數量執行 10000 次並計算其執行所花費的 CPU 時脈數。下表中的數據包括隨機產生排序資料的時間

實驗環境:

電腦:IntelR PentiumR 4 CPU 2.80GHz，504MB RAM

系統:Microsoft Windows XP Professional Version 2002 Service Pack 1

工具:Microsoft Visual C++ 6.0

| 類型 / 排序數量 | 100 | 1000 | 10000 |
|---|---|---|---|
| Selection sort | 328 | 20329 | 1993172 |

| | | | |
|---|---|---|---|
| 修改的 Selection sort | 329 | 20500 | 1995860 |

表 2 時間複雜度分析表

## 7.3 未來工作

1. 檢查註冊碼保護程式完整性的程式

   這部份如果能與編譯器結合，透過編譯器取得更多資訊，則可設計出與軟體結合更緊密的完整性檢查程式碼。

2. 更完整性的分析

   軟體含有程式碼以及其它資料諸如圖片，數據等，程式碼的比例依照軟體屬性而變，本文提出的方法只針對程式碼部份動手腳，因此缺乏對各種不同性質的軟體分析與綜合比較。

# 參考文獻


[1] 施威銘, ”IBM80x86 組合語言實務”,旗標出版有限公司,第三章,2000

[2] 羅雲彬, ”Windows 環境下 32 位元組合語言程式設計”, 全華科技圖書股份有限公司,第三章,2003

[3] 羅雲彬, ”Windows 環境下 32 位元組合語言程式設計”, 全華科技圖書股份有限公司,第一章,2003

[4] 段鋼, ”加密與解密,第二版”,全華科技圖書股份有限公司,第十七章,2004

[5] 段鋼, ”加密與解密,第二版”,全華科技圖書股份有限公司,第三,四章,2004

[6] R.L.Rivest, A.Shamir and L.Adleman, ”A Method for Obtaining Digital Signatures and Public-key Cryptosystem” ,Commun.ACM,Vol.21, pp.120-126,Feb.1978