東海大學電機工程學系

碩士論文

以軟硬體參數為基礎之資料加密法

Data Encryption based on Hardware and Software Parameters

研究生：張鉦新

指導教授：蔡坤霖 博士

中華民國 107 年 5 月

東海大學電機工程學系碩士學位

考試委員審定書

電機工程學系研究所 張鉦新 君所提之論文，

以軟硬體參數為基礎之資料加密法 ，經本考試

委員會審查，符合碩士資格標準。

學位考試委員會 召集人：張西洲 (簽章)

委 員：蘇賀宏

張延任

蔡坤霖

中 華 民 國 107 年 05 月 14 日

致謝

感謝指導教授 蔡坤霖博士，在就學期間細心指導與提供建議，以及感謝張孟洲教授、張延任教授、蔡舜宏教授在論文口試時給了許多寶貴的意見。另外感謝同學昱元、益豪、育甄在遇到問題時，一同思考解決方法，以及學妹郁淩、雅昕的幫忙。最後感謝小真與家人一直給予我督促、鼓勵和支持。

摘要

隨著科技發展，電子設備使用頻率越來越高。包含個人、政府、企業機構在內，大量的資料被放置於電子儲存設備中。然而，在網路發達的現代社會，不論是儲存在雲端電腦，或者個人電腦的資料，若未經保護，極有可能被駭客經由各種方式竊取。因此，本論文提出以軟硬體參數為基礎的資料加密方法，透過儲存設備本身的軟硬體參數建構一組環境金鑰，再透過此環境金鑰對檔案加密。在我們所提出來的方法中，被加密後的檔案只能在原有的儲存設備中解密，可確保重要檔案不慎被竊取時，無法在其他裝置中被解密。經由我們實作的結果，證實此方式能確實提供安全性高的保護機制。

關鍵字：資料洩漏、惡意攻擊、安全性、資料加密、軟硬體參數

Abstract

With the development of technology, electronic devices are used frequently in human beings' daily lives. A large amount of data is stored in various electronic devices, e.g. personal computer, smart phone, cloud device, etc. However, without encryption, these data and files may be hacked by hackers or leaked out by insiders easily. The loss of important data in the cloud may result in huge personal/business-reputation damages, credit loss, and pecuniary loss. In this paper, we propose a data encryption method based on hardware and software parameters. An environmental key is created by utilizing trusted device's hardware as well as software parameters. The important files are then encrypted by using the environmental key. In the proposed method, the encrypted files can only be decrypted in the same trusted device. Once the encrypted files are illegal copied to another untrusted device, the files cannot be decrypted since the environmental key is different. According to the experimental result, the proposed method is verified and suitable for important data encryption.

Keywords: Data leakage, Malicious attack, Security, Data encryption, Hardware/Software parameter

目錄

圖目錄

第一章 緒論

1.1 研究背景

在過去幾十年裡，人們都依賴電子設備儲存資料。特別是企業及政府的資料更是大量用電子設備存取。然而，被竊取的問題也越來越嚴重，不論是個人資料、機密文件，雖然資料儲存在電子設備有很高的便利性，但若是沒有數據加密也代表著高洩漏風險。

資料加密[1]可以分為對稱式加密[2]與非對稱式加密[3]。其中，對稱加密的特點是加密與解密都是使用同一把密鑰，例如著名的 DES[4] 與 AES 等皆是。而非對稱加密則是加解密分別各有一把密鑰，常見的有 DSA[5]與 RSA 等加密方法。

現代人們因電子設備及網際網路而必須為儲存的隱私作保護，在 1975 年美國國家標準局頒佈加密標準[6]，作為計算機系統和網際網路應用的資料加密標準。而最早的對稱加密為 DES，在使用 30 年之後，由於安全考量，漸漸被新一代加密標準 AES 所取代。

1.2 研究目的

如今，資料洩漏[7]是不同基礎產業必須共同面對的課題之一，外洩的管道非常多元，從電子郵件[8]、應用程式[9]到雲端儲存[10]及隨身碟[11]等等，都是資料外洩的管道。外洩私人資料比較常見的就是個人設備被家人、親戚或服務提供商經維修時非法竊取；而企業或政府機密文件則是經由公司內人員未經同意以可攜式裝置攜出。

因此，本研究中提出了針對上述的防盜方案，命名為以軟硬體參數為基礎之資料加密法(Data Encryption based on Hardware Parameters)，以電腦主機或其他裝置內的各個軟硬體資訊對資料加密，達到把資料綁定在當前電腦，若是以不同電腦解密，則會因為軟硬體資訊不同而無法正確開啟，進而確保資料不外洩。

1.3 章節安排

本論文共分五章，第一章為緒論，此章節包含研究背景、研究目的以及章節安排。第二章為本論文相關的研究探討，介紹各類型常見的加密技術以及與本論文研究相關的論文探討。第三章為本論文的主軸，說明環境金鑰的建構流程，以及加解密流程。第四章將實現的檔案加密技術實作並分析其安全性，最後在第五章做總結。

第二章 相關加密技術

2.1 對稱式加密

1973 年，美國國家標準局（NBS，National Bureau of Standards）現為美國國家標準技術研究所（National Institute of Standards and Technology，NIST），徵求國家密碼標準方案，IBM 公司提交了自己研製的演算法（Luciffer 演算法[12]，於 1971 年末提出）。 1977 年 7 月 15 日，該演算法被正式採納，作為美國聯邦訊息處理標準生效，並很快應用到國際商用資料加密領域，成為標準，即資料加密標準（Data Encryption Standard，DES），DES 演算法由此誕生。

DES 演算法作為現代密碼學領域中第一個官方授權的加密演算法受到全球各大密碼學機構的關注。初期 DES 演算法密鑰較短，僅 56 bits，迭代次數偏少，受到諸如差分密碼分析（Differential Cryptanalysis）[13]和線性密碼分析（Linear Cryptanalysis）[14]等各種攻擊威脅，安全性受到嚴重威脅。1998 年後，實用化 DES 演算法破譯機[15]的出現徹底宣告 DES 演算法已不具備安全性。1999 年 NIST 頒布新標準，規定 DES 演算法只能用於遺留加密系統，但不限制使 TDEA[16]演算法。以當今計算機技術能力，經 DES 演算法加密的資料在 24 小時內可能被破解[17]。由此，DES 演算法正式退出舞台成為歷史，AES 演算法成為它的替代者。即便如此，DES 演算法對於密碼學領域的貢獻確實是巨大的。各種對稱式加密演算法均由研究 DES 演算法發展而來，對後續對稱式加密演算法的發展起到奠基作用。DES 演算法實作不僅遍布軟體業，甚至很多硬體晶片本身也具備 DES 加密實作。

對於大多數對稱式加密演算法而言，解密演算法是加密演算法的逆運算，加密密鑰和解密密鑰相同。如果將 Base64 演算法[18]改良，將其字元映射成表作為密鑰保存，就可以把這個改良後的 Base64 演算法作為一種對稱式加密演算法來看。當然，加密演算法這樣改良後的安全強度還遠遠不夠，但足以讓我們認識對稱式加密演算法的特點。對稱式加密演算法易於理解，便於實作，根據加密方式又分為串流加

密[19]和分組加密[20]，其分組加密工作模式又可分為 ECB、CBC、PCBC、CFB、OFB 和 CTR，密鑰長度決定了加密演算法的安全性。對稱式加密演算法發展至今已相當完備。以最早期的 DES 演算法為例，由於密鑰長度的不足夠，衍生出了 TDEA 演算法。為了替代 DES 演算法又有了 AES 演算法。此外，還有 RC 系列演算法，包含 RC2[21]、RC4[22]、針對 32 位元及 64 位元電腦設計的 RC5[23]及 RC6[24]。除了上述演算法，較常見的還有 Blowfish、Twofish[25]和 Serpent[26]等對稱式加密演算法。

2.1.1 AES[27]

Advanced Encryption Standard(AES)，架構如圖 1。在 2001 年 11 月 26 日由 NIST 發布於 FIPS PUB 197。與 DES 不同的是使用 Substitution-Permutation Network(SPN)[28] 的結構而非 Feistel cipher[29]，其使用的密鑰可以是 128、192、256 bits，分別對應 10、12、14 rounds，加密前先將明文每 128 bits 分割成 1 個 block，在加密過程中以 4×4 的矩陣上運作，這個矩陣又稱為 state ，每個矩陣上的元素都對應明文的 1 個 byte，加密時每 1 round 均含有 4 個函數：SubBytes、ShiftRows、MixColumns 以及 AddRoundKey。

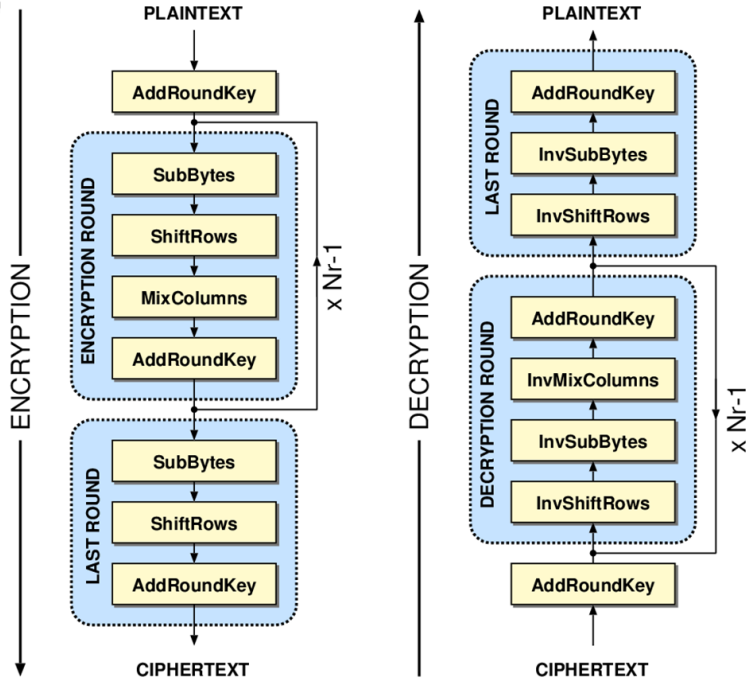


圖 1　AES 架構圖

1. SubBytes : 是一種非線性的轉換，將每 1 byte 對照 S-box[30]而得到相應的結果，必須使用可逆的 S-box，否則無法解密。

2. ShiftRows : 每一行向左並循環位移一固定值，例如：第一行不位移，第二行位移 1byte，第三行位移 2bytes，第四行位移 3bytes。

3. MixColumns : 以 Rijndael MixColumns[31]的方式為每一列四個 byte 分別定義成多項式

$$s(x) = b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0$$

的係數 b_3 、b_2 、b_1 、b_0 ，並且乘上一常數多項式

$$a(x) = 3x^3 + 1x^2 + 1x^1 + 2x^0$$

，若發生溢位則同餘(x^4+1)，可得到 $s'(x)$，意即

$$s'(x)=\alpha(x) \otimes s(x)$$

其中⊗表示乘法並 mod (x^4+1)，再將得到的 $s'(x)$係數取代 b_3 、b_2 、b_1 、b_0。

4. AddRoundKey : 在每次的加密迴圈中，皆會由主密鑰透過 key schedule[32]產生 1 把 RoundKey，此 RoundKey 會與 states 一樣，使其能夠互相對應做 XOR 運算。

5. 直到最後一次 AddRoundKey 步驟完成，便將密文輸出。

2.1.2 Blowfish[33]

在 1993 年，Bruce Schneier 希望可以替代老化的 DES，並避免與其他算法相關的問題和限制，並成功設計出 Blowfish 。在公開之後，Bruce Schneier 表示此產品為非專利，故任何人都能自由的使用。Blowfish 在軟體中提供很好的加密速度，迄今為止還沒有發現有效的密碼攻擊，後來更是有以 Blowfish 為設計基礎的 Twofish 出現。但是在使用率上目前還是以 AES 為主流。

Blowfish 是一種使用 Block cypher、S-box、Feistel cipher 以及類

似 CAST-128[34]的對稱式加密，其密鑰長度可在 32-448bits 之間，加密的 block 須為 64bits。其算法由擴展密鑰及資料加密兩部分組成。資料加密通過一個 16 輪的 Feistel cipher，每一輪都包含一個與密鑰相關的置換，以及一個密鑰與數據相關的置換。所有操作都是 XOR 和 32bits 的加法。

資料加密：

輸入是一個 64bits 的資料 x，

將 x 分成兩個 32bits 的一半：x_L、x_R，

對於 i = 1 到 16：

1. $x_L = x_L$ XOR P_i，
2. $x_R = F(x_L)$ XOR x_R，
3. 交換 x_L 和 x_R，

並重複上述 3 步驟，直到第 16 輪的第 2 步驟完成，

$x_R = x_R$ XOR P_{17}，

$x_L = x_L$ XOR P_{18}，

重新組合 x_L 和 x_R 得到密文，除了 P_1，P_2，…，P_{18} 以相反順序使用外，解密與加密完全相同。

函數 F：

將 x_L 分成 4 個 8bits 分別為：a，b，c 和 d

$$F(x_L)=((((S_1,a + S_2,b) \bmod 2^{32}) \text{XOR } S_3,c)+ S_4,d) \bmod 2^{32}$$

擴展密鑰：

Blowfish 使用大量的子密鑰並分成 P 陣列、S 陣列。這些子密鑰必須在任何數據加密或解密之前進行預先計算。其中 P 陣列由 18 個 32bits 的子密鑰組成：P_1，P_2，...，P_{18}。而 S 陣列分成 4 個 32bits 的 S-box，每個 S-box 有 256 個輸入：

S(1,0)，S(1,1)，...，S(1,255)；

S(2,0)，S(2,1)，...，S(2,255)；

S(3,0)，S(3,1)，...，S(3,255)；

S(4,0)，S(4,1)，...，S(4,255)。

1. 用固定字符串依次初始化 P 陣列和四個 S 盒。該字符串由π(小數點以下)的十六進制數字組成。例如：

P_1 = 0x243f6a88

P_2 = 0x85a308d3

P_3 = 0x13198a2e

P_4 = 0x03707344

2. P_1 XOR 密鑰的第一組 32 位，P_2 XOR 密鑰的第二組 32 位，直到把密鑰位數用盡並循環重複使用密鑰，使整個 P 陣列與密鑰 XOR。
3. 把 1.和 2.中描述的子密鑰與全零字符串使用 Blowfish 算法加密。
4. 用 3.的輸出替換掉 P_1 、P_2
5. 使用 Blowfish 演算法對修改後的子密鑰和 3.的輸出加密
6. 把 5.的輸出結果替換掉 P_3、P_4
7. 依上敘步驟，依次替換掉所有 P 陣列的內容，然後一次替換掉

2.2非對稱式加密

1976 年由 Bailey Whitfield Diffie 以及 Martin Edward Hellman 首次發表非對稱加密演算法，稱為迪菲-赫爾曼密鑰交換(Diffie–Hellman

key exchange)[35]縮寫為 D-H。1977 年，Ron Rivest、Adi Shamir、Leonard Adleman 共同提出另一非對稱加密法並稱為 RSA 加密，此後，非對稱加密逐漸為整個加密系統提供重要貢獻，除上面提到的兩種演算法，還有如：DSA、ECCDH、ElGamal[36]、SM2[37]等等許多不同演算法。其中最常見的演算法即 RSA。

2.2.1 RSA[38]

RSA 原理是使用兩個夠大的質數相乘所得出的數字，將其做質因數分解非常困難。若 A 想透過一個不可信任的管道接收 B 的訊息 m，可以先用下列方式產生一把公鑰和一把私鑰。

1. 隨意選擇兩個大的質數 p 和 q， p 不等於 q，計算 $N=pq$。

2. 根據 Euler's theorem，求得：

$$r=\varphi(N)=\varphi(p)\varphi(q)=(p-1)(q-1)$$

3. 選擇一個小於 r 的整數 e，使得 e 與 r 互質。並以下列公式求 d 值：

$$ed\equiv 1 \pmod{r}$$

4. 將 p 和 q 的記錄銷毀。

得(N,e)公鑰，可經由任意管道發送給 B；並將(N,d)私鑰保存。

B 先與 A 約定好將 m 轉換成一個小於 N 的非負整數 n，並透過下列公式以公鑰加密傳送給 A：

$$c \equiv n^e (mod\ N)$$

A 得到 B 傳送的訊息 c 後透過：

$$c^d \equiv n^{e\cdot d} (mod\ N)$$

已知$ed \equiv 1 \pmod{r}$，即 $ed = 1+h\,\varphi(N)$。由 Euler's theorem 得：

$$n^{ed} = n^{1+h\varphi(N)} = n\left(n^{\varphi(N)}\right)^{h} \equiv n(1)^{h}(mod\ N) \equiv n(mod\ N)$$

2.2.2 ECCDH[39]

全名為 Elliptic Curve Diffie–Hellman key Exchange，原理是讓使用者 A、B 都各有一對公私鑰，並以此建立共享密鑰，共享密鑰可直接用作密鑰或衍生另一把密鑰，其加密原理如下：

1. 通訊雙方先共同選擇一個系統橢圓曲線與基點 G 作為公鑰，且 G 的級數 n 要夠大(如大於 2^{160})。
2. 使用者 A 選擇密鑰正整數 x，且 $1 < x < n$，然後計算 $G_x = x \cdot G$，並將 G_x 傳送給使用者 B。
3. 使用者 B 選擇密鑰正整數 y，且 $1 < y < n$，然後計算 $G_y = y \cdot G$，並將 G_y 傳送給使用者 A。
4. 使用者 A 用收到的 G_y 與自身密鑰 x 算得 $K_{AB} = x \cdot G_y = x \cdot (y \cdot G)$，同理使用者 B 得 $K_{AB} = y \cdot G_x = y \cdot (x \cdot G)$。
5. 雙方擁有共同橢圓曲線點 $x \cdot (y \cdot G)$。此點的 x 座標即為共同金鑰 K_{AB}。

2.3 Hash Function

Hash Function 是將資料打亂混合，重新建立一個叫做 hash value 的值，通常以一個短的隨機字母和數字組成的字串來代表。Hash Function 有以下性質：1 運算速度快、2 不可逆運算、3 輸出不同，輸入必不同、4 輸出相同，輸入不一定相同。其中第 4 種性質被稱為

collision，但一個優秀的 Hash Function 很少、甚至不會發生 collision。較常看到的是 MD5[40]及 SHA-1。

2.3.1 SHA-1[40]

用在加密機制上 Hash Function 稱作 SHA(Secure Hash Algorithm)，是 FIPS 所認證的演算法，輸入的訊息限制長度不能超過 2^{64} bits，被分成數個 512bits 的區段按順序處理，其輸出結果是 1 個 160bits 的訊息摘要。

步驟一：前處理

如圖 2 將訊息本文後面加上補充位元以及訊息長度使總長度為 512bits 的倍數。補充位元從 1 到 512bits，補充的方式是先加上一個 1，再補 0 至所需要的長度。訊息長度則是一個 64bits 的資料以表示最大 2^{64}bits 的訊息本文。

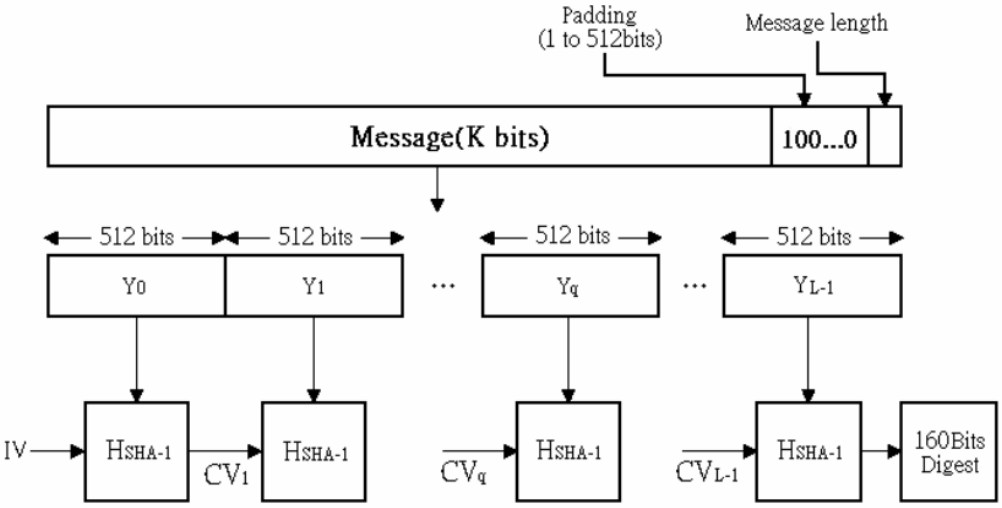


圖 2　SHA-1 加密的前處理

步驟二：設定運算暫存器內容：

我們使用 5 個 32bits 的暫存器儲存 SHA-1 的運算過程以及結果，用 A、B、C、D、E 表示，其初始值如下：

A = 0x67452301

B = 0xEFCDAB89

C = 0x98BADCFE

D = 0x10325476

E = 0xC3D2E1F0

步驟三：處理訊息分段的訊息內容。

將輸入的 512bits 按照順序分成 16 個 32bits 並命名為 W_0 -W_{15}、W_{16} -W_{31}、W_{32} -W_{47}、W_{48} -W_{63}、W_{64} -W_{79}。加密過程每一輪如下圖，並重複執行 80 次。其中第 t 輪的 W_t 將輸入的分段訊息帶入；F 為壓縮函數，因執行次數使用不同的函數，函數內邏輯運算如下：

$F = (b \cdot c) + ((\sim b) \cdot d)$ （$0 \leqq t \leqq 19$）

$F = b \oplus c \oplus d$ （$20 \leqq t \leqq 39$）

$F = (b \cdot c) + (b \cdot d) + (c \cdot d)$ （$40 \leqq t \leqq 59$）

$F = b \oplus c \oplus d$ （$60 \leqq t \leqq 79$）

K_t 為常數，因執行次數帶入不同的值，其值如下：

K_t = 0x5A827999 （$0 \leqq t \leqq 19$）

K_t = 0x6ED9EBA1 （$20 \leqq t \leqq 39$）

K_t = 0x8F1BBCDC （$40 \leqq t \leqq 59$）

K_t = 0xCA62C1D6　　（60 ≦ t ≦ 79）

最後計算完 80 次所得到的 ABCDE 共 160 位元，即為輸出。

2.4 相關論文探討

2.4.1 A Group File Encryption Method using Dynamic System Environment Key[41]

A Group File Encryption Method using Dynamic System Environment Key (GEMS)，提出一個群組內檔案加密方法，這個方法可以防止內部攻擊、數據洩漏、重送攻擊、竊聽、已知密鑰攻擊，同時還能滿足 DRM 的安全機制。架構如圖 3，GEMS 是由伺服器、可信任的電腦和複數使用者組成。使用者在加解密檔案之前必須先向

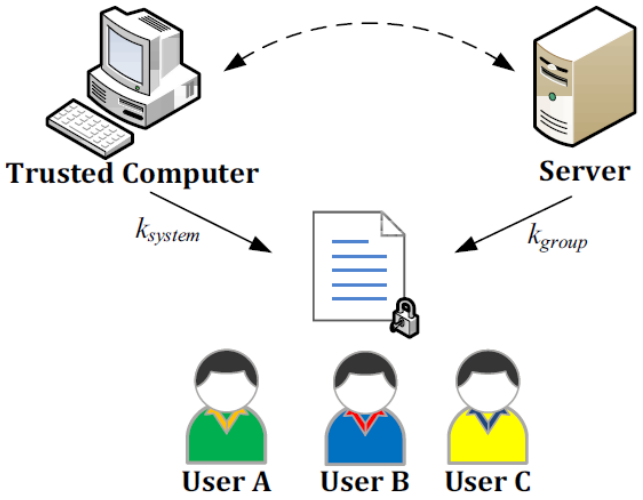


圖 3 GEMS 系統架構

伺服器註冊並授權，授權後生成用戶密碼密鑰(k_{pw})但不傳送給用戶而是直接儲存在伺服器中，並進一步生成組密鑰(k_{group})。伺服器不僅生成組密鑰，另外使用者打開受信任電腦內的文件時進行身分驗證和授

權。受信任的電腦負責生成動態系統密鑰(k_{system})並儲存加密文件。由於不同的電腦有不同的軟硬體配置，故生成的系統密鑰也會有所不同。因此，授權使用者只能從特定信任的電腦解密文件，當文件被下載到其他電腦時，便無法開啟文件。

使用者 A 若想加密一份可與使用者 B 共享的文件，須在可信任電腦加密 k_{ct} ,r_A, 並傳送 UID_A, t_{nonce} 及加密後產生的參數 r_A', R_A, Ef_A 給伺服器認證、授權，其中 t_{nonce} 為使用者 A 從可信任電腦抓取的系統時間，k_{ct} 可從 t_{nonce} 加密得到，$r_A \in Z_n^*$ 為使用者 A 在可信任電腦產生的一隨機數，UID_A 為使用者 A 的 ID。伺服器收到參數之後解密各參數並進行 t_{nonce} 時間比對、r_A 驗證可信任電腦、使用者認證、欲加密檔案確認，完成後將 k_{group} 分別作加密及 Hash 得到 $k_{user,A}$、Ψ_s 並傳回給使用者 A，以及將 k_{ct} 與 k_{group} 經 Hash 加密成 V_S、k_{group} 和 k_{ct} 結合成 Et_{file} 並儲存在伺服器資料庫，便於解密或日後查詢檔案使用紀錄。使用者 A 獲得傳回值後驗證 Ψ_s，確認後才可將解密得到的 k_{group} 與可信任電腦上產生的 k_{system} 結合並對檔案加密。

使用者B打開檔案前與使用者A相同，須在可信任的電腦加密 k'_{ct}、r_B ，並傳送 UID_B, t'_{nonce} 及加密後產生的參數 r_B', R_B, Ef_B 給伺服器認證、授權，其中 t'_{nonce} 為使用者 B 從可信任電腦抓取的系統時間，k'_{ct} 可從 t'_{nonce} 加密得到，$r_B \in Z_n^*$ 為使用者 B 在可信任電腦產生的一隨機數，UID_B 為使用者 B 的 ID。伺服器收到參數之後解密各參數並進行 t'_{nonce} 時間比對、r_B 驗證可信任電腦、使用者認證、欲加密檔案確認，完成後將 k_{group} 分別作加密及 Hash 得到 $k_{user,B}$，並將 $k_{user,B}$、V_s、Et_{file} 並傳回給使用者 B，以及將 k'_{ct} 與 k_{group} 經 Hash 加密成 V_S、k_{group} 和 k'_{ct} 結合成 Et_{file} 並儲存在伺服器資料庫，便於解密或日後查詢檔案使用紀

錄。使用者 B 獲得傳回值後驗證 V_s，確認後才可將解密得到的 k_{group} 與可信任電腦上產生的 k_{system} 結合並對檔案解密。

2.4.2 Cloud Encryption using Distributed Environmental Keys[42]

Cloud Encryption using Distributed Environmental Keys (DENK)提出一個分佈式環境密鑰，如圖 4，這些文件可以由可信任電腦上所有的授權使用者共享，且這些可信任電腦可以是複數的。文件加密密鑰是由匹配密鑰、使用者密碼和可信任電腦的環境金鑰生成；解密則是使用 ECCDH 來計算輔助密鑰及匹配密鑰獲得。他們分析所提出的系統能夠抵制特定的資料外洩、重放攻擊、竊聽攻擊、模擬攻擊。

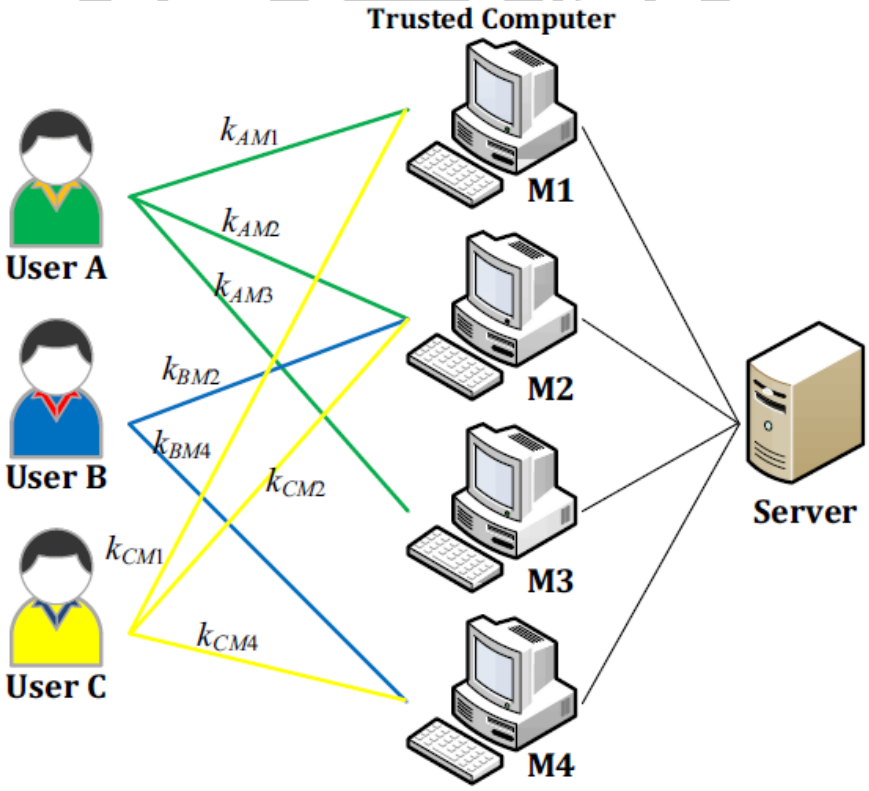


圖 4　DENK 系統架構

在最初的階段，伺服器端建立橢圓曲線加密並公開公鑰 E、P；使用者端建立環境金鑰，由於不同電腦有不同的軟硬體配置，故每台電腦都會生成不同的環境金鑰。

使用者 A 若想在可信任電腦 M1 加密檔案，須執行圖 5 步驟並傳送參數給伺服器驗證，其中 k_{pwA} 為使用者 A 的密碼，H_1 為 hash function，$F_{enviroment}$ 為環境金鑰加密，t_{nonce} 為系統時間，UID_A 及 UID_{M1} 分別為使用者 A 及可信任電腦 M1 的 ID。

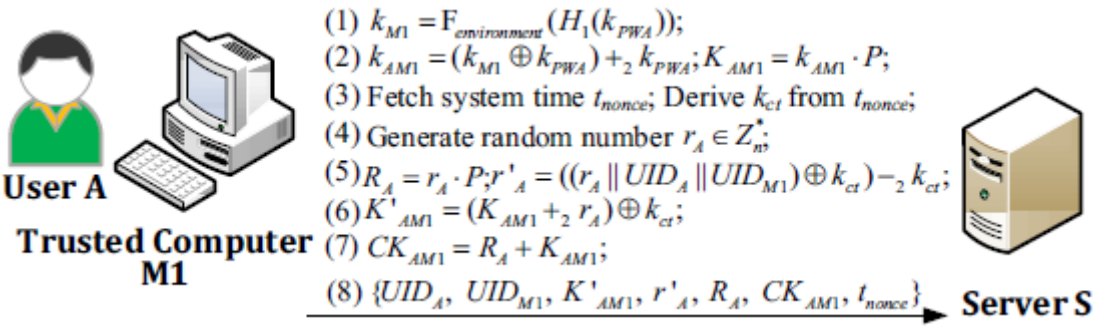


圖 5 加密使用者資料及系統參數並傳送給伺服器

伺服器收到參數後，解開各個參數並驗證，驗證成功便將檔案加密並儲存在伺服器，執行步驟如圖 6，其中 $t_{nonce,S}$ 為伺服器的系統時間，$r_{A,C}$、$R_{A,C}$、$UID_{A,C}$、$UID_{M1,C}$、$K_{AM1,C}$、$CK_{AM1,C}$ 為伺服器得出的 r_A、R_A、UID_A、UID_{M1}、$K_{AM1,C}$、$CK_{AM1,C}$。驗證成功後以同樣步驟收集所有匹配金鑰得 K_{iMj}，其中 i、j 分別為使用者 ID 及可信任電腦 ID。

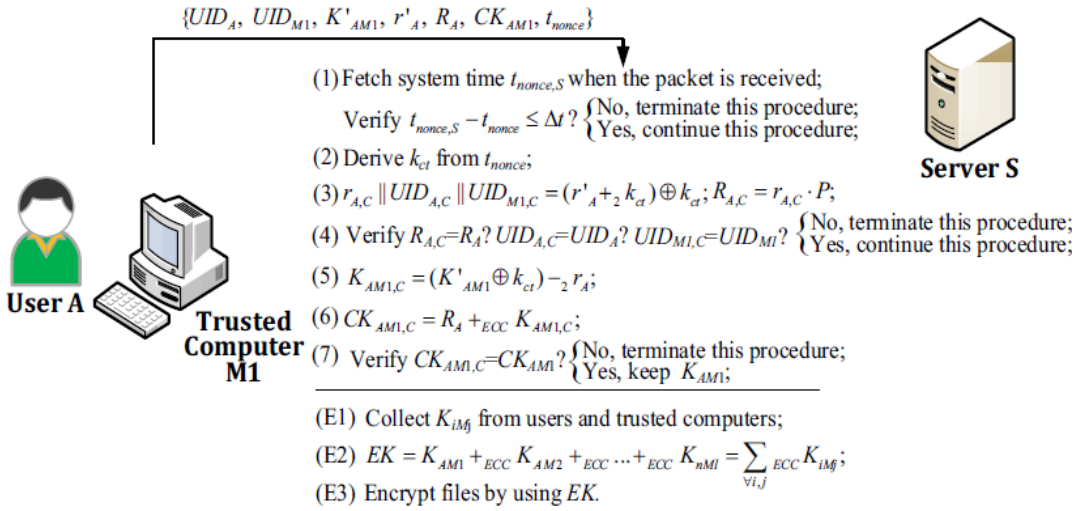


圖 6 伺服器端驗證使用者資料及回傳加密密鑰

若使用者 B 想打開文件，則必須執行如圖 7 步驟並傳送參數給伺服器驗證，其中 t'_{nonce} 為使用者 B 從可信任電腦 M3 抓取的系統時間，UID_B、UID_{M3}、FID 分別為使用者 B 的 ID、可信任電腦 M3 的 ID、檔案名稱。

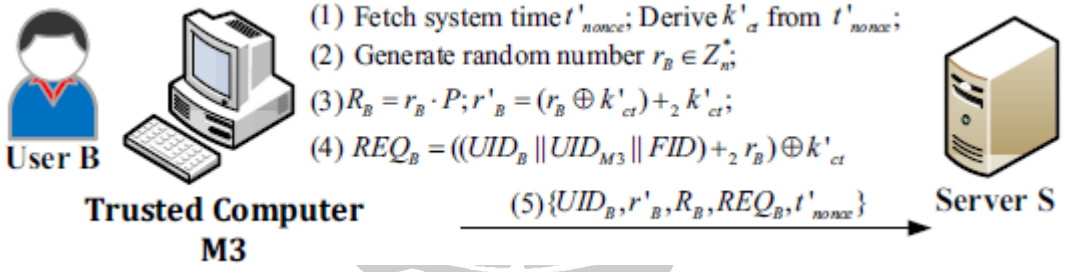


圖 7 加密使用者資料及系統參數並傳送給伺服器

伺服器端接受到使用者 B 的參數後，解開各個參數並驗證，如圖 8，驗證完成後計算得輔助密鑰，並將其傳回使用者 B，執行步驟如下圖，其中 $t'_{nonce,S}$ 為伺服器的系統時間，$r_{B,C}$、$R_{B,C}$、$UID_{B,C}$、$K_{AM1,C}$、$CK_{AM1,C}$ 為伺服器計算得出的 r_A、R_B、UID_B、$K_{AM1,C}$、$CK_{AM1,C}$。

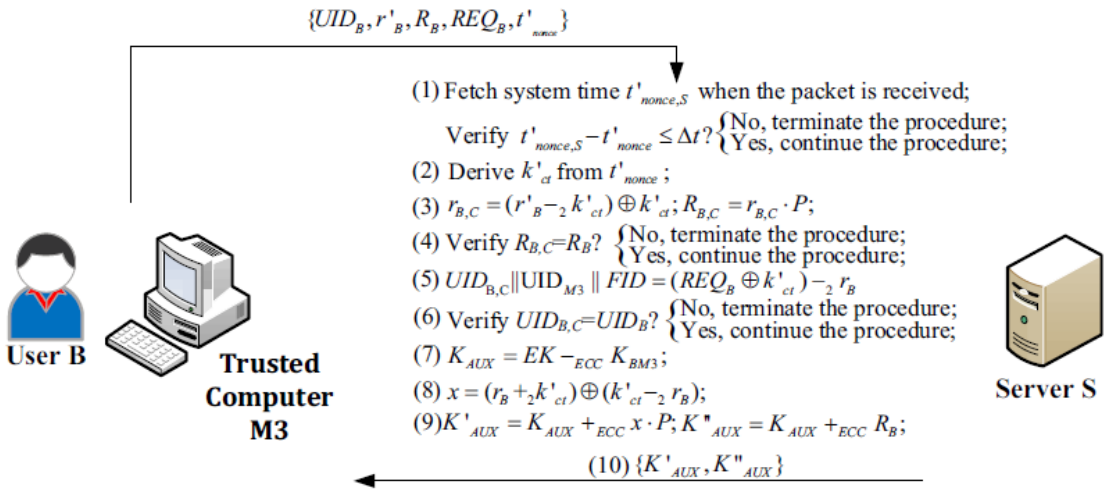


圖 8 伺服器端驗證使用者資料及回傳解密密鑰

如圖 9，使用者 B 收到伺服器回傳的輔助密鑰後，將其解開並驗證，驗證成功後，將使用者 B 的密碼、可信任電腦 M3 的環境參數與輔助密鑰結合得文件的解密密鑰，執行步驟如下圖，其中 k_{pwB} 為使用者 B 的密碼，H_1 為 hash function，$F_{enviroment}$ 為環境金鑰加密。

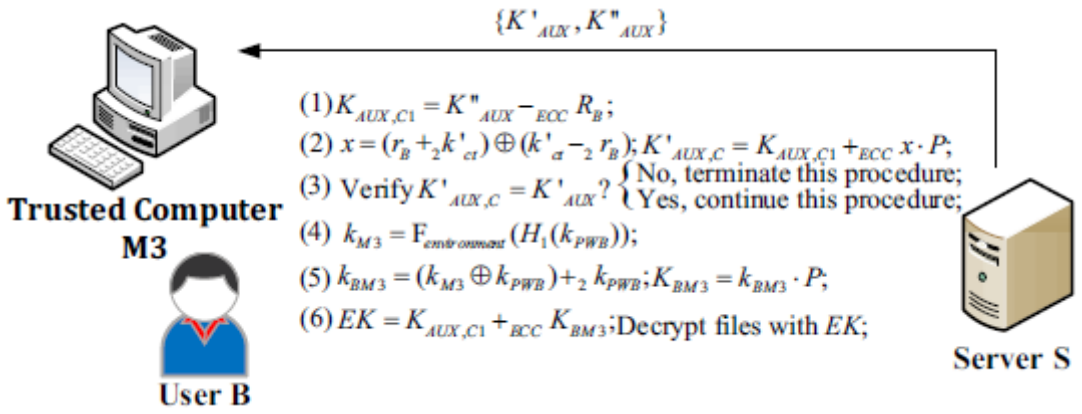


圖 9 使用者收到解密密鑰並解密檔案

第三章 以軟硬體資訊對檔案加密

3.1 系統架構

本論文系統架構流程如圖 10，在執行加解密時會先從 get hardware information 區塊抓取各個軟硬體的資訊，以及使用者可按照個人喜好輸入使用者密碼將上述各值分別輸入 SHA-256 轉換。其中使用者密碼的輸出值稱為 k_{pw}，將 k_{pw} 進制轉換做為其他 Hash Values 混合的依據，並將結果作為環境金鑰 k_{system}。後面的小節將會對各個步驟詳細介紹。

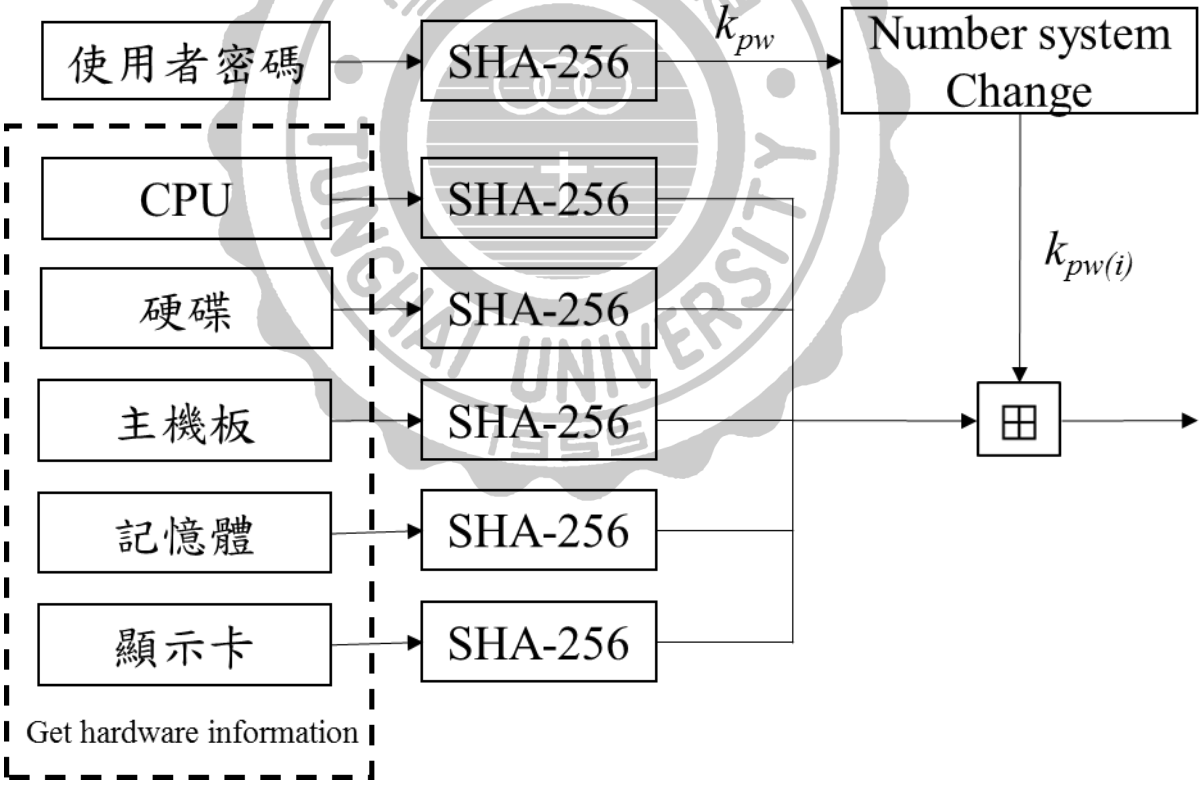


圖 10 系統架構

3.1.1 抓取軟硬體 ID

商品在生產時都附有商品編碼，這些編號可能內含國別碼、廠商識別碼、商品項目代碼、商品序號等資訊。電腦硬體也不例外，其附有的識別碼，包含 MAC、VID、PID…等，這個識別碼是唯一的，以便在需要的時候可以查詢其源頭，無論是什麼型號的電腦，都會有配置軟硬體設備。

擷取軟硬體 ID 時，先以 string 建立一個名為 *cpuid* 的暫存空間，再用 Runtime.getRuntime( )抓取 CPU 的 ID 並將值輸入進 *cpuid*，以此類推分別建立名為 *hdd*、*mobo*、*ram*、*video* 的含軟硬體 ID 字串。這些暫存的字串將在加解密結束時刪除。

3.1.2 環境金鑰建構程序

各個軟硬體的 ID，包含 cpuid、hdd、mobo、ram、video 等，輸入 Hash Function (本論文採用 SHA-256)，並將輸出值覆蓋掉原本寫有軟硬體 ID 的 string 字串。

建構環境金鑰的方法為新字串以一偽隨機數組成，此隨機數從使用者密碼經 SHA-256 得到 k_{pw}，將其依據抓取到的軟硬體參數數量 i 做進制轉換得到 $k_{pw(i)}$，各個軟硬體參數分別對應 0 - (i−1)的編號，並依 $k_{pw(i)}$的位數將各個 Hash values 以 mod 2 相加並寫入名為 *message* 的新字串，便可得到僅能從此台電腦獲得的環境金鑰 k_{system}。

3.2 檔案加密流程

在這個階段，我們將環境金鑰作為密鑰跟檔案以 AES-CTR (Counter Mode)加密，主程式演算法如下：

Encrypt(plaintext, kSystem){

令 blockSize 為 16 (16 bytes = 128 bits)。

令一 blockSize 大小的陣列為 countBlock。

抓取系統時間取到毫秒存入 nonce。

nonceMs = nonce mod 1000 ; nonceSec = nonce/1000。

以數學式產生一隨機數存入 nonceRnd。

將各項 nonce 值依序填入 counterBlock[0-7]，分別是 nonceMs，

nonceMs 右移 1bytes，nonceRnd，nonceRnd 右移 1bytes，nonceSec，nonceSec 右移 1bytes，及 nonceSec 右移 2bytes。

令 ctrTxt 為空，並將 cunterBlock[i]由 i=0 至 i=7 以字串方式填入。

將環境金鑰 kSystem 傳入 AES 加密的金鑰擴展函式可得回傳值，令為 keySchedule。

計算明文總長度共可分成多少 Block，並將數量令為 blockCount。

令 ciphertext 內容為空。

for (b=0; b< blockCount; b++) {

for (c=0; c<4; c++) counterBlock[15- c]= (b >>> c *8);

for (c=0; c<4; c++) counterBlock[11- c] =(b/ 0x100000000 >>> c*8);

令 cipherCntr 為 counterBlock 與 keySchedule 做 AES 加密的結果。

判斷 b 是否小於 blockCount-1，若是則令 blockLength = blockSize。

否則 blockLength = (plaintext.length -1) % blockSize +1。

將 cipherCntr 依據第 b 個 block 與明文第 b 個 block 做 XOR。

ciphertext = 將做完 XOR 的陣列合併寫入。

}

ciphertext = 將 ctrTxt 與 ciphertext 作字串連接。

return ciphertext。

}

3.3 檔案解密流程

當密文需要解密時，以 AES-CTR 演算法只需逆向運算即可，其過程如下：

Decrypt(ciphertext, kSystem){

令 blockSize 為 16 (16 bytes = 128 bits)。

令一 blockSize 大小的陣列為 countBlock。

從 ciphertext 提取 ctrTxt，並依序填入 counterBlock[0-7]。

將環境金鑰 kSystem 傳入 AES 加密的金鑰擴展函式可得回傳值，令為 keySchedule。

計算(密文文件長度-8)/blockSize，得密文 block 最大數量並傳值給 nBlocks。

令一 blockSize 大小的陣列為 ct，並將 ciphertext 除去 ctrTxt 的部分將密文以 block 大小傳入 ct。

令 plaintext 為空。

for (let b=0; b<nBlocks; b++) {

for (c=0; c<4; c++) counterBlock[15-c] = ((b) >>> c*8);

```
        for (c=0; c<4; c++) {
            counterBlock[11-c] = (((b+1)/0x100000000-1) >>> c*8);
        }
```

令 cipherCntr 為 keySchedule 與 counterBlock 做 AES 加密之值。

將 cipherCntr 依據第 b 個 block 與密文第 b 個 block 做 XOR，並傳值入 plaintextByte，直到長度為 ciphertext[b].length 便不再進行 XOR。

將 plaintxtByte 內容轉換成字串並存回原處。

```
        }
合併所有 plaintxtByte 陣列，寫入 plaintext。
}
return plaintext;
}
```

第四章實作與分析

4.1 加密檔案

圖 11 是程式執行加密的過程，(a)為開啟程式的畫面，包含選擇檔案按鈕、加密按鈕、解密按鈕，點選“...”按鈕並進入選擇檔案的視窗。(b)為選擇檔案的畫面，可選擇的檔案類型包含文件檔、圖檔、壓縮檔、應用程式、影片檔...等，檔案選擇完畢後按加密，只要等待完成即可，加密結束會跳出提示提醒使用者。(c)圖為點選加密且加密成功的提示視窗，並會在原檔案的資料夾內找到同樣檔名，副檔名為.enc 的檔案即為此檔案的加密檔。(d)圖為不選擇任何檔案即按下加密的結果。

(a)開啟程式畫面　(b)選擇檔案畫面

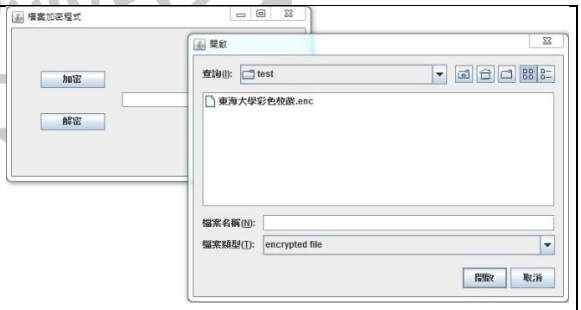
(c)加密成功　(d)無選擇檔案的加密結果

圖 11 加密執行過程

4.2 解密檔案

圖 12 是程式執行解密的過程，(a)為開啟程式的畫面，包含選擇檔案按鈕、加密按鈕、解密按鈕，點選”...”按鈕即可進入選擇檔案的視窗。(b)為選擇檔案的畫面，可以從檔案類型選擇 encrypted file 以快速尋找需解密的檔案，可解密的檔案類型僅能是.enc，檔案選擇完畢後按解密，等待完成即可，解密結束會跳出提示提醒使用者。(c)圖為點選解密且解密成功的提示視窗，並在解密檔案的資料夾內生成同樣檔名，且副檔名與加密前相同，但字頭多出 Decrypted_，此作用是避免覆蓋掉原有的檔案。(d)圖為將檔案移至其他電腦解密的失敗結果，或是在選擇檔案的時候選擇.enc 以外的檔案並執行解密。(e)圖為無選擇任何檔案解密的結果，(f)圖左為解密後的圖檔，圖中為原圖圖檔，圖右為加密後的檔案，(g)(h)圖分別為原圖檔及經加密後解密的圖檔，由此兩圖可知圖檔經加密後解密，圖形並沒有改變或模糊。

(a)開啟程式畫面



(b)選擇檔案畫面

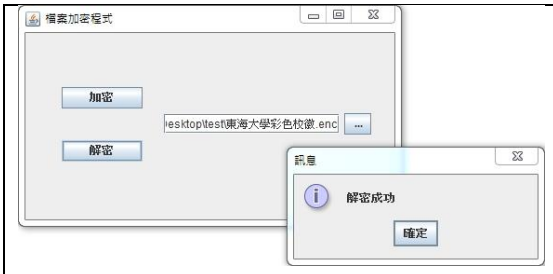
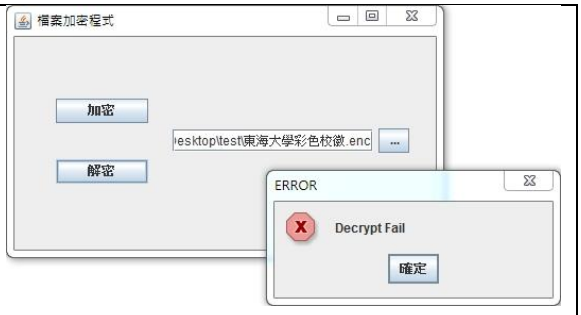
<table>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>(c)解密成功畫面</td><td>(d)解密失敗畫面</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>(e)無選擇檔案的解密結果</td><td>(f)原檔及其衍生的檔案</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>(g)原檔圖像</td><td>(h)原檔經加密後解密的圖像</td></tr>
</table>

圖 12 解密執行過程及圖檔比較

4.3 安全性

從第四章第 2 節的實作已經證實將加密的檔案從特定電腦複製出來，攻擊者仍無法在其他電腦上解密得到敏感資料。除此之外，本節也分析出幾種可行的安全性：

(1) 資料洩漏：檔案加密時，會從當前電腦抓取軟硬體參數，這些參數是唯一的，將其建構成環境金鑰加密檔案。因此，若使用者將檔案複製到其他電腦上，會因為解密時所使用的電腦軟硬體參數不同，而無法對檔案解密，進而避免資料洩漏。

(2) 動態加密：環境金鑰建構並非每次都以同樣軟硬體 ID 及同樣順序組合而成，而是由使用者輸入所希望的密碼，將其編碼做為此次建構環境金鑰的順序。每次加密解密並不儲存任何密鑰或可取得密鑰的資料。

(3) 空間共用：每臺電腦加密檔案所使用的密碼及硬體資訊皆不相同，故將這些來自不同電腦的加密檔案放入共用的隨身硬碟或雲端空間是安全的。即使未經授權下載，也同上述(1)項，而達到避免資料洩漏。

第五章結論與未來研究

本論文實現一個基於環境金鑰的本機電腦加密方案，以防止資料由內部外洩。有加密不受限於檔案類型的特點，以及可防止環境金鑰的加密程式可以輕易的放入隨身碟內，隨時能在任何電腦對檔案加解密，如此，若攻擊者在未經他人許可使用電腦也會因為電腦內沒有此程式而無法開啟檔案，並且還能防止社交工程、鍵盤側錄、竊聽攻擊...等。

未來研究希望可以將此種加密方案延伸至電腦以外的設備，如手機、平板，以及其他作業系統。另外，增加更多的軟硬體唯一識別碼，以及增加以排序方式建構環境金鑰，更能加強複雜度。

參考文獻

[1] What is encryption? , https://searchsecurity.techtarget.com/definition /encryption

[2] Chitra Biswas, Udayan Das Gupta, Md. Mokammel Haque, "A hierarchical key derivative symmetric key algorithm using digital logic," Electrical, Computer and Communication Engineering (ECCE), International Conference on, 27 April 2017.

[3] W. Diffie and M.E. Hellman, "New Directions in Cryptography," IEEE Transactions on Information Theory, IT-22, n.6, Nov 1976, pp. 644–654.

[4] D. Coppersmith, "The Data Encryption Standard (DES) and its strength against attacks," pp. 243-250, May 1994.

[5] Don Johnson, Alfred Menezes, Scott Vanstone, *"The Elliptic Curve Digital Signature Algorithm (ECDSA)," International Journal of Information Security,* Volume 1, Issue 1, pp. 36–63, August 2001.

[6] Barbara W. Tuchman, A brief history of the data encryption standard. Internet besieged: countering cyberspace scofflaws. ACM Press/ Addison-Wesley Publishing Co. New York, NY, USA: 275–280. 1997.

[7] Madhavi Suryawanshi, Sarita Patil, "Avoiding the data leakage and providing privacy to data in networking," Computing Communica -tion Control and automation (ICCUBEA), 2016 International Conference on, 12-13 Aug. 2016.

[8] Polina Zilberman, Shlomi Dolev, Gilad Katz, Yuval Elovici, Asaf Shabtai, ”Analyzing group communication for preventing data leakage via email,” Intelligence and Security Informatics (ISI), 2011 IEEE International Conference on, 10-12 July 2011.

[9] David Clark, Sebastian Hunt, Pasquale Malacaria, “Quantitative Analysis of the Leakage of Confidential Data,” Electronic Notes in Theoretical Computer Science, Volume 59, Issue 3, pp. 238-251, November 2002.

[10] Chunming Ronga, Son T.Nguyena, Martin Gilje Jaatunb, “Beyond lightning: A survey on security challenges in cloud computing,” Computers & Electrical Engineering, Volume 39, Issue 1, pp. 47-54 January 2013.

[11] Asaf Shabtai, Yuval Elovici, Lior Rokach, “*A Survey of Data Leakage Detection and Prevention Solutions*,” *Springer Science & Business Media*, 15 Mar 2012.

[12] Arthur Sorkin, “Lucifer, a cryptographic algorithm,” Cryptologia, Volume 8, Issue 1, pp.22-42, 1984.

[13] Eli Biham, Adi Shamir, “Differential cryptanalysis of DES-like cryptosystems,” Journal of Cryptology, Volume 4, Issue 1, pp 3-72, January 1991.

[14] Mitsuru Matsui, “Linear Cryptanalysis Method for DES Cipher,” Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765, T. Helleseth, Ed., Springer-Verlag, pp. 386-397, 1994.

[15] Electronic Frontier Foundation, John Gilmore, “*Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*”,

Electronic Frontier Foundation, 1998.

[16] K Lengths, "*DATA ENCRYPTION STANDARD (DES)*", *Federal Information Processing Standards (FIPS)*, 25 October 1999.

[17] San Jose, CA. "RSA Code-Breaking Contest Again Won by Distributed.Net and Electronic Frontier Foundation (EFF)," RSA DATA SECURITY CONFERENCE, 19 January 1999.

[18] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings," Internet Engineering Task Force(IETF), July 2003.

[19] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, "*Handbook of Applied Cryptography*", *CRC Press*, October 1996.

[20] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, "*Handbook of Applied Cryptography*", *CRC Press*, October 1996.

[21] R. Rivest, "A Description of the RC2(r) Encryption Algorithm," Network Working Group (NTWG), March 1998.

[22] A. Popov, "Prohibiting RC4 Cipher Suites," Internet Engineering Task Force (IETF), February 2015.

[23] Ronald L. Rivest, "The RC5 Encryption Algorithm," MIT Laboratory for Computer Science 545 Technology Square, Cambridge, Mass. 02139,

[24] Ronald L. Rivest, M.J.B. Robshaw, R. Sidney, Y.L. Yin, "The RC6™ Block Cipher," RSA Laboratories, 20 August 1998.

[25] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris

Hall, Niels Ferguson, “Twofish: A 128-Bit Block Cipher,” Schneier on Security,15 June 1998.

[26] Serpent home page, http://www.cl.cam.ac.uk/~rja14/serpent.html

[27] Joan Daemen, Vincent Rijmen, “*The Design of Rijndael: AES - The Advanced Encryption Standard*,” *Springer Science & Business Media*, 5 Mar 2013.

[28] Kam, Davida, “Structured Design of Substitution-Permutation Encryption Networks,” IEEE Transactions on Computers, Volume C-28, Issue 10, pp.747-753, October 1979.

[29] Lars R. Knudsen, “Practically secure Feistel ciphers,” International Workshop on Fast Software Encryption FSE 1993: Fast Software Encryption, pp 211-221, 08 June 2005.

[30] Serge Mister, Carlisle Adams, “Practical S-Box Design,” Workshop Record Selected Areas Cryptography(SAC'96), pp. 61-76, 15 August 1996.

[31] Joan Daemen, Vincent Rijmen, “*The Design of Rijndael: AES - The Advanced Encryption Standard*,” *Springer Science & Business Media*, 9 Mar 2013.

[32] Avi Kak, “Lecture 8: AES: The Advanced Encryption Standard,” https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf

[33] B. Schneier, “Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish),” proceedings of Fast Software Encryption, Cambridge, Lecture Notes in Computer Science, pp. 191–204, 1993.

[34] C. Adams, “The CAST-128 Encryption Algorithm,” RFC 2144, May 1997.

[35] W. Diffie, M. Hellman, “New directions in cryptography,” IEEE Transactions on Information Theory, Volume 22, Issue 6, pp.644-654, Nov 1976.

[36] T. Elgamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” IEEE Transactions on Information Theory, Volume 31, Issue 4, pp. 469-472, Jul 1985.

[37] 国家密码管理局,“SM2 椭圆曲线公钥密码算法,” 12 月 2010 年。

[38] 黃明祥、林詠章(2009)。*資訊與網路安全概論。麥格羅希爾*。

[39] 楊中皇，網路安全：理論與實務(第二版)，學貫行銷，9 月 1 號 2008 年。

[40] Burrows, James H, “Secure Hash Standard,” National Institute of Standards and Technology, 17 April 1995.

[41] Kun-Lin Tsai, Jiu-Soon Tan, Fang-Yie Leu, Yi-Li Huang,” A Group File Encryption Method using Dynamic System Environment Key,” Network-Based Information Systems (NBiS), 2014 17th International Conference on, 10-12 September 2014.

[42] Kun-Lin Tsai, Fang-Yie Leu, Yi-Fung Huang, Chi Yang, Cheng-Hsin Chang, King-Shing Yip, Yuchen Xue, Guan-Chi Lai, “Cloud Encryption Using Distributed Environmental Keys,” Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2016 10th International Conference on, 6-8 July 2016.