逢　甲　大　學

資 訊 工 程 學 系 專 題 報 告

Web Content Recovery System

學　　生：吳 嘉 倡(四甲)
張簡琮倫(四甲)

指導教授：李 維 斌

中華民國九十二年十一月

目錄

圖表目錄 …… V

摘要 …… VII

第一章 導論 …… 1

1.1 研究動機和目的 …… 1

1.2 何謂 Web Content Recovery System …… 1

1.3 系統簡介 …… 1

1.4 成果簡介 …… 2

第二章 簡述 Web Service …… 3

2.1 Request 檔案 …… 3

2.2 Response 檔案 …… 3

2.3 Request 網頁內的其他檔案 …… 3

2.4 Response 順序 …… 3

2.5 加快瀏覽速度 …… 3

2.6 續傳功能 …… 4

2.7 虛擬目錄 …… 5

2.7.1 虛擬目錄的設定 …… 5

第三章 TCP Proxy …… 9

3.1 何謂 TCP Proxy …… 9

3.2 TCP Proxy 實作 …… 9

3.2.1 TCP Proxy Server …… 10

3.2.2 TCP Proxy Client …… 10

3.2.3 彼此獨立的 TCP Proxy Client/Server…… 11

3.2.4 TCP Proxy 內部的資料傳送 …… 11

3.2.5 低階的作法 …… 12

3.2.6 可能遇到的問題 …… 12

3.3 Web Service 結合 TCP Proxy …… 13

3.3.1 目的 …… 13

3.3.2 缺點 …… 13

第四章 DES-X 介紹 …… 14

4.1 何謂 DES …… 14

4.2 DES 加密程序 …… 14

4.2.1 起始重排程序(Initial Permutation) …… 16

4.2.2 單一回合的運作細節 …… 17

4.2.3 鑰匙的產生程序 …… 22

4.3 DES 的強度 …… 23

4.3.1 DES 的崩塌效應 …… 24

4.4 Triple-DES 和 DES-X …… 26

4.4.1 Triple-DES …… 26

4.4.2 DES-X …… 27

4.5 DES 的操作模式 …… 28

4.5.1 Electronic Codebook Mode(ECB 模式) …… 29

4.5.2 Cipher Block Chaining Mode(CBC 模式)…… 29

4.5.3 Cipher Feedback Mode(CFB 模式) …… 30

4.5.4 Output Feedback Mode(OFB 模式) …… 32

第五章 SHA-1 介紹 …… 34

5.1 何謂 SHA-1 …… 34

5.2 SHA-1 原理 …… 34

5.2.1 SHA-1 訊息附加位元(Padding bits)…… 35

5.2.2 設定 Message Digest 暫存區的初值…… 35

5.2.3 處理訊息中 512 位元區段…… 35

5.2.4 SHA-1 輸出 …… 37

5.3 SHA-1 的壓縮函數 …… 38

5.4 SHA-1 和 MD5 的比較 …… 38

第六章 Web Content Recovery System 程式介紹 …… 40

6.1 安裝程式 …… 40

6.2 加密和備份程式 …… 43

6.3 解密和還原程式 …… 43

6.4 WCR 主程式 …… 44

第七章 WCR 系統架構及問題解決 …… 47

7.1 WCR 系統架構 …… 47

7.1.1 WCR 加密和備份做法說明 …… 49

7.1.2 WCR 解密和還原做法說明 …… 52

7.2 問題解決 …… 54

第八章 未來發展及心得感想 …… 56

8.1 未來發展 …… 56

8.2 心得感想 …… 57

參考資料 …… 59

圖表目錄

圖 2-1 Request 封包格式 1 …… 4

圖 2-2 Response 封包格式 2 …… 5

圖 2-3 虛擬目錄設定畫面 1 …… 6

圖 2-4 虛擬目錄設定畫面 2 …… 6

圖 2-5 虛擬目錄設定畫面 3 …… 7

圖 2-6 虛擬目錄設定畫面 4 …… 7

圖 2-7 虛擬目錄設定畫面 5 …… 8

圖 2-8 虛擬目錄設定畫面 6 …… 8

圖 3-1 TCP Proxy 作法一意示圖 …… 10

圖 3-2 TCP Proxy 作法二意示圖 …… 10

圖 4-1 DES 加密演算法的一般架構圖 …… 15

圖 4-2 DES 演算法中的單一回合流程圖 …… 17

圖 4-3 DES 每回合 S-box 的流程圖 …… 19

圖 4-4 Triple-DES 加解密流程圖 …… 27

圖 4-5 DES-X 加解密流程圖 …… 28

圖 4-6 ECB 模式操作流程圖 …… 29

圖 4-7 CBC 模式操作流程圖 …… 30

圖 4-8 CFB 模式操作流程圖 …… 31

圖 4-9 OFB 模式操作流程圖 …… 33

圖 5-1 SHA-1 演算法流程圖 …… 34

圖 5-2 SHA-1 處理一個 512 位元區段流程圖 …… 36

圖 5-3 SHA-1 的基本運算流程圖 …… 38

圖 6-1 WCR 安裝程式畫面 1 …… 40

圖 6-2 WCR 安裝程式畫面 2 …… 41

圖 6-3 WCR 安裝程式畫面 3 …… 41

圖 6-4 WCR 安裝程式畫面 4 …… 42

圖 6-5 WCR 安裝程式畫面 5 …… 42

圖 6-6 WCR 加密程式執行畫面 …… 43

圖 6-7 解密還原程式執行畫面 …… 44

圖 6-8 WCR 主程式執行畫面 …… 45

圖 6-9 WCR 在最小化在 task bar 畫面 …… 45

圖 6-10 開啟網頁測試畫面 …… 46

圖 6-11 實際在主機上的檔名畫面 …… 46

圖 6-12 直接送要求給 IIS 開啟網頁的測試畫面 …… 47

圖 7-1 WCR 系統簡單架構圖 …… 47

圖 7-2 WCR 系統處理流程圖 …… 48

圖 7-3 WCR 加密備份程式流程圖 …… 50

圖 7-4 DES-X 加密使用 CTS mode 操作流程圖 …… 52

圖 7-5 解密還原程式流程圖 …… 53

圖 7-6 DES-X 解密使用 CTS mode 操作流程圖 …… 54

摘要

近年來因為網路興起，使得家家戶戶使用網路的普遍性大為增加，而使的許多公司企業甚至個人都會在網路主機上製作屬於自己的網頁，藉以拉近和客戶或是其他民眾的距離，當然許多網頁也開始負起銷售或是售後服務等任務，可以說已經是另外一種公司企業形象的表徵，也因為如此近年來不斷發生駭客入侵公司企業的網路主機竄改其網頁，造成公司企業在形象上和客戶信任度上的損失，因此我們希望能開發一套軟體在公司企業網站遭到入侵、竄改之後能以最即時的反應將其復原或阻止其遭到竄改之網頁不被客戶或瀏覽者看到，除了兼具即時檢查檔案完整性、備分和還原之外也能以 E-mail 等方式即時通知系統管理者，以將被害損傷程度將至最低程度。

第一章 導論

1.1 研究動機和目的

因為近年來許多電腦駭客喜歡入侵公司企業甚至政府的網站主機，並置換或修改其網頁內容使的公司企業或政府遭受形象和顧客民眾的信任受損，所以有許多保護網頁的軟體或工具出現，可是大部分這類軟體都是以固定週期檢查的方式來確認網頁完整性，並且大都需要建立資料庫存放檔案相關資料(摘要值)，有鑑於此我們希望能開發一套保護網頁的軟體除了能即時檢查和還原檔案之外，也不需要另外建立資料庫存放檔案相關資料，以達到安全、快速、便利的目的，所以 Web Content Recovery System 就此而生。

1.2 何謂 Web Content Recovery System

WCR 系統主要為架構在 TCP/IP 上用來保護 IIS 所開啟網頁和確認其完整性的系統，系統分為 WCR 主程式、加密備份程式及解密還原程式三大部分，保護網頁的加密方式採檔案名稱以 DES-X 演算法加密達成，而確認網頁完整性則以 SHA-1 演算法產生摘要值來比對，並採即時檢查、即時還原遭竄改檔案、即時通知、不影響檔案內容等四項為最大特色。

1.3 系統簡介

- WinSocket API

- Microsoft Platform SDK
- Borland C++ Builder 6.0
- Internet Information Services 5.0

開發語言

專題實驗主要是以 C++來完成，介面設計則是以 Borland C++ Builder(BCB)來設計，TCP Proxy 部分以 BCB 封裝的 TCP 元件和 Winsocket API 來作，關於檔案加解密和取摘要值的 DES-X 及 SHA-1 演算法則都是以 C++撰寫。

開發環境

因為 IIS 先天環境的限制，加上 WCR 系統需要權限控制權來保護設定檔和程式，所以除了安裝 IIS 之外作業系統也必須是 Windows 2000/XP 以上，至於執行程式所需之環境則沒有特殊需求。

1.4 成果簡介

WCR 系統主要達成網頁遭竄改的即時檢查、即時還原、即時通知三項主要功能，並在加密備份網頁時不需要特別開啟專用程式，利用 Windows 作業系統內建的檔案總管或是瀏覽我的電腦按下右鍵選單即可達成，而解密還原網頁則提供 GUI 介面程式讓使用者可以看到原始檔案名稱加以選擇還原，不需要建立對照表或是整個目錄還原。

第二章 簡述 Web Service

2.1 Request 檔案

當使用者選取或輸入欲瀏覽一個網頁或檔案時，Client 端的瀏覽器會送出一個 Request 訊息，當中指明 Method 為 GET 加上所要求的網頁或檔案名稱，以及其他訊息。(圖 2-1)

2.2 Response 檔案

Web Service 收到 Request 後會將所要求的網頁內容包含檔頭資訊回傳給 Client 端，檔頭資訊會描述這個網頁或檔案的本文長度但不包含檔頭的大小。(圖 2-2)

2.3 Request 網頁內的其他檔案

當 Client 收到 Response 會再依網頁內容來提出新的 Request，如圖片、背景音樂等，直到所有檔案傳輸完為止。

2.4 Response 順序

當有多個 Request 時，其 Response 並不會指明檔名，而是按照 Request 的順序來做回應。

2.5 加快瀏覽速度

Response 還會包含一個 ETag 值以及 Date 值，以供 Client 端下次可以透過這個 2 個值來判斷檔案是否有更新，Client 會在檔頭內加上 If-Modified-Since:時間和 If-None-Match: “Etag 值”，來詢問

檔案是否有更新，如果都相同表示檔案沒有修改過，可以不必重新回傳,此時 Web Service 會回傳一個 304 Not Modified 的狀態碼給 Client 端，Client 藉此節省網路頻寬與加快瀏覽速度。

2.6 續傳功能

在 HTTP/1.1 裡支援截取檔案的部份內容，方法是在 Request 裡的 Range 欄位內指定要截取的範圍及所使用的單位，但在 HTTP/1.1 裡只支援一種單位就是 bytes，因此透過這個功能便可支援續傳的功能，或是將檔案分割傳送以加快傳送的速度。

[Method] 檔名	版本
Accept: 指定可接受的 media type	
Accept-Language: 限制所欲接收的所接收的 response 所使用的自然語言	
Accept-Encoding: 限制可使用的編碼格式（content-codings）	
If-Modified-Since: 時間(optional)	
If-None-Match: “Etag”(optional)	
Connection: 指定所要使用的連線方式	
Cache-Control:指定所需遵從的 cache 機制	

圖 2-1 Request 封包格式 1(Method 依照使用的功能有不同的設定，如果是要求檔案的話，Method 設為 GET；如果以 GET 或 POST 方式傳送附

加資訊，如留言版等，則將 Method 設為 POST。）

版本	狀態碼
Date: 說明此訊息所產生的日期和時間	
Content Type: 本文所使用的 media type	
Accept-Ranges: Range 所使用的單位	
Last-Modified: 最後一次修改的日期	
ETag: 檔案的 Tag 值	
Content-Length: 表示本文內容的長度	
本文	

圖 2-2 Response 封包格式 2

2.7 虛擬目錄

所謂虛擬目錄，就是一個實際上不在主目錄中、但是用戶端瀏覽器卻認為它在那裡的目錄。

2.7.1 虛擬目錄的設定

IIS 的虛擬目錄設定方法為：[開始]->[設定]->[控制台]->[系統管理工具]->[電腦管理]->[服務及應用程式]->[Internet Information Services] 選取”預設的 Web 站台”可看到目前設定虛擬目錄及其真實路徑的對應，以及網頁根目錄下的所有目錄及網頁(圖

2-3)。在”預設的 Web 站台”上按右鍵->[新增]->[虛擬目錄]可開啟設定虛擬目錄的功能，第一步先設定別名(圖 2-4)，也就是所要新增的虛擬目錄名稱，下一步則是選取所對應的真實目錄(圖 2-5)，再下一步則是設定目錄的權限(圖 2-6)，接下來便完成設定的動作(圖 2-7)，再來便可看到新增的虛擬目錄出現在之前的列表中(圖 2-8)。

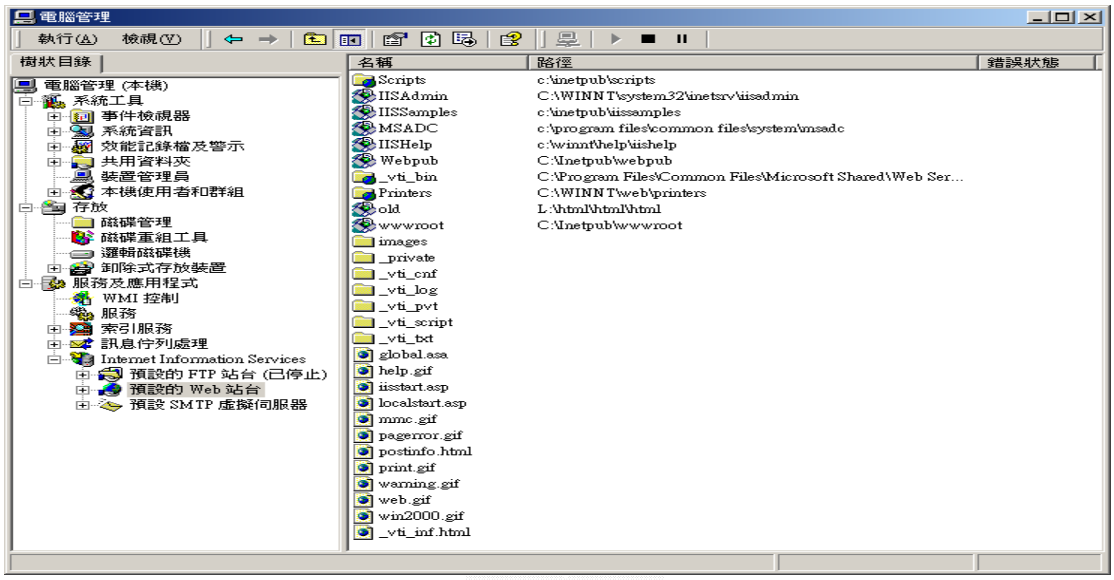


圖 2-3 虛擬目錄設定畫面 1

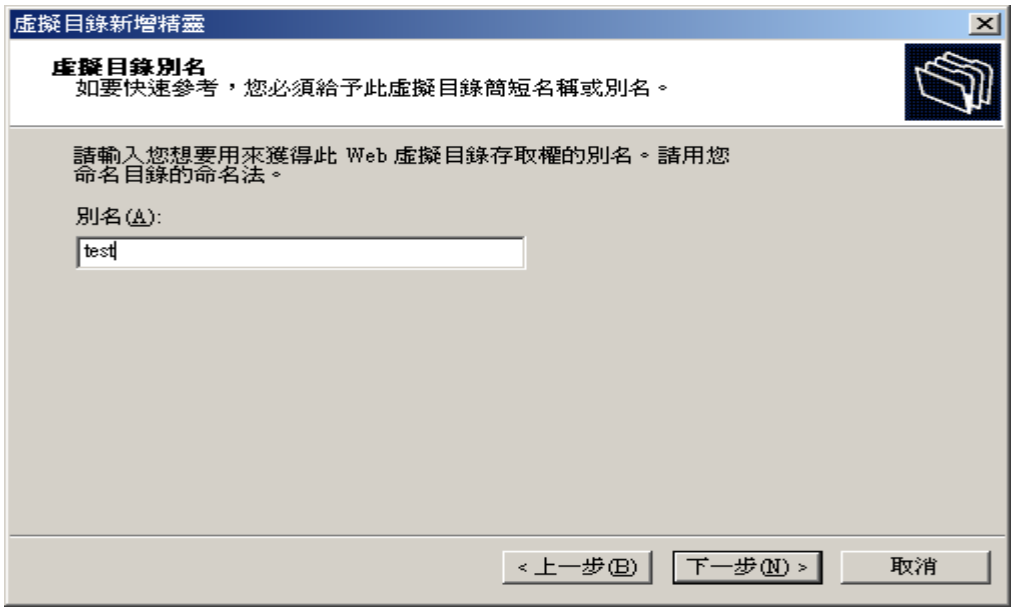


圖 2-4 虛擬目錄設定畫面 2

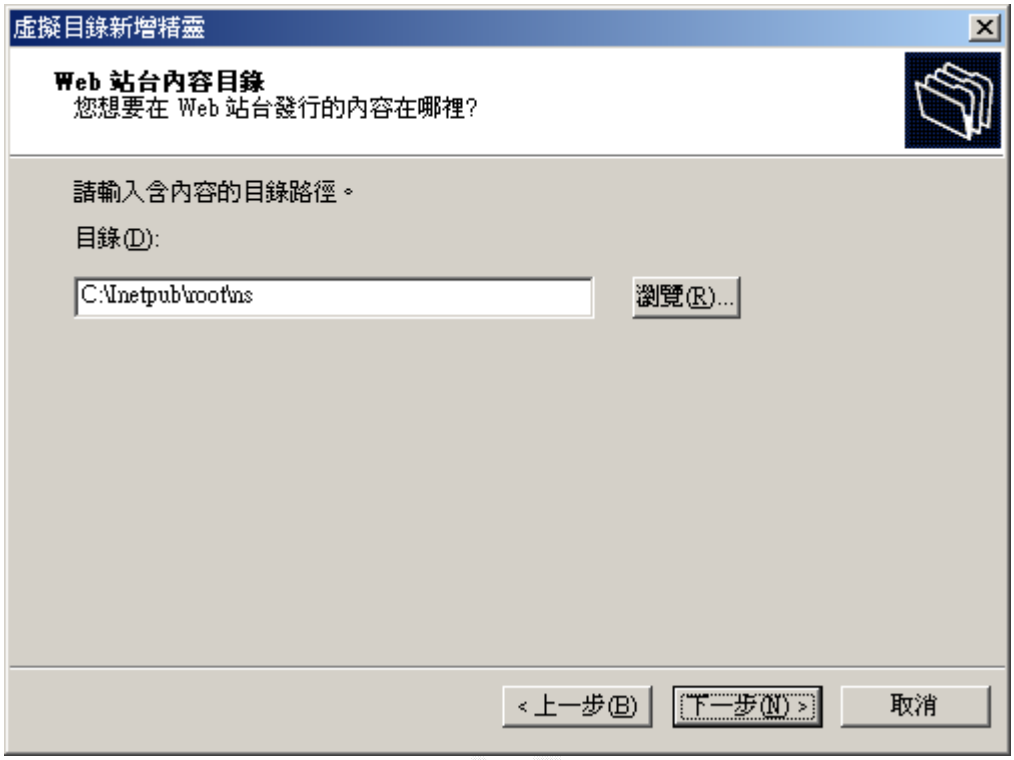


圖 2-5 虛擬目錄設定畫面 3

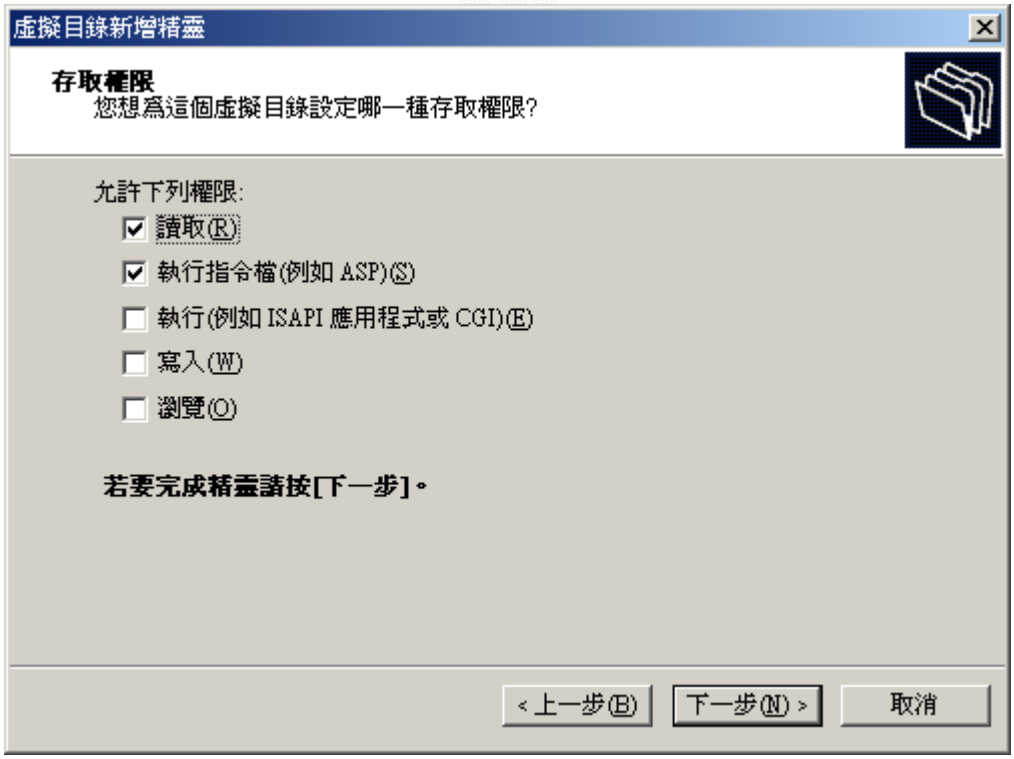


圖 2-6 虛擬目錄設定畫面 4

圖 2-7 虛擬目錄設定畫面 5

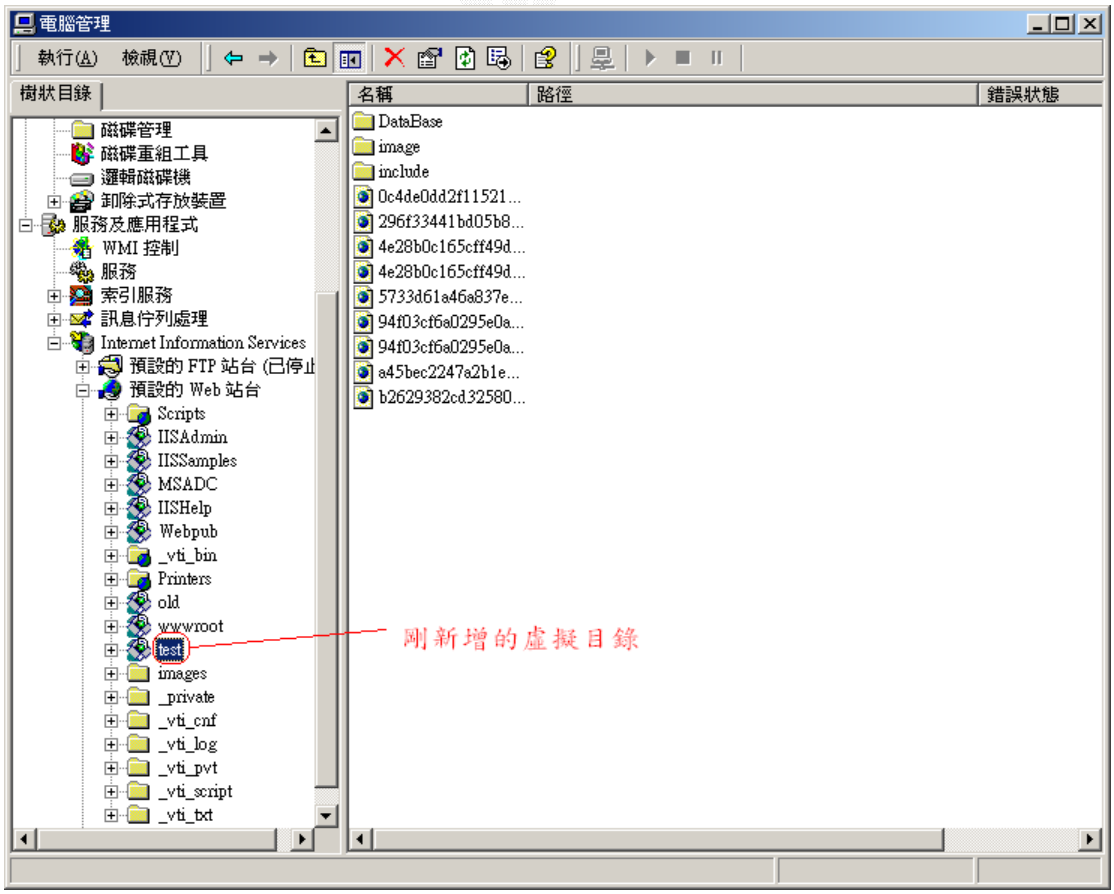


圖 2-8 虛擬目錄設定畫面 6

第三章 TCP Proxy

3.1 何謂 TCP Proxy

針對OSI model中的應用程式層之end port建立一Multithreaded Server來達成IP Tunnels,使由外界進來的封包資料能夠依照使用者設定將封包送到指定的End Port上面。

一般來說封包的裡頭會記錄目的位址與目的port，裡頭記錄的通常是其真正的目的地，也就是封包送到指定的位址與port後傳輸便完成，可能再等待對方的回應，而TCP Proxy則是在這個連線中間在作一個轉送點，也就是封包並非送給真正的目的地，而是先送給TCP Proxy再由他轉送給真正的目的地，轉送的目的通常是希望能在封包上動些手腳，如filter或流量控制等。

3.2 TCP Proxy實作

實作時所要面對的問題是如何將發送端的封包送到接收端，並將接收端的封包回傳給發送端，也就是在服務對象不止一個連線的情況下能將封包送到正確的地方，解決方法可將目的port改成真正目的地的port後再送過去，讓目的服務程式感覺不出中間的轉送，並且讓目的服務程式直接從所在port將資料送回給發送端或是直接擋在Server的前面，先將所要處理的動作完成後，再送給Server(圖3-1)，這屬於較低階的作法；或是利用Client/Server架構(如圖3-2)，並提供機制

讓每個連線獨立且互不影響，讓整個連線的動作都需透過TCP Proxy包含Request和Response。

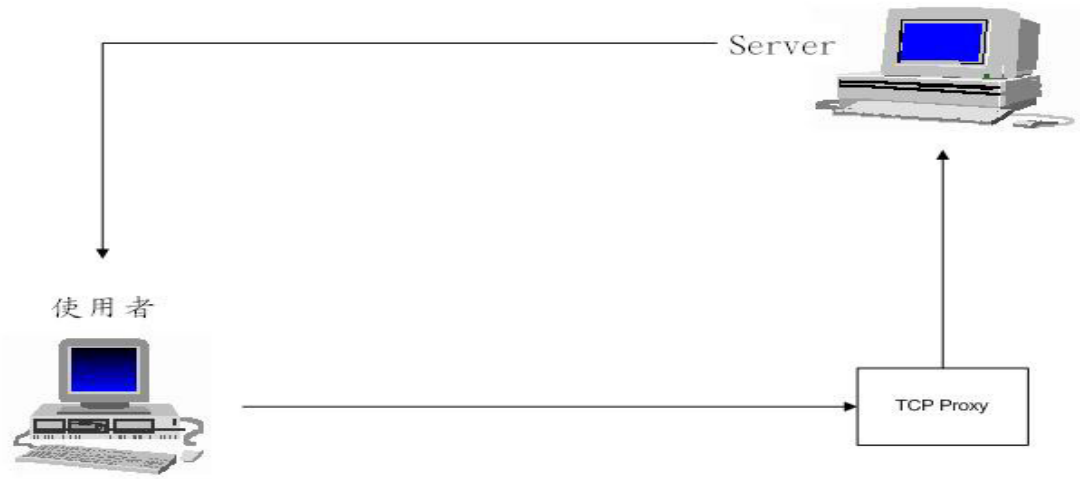


圖3-1 TCP Proxy作法一意示圖

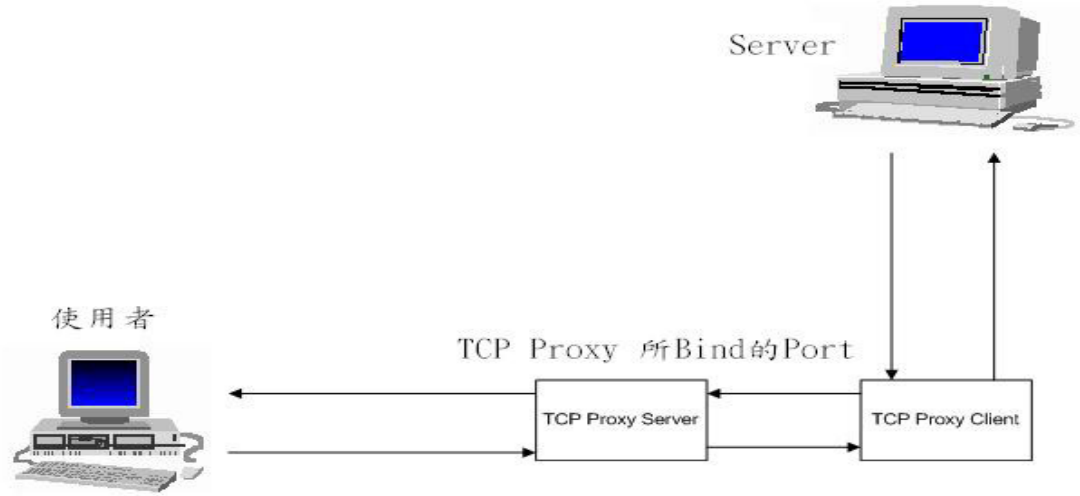


圖3-2 TCP Proxy作法二意示圖

3.2.1 TCP Proxy Server

TCP Proxy Server所要負責的工作是接收發送端的封包，並轉送給TCP Proxy Client，然後再接收由TCP Proxy Client所回傳的封包並回傳給發送端。

3.2.2 TCP Proxy Client

TCP Proxy Client所負責的工作是從TCP Proxy Server接收封包，並轉送給接收端，並將接收端回應的封包轉送給TCP Proxy Server。

3.2.3 彼此獨立的TCP Proxy Client/Server

每個TCP Proxy Client/Server為成對組合，每個連線都由一組TCP Proxy Client/Server來負責，彼此獨立互不影響，否則可能會將封包傳送到錯誤的地方，作法上可以在TCP Proxy Server收到封包後，建立一個Thread來執行TCP Proxy Client所要執行與回傳的工作，並將TCP Proxy Server的Socket或fd傳入，這樣當TCP Proxy Client收到接收端的回應時，便可透過此Socket或fd來將資料回傳給發送端，而發送端與接收端的對應也是透過此機制來維持，因為是用獨立的Thread來處理封包，因此不同的連線不會互相影響，而每個連線的對應也是透過此機制來維持，不需額外的對應表來做對應。

3.2.4 TCP Proxy 內部的資料傳送

兩者之間的傳送方式，可不需透過封包來傳送，因在同個程式內，所以只需透過簡單的參數傳送，將TCP Proxy Server(Client)收到的封包傳給TCP Proxy Client(Server)，甚至可將整個Socket傳入，將整個Socket傳入的好處是可以等初始動作完成後再來接收資料，並且當要把資料回傳時，也可直接使用此Socket，便能正確無誤的送到正

確的地方。

3.2.5 低階的作法

前面提到都是運作在AP層，所以實作起來還算簡單，移值性也較高，因為高階的Socket程式寫法在許多平台上是差不多的，或許無法直接移植，但也只需做部份修改，這種作法的適用範圍也會較大。如果實作在底層如driver或NDIS，可能得有個對應表來作轉送的查詢，做在底層的好處是可以限制無法直接由外部傳送封包到想要保護的port，做法是當封包目的port為保護的port時就將封包的port改成其他沒在用的port或是將CRC值改掉，使這個封包無效，但如果是運作在AP層的話想要封鎖port不能從外部送入的話，可能需要OS提供這樣的服務，或是用其他方法來達成。做在底層甚至可以直接擋在Server之前，做完對封包所要做的處理後，直接送給Server。

3.2.6 可能遇到的問題

a. 當收到的封包還沒送完時，又有新的封包進來，如果只有一個Buffer可能會被蓋過去，解決方法是建立一個queue來將存放封包，並等待處理，就不會讓新的封包蓋掉舊的封包。

b. 當傳送檔案時，如果檔案很大且傳送速度太快，可能網路卡的Buffer會不敷使用，而造成錯誤，此時需將程式等待一小段時間後重送，這也是為什麼要用Thread來實作TCP Proxy的理由，因為這樣

等待才不會影響主程式的運作。

c. 如果實作在AP層需提供一個機制來使的即使對方將資料直接送往Server所在的Port，也是徒勞無功，否則TCP Proxy的功能會因此消失。

3.3 Web Service 結合 TCP Proxy

3.3.1 目的

Web Service 所提供的功能是製造廠商出貨時決定的，要修改他的所提供的服務並不是簡單的事情，但如果能在 Request 進到 Web Service 前先做一點手腳，便可讓 Web Service 提供他原本的功能，但卻產生額外的附加功能，也就是在不更動原 Web Service 的情況下加入新的功能，也因非針對特定的 Web Service 來做處理，所以基本上可以適用於所有的 Web Service，但還是得做些微的修改，如各家 Web Service 的虛擬目錄的讀取方式不同等。

3.3.2 缺點

如果無法將 Web Service 所在 port 限制為只能讓本機轉送，那只要對方知道 Web Service 所在的 port，那掛在上面的 TCP Proxy 會視同無效，所以需提供一個機制來讓即使知道 Web Service 所在的 port，送出的 request 並無法得到想要的回應，也就是想要得到正確的結果一定得透過 TCP Proxy 的轉送。

第四章 DES-X 介紹

4.1 何謂 DES

DES 為 Data Encrypt Standard(資料加密標準)的縮寫，在 1997 年被美國國家標準與技術協會(National Institute of Standards and Technology，NIST)所採行；而 DES 就成為 NIST 所發布的第 46 項聯邦資訊處理標準（FIPS PUB 46)。基本上 DES 屬於對稱式加密的一種，其加密過程會使用一把 56 位元的鑰匙來對 64 位元的資料區段進行加密，其演算法會透過一連串的步驟將 64 位元的輸入轉換成 64 位元的輸出，同樣的步驟和鑰匙也被用在解密上。

4.2 DES 加密程序

DES 的加密與所有的加密架構一樣都需要兩各輸入：需要被加密的明文和鑰匙。在 DES 中明文長度是 64 位元，而鑰匙的長度是 56 位元。**圖 4-1** 是 DES 加密演算法的一般結構，我們可以看到密文處理分為三階段。首先 64 位元的明文區段會通過一個起始重排程序（Initial Permutation, IP)，IP 會重新組合其位元來產生「重新排列過的輸入」。下一個階段是由具有相同功能的 16 個回合所組成，其中包含了重排與取代這兩種函數，最後一回合（第 16 回合）所產生的 64 位元是根據明文與鑰匙所產生的，其輸出的左右兩邊交換過來之後就形成了「前期輸出（peroutput)」，這個前期輸出會經過一個重排程序(IP^{-1})來產

生 64 位元的密文。(IP^{-1} 是起始重排程序(IP)的反函數)

而右半邊顯示了 56 位元鑰匙的使用方式，一開始鑰匙會通過一各重排程序，接下來在這 16 個回合的每一個回合中都會用左旋位移和重排的方式來產生一把子鑰匙(K_i)，每個回合所用的重排程序都是一樣的，但是因為鑰匙的位元被重複地使用，所以得到的每一把子鑰匙都是不相同的。

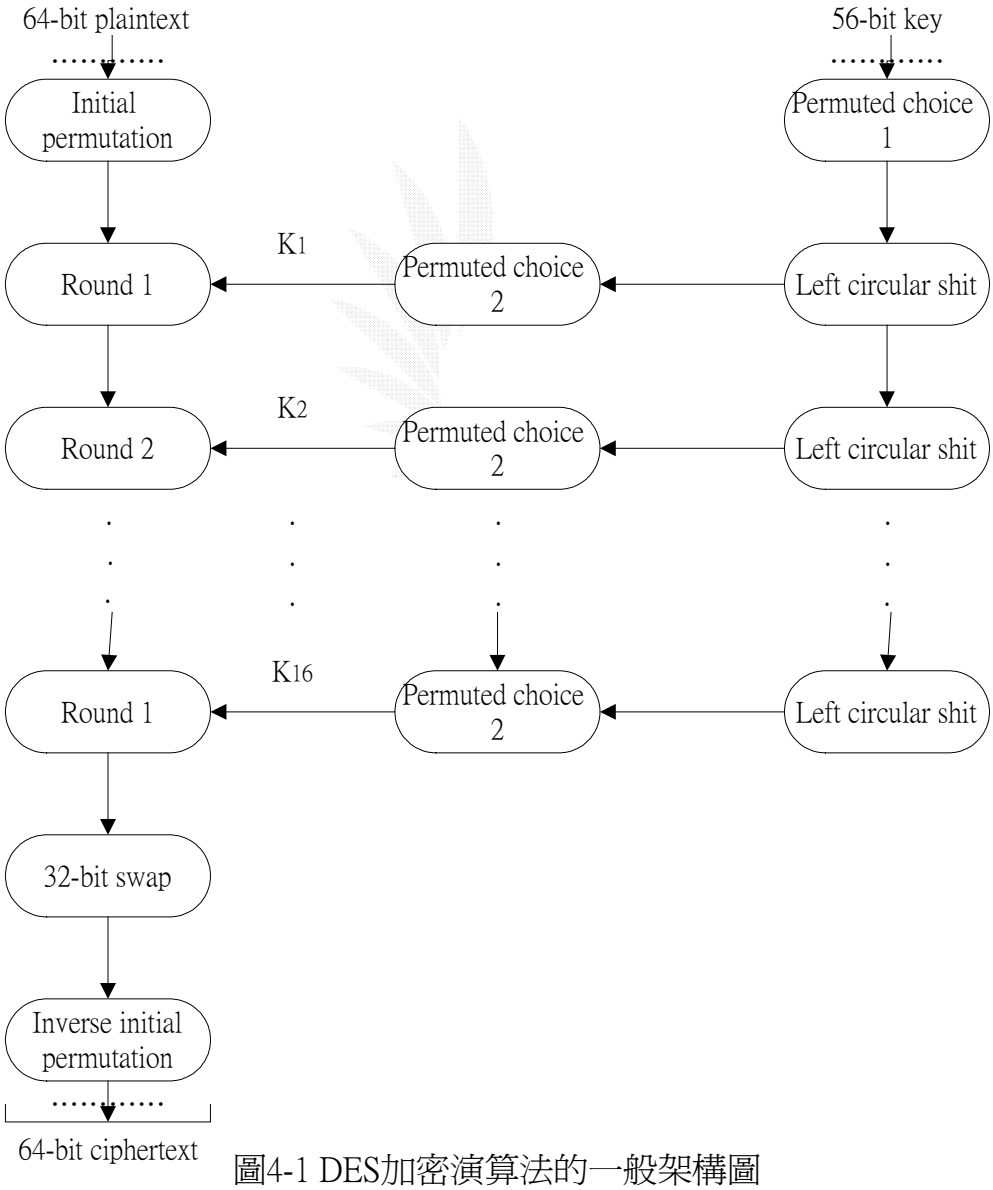


圖4-1 DES加密演算法的一般架構圖

4.2.1 起始重排程序(Initial Permutation)

起始重排與其逆運算是由表格來定義的：

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

為了確定這兩個程序的確是互為逆運算讓我們來考慮底下的64位元輸入M(依序為M_1~M_{64})，此處的每個M_i都表示一個二進位數字，則其重排的結果X=IP(M)就會如下所示：

M_{58}	M_{50}	M_{42}	M_{34}	M_{26}	M_{18}	M_{10}	M_2
M_{60}	M_{52}	M_{44}	M_{36}	M_{28}	M_{20}	M_{12}	M_4
M_{62}	M_{54}	M_{46}	M_{38}	M_{30}	M_{22}	M_{14}	M_6
M_{64}	M_{56}	M_{48}	M_{40}	M_{32}	M_{24}	M_{16}	M_8
M_{57}	M_{49}	M_{41}	M_{33}	M_{25}	M_{17}	M_9	M_1
M_{59}	M_{51}	M_{43}	M_{35}	M_{27}	M_{19}	M_{11}	M_3
M_{61}	M_{53}	M_{45}	M_{37}	M_{29}	M_{21}	M_{13}	M_5
M_{63}	M_{55}	M_{47}	M_{39}	M_{31}	M_{23}	M_{15}	M_7

所以我們將這個結果導入其逆運算，我們會得到：

$$Y = IP^{-1}(X) = IP^{-1}(IP(M))$$

很明顯的我們發現所有位元又會還原成原來的順序。

4.2.2 單一回合的運作細節

圖 4-2 顯示了單一回合的內部架構。同樣地我們先來看圖的左邊，我們將 64 位元的中間形式的左右兩半視為兩個獨立的 32 位元資料，並且分別將它們標示為 L(左)與 R(右)，每回合中的處理程序都可以用下列的方程式來表示：

$$L_n = R_{n-1}$$
$$R_n = L_{n-1} \oplus f(R_{n-1}, K_n)$$

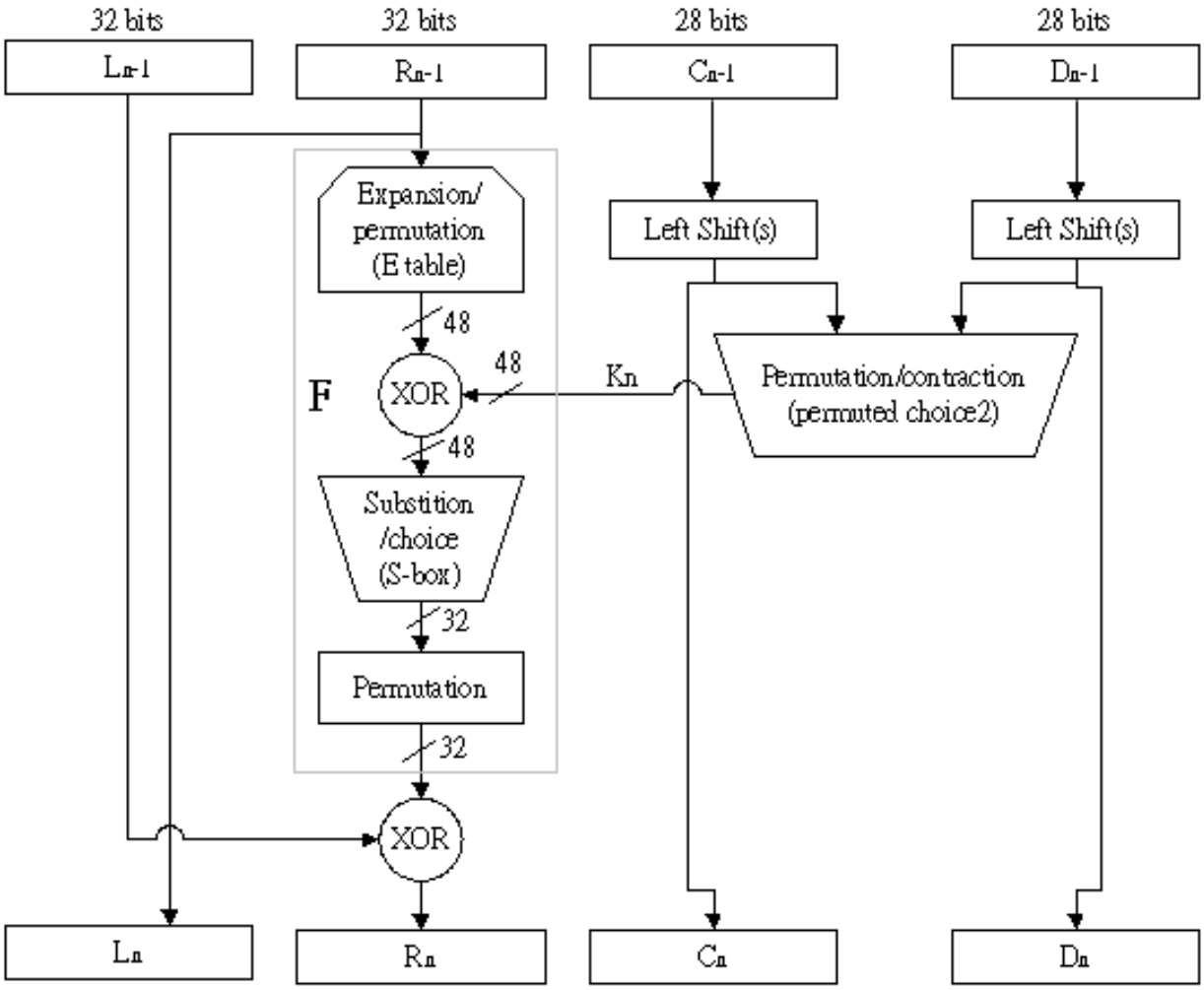


圖 4-2 DES 演算法中的單一回合流程圖

每回合中所使用的子鑰匙 Kn 是 48 位元，R 的大小是 32 位元，R 會先被擴充為 48 位元而擴充的方式是透過下面的表格來定義：

E BIT-SELECTION TABLE

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

這個表格定義了一個重排運算與一個擴充運算；這個擴充運算會複製 R 中的某 16 個位元，接下來所產生的 48 位元會被拿來產生 32 位元的輸出，最後這 32 位元的資料會依照下表所定義的方式重新排列。

重排函數(P)

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

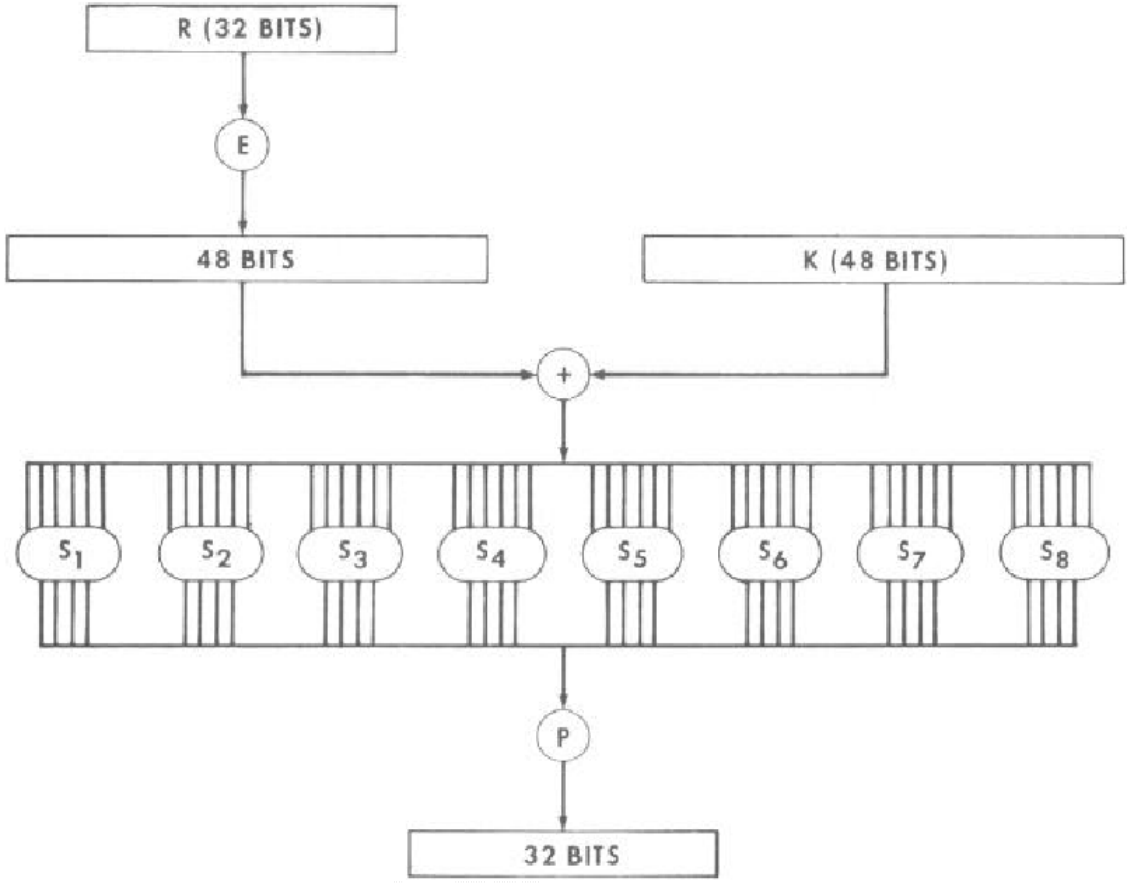


圖 4-3 DES 每回合 S-box 的流程圖

圖 4-3 所顯示的是在函數 F 中 S-box 所扮演的角色，整各取代運算是由八個 S-box 組成，每個 S-box 會接受六位元的輸入而產生四位元的輸出。下列表格定義了這八個 S-box 的運作方式：

S_1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

它們的操作方式如下：我們將 Sn 這個 box 的輸入的第一個和最後一個位元組成一個 2 位元的二進位數值，我們用這個數值來決定要選取 Sn 的表格中哪一列，中間的四個位元所形成的數值就被用來選取表格中的某一行。由此行列所選出的十進位數值就會被表示成 4 位元的二進位形式，這就是 Sn 的輸出。例如：就 S_1 而言若輸入為 011001，則被選取的列就是 01(第一列)，而被選取的行就是 1100(第 12 行)，位於第一列第十二行的數值是 9，因此 S_1 的輸出就是 1001。

S-box 的每一列都定義了一個可逆的取代法則，其取代法則的對照表如下：

明文	密文	明文	密文
0000	1110	0000	1110
0001	0100	0001	0011
0010	1101	0010	0100
0011	0001	0011	1000
0100	0010	0100	0001
0101	1111	0101	1100
0110	1011	0110	1010
0111	1000	0111	1111
1000	0011	1000	1101
1001	1010	1010	1001
1010	0110	1011	0110
1011	1100	1011	0110
1100	0101	1100	1011
1101	1001	1101	0010
1110	0000	1110	0000
1111	0111	1111	0101

4.2.3 鑰匙的產生程序

我們由圖 4-1 和 4-2 可以看到 56 位元的鑰匙會先傳給一個重排程序，這個重排程序的行為是由一個名為「重排選擇一(PC1)」的表格所決定：

***PC*-1**

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

接下來產生出來的 56 位元鑰匙會被分為兩段 28 位元的資料，我們分別將它們標示為 C_0 和 D_0，在每一回合中 Cn-1 及 Dn-1 都會分別被循環左移(或左旋轉)一個或兩各位元。至於到底是移動一個或兩個位元就由下表決定：

左移的清單

回合編號	1	2	3	4	5	7	8	9	10	11	12	13	14	15	16
旋轉的位元數	1	1	2	2	2	2	2	1	2	2	2	2	2	2	1

被移動過的值就成為下一回合的輸入，這些被移動過的值同也會是「重排選擇二(PC2)」的輸入：

PC-2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

重排選擇二會產生 48 位元的輸出，而這個輸出是函數 $F(R_{n-1}, K_n)$的輸入。

4.3 DES 的強度

自從 DES 成為標準之後，其安全層級就受到持續不斷的關切，最重要的質疑有兩點:鑰匙長度以及演算法本身的特性。

如果鑰匙的長度是 56 位元的話，那麼可能的鑰匙就有 2^{56}把，也就是大約 7.2×10^{16}那麼多把，所以表面上看來暴力攻擊似乎是不可行的，但是近年來因為電腦硬體的進步和分散式叢集系統的出現，使的暴力攻擊的可能性被提早被實現，有名的例子是 RSA 實驗室在 1997 年 1 月 29 日公佈一項懸賞找出 DES 鑰匙的題目，一位名為 Rocke Verser 的獨立顧問寫了一個暴力法程式並將此程式放到網路上，讓有興趣的

人利用自己的電腦系統加入此一行動，當有任何一台電腦加入時程式會分配一部份計算給該電腦計算並在計算完成後將結果傳回主機，這項行動由 1997 年 2 月 18 日展開，歷經 96 天之後成功的找到鑰匙，這項挑戰證明了利用分散的個人電腦來破解困難的密碼問題之威力。

當然鑰匙的破解方式不光是測試所有鑰匙這麼簡單，除非我們已經知道明文是什麼，要不然破解者必須能夠正確的辨認出明文，若是明文訊息在加密之前就經過壓縮那麼難度會更高。

另外一個考量的重點是破解者根據 DES 本身的特性來破解有多少可能性？我們關切的重點在於每回合中所使用的八個取代表格(S-box)，因為 S-box 的設計策略(其實就是整個演算法的核心)並不公開，所以我們就會懷疑在 S-box 的弱點被得知的情況下，系統有可能被對手破解，這種假設的確令人不安，但是除了近幾年來的確發現 S-box 的一些規律性以及不尋常的行為之外，目前為止倒是還沒有人發現 S-box 有致命的弱點(至少沒有人公佈)。

4.3.1 DES 的崩塌效應

所有的加密演算法都必須具備下列這個特性：明文或鑰匙的細微變動會造成密文的重大變化，尤其是改變明文或鑰匙的一個位元應該要影響密文的好幾個位元才對，如果影響很小的話那麼破解明文或鑰匙的範圍可能就回縮小了。

DES 在這崩塌效應上面當然也不例外，DES 具有很強的崩塌校應，我們可以在下面的實驗看出來：

實驗(a)改變明文		實驗(b)改變鑰匙	
回合	不同的位元數	回合	不同的位元數
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	28
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

實驗(a)中我們使用了兩段僅相差一個位元的明文：

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

所用的鑰匙是：

0000001 1001011 0100100 1100010 0011100 0011000 0011100 0110010

我們發現才經過三個回合而已這兩個區段就已經有 21 個位元不一樣

了，在整個程序結束之後這兩段密文總共有 34 個位元是不一樣的。

另外實驗(b)只改變鑰匙的測試，這次的明文是：

01101000 10000101 00101111 01111010 00010011 01110110 11101011 10100100

使用兩把只差一位元的鑰匙：

0000001 1001011 0100100 1100010 0011100 0011000 0011100 0110010

1000001 1001011 0100100 1100010 0011100 0011000 0011100 0110010

同樣地，根據結果顯示在最後的密文中大約有一半左右的位元會不一樣，並且崩塌效應在前幾回合就已經出現了。

4.4 Triple-DES 和 DES-X

因為單一次的 DES 加密安全性讓人擔心受到暴力法攻擊，所以其後衍生出 Triple-DES 和 DESX 等數種以 DES 為基礎變化的加密方式。

4.4.1 Triple-DES

Triple-DES 以字面上的意思來說就是三倍的 DES，實際上就是將明文經過三次 DES 加密的意思，不過實際上的做法是利用兩把 56 位元的鑰匙作加密動作，首先將明碼經過和第一把鑰匙作 DES 加密，接著再將得到的密文當作第二次 DES 解密的輸入並用第二把鑰匙來作，最後再將得到的結果用第一把鑰匙再作一次 DES 加密動作，其做法的流程可以簡單表示如下：

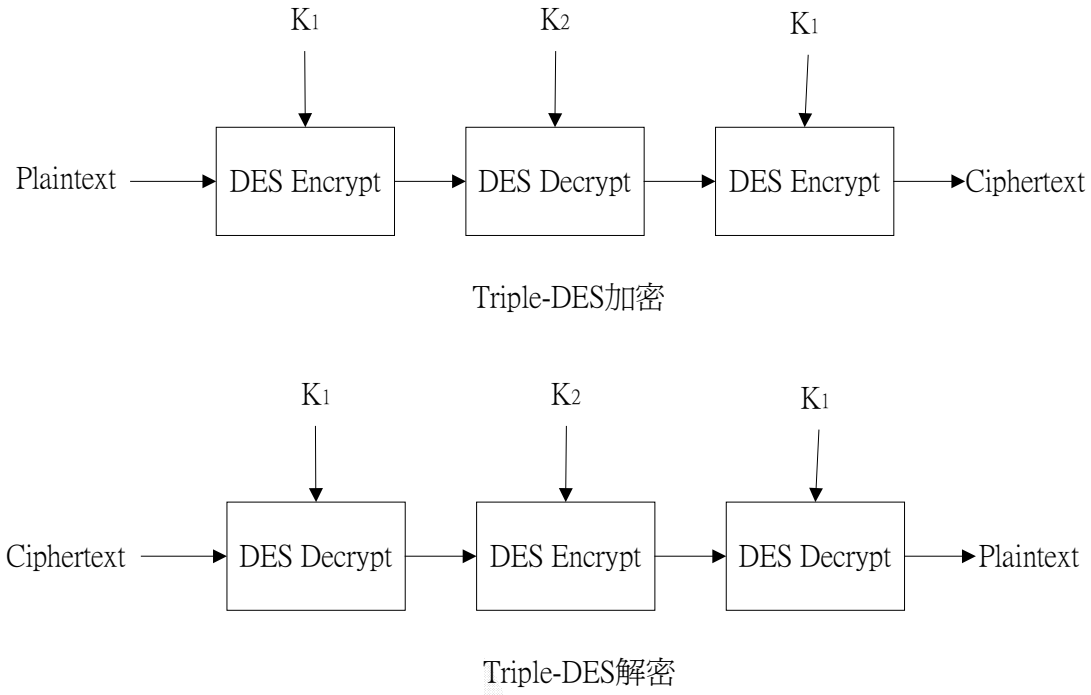
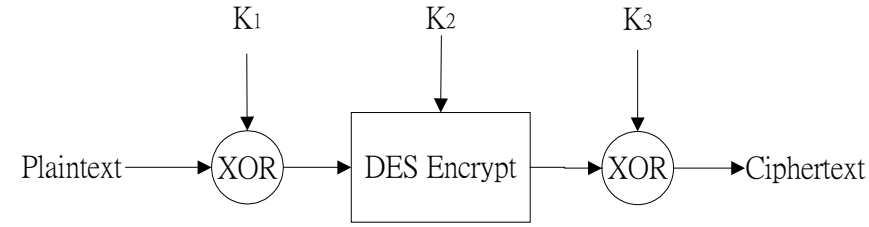


圖 4-4 Triple-DES 加解密流程圖

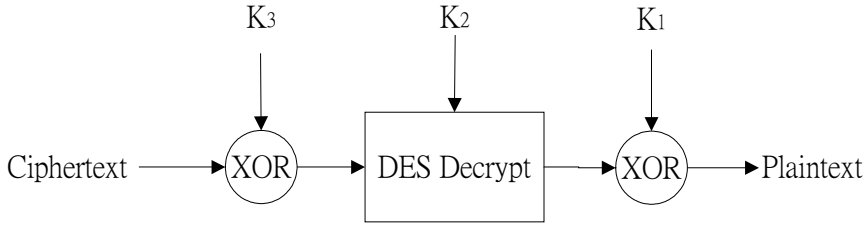
就 DES 而言，這個架構需要長度為 56 X 2 = 112 位元的鑰匙，這樣一來會大幅增加密碼學上的強度，但是因為做了三次 DES 運算相對的加密的效能和速度也大打折扣。

4.4.2 DES-X

因為 Triple-DES 雖然強度上比單次的 DES 好上許多，但是效能和速度卻也慢上許多，為了改善此一缺點並且不使強度下降 DES-X 就這樣出現，DES-X 雖然使用三把 56 位元的鑰匙也和 Triple-DES 一樣經過三個步驟，但是有兩個步驟卻是採用簡單的 XOR 運算取代 DES 加密步驟，其做法如下所示：



DES-X加密



DES-X解密

圖 4-5 DES-X 加解密流程圖

此一方式主要是利用 P⊕A⊕A = P 的特性在 DES 加解密前後多做亂碼之步驟，經過許多國內外實驗室測試 DES-X 不僅加密速度和單次的 DES 幾乎相同，強度方面也和 Triple-DES 不相上下，這也是我們採用 DES-X 來作加密的原因所在。

4.5 DES 的操作模式

DES 演算法是達成資料安全的一個建構基石，為了將 DES 結合在應用程式之中，DES 定義了四種「操作模式」(Modes of Operation, FIPS PUB 74，81)，定義這四種操作模式是為了囊括所有可能會用到 DES 的加密應用程式，而這些模式同樣也可以套用在所有對稱式區段加密的演算法上面，不侷限於只有 DES 才能使用，因為對稱式區段加密的演算法可以看成是黑盒子我們只是拿來用而已採用什麼方法並無影響。

4.5.1 Electronic Codebook Mode(ECB 模式)

ECB 模式是最簡單的一種操作方式，使用相同的鑰匙對個別的 64 位元明文區段作加密動作，架構如下所示：

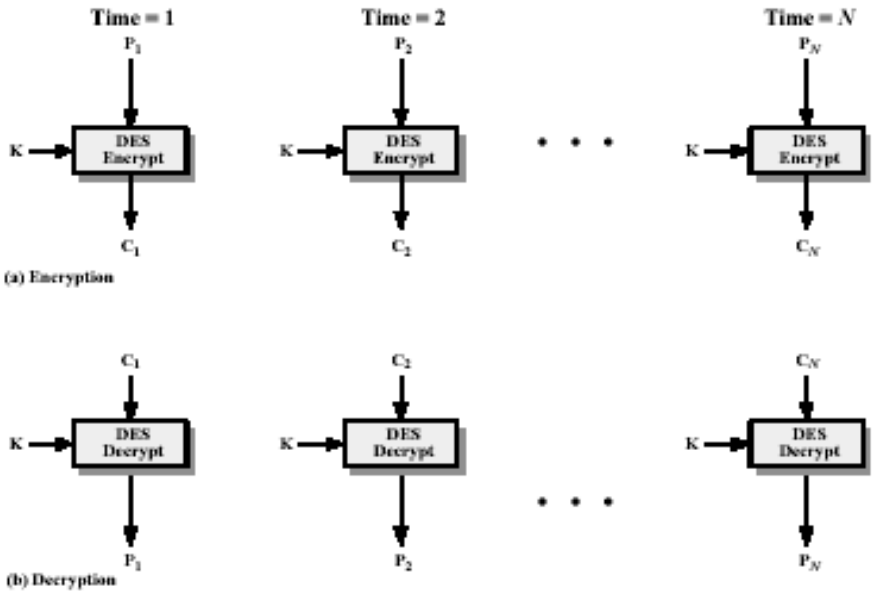


圖 4-6 ECB 模式操作流程圖

通常 ECB 模式都用來傳送單一物質，例如安全地傳送一把加密的鑰匙這樣簡短的訊息。

4.5.2 Cipher Block Chaining Mode(CBC 模式)

CBC 模式處理方式是加密演算法的輸入是由前一個密文區段的 64 位元與下一個明文區段的 64 位元經過 XOR 運算後所組成，其主要是改良 ECB 模式在同樣內容的明文重複出現的情況下，不會產生相同的密文輸出。因為一開始的明文區段並沒有前一個密文區段可以用，所以必須另外準備「起始向量」(Initialization Vector, IV)給第一段明文區段使用，而解密之時也只需要先將第一個密文區段和 IV 作 XOR 運

算之後就可以正常解密出明文，其作法如下所示：

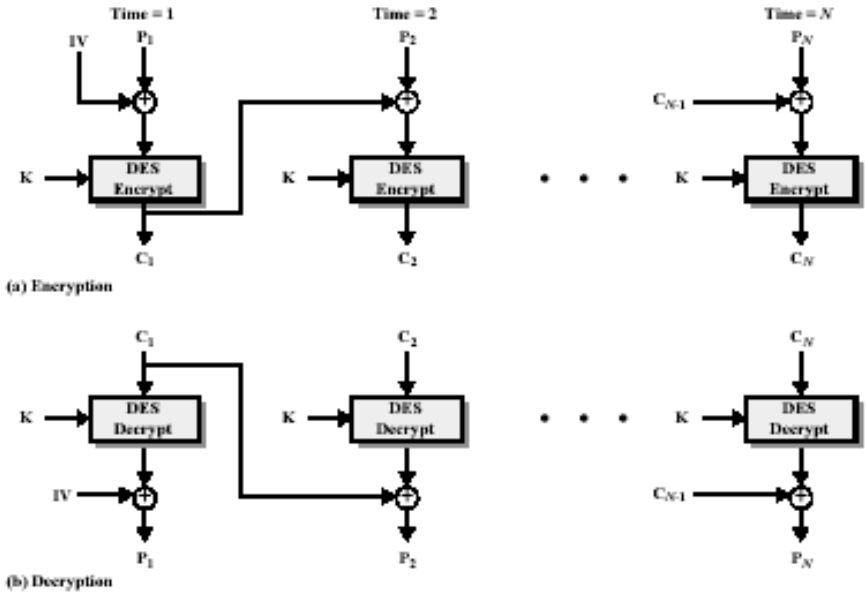


圖 4-7 CBC 模式操作流程圖

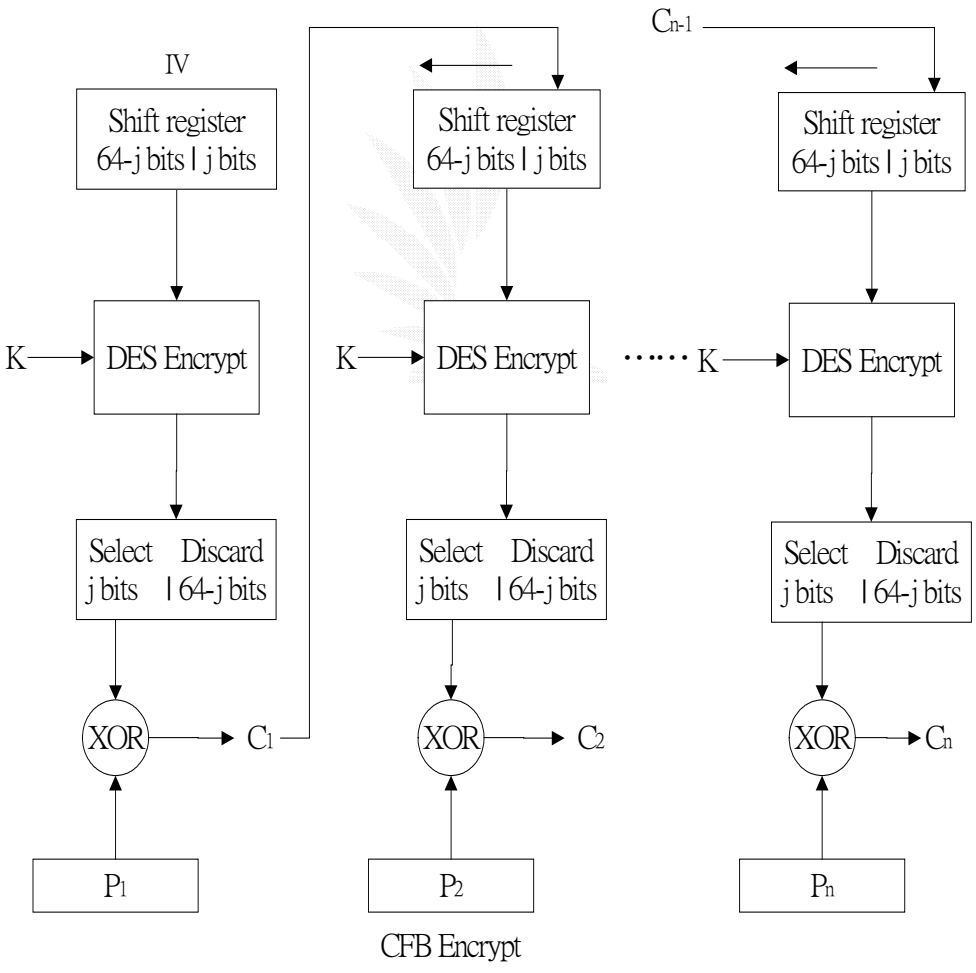
4.5.3 Cipher Feedback Mode(CFB 模式)

DES 或其他對稱式區段加密演算法都是作用在固定長度區段上的加密技巧，然而我們卻可以利用 CFB 模式將 DES 轉換成一個資料流加密法(Stream Cipher)，使用資料流加密法的話就不需要將訊息長度補足成固定區段大小的倍數，可以用即時處理的方式來運作，因此當我們要傳送一連串字元時就可以採用 CFB 模式。

資料流加密法的特質就是密文與明文長度必須相同，因此我們如果要傳送 8 位元的字元資料的話，每個字元就必須用 8 位元來加密，若是超過 8 位元來加密就浪費傳輸容量了。

CFB 模式的操作方法是加密函數輸入是一個位移暫存器(Shift Register)，這個暫存器一開始會被設定成某個起始向量 IV，加密函數

的第一次輸出的最左邊的 j 個位元會與第一個明文單元 P_1 作 XOR 運算，所得到的就是第一個密文單元 C_1，然後我們就將 C_1 傳送出去，接著將位移暫存器左移 j 個位元並將 C_1 放置在暫存器的最右邊的 j 位元內，整個加密程序就以此方式進行下去，直到明文全部被加密完為止；而解密方式也是用相同的架構，不同的是接收到密文單位會與加密函數的輸出 XOR 起來，所得到的結果就是明文單位，其做法如下所示：



CFB Encrypt

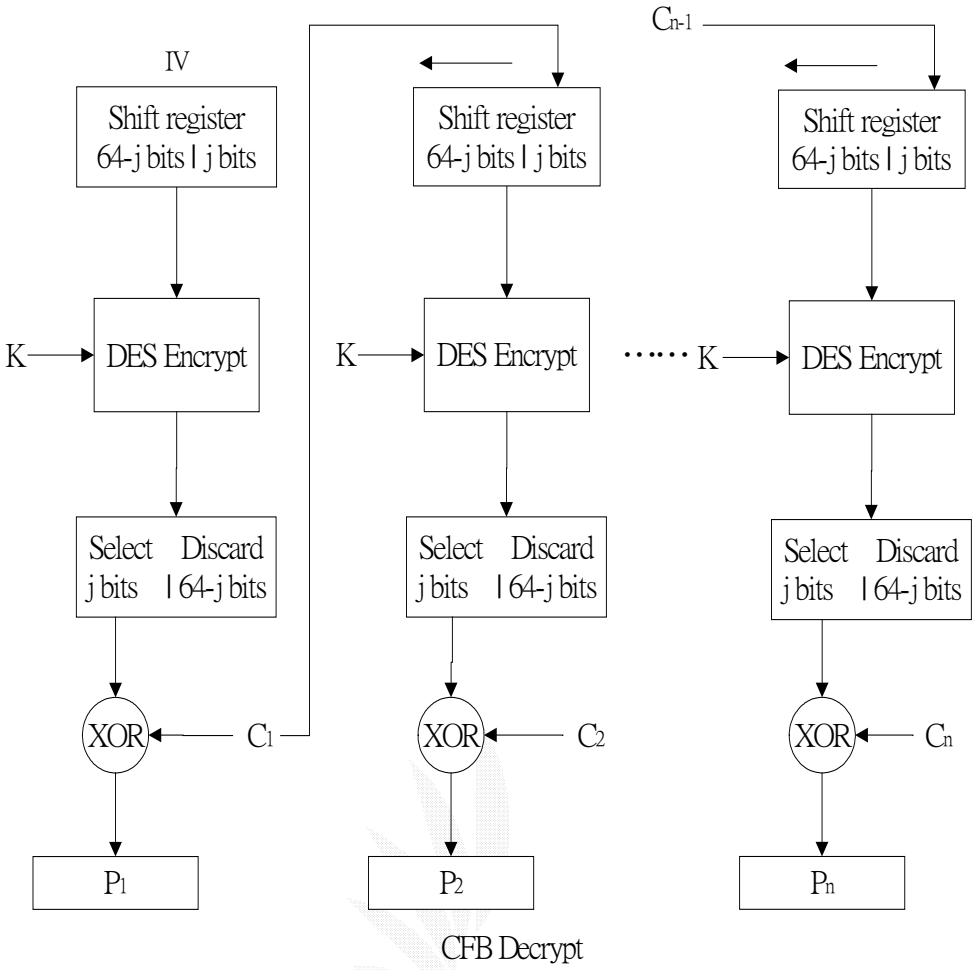


圖 4-8 CFB 模式操作流程圖

4.5.3 Output Feedback Mode(OFB 模式)

OFB 模式的結構與 CFB 模式很類似，在 OFB 中回存到位移暫存器中的是加密函數的輸出，而在 CFB 模式中回存的則是密文單位。

OFB 這個操作方法的優點在於傳輸時所發生的位元錯誤不會繼續擴散下去，舉例來說傳輸中若發生某一密文單位中的某位元錯誤，在 CFB 模式中來說將影響其後所有的密文單位，而 OFB 模式則能控制在發生錯誤的密文單位上，所以此方法常被應用於傳輸中含有大量雜訊或干擾的環境下，其作法如下所示：

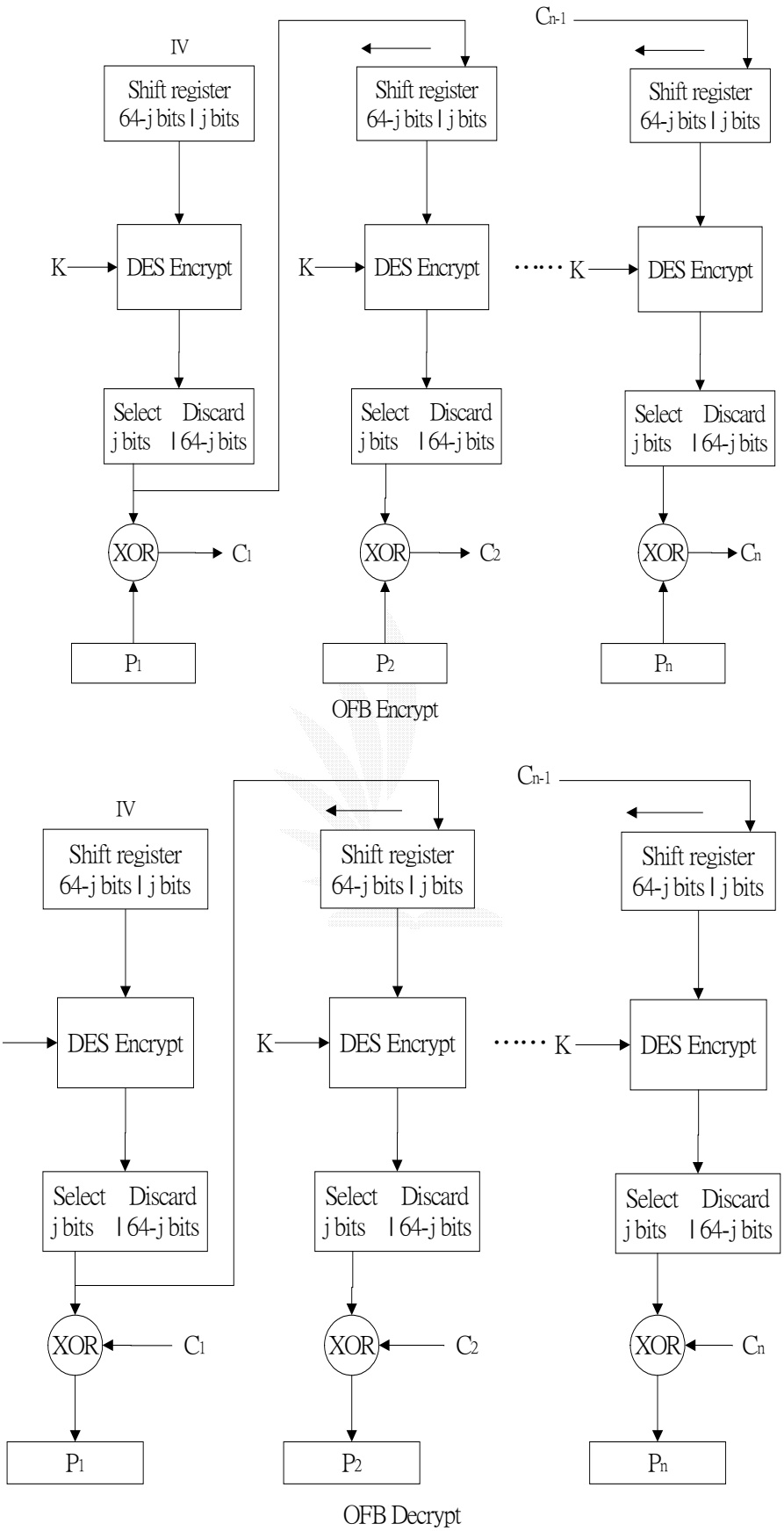


圖 4-9 OFB 模式操作流程圖

第五章 SHA-1 介紹

5.1 何謂 SHA-1

安全雜湊演算法(Secure Hash Algorithm)簡稱 SHA，是由美國國家標準與技術協會(NIST)所發展出來的，並且在 1993 年發布成為第 180 項美國聯邦資訊處理標準(FIPS PUB 180)，而在 1995 年底又將其修訂版本發布為 FIPS PUB 180-1，所以我們通稱此版本為 SHA-1，SHA 是以 MD4(Message Digest Algorithm)為基礎，並且設計方式與 MD4 也很類似。

5.2 SHA-1 原理

SHA-1 演算法所輸入的訊息不長度不能超過 2^{64} 個位元，而輸出則是一個 160 位元的訊息摘要，輸入的訊息會被分成好幾個 512 位元的區段來處理，SHA-1 的處理流程可以由圖 5-1 來了解，其區段長度為 512 位元，雜湊碼長度與串接變數長度都是 160 位元。

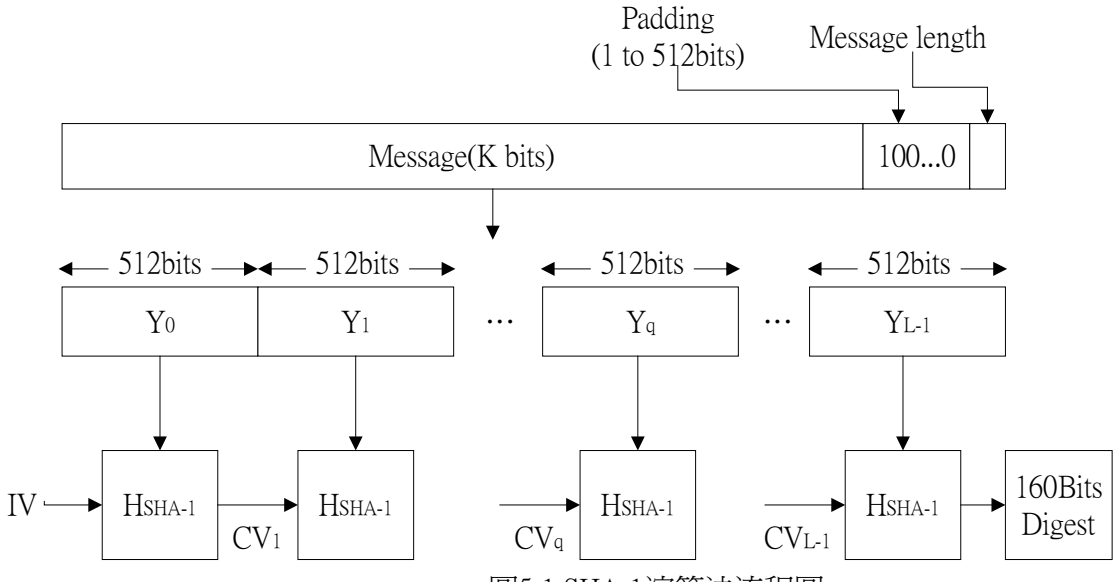


圖5-1 SHA-1演算法流程圖

5.2.1 SHA-1 訊息附加位元(Padding bits)

在訊息之後附加一些位元使訊息取 512 同餘之後等於 448，我們一定要在訊息尾端附加位元即使訊息本身長度就已經符合我們需求，因此我們附加的位元個數可以從 1 到 512 位元，而附加的方法是先加上一個 1 然後再用 0 補到需要的長度，當然這需要在訊息末端加上一段 64 位元的資料,這段資料以一非負整數表示用來記錄原來訊息的長度。

5.2.2 設定 Message Digest 暫存區的初值

我們使用一個 160 位元的暫存區來存放這個雜湊函數的中間值及最後結果，我們可以用 5 個 32 位元暫存器(A、B、C、D、E)來表示這個暫存區，而這 5 個暫存器的起始初值如下(以 16 進位表示)：

A = 67452301

B = EFCDAB89

C = 98BADCFE

D = 10325476

E = C3D2E1F0

5.2.3 處理訊息中 512 位元區段

此為 SHA-1 核心部分,這部分是由四個「處理回合」所組成的模組，每個回合有 20 個步驟，運作邏輯我們可以由圖 5-2 看到，這四個回合的結構都差不多，但是每個回合都用了一個不同的基本邏輯函數，這些邏輯函數在這個規格中分別被標示成 F_1、F_2、F_3及 F_4。

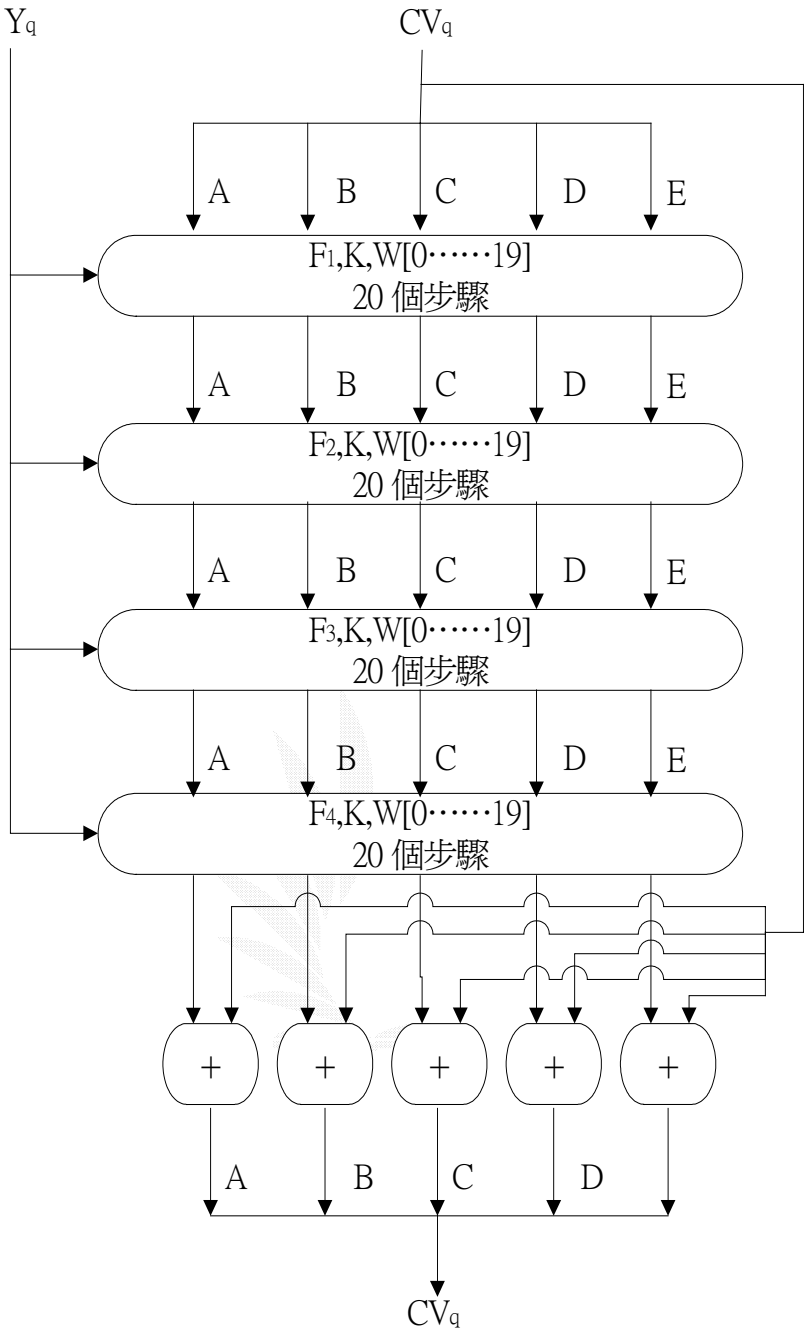


圖5-2 SHA-1處理一個512位元區段流程圖

每回合的輸入是我們正要處理的 512 位元區段(Y_q)與 160 位元的暫存區 ABCDE，這四個回合會分別更動這個暫存區的內容，每個回合還會加上常數 K_t，我們用步驟編號 t 由 0 到 79 來表示四個回合中的 80 個步驟，而實際上我們只用了四個不同的常數如下所示：

$K_t = 5A827999 \quad (0 \le t \le 19)$

$K_t = 6ED9EBA1 \quad (20 \le t \le 39)$

$K_t = 8F1BBCDC \quad (40 \le t \le 59)$

$K_t = CA62C1D6 \quad (60 \le t \le 79)$

第四個回合的輸出(第 80 個步驟的輸入)會跟第一個回合的輸入(CV_q)加在一起，所產生的結果就是 CV_{q+1}，相加的方法是暫存區中的四個字元與 CV_q 中相對應的字元相加(字元間彼此獨立相加)，並且都要取 2^{32} 的同餘就可以得到結果。

5.2.4 SHA-1 輸出

當所有 512 位元的區段都處理過之後，最後一階段產生的輸出就是我們要的 160 位元的訊息摘要值。我們可以將 SHA-1 的行為歸納如下：

$$CV_0 = IV$$

$$CV_{q+1} = SUM_{32}(CV_q, ABCDE_q)$$

$$MD = CV_L$$

此處

IV = ABCDE 暫存區的初始值

$ABCDE_q$ = 訊息中第 q 個區段最後一回合的輸出

L = 訊息中的區段個數(包含以附加的位元及長度欄位)

SUM_{32} = 將兩個輸入區段的字元分別獨立相加之再取 2^{32} 的同餘

MD = 最後的訊息摘要值

5.3 SHA-1 的壓縮函數

仔細來看 SHA-1 中每回合的每個步驟中是如何處理一個 512 位元的區段，每一個步驟都具有下列的形式：

$$A,B,C,D,E \leftarrow (E + F(t,B,C,D) + S^5(A) + W_t + K_t), A, S^{30}(B), C, D$$

每個基本函數都需要三個 32 位元的輸入字元，然後會產生一個 32 位元的輸出字元，每個函數都會執行一組位元對位元的邏輯運算，換言之輸出的第 n 個位元是由三個輸入的第 n 個位元所產生的。這些函數可以歸納如下所示：

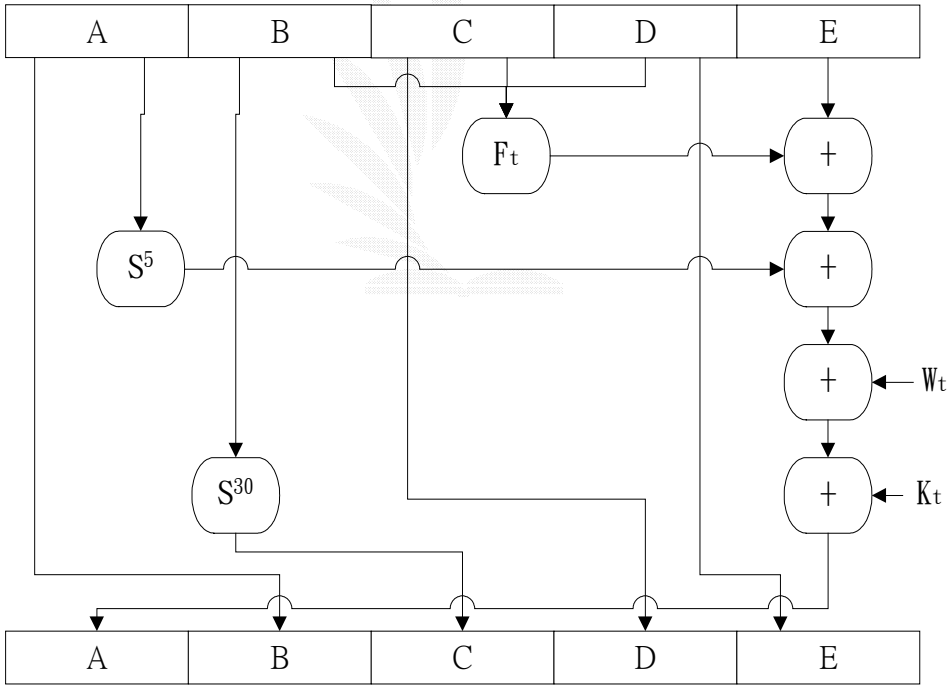


圖 5-3 SHA-1 的基本運算流程圖

5.4 SHA-1 和 MD5 的比較

為什麼我們要採用 SHA-1 而非大家慣用的 MD5，SHA-1 和 MD5 都是從 MD4 衍生而來的，所以兩者之間有許多相似之處。因此他們的強度

和特性也很相似，我們就根據 MD4 所定義的設計目標來比較這兩個演算法。

- **對抗暴力法攻擊的安全程度**：最明顯也是最重要的差別就是 SHA-1 的摘要值要比 MD5 的摘要值多了 32 個位元，如果我們想針對一摘要值來求出訊息的話，MD5 需要執行 2^{128} 個運算，而 SHA-1 則需要執行 2^{160} 個運算。此外如果想要找到兩個可以產生相同摘要值得訊息的話，那 MD5 需要執行約 2^{64} 個運算，而 SHA-1 則需要執行約 2^{80} 個運算，因此就暴力破解法而言 SHA-1 是比較安全的。
- **對抗密碼破解的安全程度**：近年來 MD5 在結構上被人發現是不安全的，而 SHA-1 尚未有發現結構上的問題，加上 SHA-1 的設計策略和 DES 的 S-box 相同不為人知，所以目前看來比 MD5 要安全許多。
- **速度**：這兩個演算法都大量的使用了取 2^{32} 同餘的加法，所以他們在 32 位元處理器上的表現都很好，但是 SHA-1 需要的步驟是 80 步而 MD5 是 64 步，在暫存區的長度上 SHA-1 的 160 位元也比 MD5 的 128 位元要來的多，所以在同樣的電腦上 SHA-1 的執行速度要比 MD5 來的慢。

由上面幾點來看雖然 SHA-1 的執行速度較 MD5 為慢，但是 SHA-1 卻具有較高的安全性強度，加上 MD5 被發現有結構上的缺點，所以我們當然是採用 SHA-1。

第六章 Web Content Recovery System 程式介紹

6.1 安裝程式

此安裝程式主要負責幾項工作：

1. 拷貝 WCR 所有程式到使用者指定目錄。
2. 設定並建立 WCR 系統的設定檔(wcr.ini)。
3. 加入系統登錄檔註冊值，使檔案總管等程式之 POP 選單增加「WCR 檔案加密」之選項。
4. 執行 Mkey.exe 來產生新的 DES 鑰匙並將其存放於指定的位置。
5. 藉由加密程式加密所有需要保護的網頁資料。
6. 建立程式集目錄和程式捷徑。

上述的步驟我們可以由下面的執行畫面 6-1~6-5 來看：

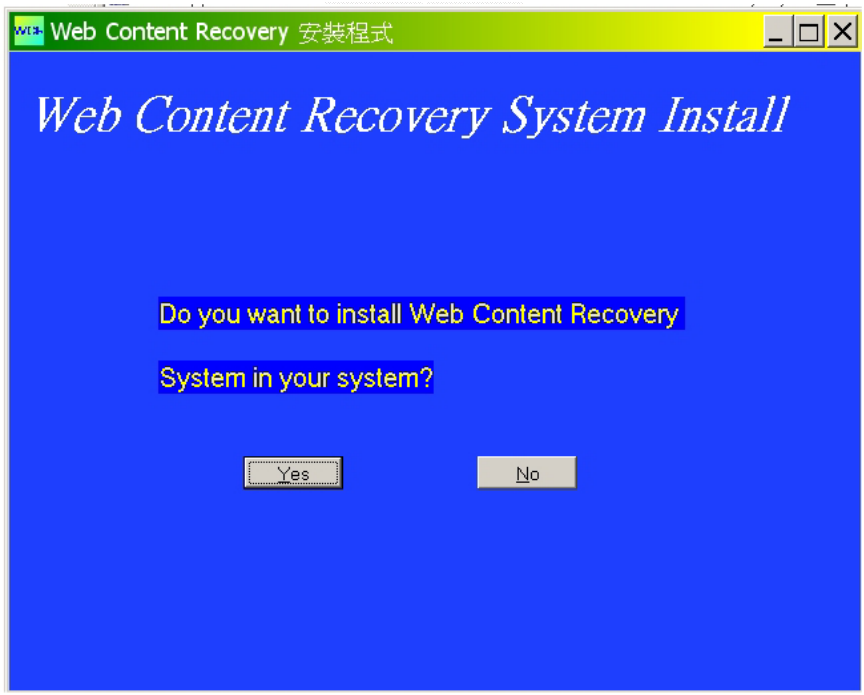


圖 6-1 WCR 安裝程式畫面 1

圖 6-2 WCR 安裝程式畫面 2

圖 6-3 WCR 安裝程式畫面 3

圖 6-4 WCR 安裝程式畫面 4

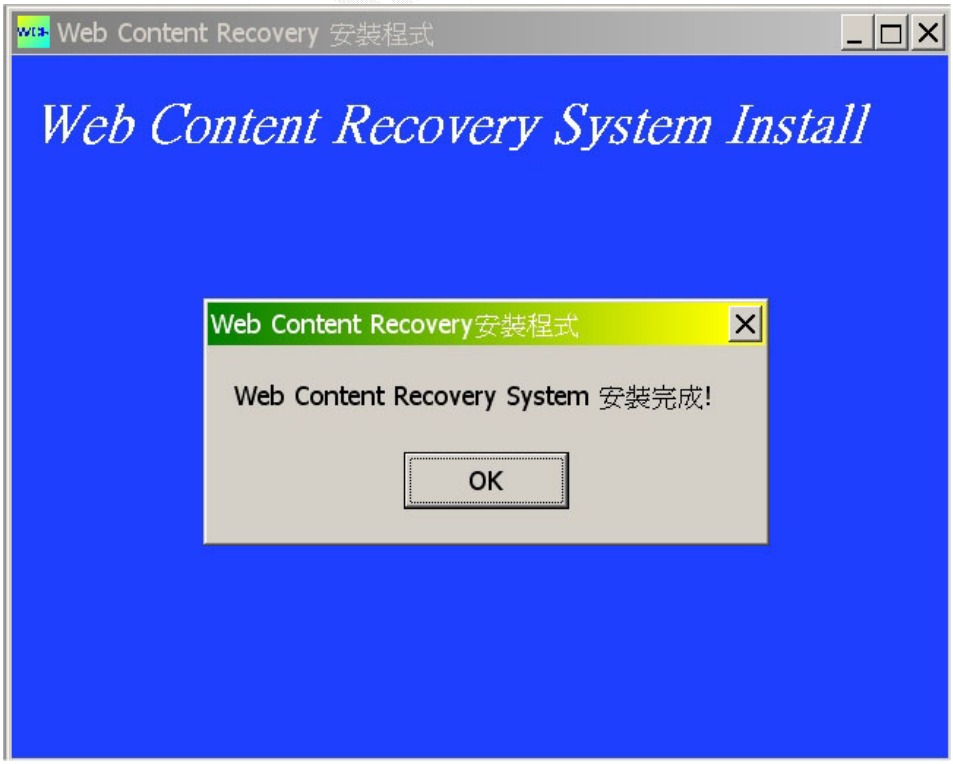


圖 6-5 WCR 安裝程式畫面 5

6.2 加密和備份程式

此程式則負責檔案的加密和加密之後的備份工作，以求發生網頁被竄改之時能有還原網頁之能力，其使用和執行介面是經由 Windows 作業系統上的檔案管理程式，不需另外開啟程式或特殊介面，我們可以由圖 6-6 看到增加到 POP 右鍵選單上的「WCR 檔案加密」選項

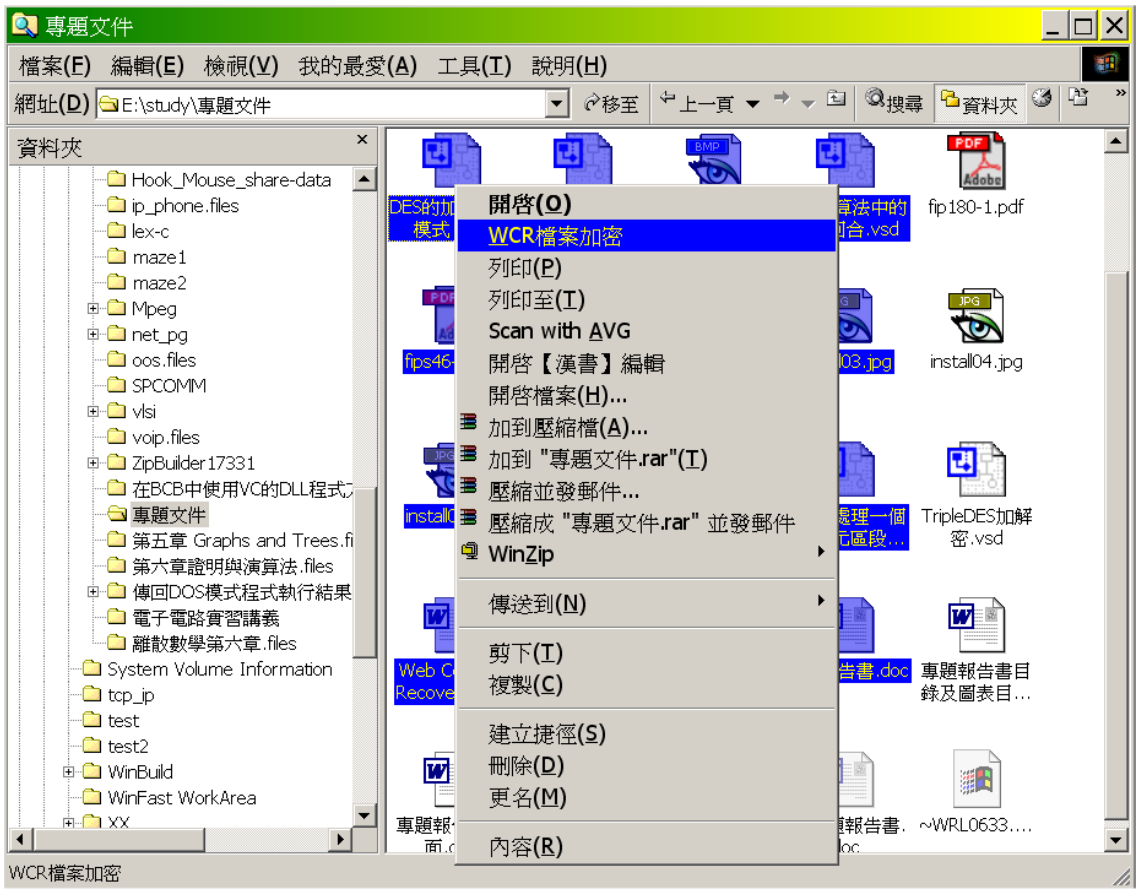


圖 6-6 WCR 加密程式執行畫面

6.3 解密和還原程式

當檔案需要解密或是由備份中移除、還原時則由此程式來做，主要用於網頁檔案 update 或是汰舊，另外因為我們的加密對象是檔案名稱，所以所有檔名皆為無法認出的密文，為了讓使用者能夠方便的找

到檔案並作解密還原的動作，所以我們的解密還原程式就在讀取檔案列表時即時的將檔名作 DES-X 解密動作，讓在 GUI 介面上看到的檔案列表是未加密前的檔名，此做法不僅方便使用者另外一方面在解密檔案時也不需要再作 DES-X 解密動作，其執行畫面如圖 6-7。

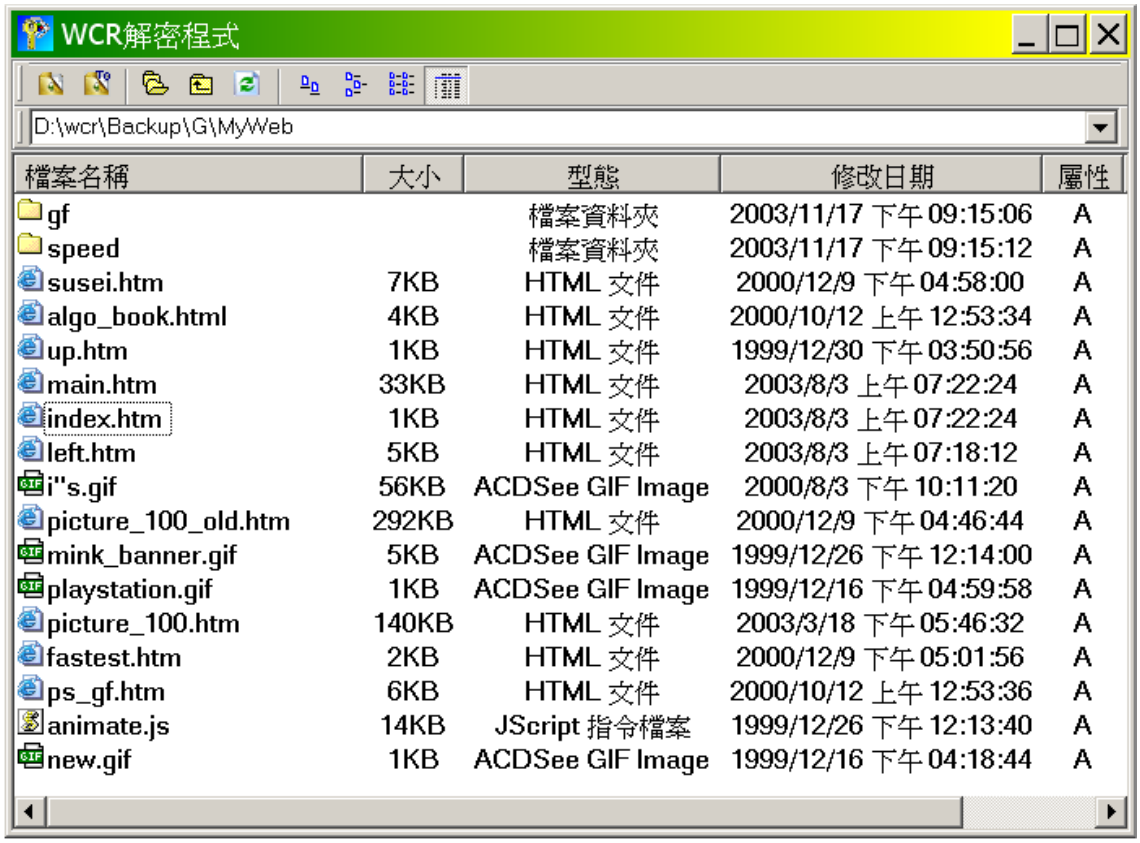


圖 6-7 解密還原程式執行畫面

6.4 WCR 主程式

功能說明

起始執行畫面如圖 6-8，主要可使用的功能有四個：

啟動：啟動 WCR TCP Proxy 的功能，啟動後才會開始執行 WCR 的功能。

重新載入虛擬目錄設定：當使用者在 WCR TCP Proxy 執行後修改虛擬

目錄的設定後，需執行此功能來重新載入設定，方能產生效果。

重新讀取設定：用來重新讀取其他細部的設定。

圖 6-8 WCR 主程式執行畫面

最小化：當點下右上角的最小化鍵後，WCR TCP Proxy 會縮到 task bar 上(如圖 6-9)，在 task bar 上的 icon 連點兩下，WCR TCP Proxy 會回復原來的大小。

圖 6-9 WCR 在最小化在 task bar 畫面

執行結果

在 WCR TCP Proxy 功能啟動後，輸入原始的檔名後，網頁正確的開啟(圖 6-10)，但實際網頁的名稱都已經經過加密(如圖 6-11)，直接到 IIS 所在的 port 來開啟同樣的網頁，結果並無法成功的開啟(如圖 6-12)。

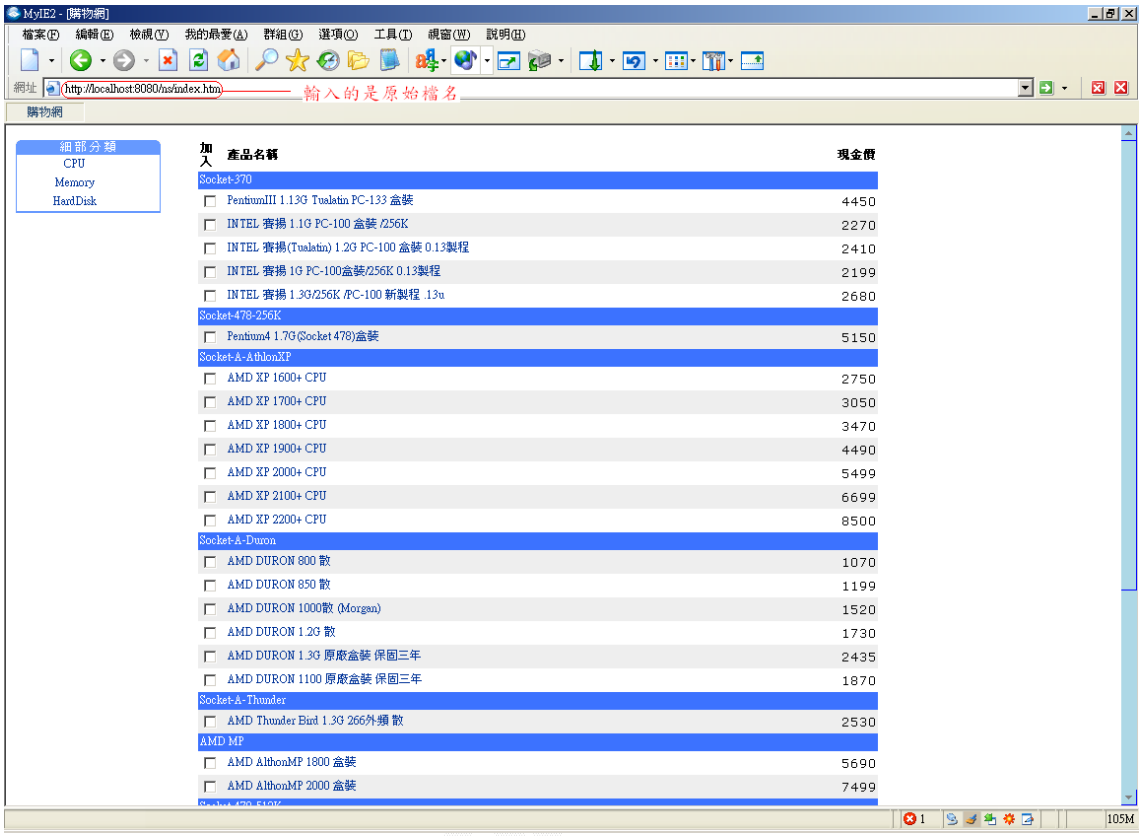


圖 6-10 開啟網頁測試畫面

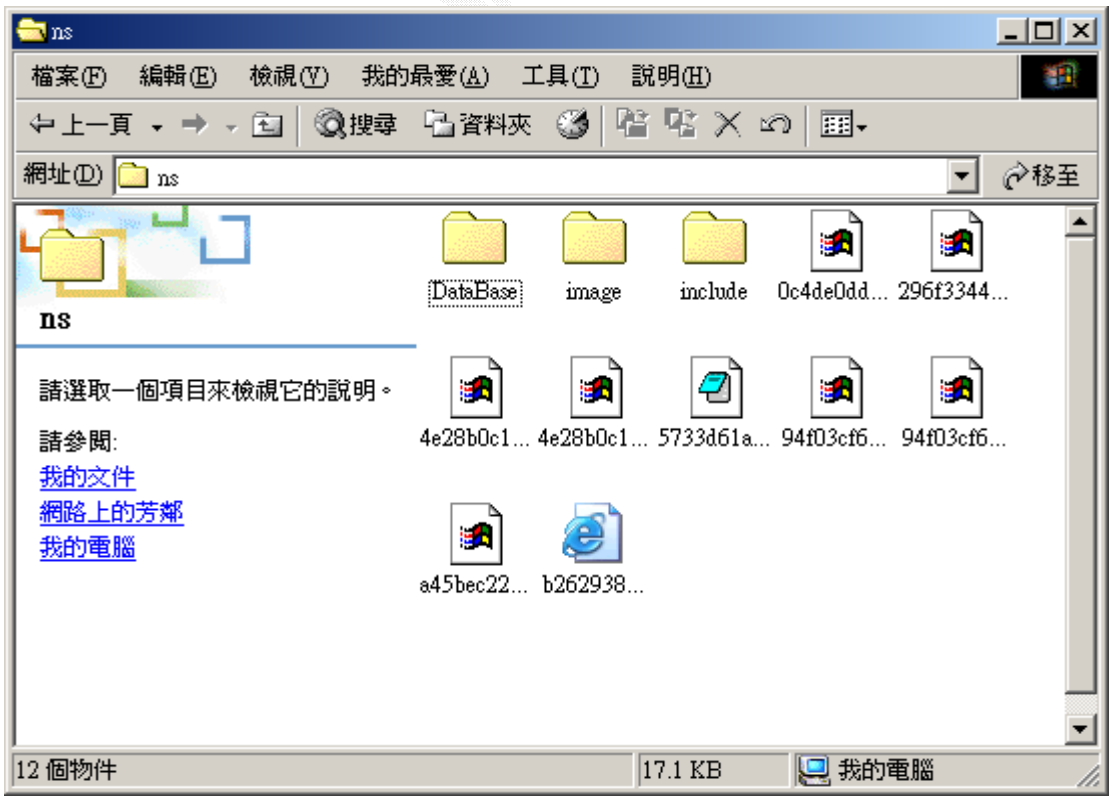


圖 6-11 實際在主機上的檔名畫面

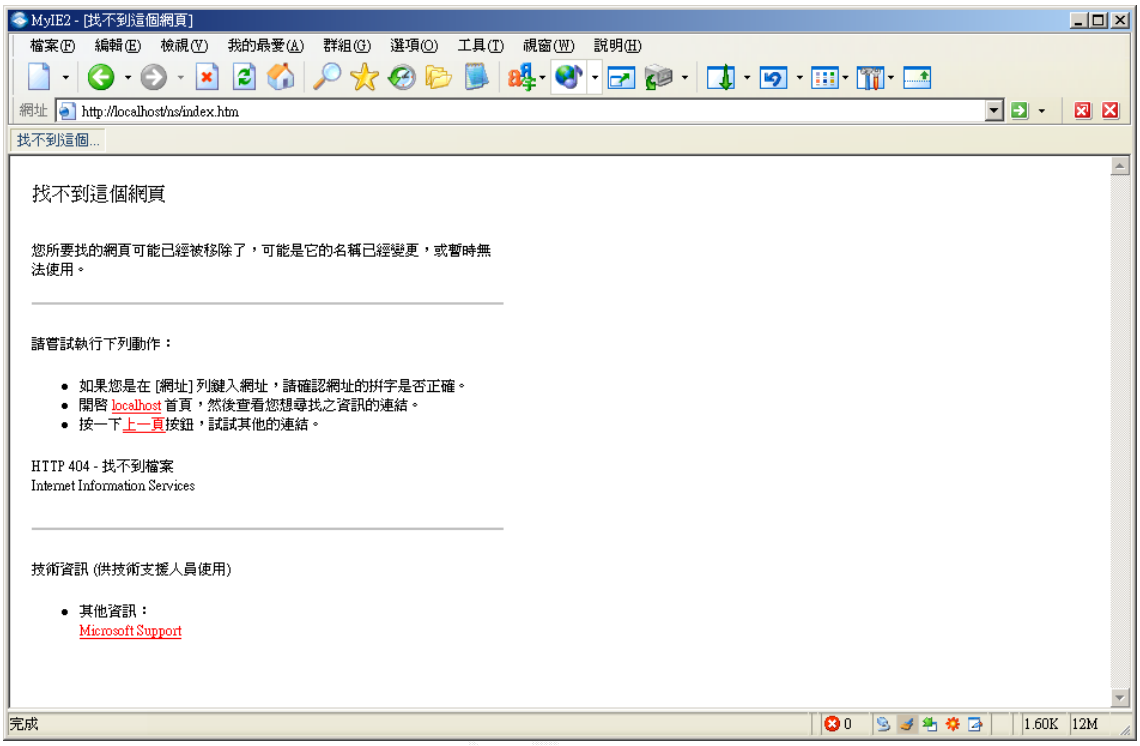


圖 6-12 直接送要求給 IIS 開啟網頁的測試畫面

第七章 WCR 系統架構和問題解決

7.1 WCR 系統架構

整體架構大致分為兩部分：On-line 的主程式和 Off-Line 的加解密及備份還原程式，其簡單的結構可由圖 7-1 來看：

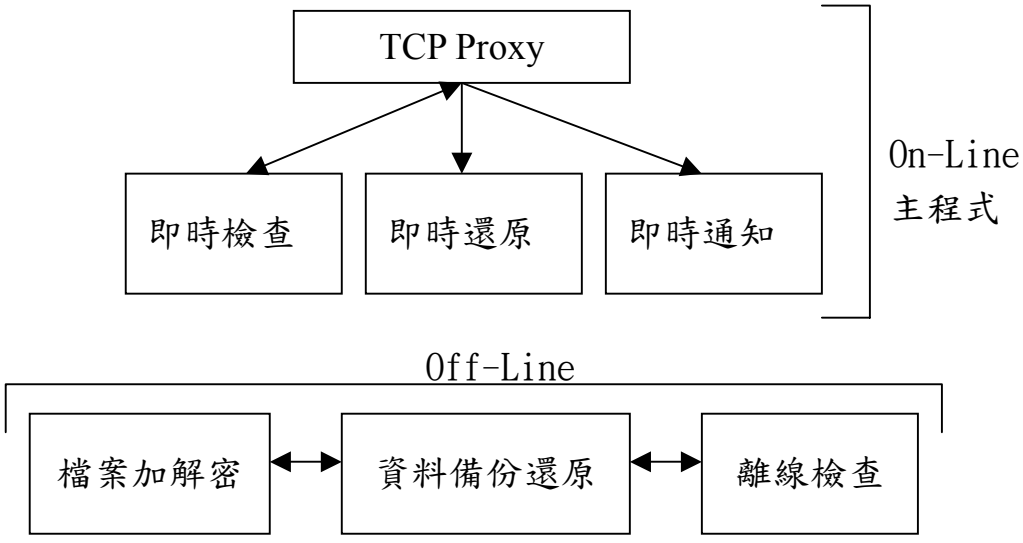


圖 7-1 WCR 系統簡單架構圖

而處理 Client 端瀏覽器 Request 的流程也可以由圖 7-2 清楚了解

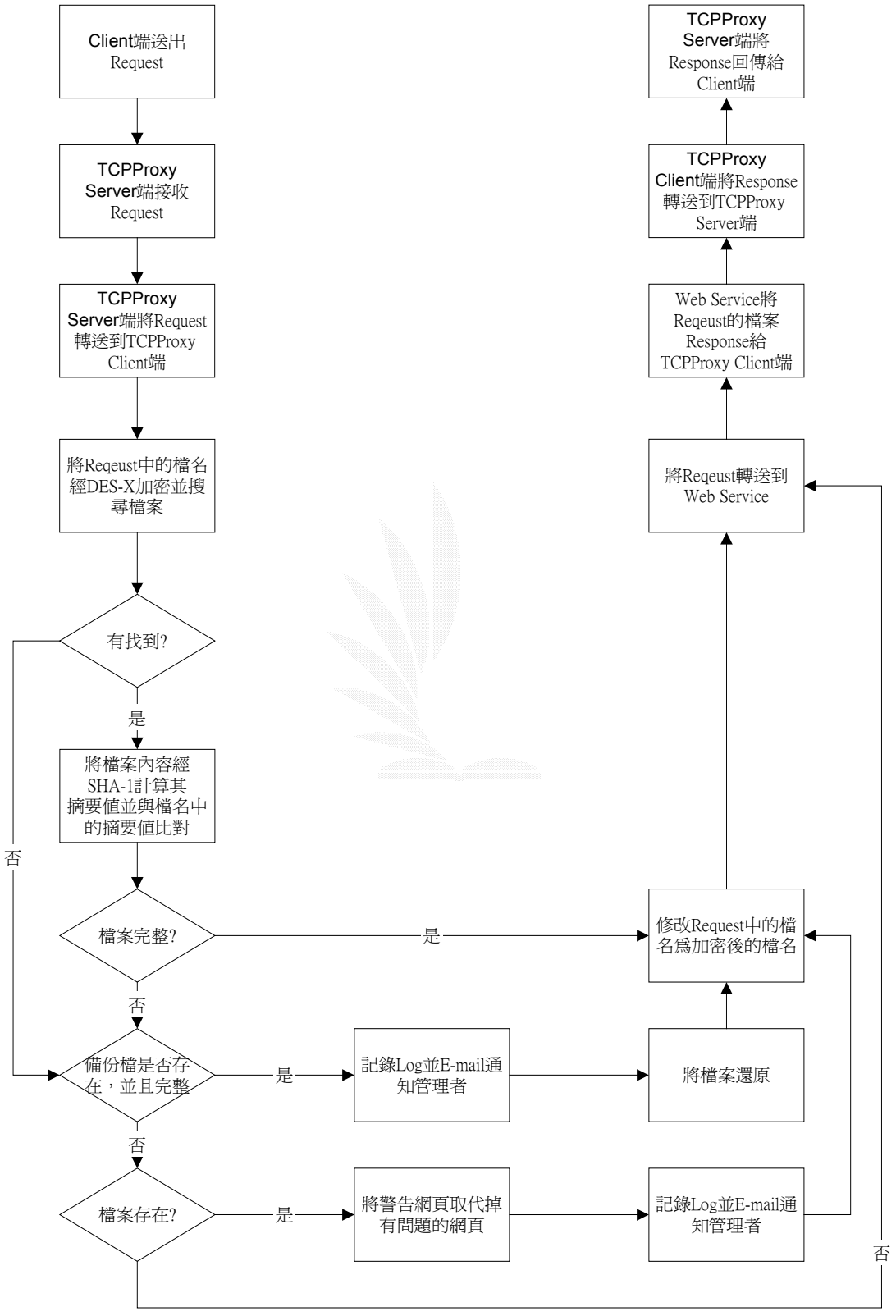


圖 7-2 WCR 系統處理流程圖

TCP Proxy 會 bind 到 80 PORT，然後當 Request 進來時，先將檔名經 DES-X 加密後再搜尋目錄下檔名以此密文開頭的檔案，找到之後再將此檔的內容經過 SHA-1 來計算其摘要值，並與檔名中的摘要值部份比對，以檢查檔案的完整性，如果沒問題再將 Request 中的檔名改為加密後的檔名，再送往 Web Service，收到 Web Service 的 Response 後將傳回給 Client 端；如果檔案有問題便即時將他從備份檔中還原，並記錄 log 再以 E-mail 方式通知管理者，最後都由 Web Service 的程式作 Response 後將傳回給 Client 端。

7.1.1 WCR 加密和備份做法說明

我們使用 DES-X 來加密檔案名稱(不包含路徑)，而以 SHA-1 來計算檔案內容的訊息摘要值，之後將兩者整合變成該加密後的檔案名稱，也就是說我們的加密處理只變動檔名部分，不會對檔案內容作任何動作或處理，至於檔案備份工作則是依照使用者設定的備份路徑將檔案依照原本所在路徑的架構完整備份下來，舉例來說：當備份路徑為 D:\WCR\Backup 而加密過檔名為 E:\e5b7cd8e07106fbda28cee94b167af6fe8d74d8a014e76387d860346.html 則備份檔案為 D:\WCR\Backup\E\e5b7cd8e07106fbda28cee94b167af6fe8d74d8a014e76387d860346.html。我們由圖 7-3 可以了解加密和備份程式處理檔案的流程。

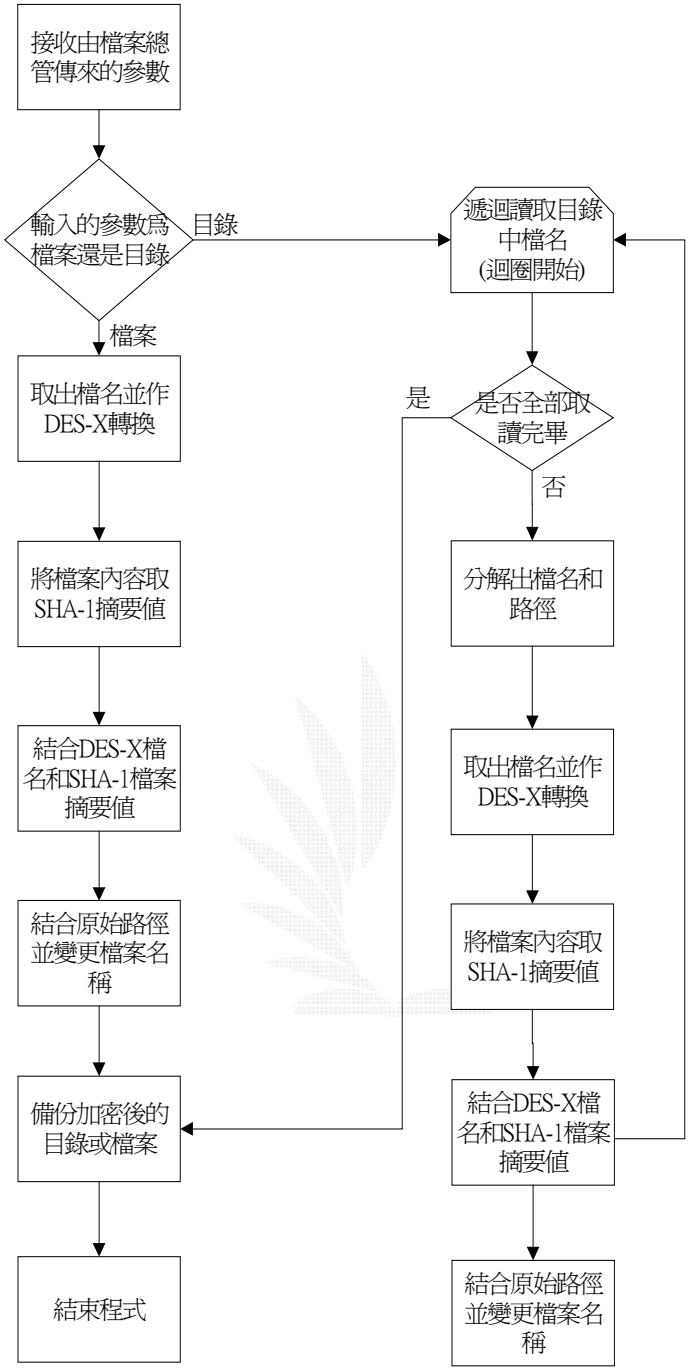


圖 7-3 WCR 加密備份程式流程圖

流程中檔案加密之做法：假設原始檔名為 filename.subname，則我們的加密程式會先將 filename 取出並將其作 DES-X 加密，但因對稱式加密的 DES-X 需要資料輸入皆為 64 位元，為了解決 padding 的問題我

們採用原本用於 RC5 上面的密文盜用模式(Ciphertext Stealing Mode，CTS)，做法如下：

1. FileName 長度小於或等於 8 bytes：

   我們會在檔案名稱後面補上不足 8 bytes 的數字，即若檔名為 a.php 則距離 8 bytes 差 3 bytes，所以檔名在 padding 之後便成 a.php333，所以當檔案名稱不足 8 bytes 時皆會補滿到 8 bytes，當然剛好為 8 bytes 時也會補上 8 個 8，不過在加密之後多補上去的部分會被去除。

2. filename 長度大於等於 8 bytes：

   若是 filename 長度大於 8 bytes 時我們的做法是：前面滿足 8 bytes 的區段部分我們都以 DES-X 的 ECB 模式來操作，而最後兩個區段(含一個未滿 8 bytes 的區段)我們就改採 CTS mode 的處理方式，其做法的流程可以分為幾個步驟：

   A. 使用傳統的 ECB 技巧對前 n-2 個區段作加密。

   B. 將 P_{n-1} 取前面一回合所產生的密文區段 C_{n-2} 作 XOR，所產生的結果為 Y_{n-1}。

   C. 將 Y_{n-1} 加密成 E_{n-1}。

   D. 將 E_{n-1} 的前 L 個位元(L 就是 P_n 的長度)取出成為 C_n。

   E. 在 P_n 的末端補 0，並且與 E_{n-1} 進行 XOR 運算來產生 Y_n。

   F. 將 Y_n 加密成 C_{n-1}

以上的步驟我們可以用下面的流程圖 7-4 表示：

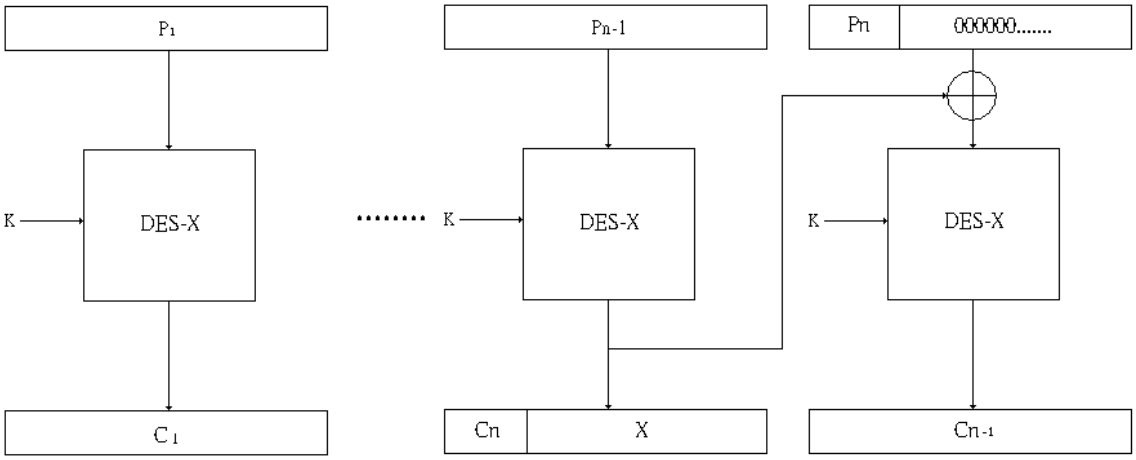
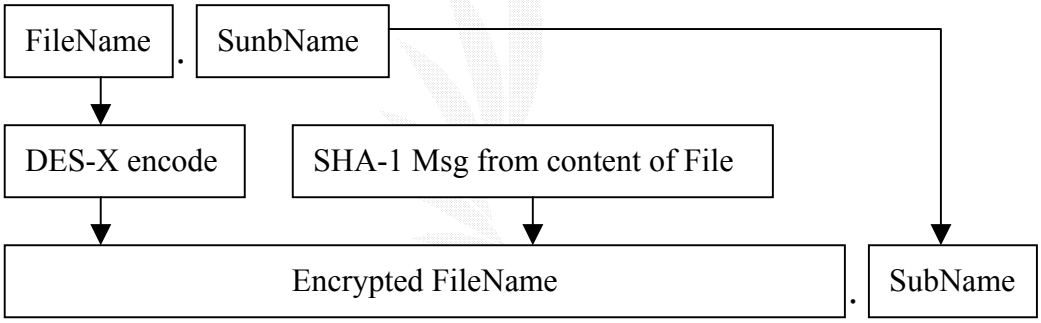


圖 7-4 DES-X 加密使用 CTS mode 操作流程圖

所以整個檔名加密架構就變成：



至於原本的 SubName 則完全保留以求和 Web Service Program 有最大相容性，因為 99%的 Web Service Program 皆以 SubName 判斷此檔案該如何處理。

7.1.2 WCR 解密和還原做法說明

為了達成讓使用者可以看到未加密前檔名並且可以直接選擇檔案或目錄還原，我們除了實作一 GUI 介面讓使用者方便操作之外，更使用 Shell API 來讀取檔案相關資料(檔案型態、屬性、修改日期等等)並在做檔案名稱列表之時就先在背景將檔案名稱作 DES-X 解密處理，

在使用者圈選檔名之後即可直接作解密還原的動作，我們由圖 7-5 可以了解解密和還原程式處理檔案的流程：

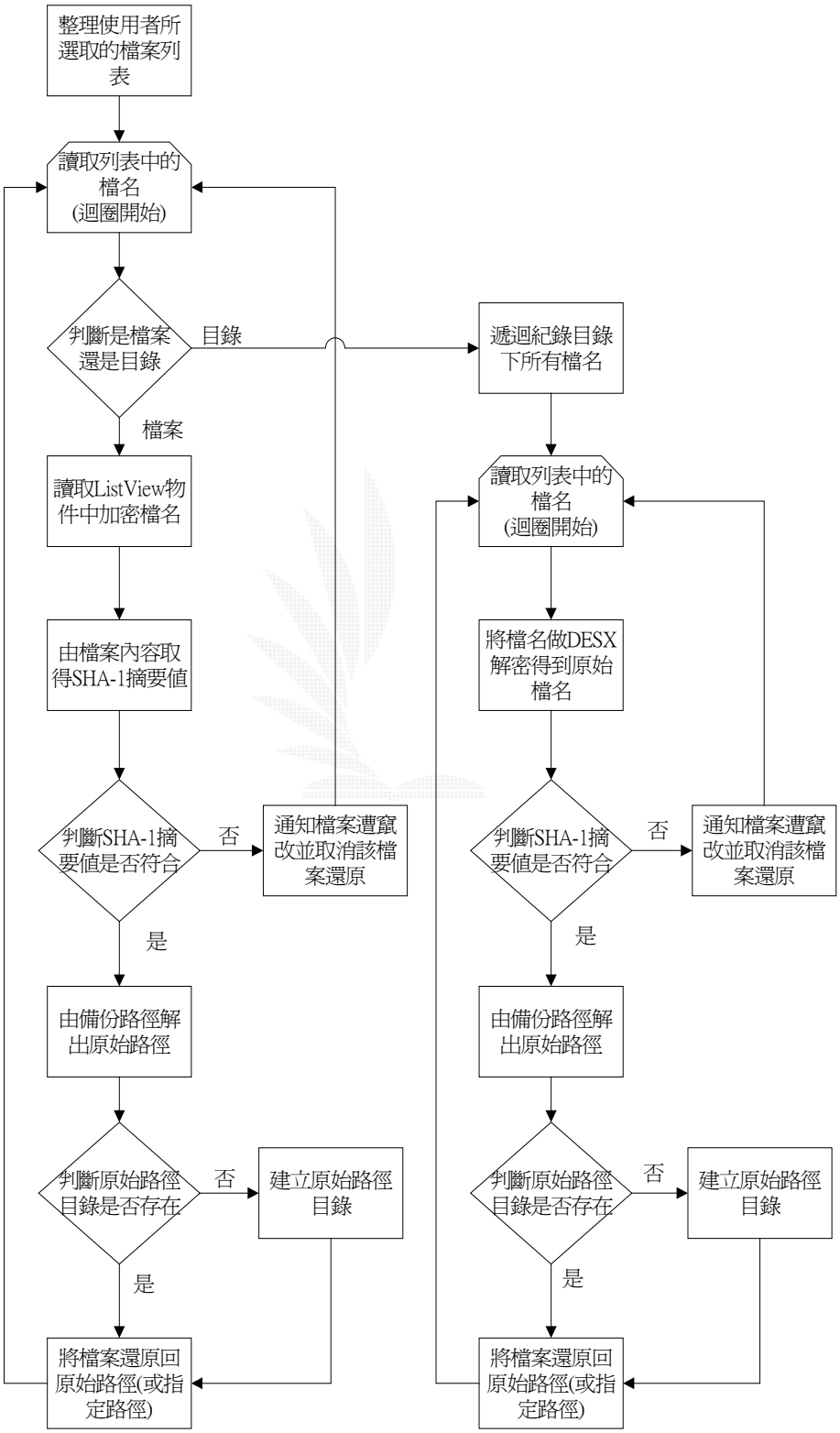


圖 7-5 解密還原程式流程圖

而其 DES-X 解密和 CTS mode 之處理流程如圖 7-6：

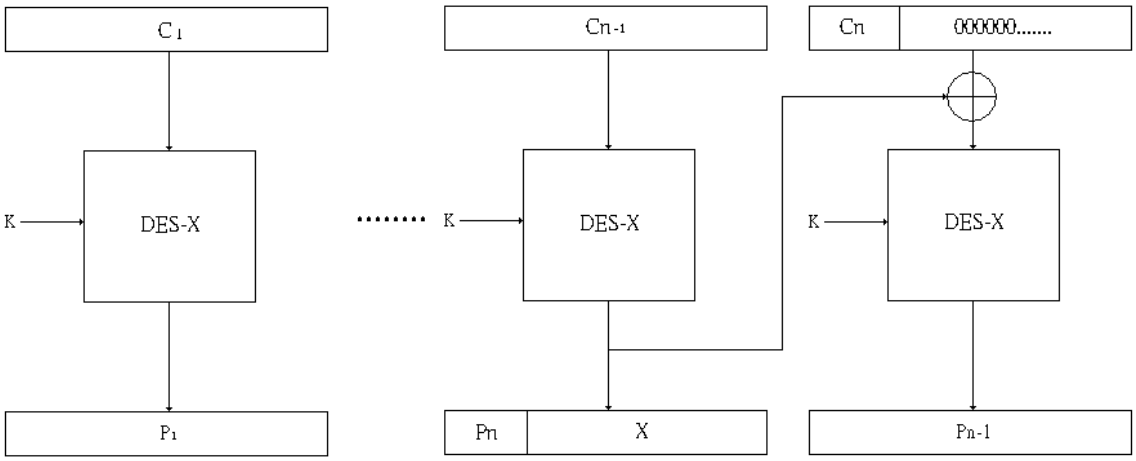


圖 7-6 DES-X 解密使用 CTS mode 操作流程圖

只要將最後兩個區段的加密過程倒過來作即可順利解密獲得明文，其做法的流程可以分為幾個步驟：

A. 使用傳統的 ECB 技巧對前 n-1 個區段作解密。

B. 在 C_n 的末端補 0。

C. 將 C_{n-1} 解密出來的資料和 C_n 作 XOR，所產生的結果為 Y_n。

D. 將 Y_n 解密成 P_{n-1}。

E. 將 C_{n-1} 解密出來的前 L 個位元(L 就是 C_n 的長度)取出成為 P_n。

7.2 問題解決

1. 在傳送資料時因傳送檔案過大，傳送速度過快，導致網路卡的 Buffer 空間不足而造成錯誤

A：解決方法是在傳送失敗時等待一小段時間再重試直到成功為止。

2. 當資料傳送給 Web Service 在資料還沒送完之前，新的資料進來把正在使用的 Buffer 蓋過去造成資料遺失。

A：解決方法是使用 Queue 來儲存 Buffer 使資料不致遺失。

3. 開發解密還原程式時因為需要讀取檔案或是目錄的詳細資料(如：檔案型態、屬性、修改日期、大小等等)，所以使用 Microsoft Platform SDK 中的 Shell API 來實作，結果一直找不到如何將設定讀取目錄資料的 Root 位置方法，只能用其內定的系統目錄當作 Root 來開啟。

A：後來由 Microsoft Platform SDK 的文件中發現其實 Shell API 尚未實作完成該函式，導致只能開啟內定的系統目錄，針對此點我們改採用 OLE 的方式繞過 Shell API，藉由 COM 技術呼叫 Windows 作業系統的 shell32.lib 程式庫直接作設定。

4. 此次使用的 SHA-1 摘要值演算法原本我們再此專題之前就已經有實作過程式，只是當時開發的環境是 C 語言，剛開始我們直接將其改寫並封裝成 C++的 Class 來使用，卻發現有部分 C 的寫法對應到 C++的環境和編譯器會發生許多問題，修改多次仍不見改善。

A：最後我們只能重新針對 C++的環境和編譯器撰寫新的 SHA-1 程式，以求得最大的相同度和較好的效能，和之前程式相比程式碼縮減 15%左右，也沒有任何使用上的問題出現。

第八章 未來發展與心得感想

8.1 未來發展

因為我們 WCR 系統的主程式架構是在 TCP/IP 的應用程式層上面，幾乎所有主程式的實作部分都可以用 Socket 來達成，所以未來主程式可以使用跨平台的 Java 來作為開發環境，如此就可以達到跨平台的程式架構，另外此次僅針對 IIS 來作處理，未來只要能加入各種 Web Service 程式(如:Apache 等)轉換虛擬目錄的對應方法就可以加以處理，而不需侷限只能搭配 IIS 來使用。

在加密保護和摘要值強度部分因為撰寫程式時都使用 C++物件導向的設計理念，並將 DES-X 和 SHA-1 演算法的程式部分封裝成 Class，這樣未來若發現需要更換演算法也只要替換演算法的 Class 即可，不需要大幅度的修改主要程式部分，甚至可以將此部份的程式包裝成動態連結程式庫來呼叫。

而在 DES-X 鑰匙的保護上面此次因為開發的語言和環境無法搭配上現有的 IC 晶片卡來運作(目前實驗室僅有支援 VB、Java 兩種語言的 IC 晶片卡)，未來若能改寫成 Java 版本或是替換新的 IC 晶片卡系統，即可利用 IC 晶片卡不易被複製和竊取內部資料的特性使 DES-X 鑰匙的保護更加安全。

最後在檔案備份的部分目前我們採用的方式是將加密過後的檔案

依照原本目錄的結構完整備份下來，未來除了可採單一壓縮檔案或是使用資料庫來存放備份檔案之外，也可以採用異地備援的方式透過網路來取得備份資料，如此都可以大大增加系統安全性。

8.2 心得感想

個人心得感想：

從三年級下學期不知道專題要做什麼題目，到題目確定甚至四年級剛開學換了題目最後能順利完成，這中間的經歷就非課堂上聽老師授課或是自行研究可以得到的，縱然曾經有過軟體開發的經驗但是面對許多專題真正在 Run 的過程中遇到的困難和其他阻力有時候還是不知如何解決，中間經過指導老師的開導和小組間的討論最後將專題順利的完成，其中學到的觀念和經驗都是個人寶貴的資產，也能成為日後工作或是研究上借鏡。

這次的專題感謝組長給與的大量技術支援，減少了許多的開發時間，大體上在程式架構確定後，程式的寫作便很快的進行，其中遇到的問題大多是寫作上的粗心所造成的錯誤，而非技術上的不足所造成的，雖然技術上都還可以，但還是學到了不少東西，畢竟有些東西是寫下去才會發現的問題，一開始曾經考慮要使用低階的方法來寫，但考量的時間上來不及所以改採用高階的作法，也就是目前所使用的架

構。專題令我最困擾的不是程式的方面而是文件的寫作，因為沒有相關的經驗所以在文件的撰寫上顯的相當生疏。

小組心得感想：

我們的專題原先是”Transfer for Telephone to IP Phone”想藉由嵌入式系統來實作，達到由一般電話轉接到 IP Phone 上面的功能，在開始階段的規劃和程式設定、撰寫都算順利，但是最後卻發生我們選定的嵌入式整合系統無發搭配上 A/D Converter，國內又有許多 IC 需要大量訂購才能買到,最後改換題目為現在的 Web Content Recovery System，縱使沒能完成預定目標但是其中我們也對 IP Phone、嵌入式系統、H.323 標準等等有更深入的認識和了解，而其後的 WCR 系統雖然開發時間較為短，但是我們也竭盡全力讓系統能夠完備，除了考量未來發展以較高階的 Socket 為開發基礎，更將許多部分都預留可擴充和改良的空間，藉由如此的過程將課堂上所學的系統規劃和軟體工程應用於其上，雖然最後因為專題時程關係沒能將 IC 晶片卡和異地備援的功能加入不過我們依然收穫良多，最後由衷感謝長期給我們指導與鼓勵的指導老師，沒有老師的指導我們的專題也無法順利完成，以上即是我們專題的心得與感想。

參考資料：

1. FIPS PUB 46-3,Data Encrypt Standsrd(DES)
http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf
2. RSA's FAQ about Today's Cryptography (DES section)
http://www.rsasecurity.com/rsalabs/faq/3-2-1.html
3. FIPS PUB 81,DES Mode of Operation
http://www.itl.nist.gov/fipspubs/fip81.htm
4. Block ciphers and modes of operation. DES, AES. External ...
http://www.cs.ucd.ie/staff/vmatyas/home/COMP4015_lect04.pdf
5. What is DESX
http://www.rsasecurity.com/rsalabs/faq/3-2-7.html
6. The Data Encryption Standard's Volume.
http://www.ams.org/notices/200003/fea-landau.pdf
7. FIPS PUB 180-1, Secure Hash Standard
http://www.itl.nist.gov/fipspubs/fip180-1.htm
8. Secure Hash Standard
http://www.jura.ch/lcp/cours/dm/codage/moderne/fips180-2.pdf
9. CRYPTOGRAPHY AND NETWORK SECURITY Second Edition，作者：William Stallings，pp.64-117、pp.358-376，編號：ISBN 975-566-870-7
10. RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1
ftp://ftp.isi.edu/in-notes/rfc2616.txt
11. IIS 5.0 線上說明文件，Internet Information Service 5.0