行政院國家科學委員會專題研究計劃成果報告

****

* PGP 改進的研究及實作 *

****

計劃類別：個別型計劃

計劃編號：NSC 89-2213-E-009-005

執行期間：八十八年八月一日 至 八十九年七月三十一日

個別型計劃： 計劃主持人：葉義雄 交通大學資訊工程系副教授

共同主持人：林祝興 東海大學資訊科學系副教授

處理方式：一年後可對外提供參考

執行單位：交通大學資訊工程系 東海大學資訊科學系

中華民國 八十九 年 七 月 三十一 日

PGP 改進的研究及實作

目錄

誌　　謝

仰蒙行政院國家科學委員會之經費補助，使本計劃得以推動進行，並且順利完成，謹此誌謝。

計劃編號：NSC 89-2213-E-009-005

PGP 改進的研究及實作

葉義雄

交通大學資訊工程系
新竹市大學路 1001 **號**

E-mail：ysyeh@csie.nctu.edu.tw
Phone：(03)5712121 - 31813
Fax：(03)5724176

林祝興

東海大學資訊科學系
台中市西屯區台中港路三段 181 **號**

E-mail：chlin@mail.thu.edu.tw
Phone：(04)3590121 - 3287
Fax：(04)3596557

Correspondence Address：
Dr. Yi-Shiung Yeh
Chiao Tung University
Hsinchu, Taiwan 30050, ROC
E-mail : ysyeh@csie.nctu.edu.tw

Correspondence Address：
Dr. Chu-Hsing Lin
Box 808, Tunghai University
Taichung, Taiwan 407, ROC
E-mail：chlin@mail.thu.edu.tw

PGP 改進的研究及實作

摘要

PGP 在 1991 年時由 Philip Zimmermann 獨自開發成功並且公諸於世。經過八年來不斷的更新版本，PGP 已成為目前全世界流通最廣的加密軟體。可預見的是，在當今通訊保密問題日漸受到人們重視之際，PGP 將挾其廣大用戶的既有優勢，對未來保密通訊扮演極重要的角色。

本計劃將以 PGP 現有功能為基礎。嘗試以 AESII 公佈的五個候選演算法(Rijndael、Serpent、Twofish、RC6、Mars)等密碼方法來取代現今 PGP 中所使用的 IDEA 的對稱式加密法(Symmetric cryptographic algorithm)並新增單向雜湊函數(One-way hash function) RIPEMD160 的變形函數 RIPEMD-X 作為產生數位簽章(Digital Signature)的演算法，並建議各以 SHA-1、RIPEMD-X 為 DH/DSS、RSA 金鑰為基礎來產生簽章。主要是希望藉由這些新的密碼模組使用，以提高 PGP 軟體的安全等級，以因應未來的使用。

關鍵字

PGP (Pretty Good Privacy)、RIPEMD、Rijndael、Serpent、Twofish、RC6、Mars、MD5、IDEA、One-way hash function、Symmetric cryptographic algorithm、GNU GPL(General Public License)、Digital Signature。

1. 簡介

在網際網路基礎建設廣為架設與網際網路應用的蓬勃發展的時代，E 時代的已悄然來臨了，E 代表了 Easy、Economy；在網際網路不再侷限於學術領域運用時，商業用途的網際網路運用已取而代之為一主要的網際網路運用，因此網路安全與私密的要求已屬刻不容緩的必備條件。網路駭客、病毒、匿名攻擊更進一部威脅著網際網路的實用性與可靠性。因此如何提昇資料與資訊傳遞的安全性，早已是無庸置疑的研究課題。

資訊安全分可細分為幾個重要的課題：

・系統安全:

系統安全需具備防禦不當存取的與惡意的破壞行為，在一個不具安全防禦的作業系統，網路駭客可以輕易透過網際網路入侵電腦進行破壞。

・網路通訊協定:

網路通訊協定所必須具備的條件是保護通訊資訊的安全與私密性，它的功能著重在通訊協定的設計，以目前的趨勢而言包含的網路通訊安全協定包括 SSL、IPSEC、TLS 等等。

・密碼系統:

密碼系統包括對稱型加密系統、非對稱型加密系統及雜湊函數等，其功能包含資料保密、資料完整性、身分驗證與數位簽章，目前的對稱型加密系統有 DEA、IDEA、AES（有鑒於 DES 的安全性即將不敷使用，NIST 已經過三次的公開會議選定 Rijndael 為以後對稱型加密系統的標準）等。

PGP 在 1991 年時由 Philip Zimmermann 獨自開發成功並且公諸於世。經過八年來不斷的更新版本，且 PGP 是屬於 GNU GPL(General Public License)[15]的共享軟體，因此 PGP 已成為目前全世界流通最廣的加密軟體。可預見的是，在當今通訊保密問題日漸受到人們重視之際，PGP 將挾其廣大用戶的既有優勢，對未來保密通訊扮演極重要的角色。

PGP（Pretty Good Privacy）是可以讓電子郵件或檔案具有保密功能的程式，其背後所採用的加密技術就是公開金鑰加密。換言之 PGP 就是公開金鑰加密系統的一個實際產品，具有簡易的操作及金鑰的管理介面，提供一個資料加密與解密的環境讓我們來使用。

本計劃將以 PGP 現有功能為基礎。嘗試以 AESII 公佈的五個候選演算法(Rijndael、Serpent、Twofish、RC6、Mars)等密碼方法來取代現今 PGP 中所使用的 IDEA 的對稱式加密法(Symmetric cryptographic algorithm)並新增 RIPEMD-160 單向雜湊函數(One-way

hash function)的變形函數 RIPEMD-X 模組；希望能由所採用的這些密碼方法，獲得更適合在未來環境中使用的 PGP 軟體。

PGP 可以對電子文件利用 IDEA 做加密的運算。發送者先任選一個 128 位元的亂數做為 IDEA 的加密金鑰，對明文進行加密。發送者在將此任選的加密金鑰，利用接收者的 RSA 公開金鑰加密。在將文件的密文及密鑰密文傳送給對方。接收者先利用 RSA 私有金鑰，將 IDEA 的加密金鑰解出，再利用此解出金鑰來解讀密文。發送者也可以利用 PGP 來對電子文件產生屬於發送者的電子簽章。接收者利用發送者的公開金鑰，可以藉由電子簽章驗證電子文件的完整性。此外，PGP 軟體對公開金鑰的傳送及管理問題上，也提供了一整套非常有效率的解決辦法。

總而言之，PGP 提供了三種安全服務：文件識別，文件加密，金鑰交換。

另外，單向雜湊函數的觀念非常類似於資料壓縮，所差別者為單向雜湊函數是一種不可解壓縮的資料壓縮法。對於一段明文，單向雜湊函數可以由此明文產生一雜湊值，且此雜湊值的長度遠比明文短，並且無法由雜湊值求得明文的資訊。

目前 PGP 所使用的單向雜湊函數為 MD5，安全性對現今而言已不夠，有許多實例證明可以偽造 MD5 的雜湊值，因此本計劃建議各以 SHA-1、RIPEMD-X 為 DH/DSS、RSA 金鑰基礎以用來產生數位簽章，以謀求更高的安全性。

另外，對稱式加密演算法的意義在於，對於此演算法，若知道加密金鑰，則必能得知解密金鑰；換句話說，加密金鑰和解密金鑰是對稱的。一般而言，在對稱式加密演算法中，加密金鑰等於解密金鑰。

目前 PGP 所始用的對稱式加密演算法是 IDEA，在這個計劃中，我們將會以 1999 年公佈的 AESII 候選演算法之一的 RC6 來取代 IDEA，另外也將其他的候選演算法也整合進 PGP 系統內，以因應未來所需，提高安全等級。

本計劃將以 PGP6.5.1 的版本來作修正以期使 PGP 能接受更多的密碼模組，這也是本計劃所預期的研究成果，此成果期望藉由密碼模組的加強來強化密碼系統的安全強度，而不在於期望修正 PGP 的通訊協定來加強通訊機制的強度，因此本計劃將針對密碼系統的提供更多的密碼模組以期能加強現有 PGP 系統的安全，在本計劃所將新增加的對稱型加密系統有 RC6、Rijndael、Serpent、Twofish、Mars，針對單向雜湊函數我們將新增 RIPEMD-X(128/192/256)的模組以改良 RIPEMD-160。

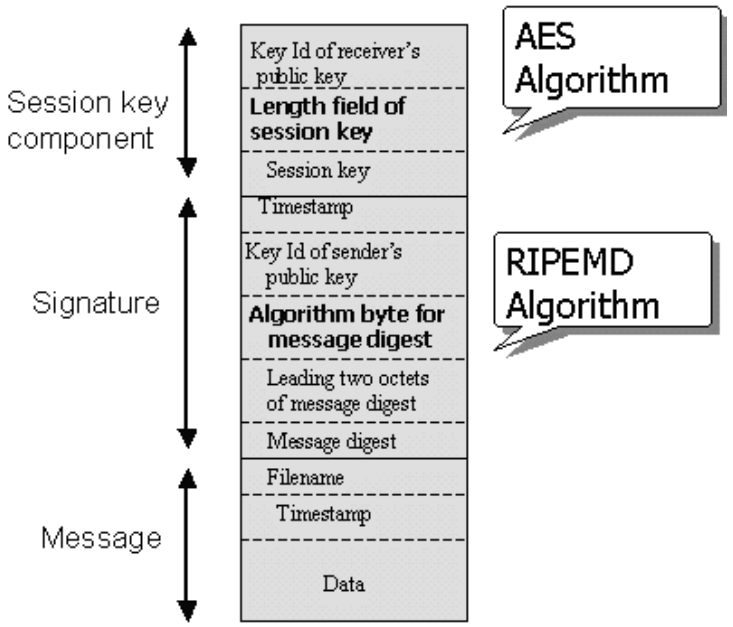

圖 1.1

圖 1.1 **為本計劃系統架構的示意圖，本圖僅是針對密碼系統來作觀念上的萃取示意圖, 如圖所示本計劃需修正** PGP6.5.1 **系統的部分包括負責** 1.command parser **工作的部分**: **必須修正此部份如此修正後的系統才能接受新增密碼器的編號。**2.**新增密碼模組的部分，我們必須建立符合此系統的呼叫介面如此才能將新的密碼模組加入。如此我們所修正的系統將可接受新增的密碼模組並建立以後再加入更多密碼模組的相關技術，此一系統將藉由對現有的** PGP **系統的修正來建立一符合我們需求的** PGP **安全系統而不再受限於現有** PGP **系統的安全性假設條件之下。**

2 背景介紹

2.1 PGP 運用的加密演算法

首先先解釋何謂對稱式加密演算法(symmetric cryptographic algorithm)**，對稱式加密演算法的意義在於，對於此演算法，若知道加密金鑰，則必能得知解密金鑰；換句話說，加密金鑰和解密金鑰是對稱的。一般而言，在對稱式加密演算法中，加密金鑰等於解密金鑰。**

而非對稱式加密演算法(Asymmetric cryptographic algorithm)**與對稱式加密演算法最大的不同處在於，其加密及解密所用的金鑰不同，而且很難由其中一個推算出另外一個。這兩把金鑰分別稱之『公開金鑰**(Public key)**』及『私密金鑰**(Private key)**』**

現在，我們來介紹 PGP **的加解密系統，** PGP **採用對稱與非對稱型的結合密碼系統。圖** 2.1.1(**參考**[3])**的簡單地將此概念表現出來：**

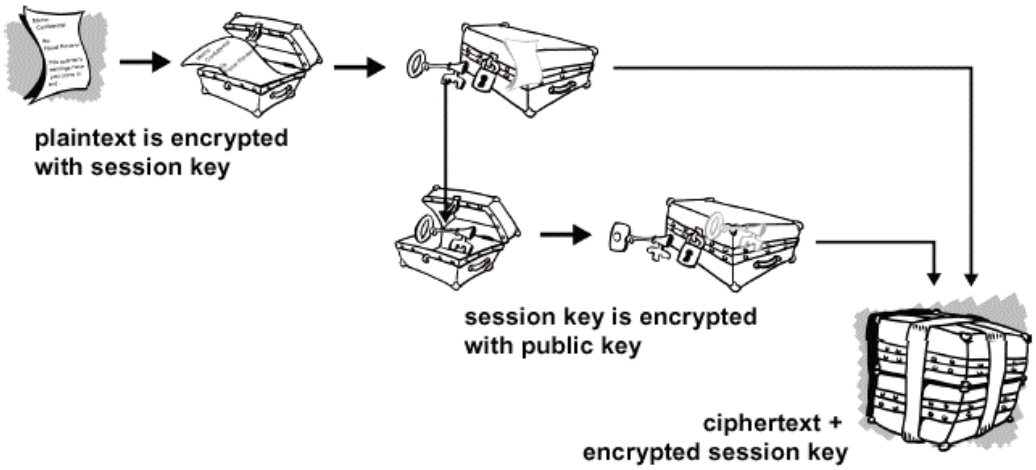

圖 2.1.1

上圖為 PGP **結合對稱及非對稱型的加密系統，利用隨機產生的** session key **來加密檔案(使用對稱性加密法)：**$A = E_{session\_key}[Message]$**；再用非對稱加密法將此金鑰加密：**$B = E_{receiver\_publickey}[session\ key]$**，如此只有接收端才能將此** session key **解開：**$D_{receiver\_privatekey}[B] =$ session key**；得到** session key **後，就可以用此** session key **將檔案解開：**$D_{session\_key}[A] =$ Message**。之所以使用混合型的加密系統，原因主要在於速度的考量，因為使用非對稱性加密法速度遠比對稱性慢的許多，因此只將** session key **使用非對稱法加密，而檔案的加解密就使用對稱性的加密法。**

2.2 單向雜湊函數(one-way hash function)

單向雜湊函數的觀念非常類似於資料壓縮，所差別者為單向雜湊函數是一種不可解壓縮的資料壓縮法。對於一段明文，單向雜湊函數可以由此明文產生一雜湊值，且此雜湊值的長度遠比明文短，並且無法由雜湊值求得明文的資訊。

目前 PGP 所使用的單向雜湊函數為 MD5 安全性已遭爭議，因此本計劃將新增 |RIPEMD-160 的變形函數 RIPEMD-X(128/192/256)單向雜湊函數模組，以謀求更高的安全性；並建議各以 SHA-1、RIPEMD-X 為 DH/DSS、RSA 金鑰為基礎來產生數位簽章。

2.3 電子簽章(digital signature)

電子簽章為近代密碼學與資訊安全最重要的應用技術之一。經過二十幾年的演進，電子簽章技術已漸趨成熟，且已可被廣泛接受及應用於電子商務中。如電子銀行、電子購物、電子錢包等。電子簽章技術依其應用又有許多變形。如多重電子簽章(Multi-signature)、盲目簽章(Blind signature)及不可拒絕簽章(Undeniable signature)等。值得注意的是，電子簽章本身並無法達到隱私性的功能；亦即電子簽章須配合加解密演算法使用。

2.4 PGP 簡介

2.4.1 PGP 加密流程

若以 pgp6.5.1i 為例，我們直接用圖來說明 PGP 的加密流程，如圖 2.4.1.1 所示：

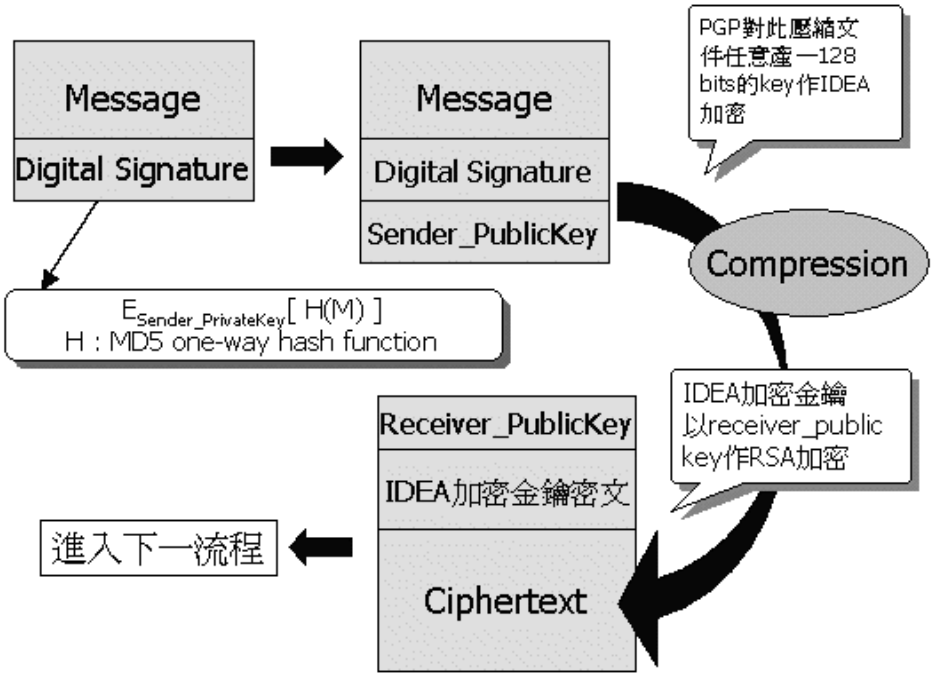

圖 2.4.1.1

使用者先對 Message **產生** Signature，**在** pgp6.5.1i **中**，PGP **會先使用** MD5 Hash Function **對** Message **產生** Hash --- H(M)，**之後用使用者的私鑰作加密形成簽章** --- $E_{signer\_privatekey}$[H(M)]，**之後再附上自己的公開金鑰；完成後，此時** PGP **會作壓縮的動作，以減低簽章所造成的影響；接下來** PGP **會對此壓縮文件任意產生一把** 128bits **的** session key，**再使用這把** session key **作** IDEA **加密，動作結束後，原** Message **已加密成密文** (CipherText)；**而這把用來加密** Message **的** session key，**也將利用非對稱性加密法加密** --- $E_{receiver\_publickey}$[Session key]；**最後再附上接收端的公開金鑰。這就形成了一個** package，**接下來以此** package **為單位進入下一個流程。**

2.4.2 基底 64 的轉換

首先我們先來說明基底 64 **轉換的目的，它主要是要使** PGP **處理過後的文件能在一般電子郵件系統中傳送，因為所有經由電子郵件傳送的文件內容，都是可打印的字母，而電子郵件系統就是利用那些不可打印的字母當作控制碼，所以避免密文內容與控制碼混淆，因此將字元重新編碼。圖** 2.4.2.1 **為基底** 64 **轉換的概念圖。**

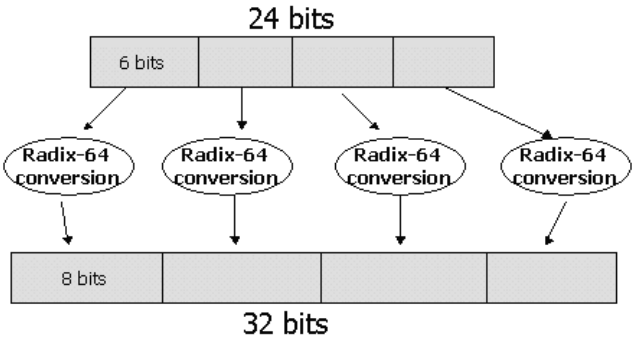

圖 2.4.2.1

而 PGP **如何去做基底轉換，只要參照以下的表** 2.4.2.2 **即可。**

表 2.4.2.2

6-bit character value	encoding	6-bit character value	encoding	6-bit character value	encoding	6-bit character value	encoding
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
1	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

另外雖然作基底 64 **轉換會使檔案大小增大 (增加** 3/2)**，但是之前有做過壓縮的處理 (減小** 1/2)**，所以最後轉換處理後，整個檔案大小將為原來的** 3/4**，**

2.4.3 PGP **的分段**

另外，因為許多電子郵件系統對文件的長度多有限制，因此 PGP **對於長度過長的文件也會自動分段。而接收端收到後，先將** header **刪去，再作重組的動作依順序還原。**

由以上這些小節可以將 PGP **處理文件的流程整理如下，如圖** 2.4.1 **所示：**

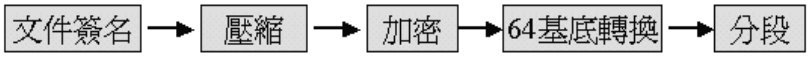

圖 2.4.1

此外，我們使用的 PGP **版本為** pgp6.5.1i**，為最新的原始碼公開的版本；**
使用的硬體平台為 Intel Pentium Pro 200**，**64MB Ram**；**
使用的作業平台為 Linux RedHat6.0+0.6CLE**；**

GNU **程式發展環境及工具：**

- egcs-2.91.66 (gcc version)
- GNU make version 3.77
- gdb 4.17.0.11
- ddd-3.2.1
- mcrypt-2.5.3

3. 改良 PGP 的發展過程

3.1 GNU 程式設計

因為 PGP 是屬於 GNU GPL 的軟體，因此我們特地介紹何謂 GNU GPL(通用公共許可證)，所謂 GNU Generation Public License〔15〕是強調使用者有共享和修改軟體的自由。由於我們架設在此環境下，因此可降低成本，並且可修改發展工具原始碼以幫助我們更了解流程，另外本計劃的架設環境為 linux RedHat6.0，因此發展工具為 unix 下的 C 語言。接下來介紹如何發展一個 GNU 的程式。『configure』目的產生一個 Make 檔案，主要根據我們所設定的參數(例如：函式庫路徑等等)而產生出 Make File；『make』是 unix 下一個強大的編譯工具，它會根據 make file 的設定去產生所需要的 library，另外，它還有一個重要的特性，若 make file 要產生的 library 已存在，它不會重新 make 一份新的 library，這對於大型系統的偵錯有相當大的幫助。

3.2 PGP Software Developer's Kit 簡介

在本計劃中，新增 Crypto 的模組是屬於 PGP Software Developer's Kit 的最重要部分。而 PGP 它是屬於一個大型加密系統，因此包括了許多 API，這些部分由於可以自由地讓程式設計師發展成符合自己或企業所需的系統，因此 PGP 另外在此部份提出文件 --- PGP Software Developer's Kit --- 它除了方便讓程式設計師輕易地將 PGP 的函式整合進自己的加密系統；另外 PGP 由於將所有的函式包成相同格式的 C 語言 Header，因此更容易作整合或新增的工作，換句話說：PGP 將所有的函式分成幾個部分的 API，並且呼叫的方式相同，以方便發展者呼叫到其他函式。

另外，由於 PGP 是屬於加解密系統，最主要的目的是資料的安全，因此在 PGPsdk 中除了教導每個 API 主要負責的功能也提醒程式設計師在每個 API 可能會陷入的安全陷阱中，當然最直接的做法是提供另一份描述 PGP 加解密系統的文件[3]。
還有在第二節有提及 PGP 是屬於 GNU GPL 的軟體，因此程式原始碼是公開的，這代表的意義是使用者可以完全相信這套系統，它不會有 Trojan horse 或 Trap 的存在，而資料安全完全建立在加密的演算法上。也因為原始碼公開及 PGP 屬 GNU GPL 軟體的原因使得 PGP 會如此廣泛的原因。

而如何改良 PGP 系統以符合我們所需，在 PGPsdk 中也清楚地提出程式設計師應該對所需更改的 API 直接作修正，而不須對整個 PGP 系統作修改。整個 PGP 系統有標準的函式呼叫，程式設計師只要針對所屬的 API 內的函式及所傳送的參數作深入的了解即可，不需鑽研到其他 API 內的函式呼叫，因為 PGP 雖然由 C 語言撰寫，但已有 API 的觀念，所以每個 API 間的參數傳遞是非常完整且可靠的。

而 PGP **系統主要的幾個** API **我們也詳述如下，包括了函式庫及重要的** Header File (**本表參考於** PGPsdkUserGuide[2])**：**

表 3.2.1

Subject Area	Header File	Unix Library
Option lists	pgpOptionList.h	(various)
Local key management	pgpKeys.h	PGPsdk.a
Groups	pgpGroups.h	
Ciphering & Authentication	pgpCBC.h pgpCFC.h pgpEncode.h pgpHash.h pgpHMAC.h pgpPublicKey.h pgpSymmetricCipher.h	
Feature query	pgpFeatures.h	
Utilities	pgpMemoryMgr.h pgpPubTypes.h pgpSDKPrefs.h pgpUtilities.h	(various)
Random number generation	pgpRandomPool.h	PGPsdk.a
User interface	pgpUserInterface.h	(notavailable)
Key server access	pgpKeyServer.h	PGPsdkNetworkLib.a
TLS	pgpTLS.h	
Network sockets	pgpSockets.h	
Big number management	pgpBigNum.h	PGPsdk.a
Error codes	pgpErrors.h pgpPFLErrors.h	

本計劃中，我們主要是深入 PGP **的核心部分** --- Ciphering and Authentication Functions ---**此** API **最重要的函式呼叫為** PGPEncode()**，可以將此函式看成作加解密的** API**，以下為一例：**

```
Err=PGPEncode( context,
        PGPOEncryptToKeySet(context, foundUserKey),
        PGPOInputFile(context, inFileRef),
        PGPOOutputFile(context,outFileRef),
        PGPOLastOption(context)  );
```

傳入的參數皆是負責某一功能的函式：context **是一個指向存放全域變數記憶體的指標；** PGPOEncryptToKeySet(context, foundUserKey)**是用來取得加解密金鑰的函式；** PGPOInputFile(context, inFileRef)**是取得輸入檔案的** Reference **指標，這當然包括了加解密、簽章等等輸入檔案(不是由** PGP **系統產生的檔案)；而** PGPOOutputFile(context,outFileRef) **是最後經由** PGP **運算處理(例如：加密、解密、簽章等等)後要指向的檔案指標；**PGPOLastOption(context)**是指向一塊記憶體的指標，這塊記憶體存放了不是預設值的功能選項(例如：使用** RC6 **演算法作加密)。**

3.3 PGP's Configuration File --- pgp.cfg

PGP 可以儲存使用者自訂的變數來符合使用者的需求, 可以方便的紀錄使用者訂定的旗號(Flags)和參數(Parameters)，本檔案 pgp.cfg 位於./pgp 目錄下。若不使用 PGP 預設值就必須設定組態檔，例如：PGP6.5.1i 在對稱性加密是以 IDEA 當作其預設值，若欲以 3DES 為預設值就必須撰寫此檔案。以下為一個簡單的 PGP 組態檔格式：

```
armor=on
ciphernum=2
compress=off
```

其他的參設設定還包括了 armorlines、cert_depth、clearsig、comment、compatible、completes_needed、encrypttoself、fastkeygen、hashnum、interactive、keyserver_url、marginals_needed、myname、pager、pgp_mime、pgp_mimeparse、pubring、randomdevice、randseed、secring、showpass、textmode、tzfix、verbose 等等…，使用者可以根據本身所需來設定參數，相關參數所代表的意義可以參考 PGPCmdLineGuide 的第四節[4]。

另外，若只是想單次使用不同預設值的參數，只需要在命令列下達選項(Option)的指令即可，例如：想以 3DES 取代預設值的 IDEA 當作對稱加密演算法只需下達『pgp –c +ciphernum=2 filename』。但是在以下這幾種情況，PGP 系統將仍使用預設值：1.組態參數未設定、2.組態檔不存在、3.PGP 找不到組態檔 (可能路徑或檔名不正確所造成)。

至於本次計劃著重的目標在於加密演算法及雜湊函數模組的新增，因此我們將『ciphernum』及『hashnum』此參數所支援的演算法拿出來加以討論。

在『ciphernum』部分：在 pgp6.5.1i 的對稱加密演算法包括了 IDEA、3DES、CAST5，下達的選項參數如下：

- 若以 IDEA 為對稱式加密演算法：+ciphernum=1；
- 若以 3DES 為演算法：+ciphernum=2；
- 若以 CAST5 為演算法：+ciphernum=3；

由此可知，pgp6.5.1i 所支援的對稱性加密演算法只有三種，而此三種演算法的安全性已遭受懷疑，因此本計劃在提出之際時，美國國家標準局 NIST (National Institute of Standards and Technology)在 1999 年在 15 個加密演算法選出 5 個對稱加密演算法作為 AES (Advanced Encryption Standard)的候選演算法。而前一節也提及 PGP 的資料安全是建立在這些演算法上, 因此本計劃決定將這些演算法整合於 PGP 系統當中以提昇我們的安全等級並因應未來的趨勢。

在『hashnum』部分，pgp6.5.1i 預設支援的有 MD5、SHA-1、RIPEMD160，但必須注意

的一點是 RIPEMD160 **只支援以** RSA **為金鑰基礎，意指要使用** RIPEMD160 **產生數位簽章，其使用者的金鑰需使用** RSA**，而不是** DH/DSS**。另外，由於** MD5 **安全性已遭爭議，且** RIPEMD **的複雜度比** SHA-1 **高。因此本計劃建議以** DH/DSS **產生金鑰的使用者使用** SHA-1 **產生簽章；以** RSA **產生金鑰的使用者用** RIPEMD-X **產生簽章。**

在原 PGP **系統其『**hashnum**』下達方式：**

‧使用 MD5 **為雜湊函數：** +hashnum=1**；**

3.4 PGP 加解密及簽章步驟

我們新增密碼模組是屬於『Ciphering & Authentication**』的範圍，因此必須先了解整個系統是如何在作加解密的動作。我們將步驟簡單地描述如下：**

‧加密過程：

1. Setup
2. Initializing the PGPsdk library
3. Creating a PGPContext
4. Obtaining the recipients's public key
5. Performing the key search
6. Creating the input and output file references
7. Forming options and ciphering the message
8. Call PGPEncode()
9. Cleanup
10. PGPShutdown

首先必須先作函式庫的初始化，之後必須建立一個指標指向一塊記憶體位置，此記憶體存放全域變數，換言之，儲存了所有使用者動作的資訊，以下為最基本的呼叫方式：

```
err = PGPNewContext( kPGPsdkAPIVersion, &context );
if( IsPGPError( err ) )     goto Exit;
```

接下來，如果不論使用者使用對稱性或非對稱性加密法都必須有一把金鑰用來作加密金鑰，PGP **順利取得金鑰後就必須在建立兩個指標用來連結檔案，第一個指標指向原使用者(未經** PGP **處理過)的檔案；第二個指標指向** PGP **產生的檔案(如：加密檔、簽章檔)，以下為最基本的呼叫方法：**

```
err = PGPNewFileSpecFromFullPath( context, inFileName, &inFileRef );
if( IsPGPError( err ) )   goto Exit;
err = PGPNewFileSpecFromFullPath( context, outFileName, &outFileRef );
if( IsPGPError( err ) )  goto Exit;
```

除了這些基本的步驟，另外必須也有一塊記憶體去儲存使用者自訂的選項參數；所有的參數準備就緒後，就進入加密的函式呼叫，在 PGP 加密最重要函式為 PGPEncode()，此函式會呼叫其他的子函式去完成參數的準備工作，包括了 pgpEncodeInternal()、pgpEncryptPipelineCreate()、pgpCipherModEncryptCreate()、pgpCFBEncryptInternal()、pgpCFBEncrypt()，之後才會呼叫使用者所訂定的加密演算法，若使用 PGP 預設值，接下來將呼叫 pgpIDEA.c 中的函式，以下為 PGPEncode 簡單的呼叫方式：

```
err = PGPEncode( context,
    PGPOEncryptToKeySet( context, foundUserKeys ),
        PGPOInputFile( context, inFileRef ),
        PGPOOutputFile( context, outFileRef ),
        PGPOLastOption( context ) );
```

所有加密動作完成後，最後就必須釋放所有的資源(如：記憶體空間)。

另外，解密、簽章及驗證簽章過程的方式都大同小異，因此不加贅述，只簡略地列出步驟：

・解密過程

1. Setup
2. Initializing the PGPsdk library
3. Creating the input and output file references
4. Providing access to your user’s private key
5. Passphrases control access to private decryption keys
6. Forming options and deciphering the message
7. Calling PGPDecode()
8. Cleanup
9. Even handler function

・簽章過程

1. Setup
2. Creating the input and output file references
3. Accessing your user’s private key
4. Forming options and signing the message
5. Calling PGPEncode()
6. Cleanup

・驗證簽章

1. Setup
2. Creating the file references
3. Accessing a key database
4. Forming options and verifying the signature

5. Verification options
6. Calling PGPDecode()
7. Even handler function

3.5 RC6 對稱式加密演算法

本計劃著重於 1999 **年發布的** AES **候選演算法的整合。在了解整個** PGP **系統作加解密的流程後，我們必須先就單一個演算法作研究，才能正確地將密碼模組整合進** PGP **系統內，因為一旦整合成功，代表我們已對** PGP **中作加解密的參數有完全的了解，接下來如果要新增其他的密碼模組，相信對我們而言並非難事。**

3.5.1 RC6-w/r/b 參數的基本描述

RC6-w/r/b parameters:

–Word size in bits: w (32)(lg(w) = 5)

–Number of rounds: r (20)

–Number of key bytes: b (16, 24, or 32)

Key Expansion:

–Produces array S[0 … 2r + 3] of w-bit round keys.

Encryption and Decryption:

–Input/Output in 32-bit registers A,B,C,D

3.5.2 RC6 基本運算

A + B	integer addition modulo 2^w
A – B	integer subtraction modulo 2^w
A⊕B	bitwise excusive-or of w-bit words
A x B	integer multiplication modulo 2^w
A <<< B	rotate the w-bit word a to the left by the amount given by the least significant lg(w) bits of B
A >>> B	rotate the w-bit word a to the right by the amount given by the least significant lg(w) bits of B

3.5.3 Key Scheduling

Define P32 = B7E15163 (hex)
Q32 = 9E3779B9 (hex).

Input: User-supplied b byte key preloaded into c-word array L[0, ... , c-1]
Number r of rounds

Output: w-bit round keys S[0, ... , 2r+3]

Procedure:

S[0]=Pw

for i = 1 to 2r + 3 do

S[i] = S[i-1] + Qw

A = B = i = j = 0

v = 3 x max{c, 2r+4}

for s = 1 to v do {

A = S[i] = (S[i] + A + B) <<< 3

B = L[j] = (L[j] + A + B) <<< (A+B)

i = (i + 1) mod (2r + 4)

j = (j + 1) mod c

}

3.5.4 RC6 Encryption

Input: Plaintext stored in four w-bit input registers A,B,C,D.

Number r of rounds.

w-bit round keys S[0, ... , 2r+3] (generated by the key schedule with primary key of b-byte).

Output: Ciphertext stored in A,B,C,D.

Procedure:

B = B + S[0]

D = D + S[1]

for i = 1 to r do

t = (B x (2B + 1)) <<< lg(w)

u = (D x (2D + 1)) <<< lg(w)

A = ((A⊕t) <<< u) + S[2i]

C = ((C⊕u) <<< t) + S[2i+1]

(A, B, C, D) = (B, C, D, A)

}

A = A + S[2r + 2]

C = C + S[2r + 3]

3.5.5 RC6 Decryption

Input: Ciphertext stored in four w-bit input registers A,B,C,D.

Number r of rounds.

w-bit round keys S[0, ... , 2r+3] (generated by the key schedule with primary key of b-byte).

Output: Plaintext stored in A,B,C,D.

Procedure:

C = C - S[2r + 3]
A = A - S[2r + 2]
for i = r downto 1 do {
(A, B, C, D) = (D, A, B, C)
u = (D x (2D + 1)) <<< lg(w)
t = (B x (2B + 1)) <<< lg(w)
C = ((C - S[2i+1]) >>> t) ⊕ u
A = ((A - S[2i]) >>> u) ⊕ t
}
D = D - S[1]
B = B - S[0]

3.6 其他新增的對稱式加密演算法

除 RC6 **之外，**AES **還有另四個候選演算法，分別為** Rijndael、Serpent、Twofish、Mars**。但是就誠如以上所述，一旦** RC6 **整合成功，我們已完成了解整個** PGP **在加解密部分的流程以及傳遞參數所代表的意義，因此我們只需了解這些演算法的基本描述。**

3.7 RIPEMD 演算法的改進

由於 PGP **預設處理** Hash function **的** MD5 **在近來的安全性已遭爭議因此我們決定使用** RIPMD **來取代** MD5**，但是在** PGP **系統中(指** pgp5.x version)**雖然有支援** RIPEMD**，但是只能使用在** DH/DSS **的金鑰上,** SHA-1 **可以使用在** DS/DSS **及** RSA **的金鑰, 而** RIPEMD **只能使用在** DH/DSS **上, 因此本計劃為增加安全性, 提出** RIPEMD-160 **的變形** RIPEMD-X **來增加複雜度。**

3.7.1 RIPEMD-160

以下簡單的介紹 RIPEMD-160 **的演算法，輸入皆是** 512 bits**，經過** RIPEMD-160 **雜湊函數的計算，輸出** 160bits **的雜湊值。**

- RIPEMD-160: definitions

nonlinear functions at bit level: exor, mux, -, mux, -

f(j, x, y, z) = x XOR y XOR z (0 <= j <= 15)
f(j, x, y, z) = (x AND y) OR (NOT(x) AND z) (16 <= j <= 31)
f(j, x, y, z) = (x OR NOT(y)) XOR z (32 <= j <= 47)
f(j, x, y, z) = (x AND z) OR (y AND NOT(z)) (48 <= j <= 63)
f(j, x, y, z) = x XOR (y OR NOT(z)) (64 <= j <= 79)

added constants (hexadecimal)

```
K(j) = 0x00000000      (0 <= j <= 15)
K(j) = 0x5A827999     (16 <= j <= 31)      int(2**30 x sqrt(2))
K(j) = 0x6ED9EBA1     (32 <= j <= 47)      int(2**30 x sqrt(3))
K(j) = 0x8F1BBCDC     (48 <= j <= 63)      int(2**30 x sqrt(5))
K(j) = 0xA953FD4E     (64 <= j <= 79)      int(2**30 x sqrt(7))
K'(j) = 0x50A28BE6     (0 <= j <= 15)      int(2**30 x cbrt(2))
K'(j) = 0x5C4DD124    (16 <= j <= 31)      int(2**30 x cbrt(3))
K'(j) = 0x6D703EF3    (32 <= j <= 47)      int(2**30 x cbrt(5))
K'(j) = 0x7A6D76E9    (48 <= j <= 63)      int(2**30 x cbrt(7))
K'(j) = 0x00000000    (64 <= j <= 79)
```

selection of message word

```
r(j)      = j                   (0 <= j <= 15)
r(16..31) = 7, 4, 13, 1, 10, 6, 15, 3, 12, 0, 9, 5, 2, 14, 11, 8
r(32..47) = 3, 10, 14, 4, 9, 15, 8, 1, 2, 7, 0, 6, 13, 11, 5, 12
r(48..63) = 1, 9, 11, 10, 0, 8, 12, 4, 13, 3, 7, 15, 14, 5, 6, 2
r(64..79) = 4, 0, 5, 9, 7, 12, 2, 10, 14, 1, 3, 8, 11, 6, 15, 13
r'(0..15) = 5, 14, 7, 0, 9, 2, 11, 4, 13, 6, 15, 8, 1, 10, 3, 12
r'(16..31)= 6, 11, 3, 7, 0, 13, 5, 10, 14, 15, 8, 12, 4, 9, 1, 2
r'(32..47)= 15, 5, 1, 3, 7, 14, 6, 9, 11, 8, 12, 2, 10, 0, 4, 13
r'(48..63)= 8, 6, 4, 1, 3, 11, 15, 0, 5, 12, 2, 13, 9, 7, 10, 14
r'(64..79)= 12, 15, 10, 4, 1, 5, 8, 7, 6, 2, 13, 14, 0, 3, 9, 11
```

amount for rotate left (rol)

```
s(0..15)  = 11, 14, 15, 12, 5, 8, 7, 9, 11, 13, 14, 15, 6, 7, 9, 8
s(16..31) = 7, 6, 8, 13, 11, 9, 7, 15, 7, 12, 15, 9, 11, 7, 13, 12
s(32..47) = 11, 13, 6, 7, 14, 9, 13, 15, 14, 8, 13, 6, 5, 12, 7, 5
s(48..63) = 11, 12, 14, 15, 14, 15, 9, 8, 9, 14, 5, 6, 8, 6, 5, 12
s(64..79) = 9, 15, 5, 11, 6, 8, 13, 12, 5, 12, 13, 14, 11, 8, 5, 6
s'(0..15) = 8, 9, 9, 11, 13, 15, 15, 5, 7, 7, 8, 11, 14, 14, 12, 6
s'(16..31)= 9, 13, 15, 7, 12, 8, 9, 11, 7, 7, 12, 7, 6, 15, 13, 11
s'(32..47)= 9, 7, 15, 11, 8, 6, 6, 14, 12, 13, 5, 14, 13, 13, 7, 5
s'(48..63)= 15, 5, 8, 11, 14, 14, 6, 14, 6, 9, 12, 9, 12, 5, 15, 8
s'(64..79)= 8, 5, 12, 9, 12, 5, 14, 6, 8, 13, 6, 5, 15, 13, 11, 11
```

initial value (hexadecimal)

```
h0 = 0x67452301; h1 = 0xEFCDAB89; h2 = 0x98BADCFE; h3 = 0x10325476; h4 = 0xC3D2E1F0;
```

• RIPEMD-160: pseudo-code

It is assumed that the message after padding consists of t 16-word blocks
that will be denoted with X[i][j], with 0 <= i <= t-1 and 0 <= j <= 15.
The symbol [+] denotes addition modulo 2**32 and rol_s denotes cyclic left
shift (rotate) over s positions.

```
for i := 0 to t-1 {
   A := h0;
   B := h1;
   C := h2;
   D = h3;
   E = h4;
```

```
    A' := h0;
    B' := h1;
    C' := h2;
    D' = h3;
    E' = h4;
    for j := 0 to 79 {
       T := rol_s(j)(A [+] f(j, B, C, D) [+] ,X[i][r(j)] [+] K(j)) [+] E;

       A := E;
       E := D;
       D := rol_10(C);
       C := B;
       B := T;

       T := rol_s'(j)(A' [+] f(79-j, B', C', D') [+] X[i][r'(j)] [+] K'(j)) [+] E';

       A' := E';
       E' := D';
       D' := rol_10(C');
       C' := B';
       B' := T;
    }
    T := h1 [+] C [+] D';
    h1 := h2 [+] D [+] E';
    h2 := h3 [+] E [+] A';
    h3 := h4 [+] A [+] B';
    h4 := h0 [+] B [+] C';
    h0 := T;
}
```

3.7.2 RIPEMD 的變形 ---『RIPEMD-X』

我們將 RIPEMD-160 **改良成** RIPEMD-X **以增加複雜度。主要的概念如何將輸出的雜湊值增加，我們的做法是將暫存器增加並更改演算法以符合所需。以下為我們所提出的變形函數** RIPEMD-X**：**

Algorithm RIPEMD-X

Input: X, M

Begin

n = X divide 32

Divide message into t blocks, Xi,i=0 to t-1

for i=0 to t-1 {

CASE X=128 ($h_0h_1h_2h_3$)

$A=h_0$ $A'=h_1$

$B=h_1$ $B'=h_2$

$C=h_2$ $C'=h_3$

$D=h_3$ $D'=h_0$

$E=h_0$ $E'=h_1$

CASE X=192 ($h_0h_1h_2h_3h_4h_5$)

$A=h_0$ $A'=h_5$

B=h_1 B'=h_0

C=h_2 C'=h_1

D=h_3 D'=h_2

E=h_4 E'=h_3

CASE X=256 ($h_0h_1h_2h_3h_4h_5h_6h_7$)

A=h_0 A'=h_5

B=h_1 B'=h_6

C=h_2 C'=h_7

D=h_3 D'=h_0

E=h_4 E'=h_1

For j=0 to 79 {

T=$rol_{s(j)}$(A+f(j,B,C,D)+x_i[r(j)]+k(j))+E;

A=E,E=D,D=ROL10(C);C=B;B=T

T=$rol_{s'(j)}$(A'+f(79-j,B',C',D')+x_i[r'(j)]+k'(j))+E';

A'=E;E'=D';D'=ROLS10(C');C'=B';B'=T;

}

CASE X=128

h_0=h_0+A+E+D';

h_1=h_1+B+A'+E';

h_2=h_2+C+B';

h_3=h_3+D+C';

CASE X=192

h_0=h_0+A+B';

h_1=h_1+B+C';

h_2=h_2+C+D';

h_3=h_3+D+E';

h_4=h_4+E

h_5=h_5+A'

CASE X=256

h_0=h_0+A+D'

h_1=h_1+B+E;

h_2=h_2+C;

h_3=h_3+D;

h_4=h_4+E

h_5=h_5+A'

h_6=h_6+B'

h_7=h_7+C'

}

End.

Ouput:$h_0h_1h_2….h_{n-1}$

輸出的值即為輸入 512bits **的雜湊值，根據使用者所需有** 128bits、192bits、256bits**，當然也包括了原來的** 160bits**。**

4.PGP 改進的實作

4.1 相關函式的呼叫

我們將相關的函式呼叫整理如下，可以粗略地了解 PGP **如何去呼叫一個加密演算法，這些動作有助於我們去增加一個新的** CIPHERNUM**，**

1.『clients/pgp/cmdline/doencode.c』 Ğ conventionalEncryptFile

2.『libs/pgpcdk/priv/clientlib/pgpClientEncode.c』 Ğ doEncode
in doEncode have a parameter 『PGPOConventionalEncrypt』 to set pgp cipher algorithm

3.『libs/pgpcdk/priv/encrypt/pgpEncode.c』 Ğ pgpEncodeInternal
the cipher algorithm is decide by 『pgp.cfg』 of 『CIPHERNUM=』

4.『doencode.c』 is the cmdline interface,
and the kernel of the pgp is in the 『libs/』
doencode.c will call doEncode in 『pgpClientEncode.c』

5.『pgpClientEncode.c』 just convert xxx(xxx,...) to only one method, check line 33 and 57
the parameter use firstOption and lastOption to make more parameter and use another function to parse it, and generate args
funally, call『pgpEncodeInternal』in『pgpEncode.c』

6.『pgpEncodeInternal』is the kernel encoding function,
it encode everything decide by the content of context
it first set conventionalCipheralgorithm 『1078:pgpSetupConventionalEncryption』then encode it

4.2 新增一個『CIPHERNUM』選項

我們必須讓 PGP **加密系統先接受一個新的演算法，由** PGPCmdLine[2]**可知，指定** PGP **使用哪一個加密演算法的選項參數設定為**『CIPHERNUM』**，下達的指令為**『pgp –c +ciphernum=? filename』**，其中**?**可以**『1、2、3』**代替；另外從** PGPCmdLine[2]**會提供使用者可加密演算法的資訊或從**『./pgp/libs/pgpcdk/pub/include/pgpPubTypes.h』**可以看出** PGP **系統在對稱加密提供了哪些演算法，在現行的** pgp6.5.1i **中，**PGP **提供了** IDEA、

3DES、CAST5 的對稱性加密演算法。因此我們必須讓 CIPHERNUM 可以接受到『4』，如此一來，才完成新增一個 CIPHERNUM 的工作。為了偵錯方便使用，我們以其中一個演算法作為樣板，真正更改的是 Header 及 Exported Function，下達的指令為『pgp –c +ciphernum=4 filename』，若加解密無誤，表示我們以成功加入新的 CIPHERNUM。

底下以一個實例來說明，也藉由這個實例實際地去了解 PGP 新增密碼模組的流程及所用到的 Header 或 Function。

實例說明：

1. libs/pgpcdk/pub/include/pgpPubTypes.h

add 153 kPGPCipherAlgorithm_TEST = 4,

change 156 kPGPCipherAlgorithm_Last = kPGPCipherAlgorithm_TEST,

新增一個 Cipher Algorithm，假設演算法名稱為 TEST，並將選項參數設為『4』，另外，將 kPGPCipheralgorithm_Last 這個指標指向 kPGPCipherAlgorithm_TEST，以供接下來程式碼計算加密演算法的數量。

2. libs/pgpcdk/priv/include/pgpSDKBuildFlags.h

add 28 #define PGP_TEST

pgpSDKBuildFlags.h 是一個控制 Flag 的 Header File，必須在此檔案將 PGP_TEST 這個演算法的 Flag 轉為 true，此演算法才可被使用。

3. libs/pgpcdk/priv/crypto/cipher/pgpSymmetricCipher.c

add 21 #include "pgpTEST.h"

add 460 &cipherTEST,

將會使用到的對稱性加密演算法的 Header 宣告在 pgpSymmetricCipher.c 中，另外必須在將&cipherTEST 定義，此變數將來會 export 供其他函式呼叫。

4. libs/pgpcdk/priv/crypto/cipher/

copy pgpIDEA.c pgpIDEA.c

copy pgpIDEA.h pgpTEST.h

我們使用 IDEA 演算法來作測試的動作。

5 .libs/pgpcdk/priv/crypto/cipher/pgpTEST.h

change all IDEA to TEST

6 .libs/pgpcdk/priv/crypto/cipher/pgpTEST.c

change 52 #ifndef PGP_TEST

change 56 #if PGP_TEST /* [*/

change 62 #include "pgpTEST.h"

change 421 PGPCipherVTBL const cipherTEST = {

change 422 "TEST",
change 423 kPGPCipherAlgorithm_TEST,

7. libs/pgpcdk/priv/crypto/cipher/Makefile.in
change 8 pgpCAST5.o pgpCBC.o pgpTwofish.o pgpRC2.o pgpTEST.o
change 11 pgpTwofishPlatform.h pgpTwofishTable.h pgpTEST.h
更改 Makefile**，在執行『**make**』後以產生** pgpTEST.o **此** obj **檔。**

8. back to libs/pgpcdk
and execute
./configure
make headers
make

4.3 PGP 如何去呼叫指定的加密演算法

底下為幾個叫重要的檔案，我們將分述如下：

[1] is pgp cipher algorithm list, and kPGPCipherAlgorithm_Last should point to the Last algorithm for counting total numbers of algorithm
[2] is a switch of using pgp cipher algorithm if you set PGP_TEST=0 and you will never pass the compile
[3] is the cipher algorithm which to map to the position of this algorithm must the same to [1]

when setting algorithm, pgp will get algorithm number from your pgp.cfg,
map to PGPCipherAlgorithm in [1]

```
enum PGPCipherAlgorithm_
{
    /* do NOT change these values */
    kPGPCipherAlgorithm_None      = 0,
    kPGPCipherAlgorithm_IDEA      = 1,
    kPGPCipherAlgorithm_3DES      = 2,
    kPGPCipherAlgorithm_CAST5     = 3,
    kPGPCipherAlgorithm_TEST      = 4,

    kPGPCipherAlgorithm_First           = kPGPCipherAlgorithm_IDEA,
    kPGPCipherAlgorithm_Last            = kPGPCipherAlgorithm_TEST,
```

```
        PGP_ENUM_FORCE( PGPCipherAlgorithm_ )
    };
    PGPENUM_TYPEDEF( PGPCipherAlgorithm_, PGPCipherAlgorithm );
```

then in [3], compare the position of 1 and sCipherList[] in [3],
and get hte reference from sCipherList[]

```
    static PGPCipherVTBL const * const sCipherList[] =
    {
    #if PGP_IDEA
        &cipherIDEA,
    #endif
        &cipher3DES,
        &cipherCAST5,
        &cipherTEST,
    };
```

then it will lookup the xxx.h(pgpIDEA.h, pgpDES3.h, pgpCAST5.h, pgpTEST.h)
to find the reference 『PGPCipherVTBL』
in xxx.c will declear this 『PGPCipherVTBL』, and operation

```
    PGPCipherVTBL const cipherIDEA = {
        "IDEA",
        kPGPCipherAlgorithm_IDEA,
        8,              /* Blocksize */
        16,             /* Keysize */
        IDEA_KEYBYTES + 1,      /* Last one remembers encrypt vs decrypt */
        alignof(PGPUInt16),
        ideaKey,
        ideaEncrypt,
        ideaDecrypt,
        ideaWash
    };
```

if we want to write cipher algorithm
we should follow their way to rewrite our code

4.4 RC6 演算法的實作

根據 3.5 **節，我們先實作出** RC6 **演算法，之後再加上符合** pgp **呼叫參數的格式及型態。以下為實作** RC6 **演算法的原始碼。**

RC6 演算法的實作：

```
#include <stdio.h>

/* RC6 is parameterized for w-bit words, b bytes of key, and
 * r rounds.  The AES version of RC6 specifies b=16, 24, or 32;
 * w=32; and r=20.
 */

#define w 32        /* word size in bits */
#define r 20        /* based on security estimates */

#define P32 0xB7E15163      /* Magic constants for key setup */
#define Q32 0x9E3779B9

/* derived constants */
#define bytes   (w / 8)                     /* bytes per word */
#define c       ((b + bytes - 1) / bytes)       /* key in words, rounded up */
#define R24     (2 * r + 4)
#define lgw     5                       /* log2(w) -- wussed out */

/* Rotations */
#define ROTL(x,y) (((x)<<(y&(w-1))) | ((x)>>(w-(y&(w-1)))))
#define ROTR(x,y) (((x)>>(y&(w-1))) | ((x)<<(w-(y&(w-1)))))

unsigned int S[R24 - 1];                /* Key schedule */

void rc6_key_setup(unsigned char *K, int b) {
        int i, j, s, v;
        unsigned int L[(32 + bytes - 1) / bytes]; /* Big enough for max b */
        unsigned int A, B;

        L[c - 1] = 0;
        for (i = b - 1; i >= 0; i--)
                L[i / bytes] = (L[i / bytes] << 8) + K[i];

        S[0] = P32;
        for (i = 1; i <= 2 * r + 3; i++)
                S[i] = S[i - 1] + Q32;

        A = B = i = j = 0;
        v = R24;
        if (c > v) v = c;
        v *= 3;

        for (s = 1; s <= v; s++)
        {
                A = S[i] = ROTL(S[i] + A + B, 3);
                B = L[j] = ROTL(L[j] + A + B, A + B);
                i = (i + 1) % R24;
                j = (j + 1) % c;
        }
}
```

```
void rc6_block_encrypt(unsigned int *pt, unsigned int *ct) {
	unsigned int A, B, C, D, t, u, x;
	int i, j;

	A = pt[0];
	B = pt[1];
	C = pt[2];
	D = pt[3];
	B += S[0];
	D += S[1];
	for (i = 2; i <= 2 * r; i += 2) {
		t = ROTL(B * (2 * B + 1), lgw);
		u = ROTL(D * (2 * D + 1), lgw);
		A = ROTL(A ^ t, u) + S[i];
		C = ROTL(C ^ u, t) + S[i + 1];
		x = A;
		A = B;
		B = C;
		C = D;
		D = x;
	}
	A += S[2 * r + 2];
	C += S[2 * r + 3];
	ct[0] = A;
	ct[1] = B;
	ct[2] = C;
	ct[3] = D;
}

void rc6_block_decrypt(unsigned int *ct, unsigned int *pt) {
	unsigned int A, B, C, D, t, u, x;
	int i, j;

	A = ct[0];
	B = ct[1];
	C = ct[2];
	D = ct[3];
	C -= S[2 * r + 3];
	A -= S[2 * r + 2];
	for (i = 2 * r; i >= 2; i -= 2) {
		x = D;
		D = C;
		C = B;
		B = A;
		A = x;
		u = ROTL(D * (2 * D + 1), lgw);
		t = ROTL(B * (2 * B + 1), lgw);
		C = ROTR(C - S[i + 1], t) ^ u;
		A = ROTR(A - S[i], u) ^ t;
	}
	D -= S[1];
	B -= S[0];
	pt[0] = A;
	pt[1] = B;
	pt[2] = C;
	pt[3] = D;
}
```

4.5 Add crypto module mapping CIPHERNUM – RC6 Algorithm

需將模組程式放置以下路徑：『pgp6.5.1i/libs/pgpcdk/priv/crypto/cipher/』

- Header file : pgpRC6.h

```
#ifndef Included_pgpRC6_h
#define Included_pgpRC6_h
#include "pgpSymmetricCipherPriv.h"
PGP_BEGIN_C_DECLARATIONS
extern PGPCipherVTBL const cipherRC6;
PGP_END_C_DECLARATIONS
#endif /* Included_pgpRC6_h */
```

- Define a Cipher for generic cipher (main structure)

```
PGPCipherVTBL const cipherRC6 = {
"RC6",
kPGPCipherAlgorithm_RC6,
16,             /* Blocksize */
16,             /* Keysize */
RC6_KEYBYTES,
alignof(PGPUInt32),
rc6Key,
rc6Encrypt,
rc6Decrypt,
NULL  //RC6Wash
};
```

- pointer priv* point a memory address **big enough** to keep subkey or key schedule boxes.
  RC6_KEYBYTES : have to **point out the priv* size**
  E.g

```
#define RC6_ROUNDS 20
#define RC6_KEYLEN (2*RC6_ROUNDS+4)
#define RC6_KEYBYTES (sizeof(PGPUInt32) * RC6_KEYLEN)
```

- Exported **three** main functions

```
static void  rc6Key(void *priv, void const *key)  {
        rc6_key_setup((const PGPUInt32 *) key, (PGPUInt32 *)priv, 16);
}
```

*static void rc6_key_setup(PGPUInt32 const *L, PGPUInt32 *xkey, int b)*

```
static void rc6Encrypt(void *priv, void const *in, void *out)  {
```

```
    rc6_block_encrypt((const PGPUInt32 *) in, (PGPUInt32 *) out, (PGPUInt32
    *)priv);
}
    static void rc6_block_encrypt(PGPUInt32 const *pt,
                                  PGPUInt32  *ct,   PGPUInt32 const *xkey)

static void rc6Decrypt(void *priv, void const *in, void *out)  {
    rc6_block_decrypt((const PGPUInt32 *) in, (PGPUInt32 *) out, (PGPUInt32
*)priv);
}
    static void rc6_block_decrypt(PGPUInt32 const *ct,
                                  PGPUInt32 *pt, PGPUInt32 const *xkey)
```

4.6 pgpRC6 演算法的實作

4.6.1 pgpRC6.h

```
#ifndef Included_pgpRC6_h
#define Included_pgpRC6_h

#include "pgpSymmetricCipherPriv.h"

PGP_BEGIN_C_DECLARATIONS

extern PGPCipherVTBL const cipherRC6;

PGP_END_C_DECLARATIONS

#endif /* Included_pgpRC6_h */
```

4.6.2 pgpRC6.c

```
#include "pgpSDKBuildFlags.h"

#ifndef PGP_RC6
#error you must define PGP_RC6 one way or the other
#endif

#if PGP_RC6      /* [ */

#include <string.h>
#include "pgpConfig.h"

#include "pgpSymmetricCipherPriv.h"
#include "pgpRC6.h"
#include "pgpMem.h"
#include "pgpUsuals.h"


// A few handy definitions
#define RC6_ROUNDS 20
```

```
#define RC6_KEYLEN (2*RC6_ROUNDS+4)
#define RC6_KEYBYTES (sizeof(PGPUInt32) * RC6_KEYLEN)


//#include <stdio.h>

/* RC6 is parameterized for w-bit words, b bytes of key, and
 * r rounds.  The AES version of RC6 specifies b=16, 24, or 32;
 * w=32; and r=20.
 */

#define w 32        /* word size in bits */
//#define r 20      /* based on security estimates */

#define P32 0xB7E15163      /* Magic constants for key setup */
#define Q32 0x9E3779B9

/* derived constants */
#define bytes   (w / 8)                         /* bytes per word */
//#define RC6_KEYBYTES    ((16 + bytes - 1) / bytes)      /* key in words, rounded up  b=16*/
#define R24     (2 * RC6_ROUNDS + 4)
#define lgw     5                       /* log2(w) -- wussed out */

/* Rotations */
#define ROTL(x,y) (((x)<<(y&(w-1))) | ((x)>>(w-(y&(w-1)))))
#define ROTR(x,y) (((x)>>(y&(w-1))) | ((x)<<(w-(y&(w-1)))))

//unsigned int S[R24 - 1];              /* Key schedule */



/* Private functions */

static void rc6_key_setup(PGPUInt32 const *L, PGPUInt32 *xkey, int b)
{
        int i, j, s, v;
//      unsigned int L[(16 + bytes - 1) / bytes]; /* Big enough for max b--16 */
        PGPUInt32 A, B;

/*      L[RC6_KEYBYTES - 1] = 0;
        for (i = b - 1; i >= 0; i--)
                L[i / bytes] = (L[i / bytes] << 8) + k[i];
*/
        xkey[0] = P32;
        for (i = 1; i <= 2 * RC6_ROUNDS + 3; i++)
                xkey[i] = xkey[i - 1] + Q32;

        A = B = i = j = 0;
        v = R24;
        if (RC6_KEYLEN > v) v = RC6_KEYLEN;
        v *= 3;

        for (s = 1; s <= v; s++)
        {
                A = xkey[i] = ROTL(xkey[i] + A + B, 3);
                B = L[j] = ROTL(L[j] + A + B, A + B);
                i = (i + 1) % R24;
                j = (j + 1) % (b/4);
        }
```

```
}

static void rc6_block_encrypt(PGPUInt32 const *pt, PGPUInt32 *ct,PGPUInt32 const *xkey)
{
	PGPUInt32 A, B, C, D, t, u, x;
	int i;

	A = pt[0];
	B = pt[1];
	C = pt[2];
	D = pt[3];
	B += xkey[0];
	D += xkey[1];
	for (i = 2; i <= 2 * RC6_ROUNDS; i += 2)
	{
		t = ROTL(B * (2 * B + 1), lgw);
		u = ROTL(D * (2 * D + 1), lgw);
		A = ROTL(A ^ t, u) + xkey[i];
		C = ROTL(C ^ u, t) + xkey[i + 1];
		x = A;
		A = B;
		B = C;
		C = D;
		D = x;
	}
	A += xkey[2 * RC6_ROUNDS + 2];
	C += xkey[2 * RC6_ROUNDS + 3];
	ct[0] = A;
	ct[1] = B;
	ct[2] = C;
	ct[3] = D;
}

static void rc6_block_decrypt(PGPUInt32 const *ct, PGPUInt32 *pt, PGPUInt32 const *xkey)
{
	PGPUInt32 A, B, C, D, t, u, x;
	int i;

	A = ct[0];
	B = ct[1];
	C = ct[2];
	D = ct[3];
	C -= xkey[2 * RC6_ROUNDS + 3];
	A -= xkey[2 * RC6_ROUNDS + 2];
	for (i = 2 * RC6_ROUNDS; i >= 2; i -= 2)
	{
		x = D;
		D = C;
		C = B;
		B = A;
		A = x;
		u = ROTL(D * (2 * D + 1), lgw);
		t = ROTL(B * (2 * B + 1), lgw);
		C = ROTR(C - xkey[i + 1], t) ^ u;
		A = ROTR(A - xkey[i], u) ^ t;
	}
	D -= xkey[1];
	B -= xkey[0];
	pt[0] = A;
	pt[1] = B;
```

```
	pt[2] = C;
	pt[3] = D;
}

/*
 * Exported functions
 */

static void
rc6Key(void *priv, void const *key)
{

	rc6_key_setup((const PGPUInt32 *) key, (PGPUInt32 *)priv, 16);

}

static void
rc6Encrypt(void *priv, void const *in, void *out)
{

	rc6_block_encrypt((const PGPUInt32 *) in, (PGPUInt32 *) out, (PGPUInt32 *)priv);

}

static void
rc6Decrypt(void *priv, void const *in, void *out)
{
	rc6_block_decrypt((const PGPUInt32 *) in, (PGPUInt32 *) out, (PGPUInt32 *)priv);

}


/*
 * Define a Cipher for the generic cipher.  This is the only
 * real exported thing -- everything else can be static, since everything
 * is referenced through function pointers!
 */
PGPCipherVTBL const cipherRC6 = {
	"RC6",
	kPGPCipherAlgorithm_RC6,
	16,			/* Blocksize */
	16,			/* Keysize */
	RC6_KEYBYTES,
	alignof(PGPUInt32),
	rc6Key,
	rc6Encrypt,
	rc6Decrypt,
	NULL
	//ideaWash
};


#endif /* ] PGP_RC6 */
```

4.7 加入其它的對稱式加密演算法

我們使用現有的密碼模組 --- mlibcrypto **的模組來實作其他** AESII **的四個候選演算法，並**

根據之前的說明而加以更改以整合進 PGP **的系統中。此處要再強調一點，凡屬於** GNU GPL **的程式設計，都必須將原始碼公開，必須所有的使用者都有修改原始碼的權利，但一旦原始碼有經修改，都必須再將新版本的原始碼公開，如此才符合** GNU GPL **的精神。**

4.8 RIPEMD 雜湊函數的改良過程

4.8.1 新增一個『HASHNUM』選項

先以一個 TEST **演算法作為新增** HASHNUM **的測試，一旦成功再將原** TEST **改成我們所要的演算法。以下為新增一個** HASHNUM **的步驟：**

1. libs/pgpcdk/pub/include/pgpPubTypes.h

```
add     181     kPGPHashAlgorithm_TEST  = 4,

change  183  kPGPHashAlgorithm_First    = kPGPHashAlgorithm_MD5,
change  184     kPGPHashAlgorithm_Last    = kPGPHashAlgorithm_TEST,
```

2. libs/pgpcdk/priv/crypto/hash/pgpHash.c

```
add     17      #include "pgpTEST.h"
change  337     static PGPHashVTBL const * const sHashList[]  =
add     343     &HashTEST,
```

3. libs/pgpcdk/priv/crypto/hash/

```
copy pgpMD5.c pgpTEST.c
copy pgpMD5.h pgpTEST.h
```

4 .libs/pgpcdk/priv/crypto/hash/pgpTEST.h

```
change all MD5 to TEST
```

5 .libs/pgpcdk/priv/crypto/hash/pgpTEST.c

```
change  18   #include "pgpTEST.h"
change  499  struct PGPHashVTBL const HashTEST = {
change  500  "TEST",
change  501  kPGPHashAlgorithm_TEST,
```

6. and then change the file 『Makefile.in』in『libs/pgpcdk/priv/crypto/hash』directory

```
add     8       pgpTEST.o
add     10      pgpTEST.h
```

7. back to libs/pgpcdk
and execute
./configure
make headers

一旦新增『HASHNUM**』成功，我們就可以將以相同的方法增加** pgpRIPEMD128、pgpRIPEMD192、pgpRIPEMD256 **的** HASHNUM**，並開始** RIPEMD-X **的演算法實作部分。**

4.8.2 Add hash function mapping HASHNUM --- RIPEMD Algorithm

需將模組程式放置以下路徑：『pgp6.5.1i/libs/pgpcdk/priv/crypto/hash/』

- Header file : pgpRIPEMD128.h

```
#ifndef Included_pgpRIPEMD128_h
#define Included_pgpRIPEMD128_h
#include "pgpSymmetricCipherPriv.h"
PGP_BEGIN_C_DECLARATIONS
extern PGPHashVTBL const HashRIPEMD128;
PGP_END_C_DECLARATIONS
#endif /* Included_pgpRIPEMD128_h */
```

- Define a Hash function module for generic hash function (main structure)

```
struct PGPHashVTBL const HashRIPEMD128 = {
"RIPEMD128",
kPGPHashAlgorithm_RIPEMD128,
RIPEMD128DERprefix,
sizeof(RIPEMD128DERprefix),
RIPEMD128_HASHBYTES,
sizeof(struct RIPEMD128Context),
sizeof(struct{char _a; struct RIPEMD128Context _b;}) -
    sizeof(struct RIPEMD128Context),
rmd128Init,
rmd128Update,
rmd128Final
};
```

- pointer priv* point a memory address **big enough** to keep subkey or key schedule boxes.
RIPEMD128_HASHBYTES : have to **point out the priv* size**
E.g

```
#define RIPEMD128_HASHBYTES    16    //output 16*8=128 bits
#define RIPEMD128_HASHWORDS   4      //output 4 words
```

• Exported **three** main functions

```
static  void      rmd128Init(void *priv)
static  void      rmd128Update(void *priv, void const *bufIn, PGPSize len)
static  void const *  rmd128Final(void *priv)
```

• change the initial register value

```
static void RMDinit(PGPUInt32 *MDbuf) {
  MDbuf[0] = 0x67452301UL;
  MDbuf[1] = 0xefcdab89UL;
  MDbuf[2] = 0x98badcfeUL;
  MDbuf[3] = 0x10325476UL;
  return;
}
```

• change the main compress function

主要的演算法不變，主要是將暫存器增加/減少及輸出部分重新排列以符合我們所需。以下以 128 bits **為例：**

```
static void RMDcompress(PGPUInt32 *MDbuf, PGPUInt32 *X) {
  PGPUInt32 aa = MDbuf[0],   bb = MDbuf[1],   cc = MDbuf[2],
            dd = MDbuf[3],   ee = MDbuf[0];
  PGPUInt32 aaa = MDbuf[1],  bbb = MDbuf[2],  ccc = MDbuf[3],
            ddd = MDbuf[0],  eee = MDbuf[1];

    ……

  /* combine results */
  ddd += cc + MDbuf[1];                 //     T=h1+C+D'
  MDbuf[1] = MDbuf[2] + dd + eee;       //     h1=h2+D+E'
  MDbuf[2] = MDbuf[3] + ee + aaa;       //     h2=h3+E+A'
  MDbuf[3] = MDbuf[0] + aa + bbb;       //     h3=h0+A+B'
  MDbuf[0] = ddd;                       //     h0=T

  return;
}
```

4.8.3 pgpRIPEMD128 的實作

- pgpRIPEMD128.h

```
#ifndef Included_pgpRIPEMD128_h
#define Included_pgpRIPEMD128_h

#include "pgpHashPriv.h"

PGP_BEGIN_C_DECLARATIONS

extern PGPHashVTBL const HashRIPEMD128;

PGP_END_C_DECLARATIONS

#endif /* !Included_pgpRIPEMD128_h */
```

- pgpRIPEMD128.c

```
#include "pgpConfig.h"

#include <string.h>

#include "pgpHash.h"
#include "pgpRIPEMD128.h"
#include "pgpUsuals.h"
#include "pgpMem.h"
#include "pgpDebug.h"

#define RIPEMD128_BLOCKBYTES 64
#define RIPEMD128_BLOCKWORDS        16

#define RIPEMD128_HASHBYTES    16     //output 16*8=128 bits
#define RIPEMD128_HASHWORDS  4       //output 4 words

typedef struct RIPEMD128Context {
    PGPUInt32 key[RIPEMD128_BLOCKWORDS];
    PGPUInt32 iv[RIPEMD128_HASHWORDS];
    PGPUInt32 bytesHi, bytesLo;
    DEBUG_STRUCT_CONSTRUCTOR( RIPEMD128Context )
} RIPEMD128Context;



/********************** File rmd128.h **********************/
/* Extracted from rmd128.h in the reference implementation */

/* macro definitions */

/* ROL(x, n) cyclically rotates x over n bits to the left */
/* x must be of an unsigned 32 bits type and 0 <= n < 32. */
#define ROL(x, n)       (((x) << (n)) | ((x) >> (32-(n))))

/* the three basic functions F(), G() and H() */
#define F(x, y, z)       ((x) ^ (y) ^ (z))
#define G(x, y, z)       (((x) & (y)) | (~(x) & (z)))
#define H(x, y, z)       (((x) | ~(y)) ^ (z))
#define I(x, y, z)       (((x) & (z)) | ((y) & ~(z)))
#define J(x, y, z)       ((x) ^ ((y) | ~(z)))
```

```
/* the eight basic operations FF() through III() */
#define FF(a, b, c, d, e, x, s)        {\
     (a) += F((b), (c), (d)) + (x);\
     (a) = ROL((a), (s)) + (e);\
     (c) = ROL((c), 10);\
   }
#define GG(a, b, c, d, e, x, s)        {\
     (a) += G((b), (c), (d)) + (x) + 0x5a827999UL;\
     (a) = ROL((a), (s)) + (e);\
     (c) = ROL((c), 10);\
   }
#define HH(a, b, c, d, e, x, s)        {\
     (a) += H((b), (c), (d)) + (x) + 0x6ed9eba1UL;\
     (a) = ROL((a), (s)) + (e);\
     (c) = ROL((c), 10);\
   }
#define II(a, b, c, d, e, x, s)        {\
     (a) += I((b), (c), (d)) + (x) + 0x8f1bbcdcUL;\
     (a) = ROL((a), (s)) + (e);\
     (c) = ROL((c), 10);\
   }
#define JJ(a, b, c, d, e, x, s)        {\
     (a) += J((b), (c), (d)) + (x) + 0xa953fd4eUL;\
     (a) = ROL((a), (s)) + (e);\
     (c) = ROL((c), 10);\
   }
#define FFF(a, b, c, d, e, x, s)       {\
     (a) += F((b), (c), (d)) + (x);\
     (a) = ROL((a), (s)) + (e);\
     (c) = ROL((c), 10);\
   }
#define GGG(a, b, c, d, e, x, s)       {\
     (a) += G((b), (c), (d)) + (x) + 0x7a6d76e9UL;\
     (a) = ROL((a), (s)) + (e);\
     (c) = ROL((c), 10);\
   }
#define HHH(a, b, c, d, e, x, s)       {\
     (a) += H((b), (c), (d)) + (x) + 0x6d703ef3UL;\
     (a) = ROL((a), (s)) + (e);\
     (c) = ROL((c), 10);\
   }
#define III(a, b, c, d, e, x, s)       {\
     (a) += I((b), (c), (d)) + (x) + 0x5c4dd124UL;\
     (a) = ROL((a), (s)) + (e);\
     (c) = ROL((c), 10);\
   }
#define JJJ(a, b, c, d, e, x, s)       {\
     (a) += J((b), (c), (d)) + (x) + 0x50a28be6UL;\
     (a) = ROL((a), (s)) + (e);\
     (c) = ROL((c), 10);\
   }



/************************ File rmd128.c ************************/
/* Extracted from rmd128.c in the reference implementation */


/*
 *  initializes MDbuffer to "magic constants"
```

```
 */
static void RMDinit(PGPUInt32 *MDbuf)
{
  MDbuf[0] = 0x67452301UL;
  MDbuf[1] = 0xefcdab89UL;
  MDbuf[2] = 0x98badcfeUL;
  MDbuf[3] = 0x10325476UL;
//  MDbuf[4] = 0xc3d2e1f0UL;

  return;
}


/*
 *  the compression function.
 *  transforms MDbuf using message PGPBytes X[0] through X[15]
 */
static void RMDcompress(PGPUInt32 *MDbuf, PGPUInt32 *X)
{
  PGPUInt32 aa = MDbuf[0],  bb = MDbuf[1],  cc = MDbuf[2],
       dd = MDbuf[3],  ee = MDbuf[0];
  PGPUInt32 aaa = MDbuf[1], bbb = MDbuf[2], ccc = MDbuf[3],
       ddd = MDbuf[0], eee = MDbuf[1];

  /* round 1 */
  FF(aa, bb, cc, dd, ee, X[ 0], 11);
  FF(ee, aa, bb, cc, dd, X[ 1], 14);
  FF(dd, ee, aa, bb, cc, X[ 2], 15);
  FF(cc, dd, ee, aa, bb, X[ 3], 12);
  FF(bb, cc, dd, ee, aa, X[ 4],  5);
  FF(aa, bb, cc, dd, ee, X[ 5],  8);
  FF(ee, aa, bb, cc, dd, X[ 6],  7);
  FF(dd, ee, aa, bb, cc, X[ 7],  9);
  FF(cc, dd, ee, aa, bb, X[ 8], 11);
  FF(bb, cc, dd, ee, aa, X[ 9], 13);
  FF(aa, bb, cc, dd, ee, X[10], 14);
  FF(ee, aa, bb, cc, dd, X[11], 15);
  FF(dd, ee, aa, bb, cc, X[12],  6);
  FF(cc, dd, ee, aa, bb, X[13],  7);
  FF(bb, cc, dd, ee, aa, X[14],  9);
  FF(aa, bb, cc, dd, ee, X[15],  8);

  /* round 2 */
  GG(ee, aa, bb, cc, dd, X[ 7],  7);
  GG(dd, ee, aa, bb, cc, X[ 4],  6);
  GG(cc, dd, ee, aa, bb, X[13],  8);
  GG(bb, cc, dd, ee, aa, X[ 1], 13);
  GG(aa, bb, cc, dd, ee, X[10], 11);
  GG(ee, aa, bb, cc, dd, X[ 6],  9);
  GG(dd, ee, aa, bb, cc, X[15],  7);
  GG(cc, dd, ee, aa, bb, X[ 3], 15);
  GG(bb, cc, dd, ee, aa, X[12],  7);
  GG(aa, bb, cc, dd, ee, X[ 0], 12);
  GG(ee, aa, bb, cc, dd, X[ 9], 15);
  GG(dd, ee, aa, bb, cc, X[ 5],  9);
  GG(cc, dd, ee, aa, bb, X[ 2], 11);
  GG(bb, cc, dd, ee, aa, X[14],  7);
  GG(aa, bb, cc, dd, ee, X[11], 13);
  GG(ee, aa, bb, cc, dd, X[ 8], 12);
```

```
/* round 3 */
HH(dd, ee, aa, bb, cc, X[ 3], 11);
HH(cc, dd, ee, aa, bb, X[10], 13);
HH(bb, cc, dd, ee, aa, X[14],  6);
HH(aa, bb, cc, dd, ee, X[ 4],  7);
HH(ee, aa, bb, cc, dd, X[ 9], 14);
HH(dd, ee, aa, bb, cc, X[15],  9);
HH(cc, dd, ee, aa, bb, X[ 8], 13);
HH(bb, cc, dd, ee, aa, X[ 1], 15);
HH(aa, bb, cc, dd, ee, X[ 2], 14);
HH(ee, aa, bb, cc, dd, X[ 7],  8);
HH(dd, ee, aa, bb, cc, X[ 0], 13);
HH(cc, dd, ee, aa, bb, X[ 6],  6);
HH(bb, cc, dd, ee, aa, X[13],  5);
HH(aa, bb, cc, dd, ee, X[11], 12);
HH(ee, aa, bb, cc, dd, X[ 5],  7);
HH(dd, ee, aa, bb, cc, X[12],  5);

/* round 4 */
II(cc, dd, ee, aa, bb, X[ 1], 11);
II(bb, cc, dd, ee, aa, X[ 9], 12);
II(aa, bb, cc, dd, ee, X[11], 14);
II(ee, aa, bb, cc, dd, X[10], 15);
II(dd, ee, aa, bb, cc, X[ 0], 14);
II(cc, dd, ee, aa, bb, X[ 8], 15);
II(bb, cc, dd, ee, aa, X[12],  9);
II(aa, bb, cc, dd, ee, X[ 4],  8);
II(ee, aa, bb, cc, dd, X[13],  9);
II(dd, ee, aa, bb, cc, X[ 3], 14);
II(cc, dd, ee, aa, bb, X[ 7],  5);
II(bb, cc, dd, ee, aa, X[15],  6);
II(aa, bb, cc, dd, ee, X[14],  8);
II(ee, aa, bb, cc, dd, X[ 5],  6);
II(dd, ee, aa, bb, cc, X[ 6],  5);
II(cc, dd, ee, aa, bb, X[ 2], 12);

/* round 5 */
JJ(bb, cc, dd, ee, aa, X[ 4],  9);
JJ(aa, bb, cc, dd, ee, X[ 0], 15);
JJ(ee, aa, bb, cc, dd, X[ 5],  5);
JJ(dd, ee, aa, bb, cc, X[ 9], 11);
JJ(cc, dd, ee, aa, bb, X[ 7],  6);
JJ(bb, cc, dd, ee, aa, X[12],  8);
JJ(aa, bb, cc, dd, ee, X[ 2], 13);
JJ(ee, aa, bb, cc, dd, X[10], 12);
JJ(dd, ee, aa, bb, cc, X[14],  5);
JJ(cc, dd, ee, aa, bb, X[ 1], 12);
JJ(bb, cc, dd, ee, aa, X[ 3], 13);
JJ(aa, bb, cc, dd, ee, X[ 8], 14);
JJ(ee, aa, bb, cc, dd, X[11], 11);
JJ(dd, ee, aa, bb, cc, X[ 6],  8);
JJ(cc, dd, ee, aa, bb, X[15],  5);
JJ(bb, cc, dd, ee, aa, X[13],  6);

/* parallel round 1 */
JJJ(aaa, bbb, ccc, ddd, eee, X[ 5],  8);
JJJ(eee, aaa, bbb, ccc, ddd, X[14],  9);
JJJ(ddd, eee, aaa, bbb, ccc, X[ 7],  9);
JJJ(ccc, ddd, eee, aaa, bbb, X[ 0], 11);
JJJ(bbb, ccc, ddd, eee, aaa, X[ 9], 13);
```

```
JJJ(aaa, bbb, ccc, ddd, eee, X[ 2], 15);
JJJ(eee, aaa, bbb, ccc, ddd, X[11], 15);
JJJ(ddd, eee, aaa, bbb, ccc, X[ 4],  5);
JJJ(ccc, ddd, eee, aaa, bbb, X[13],  7);
JJJ(bbb, ccc, ddd, eee, aaa, X[ 6],  7);
JJJ(aaa, bbb, ccc, ddd, eee, X[15],  8);
JJJ(eee, aaa, bbb, ccc, ddd, X[ 8], 11);
JJJ(ddd, eee, aaa, bbb, ccc, X[ 1], 14);
JJJ(ccc, ddd, eee, aaa, bbb, X[10], 14);
JJJ(bbb, ccc, ddd, eee, aaa, X[ 3], 12);
JJJ(aaa, bbb, ccc, ddd, eee, X[12],  6);

/* parallel round 2 */
III(eee, aaa, bbb, ccc, ddd, X[ 6],  9);
III(ddd, eee, aaa, bbb, ccc, X[11], 13);
III(ccc, ddd, eee, aaa, bbb, X[ 3], 15);
III(bbb, ccc, ddd, eee, aaa, X[ 7],  7);
III(aaa, bbb, ccc, ddd, eee, X[ 0], 12);
III(eee, aaa, bbb, ccc, ddd, X[13],  8);
III(ddd, eee, aaa, bbb, ccc, X[ 5],  9);
III(ccc, ddd, eee, aaa, bbb, X[10], 11);
III(bbb, ccc, ddd, eee, aaa, X[14],  7);
III(aaa, bbb, ccc, ddd, eee, X[15],  7);
III(eee, aaa, bbb, ccc, ddd, X[ 8], 12);
III(ddd, eee, aaa, bbb, ccc, X[12],  7);
III(ccc, ddd, eee, aaa, bbb, X[ 4],  6);
III(bbb, ccc, ddd, eee, aaa, X[ 9], 15);
III(aaa, bbb, ccc, ddd, eee, X[ 1], 13);
III(eee, aaa, bbb, ccc, ddd, X[ 2], 11);

/* parallel round 3 */
HHH(ddd, eee, aaa, bbb, ccc, X[15],  9);
HHH(ccc, ddd, eee, aaa, bbb, X[ 5],  7);
HHH(bbb, ccc, ddd, eee, aaa, X[ 1], 15);
HHH(aaa, bbb, ccc, ddd, eee, X[ 3], 11);
HHH(eee, aaa, bbb, ccc, ddd, X[ 7],  8);
HHH(ddd, eee, aaa, bbb, ccc, X[14],  6);
HHH(ccc, ddd, eee, aaa, bbb, X[ 6],  6);
HHH(bbb, ccc, ddd, eee, aaa, X[ 9], 14);
HHH(aaa, bbb, ccc, ddd, eee, X[11], 12);
HHH(eee, aaa, bbb, ccc, ddd, X[ 8], 13);
HHH(ddd, eee, aaa, bbb, ccc, X[12],  5);
HHH(ccc, ddd, eee, aaa, bbb, X[ 2], 14);
HHH(bbb, ccc, ddd, eee, aaa, X[10], 13);
HHH(aaa, bbb, ccc, ddd, eee, X[ 0], 13);
HHH(eee, aaa, bbb, ccc, ddd, X[ 4],  7);
HHH(ddd, eee, aaa, bbb, ccc, X[13],  5);

/* parallel round 4 */
GGG(ccc, ddd, eee, aaa, bbb, X[ 8], 15);
GGG(bbb, ccc, ddd, eee, aaa, X[ 6],  5);
GGG(aaa, bbb, ccc, ddd, eee, X[ 4],  8);
GGG(eee, aaa, bbb, ccc, ddd, X[ 1], 11);
GGG(ddd, eee, aaa, bbb, ccc, X[ 3], 14);
GGG(ccc, ddd, eee, aaa, bbb, X[11], 14);
GGG(bbb, ccc, ddd, eee, aaa, X[15],  6);
GGG(aaa, bbb, ccc, ddd, eee, X[ 0], 14);
GGG(eee, aaa, bbb, ccc, ddd, X[ 5],  6);
GGG(ddd, eee, aaa, bbb, ccc, X[12],  9);
GGG(ccc, ddd, eee, aaa, bbb, X[ 2], 12);
```

```
  GGG(bbb, ccc, ddd, eee, aaa, X[13],  9);
  GGG(aaa, bbb, ccc, ddd, eee, X[ 9], 12);
  GGG(eee, aaa, bbb, ccc, ddd, X[ 7],  5);
  GGG(ddd, eee, aaa, bbb, ccc, X[10], 15);
  GGG(ccc, ddd, eee, aaa, bbb, X[14],  8);

  /* parallel round 5 */
  FFF(bbb, ccc, ddd, eee, aaa, X[12] ,  8);
  FFF(aaa, bbb, ccc, ddd, eee, X[15] ,  5);
  FFF(eee, aaa, bbb, ccc, ddd, X[10] , 12);
  FFF(ddd, eee, aaa, bbb, ccc, X[ 4] ,  9);
  FFF(ccc, ddd, eee, aaa, bbb, X[ 1] , 12);
  FFF(bbb, ccc, ddd, eee, aaa, X[ 5] ,  5);
  FFF(aaa, bbb, ccc, ddd, eee, X[ 8] , 14);
  FFF(eee, aaa, bbb, ccc, ddd, X[ 7] ,  6);
  FFF(ddd, eee, aaa, bbb, ccc, X[ 6] ,  8);
  FFF(ccc, ddd, eee, aaa, bbb, X[ 2] , 13);
  FFF(bbb, ccc, ddd, eee, aaa, X[13] ,  6);
  FFF(aaa, bbb, ccc, ddd, eee, X[14] ,  5);
  FFF(eee, aaa, bbb, ccc, ddd, X[ 0] , 15);
  FFF(ddd, eee, aaa, bbb, ccc, X[ 3] , 13);
  FFF(ccc, ddd, eee, aaa, bbb, X[ 9] , 11);
  FFF(bbb, ccc, ddd, eee, aaa, X[11] , 11);

  /* combine results */
  ddd += cc + MDbuf[1];               //      T=h1+C+D'
  MDbuf[1] = MDbuf[2] + dd + eee;     //      h1=h2+D+E'
  MDbuf[2] = MDbuf[3] + ee + aaa;     //      h2=h3+E+A'
  MDbuf[3] = MDbuf[0] + aa + bbb;     //      h3=h0+A+B'
  MDbuf[0] = ddd;                     //      h0=T

  return;
}


/*
 *  puts bytes from strptr into X and pad out; appends length
 *  and finally, compresses the last block(s)
 *  note: length in bits == 8 * (lswlen + 2^32 mswlen).
 *  note: there are (lswlen mod 64) bytes left in strptr.
 */
static void RMDfinish(PGPUInt32 *MDbuf, PGPByte *strptr, PGPUInt32 lswlen,
    PGPUInt32 mswlen)
{
  PGPUInt32       i;                  /* counter       */
  PGPUInt32       X[16];                  /* message words */

  pgpClearMemory( X,  16*sizeof(PGPUInt32));

  /* put bytes from strptr into X */
  for (i=0; i<(lswlen&63); i++) {
    /* byte i goes into word X[i div 4] at pos.  8*(i mod 4)  */
    X[i>>2] ^= (PGPUInt32) *strptr++ << (8 * (i&3));
  }

  /* append the bit m_n == 1 */
  X[(lswlen>>2)&15] ^= (PGPUInt32)1 << (8*(lswlen&3) + 7);

  if ((lswlen & 63) > 55) {
    /* length goes to next block */
```

```
    RMDcompress(MDbuf, X);
     pgpClearMemory( X,  16*sizeof(PGPUInt32));
  }

  /* append length in bits*/
  X[14] = lswlen << 3;
  X[15] = (lswlen >> 29) | (mswlen << 3);
  RMDcompress(MDbuf, X);

  return;
}

/************************ end of file rmd128.c ************************/
/* Remainder provides common interface used by PGP library */


/*
 * Shuffle the bytes into little-endian order within 32-bit words,
 * as per the RIPEMD-128 spec (which follows MD4 conventions).
 */
static void
rmd128ByteSwap(PGPUInt32 *dest, PGPByte const *src, unsigned words)
{
    do {
        *dest++ = (PGPUInt32)((unsigned)src[3] << 8 | src[2]) << 16 |
                    ((unsigned)src[1] << 8 | src[0]);
        src += 4;
    } while (--words);
}


/* Initialize the RIPEMD-128 values */

static void
rmd128Init(void *priv)
{
    struct RIPEMD128Context *ctx = (struct RIPEMD128Context *)priv;

    /* Set the h-vars to their initial values */
    RMDinit (ctx->iv);

    /* Initialise bit count */
    ctx->bytesHi = 0;
    ctx->bytesLo = 0;
}

/* Update the RIPEMD-128 hash state for a block of data. */

static void
rmd128Update(void *priv, void const *bufIn, PGPSize len)
{
    struct RIPEMD128Context *ctx = (struct RIPEMD128Context *)priv;
    unsigned i;
    PGPByte *buf = (PGPByte *) bufIn;

    /* Update bitcount */

    PGPUInt32 t = ctx->bytesLo;
    if ( ( ctx->bytesLo = t + len ) < t )
        ctx->bytesHi++;  /* Carry from low to high */
```

```
    i = (unsigned)t % RIPEMD128_BLOCKBYTES; /* bytes already in ctx->key */

    /* i is always less than RIPEMD128_BLOCKBYTES. */
    if (RIPEMD128_BLOCKBYTES-i > len) {
        memcpy((PGPByte *)ctx->key + i, buf, len);
        return;
    }

    if (i) {        /* First chunk is an odd size */
        memcpy((PGPByte *)ctx->key + i, buf, RIPEMD128_BLOCKBYTES - i);
        rmd128ByteSwap(ctx->key, (PGPByte *)ctx->key, RIPEMD128_BLOCKWORDS);
        RMDcompress(ctx->iv, ctx->key);
        buf += RIPEMD128_BLOCKBYTES-i;
        len -= RIPEMD128_BLOCKBYTES-i;
    }

    /* Process data in 64-byte chunks */
    while (len >= RIPEMD128_BLOCKBYTES) {
        rmd128ByteSwap(ctx->key, buf, RIPEMD128_BLOCKWORDS);
        RMDcompress(ctx->iv, ctx->key);
        buf += RIPEMD128_BLOCKBYTES;
        len -= RIPEMD128_BLOCKBYTES;
    }

    /* Handle any remaining bytes of data. */
    if (len)
        memcpy(ctx->key, buf, len);
}


/* Final wrapup - MD4 style padding on last block. */

static void const *
rmd128Final(void *priv)
{
    struct RIPEMD128Context *ctx = (struct RIPEMD128Context *)priv;
    PGPByte *digest;
    int i;
    PGPUInt32 t;

    RMDfinish(ctx->iv, (PGPByte *)ctx->key, ctx->bytesLo, ctx->bytesHi);

    digest = (PGPByte *)ctx->iv;
    for (i = 0; i < RIPEMD128_HASHWORDS; i++) {
        t = ctx->iv[i];
        digest[0] = (PGPByte)t;
        digest[1] = (PGPByte)(t >> 8);
        digest[2] = (PGPByte)(t >> 16);
        digest[3] = (PGPByte)(t >> 24);
        digest += 4;
    }
    /* In case it's sensitive */
/* XXX   pgpClearMemory( ctx, sizeof(ctx)); */

    return (PGPByte const *)ctx->iv;
}
```

```
/*
 * RIPEM OID is 1.3.36.3.2.1, from URL above.
 * The rest of the format is stolen from MD5.  Do we need the NULL
 * in there?
 */
static PGPByte const RIPEMD128DERprefix[] = {
    0x30, /* Universal, Constructed, Sequence */
    0x21, /* Length 33 (bytes following) */
        0x30, /* Universal, Constructed, Sequence */
        0x09, /* Length 9 */
            0x06, /* Universal, Primitive, object-identifier */
            0x05, /* Length 8 */
                43, /* 43 = ISO(1)*40 + 3 */
                36,
                3,
                2,
                1,
            0x05, /* Universal, Primitive, NULL */
            0x00, /* Length 0 */
        0x04, /* Universal, Primitive, Octet string */
        0x14 /* Length 20 */
            /* 20 RIPEMD-128 digest bytes go here */
};

struct PGPHashVTBL const HashRIPEMD128 = {
    "RIPEMD128",
    kPGPHashAlgorithm_RIPEMD128,
    RIPEMD128DERprefix,
    sizeof(RIPEMD128DERprefix),
    RIPEMD128_HASHBYTES,
    sizeof(struct RIPEMD128Context),
    sizeof(struct{char _a; struct RIPEMD128Context _b;}) -
        sizeof(struct RIPEMD128Context),
    rmd128Init,
    rmd128Update,
    rmd128Final
};
```

4.8.4 pgpRIPEMD192 及 pgpRIPEMD256 的實作

由於實作方式與 pgpRIPEMD128 **極為相似，所以只要根據** 3.7.2、4.8.1 **及** 4.8.2 **所述，即可增加此兩種變形雜湊函數。此外必須注意一點，因為** RIPEMD192、RIPEMD256 **都增加了暫存器的數量，因此這些暫存器的初始直我們也必須加以定義。**

- RIPEMD192：

```
static void RMDinit(PGPUInt32 *MDbuf) {
    MDbuf[0] = 0x67452301UL;
    MDbuf[1] = 0xefcdab89UL;
    MDbuf[2] = 0x98badcfeUL;
    MDbuf[3] = 0x10325476UL;
    MDbuf[4] = 0xc3d2e1f0UL;
    MDbuf[5] = 0x7c4a19b4UL;
    return;
```

```
}
```

- RIPEMD256：

```
static void RMDinit(PGPUInt32 *MDbuf) {
  MDbuf[0] = 0x67452301UL;
  MDbuf[1] = 0xefcdab89UL;
  MDbuf[2] = 0x98badcfeUL;
  MDbuf[3] = 0x10325476UL;
  MDbuf[4] = 0xc3d2e1f0UL;
  MDbuf[5] = 0x7c4a19b4UL;
  MDbuf[6] = 0x639ba0e2UL;
  MDbuf[7] = 0xc804abf5UL;
    return;
}
```

4.9 新增演算法的 Mapping Number

4.9.1 對稱式加密演算法

本計劃主要新增 AESII **的加密演算法，以下為新增的列表：**

Rijndael ：ciphernum = 4 ；
Serpent ：ciphernum = 5 ；
Twofish ：ciphernum = 6 ；
RC6 ：ciphernum = 7 ；
MARS ：ciphernum = 8 。

4.9.2 RIPEMD-X **改良型雜湊函數**

本計劃主要改良 RIPEMD-160 **成為** RIPEMD-X**，以下為新增模組的列表：**

RIPEMD128 ： = 4 ；
RIPEMD192 ： = 6 ；
RIPEMD256 ： = 7 。

5. 結論

本計劃主要是改良 PGP 以更符合未來的需求，以往的加密系統已不安全，而 PGP 系統又是許多人使用的加解密軟體, 因此我們選擇更換及改良 PGP 系統內的加解密及雜湊函數的模組來作為本次計劃的主要目標。

PGP 系統本身的功能完備，所以涵蓋的 API 甚廣，因此如何在此系統找到進入點也成為本計劃的重點之一；另外，找到進入點又如何將提出的演算法改寫成 PGP 系統可接受的格式也是另一重要課題。所以，本計劃的其中一個目的也是『研究 PGP 如何接受其他的演算法』。

了解 PGP 如何呼叫相對應的函數後, 我們就以提出的演算法(AESII, 包括 Rijndael、Serpent、Twofish、RC6、Mars)來替換原先的對稱加密演算法及新增 RIPEMD 的變形函數---RIPEMD-X(RIPME128、RIPEMD192、RIPEMD256)。

最後，也希望經過本計劃的研究及實作，真正為 PGP 系統增加安全度也更因應未來的所需。

6. 參考文獻

[1] Online Help, “PGPsdkReferenceGuide”, pgp6.5.1i/docs.

[2] Online Help, “PGPsdkUsersGuide”, pgp6.5.1i/docs.

[3] Online Help, “IntrotoCrypto”, pgp6.5.1i/docs.

[4] Online Help, “PGPCmdLineGuide”, pgp6.5.1i/docs.

[5] R.L. Rivest, “The MD5 Message Digest Algorithm,” RFC 1321, Apr 1992.

[6] National Institute of Standards and Technology, NIST FIPS PUB 186, “Digital Signature Standard,” U.S. Department of Commerce, May 1994.

[7] R.L. Rivest, A. Shamir, and L.M. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,” Communications of the ACM, v. 21, n. 2, Feb 1978, pp. 120-126.

[8] H.W. Lenstra Jr. “Elliptic Curves and Number-Theoretic Algorithms,” Report 86-19, Mathematisch Instituut, Universiteit van Amsterdam, 1986.

[9] X. Lai, “On the Design and Security of Block Ciphers,” ETH Series in Information Processing, v. 1, Konstanz: Hartung-Gorre Verlag, 1992.

[10] Ronald L. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Yin, “The RC6 Block Cipher,” by http://theory.lcs.mit.edu/~rivest/rc6.ps

[11] P.R. Zimmermann, “The Offical PGP User’s Guide,” Boston: MIT Press, 1995.

[12] **賴溪松，韓亮，武怡先，**”**網路安全**—PGP **的原理、安裝與操作**”**，台北，財團法人資訊工業策進會 資訊與電腦出版社，**1997**。**

[13] **交大密碼模組** by http://crypto13.csie.nctu.edu.tw

[14] Mike Lonkides & Andy Oram, “GNU **程式設計**”, O’Reilly, Oct 1998.

[15] GNU GPL **相關說明**, by http://www.gnu.org

[16] RSA Laboratories, “The RC6 Block Cipher”, by http://www.rsasecurity.com/rsalabs/aes

[17] Joan Daemen, Vincent Rijmen, “The Rijndael Block Cipher”, by AES Document version2, by http://www.esat.kuleuven.ac.be/~rijmen/rijndael/

[18] Ross Anderson, Eli Biham, Lars Knudsen, “Serpent : A Proposal for the Advanced Encryption Standard”, by http://www.cl.cam.ac.uk/~rja14/serpent.html

[19] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, “Twofish : A 128-Bit Block Cipher”, by http://www.counterpane.com/twofish.html, 15 June 1998

[20] IBM Corporation, “MARS – a candidate cipher for AES”, by http://www.research.ibm.com/security/mars.html, September 22 1999

[21] **謝志敏，葉義雄，周志賢，**”**資訊安全通訊**”**，第五卷第三期** 88.06

[22] Bruce Schneier, “Applied Cryptography,” John Wiley &Sons, Inc., 1996.