翻轉電子書系列：資訊與網路安全概論 回首頁

翻轉工作室：粘添壽

第四章 雜湊與亂數演算法

人可以利用獨一無二的『指紋』來證明自己的特徵；文件也可以利用雜湊碼來證實其唯一性，但雜湊值真的安全？不可以偽造嗎？

4-1 雜湊函數簡介

所謂『**雜湊函數**』（**Hash Function**），是將不定長度訊息的輸入，演算成固定長度雜湊值的輸出，且所計算出來的雜湊值必須符合**兩個主要條件：(1) 由雜湊值是無法反推出原來的訊息，(2) 雜湊值必須隨明文改變而改變**。也就是說，不同明文所計算出來的雜湊值必須是不相同的，甚至僅改變明文中任何一個字元時，雜湊值的輸出也必須差異很大。由此可見，期望中的雜湊值就好像是一個無法冒充的識別碼，且每一份文件都有其特殊的雜湊值，且無法被偽造，故有『**數位指紋**』（**Digital Fingerprint**）之稱。雜湊函數在資訊安全技術上所扮演的角色極為重要，應用範圍很廣泛。本章除了介紹一些標準化的雜湊演算法，也會探討其安全性。

雜湊函數的運算及參數表示如下：

訊息輸入：M

雜湊函數：H

雜湊值輸出：h = H(M)

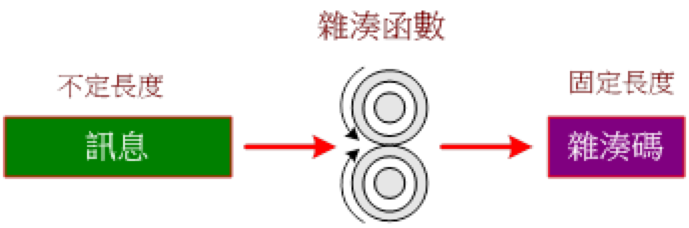

圖 4-0

我們期望輸出的雜湊值 h，必須隨輸入訊息 M 而改變，而且該值是無法仿冒的，如人類『**指紋**』功能一般。當然要達到這個功能，主要關鍵在於雜湊函數的複雜程度如何。一般雜湊函數必須具備下列功能：

- **雜湊函數必須對任意長度的訊息輸入，產生固定長度的雜湊值輸出。**
- **對於任意訊息 M，雜湊函數可以輕易的計算出 H(M)**，並且可經由硬體或軟體來實現。
- **如果給予雜湊值 h，在計算上是無法找出原訊息 M，使其符合 h = H(M)，此特性稱之為『單向雜湊』（One-way Hash）**。
- **對於一個訊息 M_1，在計算上是無法找出另一個訊息 M_2（≠ M_1），使得 $H(M_1) = H(M_2)$**。也就是說，給定一個雜湊值（$H(M_1)$），須無法找出另一個訊息（M_2），使其所產生的雜湊值相同。
- **就兩訊息 M_1、M2 而言，若他們的雜湊值相等，亦即 $H(M_1) = H(M_2)$，則 M_1 與 M_2 兩訊息也一定相等（$M_1 = M_2$）**；同理，**若 $H(M_1) \neq H(M_2)$，則 $M_1 \neq M_2$**。也就是說，給定一個明文與雜湊值，須保證無法找出另一個訊息來產生同樣的雜湊值。

第4項是雜湊函數最基本功能，亦即給予一段不定長度的訊息，能輕易計算出一個固定長度的雜湊值。如果反過來，給予一個固定長度的雜湊值，是無法計算出原來訊息的，也無法找出可以產生同樣雜湊值的另一段訊息，如能達到此功能，一般稱之為**『弱雜湊函數』（Weak Hash Function）**。若再涵蓋第 5 項功能，則稱為**『強雜湊函數』（Strong Hash Function）**，其表示有了明文與雜湊值，也無法另外偽造一個明文來產生相同的雜湊值。由此可見，欲達到強雜湊函數的功能實不容易，但它可以克服『生日攻擊法』的攻擊（容後介紹）。

4-2 簡單的雜湊函數

其實在網路通訊方面，雜湊函數常被用來偵測傳輸資料是否有發生錯誤，如圖 4-1 所示，發送端傳送資料之前，先將資料經過某一種雜湊函數計算，而得到一個雜湊值。然後將這雜湊值附加在資料後面，一併傳送給接收端；接收端收到資料之後，也以相同的雜湊函數計算出另一個雜湊值；如果此雜湊值與傳送端所計算的相同，則表示資料並沒有發生錯誤。一般來講，在較低層次的通訊協定（如 Ethernet 層）都使用 **CRC（Cyclic Redundancy Check）**除法器，來產生一個檢查資料的 FCS（Frame Check Sequence）；至於較高層次（如 TCP/IP 層）則採用檢查集（Checksum）方法居多。

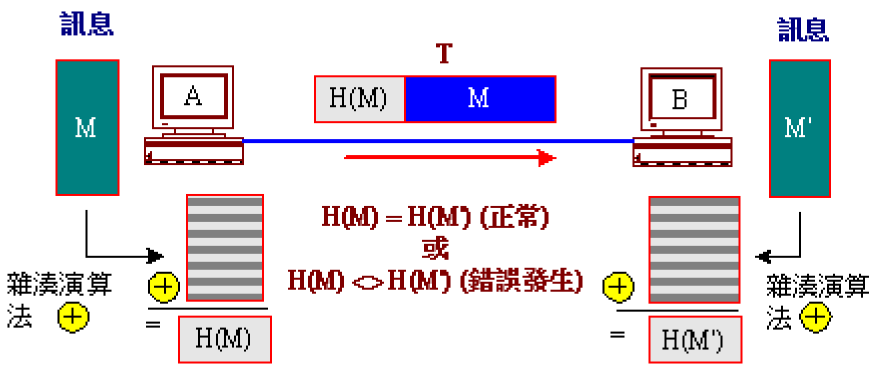

圖 4-1 簡單的雜湊函數

以下就檢查集為例，來說明簡單雜湊函數的製作方法，不難發現這種雜湊函數的脆弱性。它的運作方式如圖 5-2 所示，利用所有字組成的相對位元之間做 XOR（符號為 ⊕）運算，稱為**『位元操作- XOR』（Bitwise-XOR）**。首先將訊息以字元（如 32、64 或 128 bits）區塊排列，再區塊之間相對位元做 XOR運算。假設將訊息切乘 m 個區塊，每個區塊有 n 個位元（如 32 bits），則所產生的檢查集（或稱雜湊值，也是 32 bits）為：

$C_j = b_{j1} \oplus b_{j2} \oplus, \ldots, \oplus b_{jm}$ ，$j = 1, 2, \ldots, n$。

	位元 1	位元 2	位元 3		位元 n
字元 1	B_{11}	B_{21}	B_{31}	……	B_{n1}
字元 2	B_{12}	B_{22}	B_{32}	……	B_{n2}
	….	…..	……	…..	
字元 m	B_{1m}	B_{2m}	B_{3m}	……	B_{nm1}
雜湊值	C_1	C_2	C_3	……	C_n

圖 4-2 檢查集的雜湊函數

由上圖 4-2 可知，每一雜湊值的位元（C_j）是由該行所有位元做 XOR 運算而產生，如同位位元（Parity Bit）一般，其中如果有偶數個位元發生錯誤（由 1 → 0、或 0 → 1），則該位元的雜湊值將不會發生變化，因此，它的偵錯能力是非常薄弱的。

4-3 MD5 訊息摘要

『訊息摘要』（Message Digest）是由 Ron Rivest（RSA 中的 “R”）在 MIT 所發展的一系列雜湊演算法，其中較著名的有 MD2（RFC 1319）、MD4（RFC 1320）與 MD5（RFC 1321）。MD2 主要是以 8 個位元為單位的計算方式，複雜度較弱，一般都將該演算法崁入於數位晶片上，如 IC 卡。MD4 與 MD5 都是針對 32 位元 CPU 所設計的；MD5 是由 MD4 改良得來，主要為了增加複雜度，就目前而言，MD5 早已凌駕 MD4 之上。

4-3-1 MD5 運作原理

MD5 是將不定長度的訊息，演算成一個 128 個位元長的訊息摘要（或稱雜湊碼）；計算方式是屬於串接式的區塊演算法，如圖 4-3 所示。處理步驟如下：

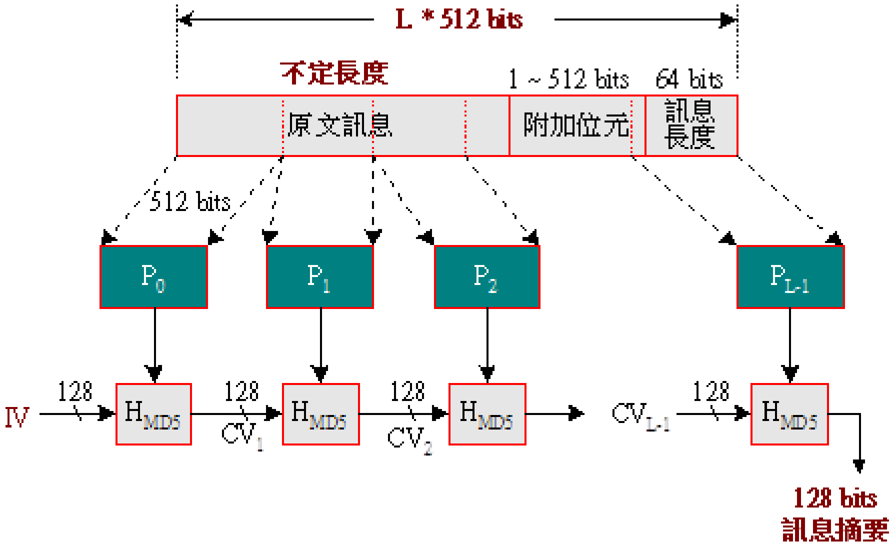

圖 4-3 利用 MD5 產生訊息摘要

- **步驟 1：**加上一些附加位元（Padding Bits），使得訊息長度除以512餘數為448(448 mod 512)。如果訊息的長度剛好滿足448 mod 512時，仍必須加入 512 bits 的附加位元。所附加位元的資料除第一個位元為1，其餘皆為0。
- **步驟 2：**加上 64 bits 長度欄位，其內容表示原訊息的長度，以位元為單位。如果長度超過 2^{64} 個位元，則只取最低 64 位元的資料；亦即 mod 2^{64}。
- **步驟 3：**以每 512 bits 為單位，將訊息分割為L個區塊 $\{P_0, P_1, .., P_q, \ldots, P_{L-1}\}$。
- **步驟 4：**將第一區塊（P_0）與起始向量（Initial Vector, IV，128 bits）輸入到 MD5 演算法中，輸出為 CV_1（128 bits）；CV_1 則作為下一個區塊P_1的輸入向量，依此類推。
- **步驟 5：**最後區塊（P_{L-1}）與前一個 CV_{L-1} 經由 MD5 演算後的輸出值，即為該訊息的訊息摘要（或稱雜湊值，128 bits）。

起始向量（IV）是一個重要的參數，每次演算時加入不同的 IV 值，可以增加破解的困難度。一般來講，IV 值只要在雙方協商時，以明文傳送即可（亦可加入於原訊息中一起加密）。

4-3-2 MD5 演算法

接下來，我們來探討圖 5-3 中的 MD5 演算法，他有**兩組輸入：一組為 512 bits 的明文區塊（P_q），另一組為上一區塊所計算出來 128 bits 的雜湊值（CV_q）（第一個區塊為 IV 值）**。經由 MD5 演算法計算出 128 bits 的雜湊值（CV_{q+1}）後，繼續傳送給下一個區塊直到結束。其實，圖 5-3

中只有一個 MD5 演算法，亦即各個區塊不斷重複使用同一個 MD5 演算法。MD5 演算法共區分為四個步驟，每個步驟須重複運算 16 次，合計共有 64 次的運算。其計算方式是將 512 bits 區分為 4 個 128 bits 的運算單位，分別存入 4 個暫存器內，在 64 次計算當中，都依照暫存器的內容來運算，並分別加入前一次的雜湊值，不僅如此，還加入 sin(x) 函數的運算，如此說來，MD5 應該夠複雜了。

圖 4-4 為 MD5 演算法的功能圖，共分為 4 個回合計算，每一回合計算 16 次。處理步驟如下：

- **步驟 1：**將輸入明文（512 bits）以每 32 bits 為單位，分別存入X[k]中，其中 k =0, 1, 2, .., 15；每一回合的每一次計算分別選入不同的 X[k] 值（容後介紹）。
- **步驟 2：**初始化 A、B、C 與 D 暫存器。MD5 必須產生 128 bits 的訊息摘要，而這些摘要必須利用4 個暫存器來儲存及運算，因此每個暫存器的空間是 32 bits；其初始值的設定分別如下：（16 進位表示）

  A：01 23 45 67
  B：89 ab cd ef
  C：fe dc ba 98
  D：76 54 32 10

  其中數值皆以較小位元在前面的位元組（Little-endian）方式來儲存。

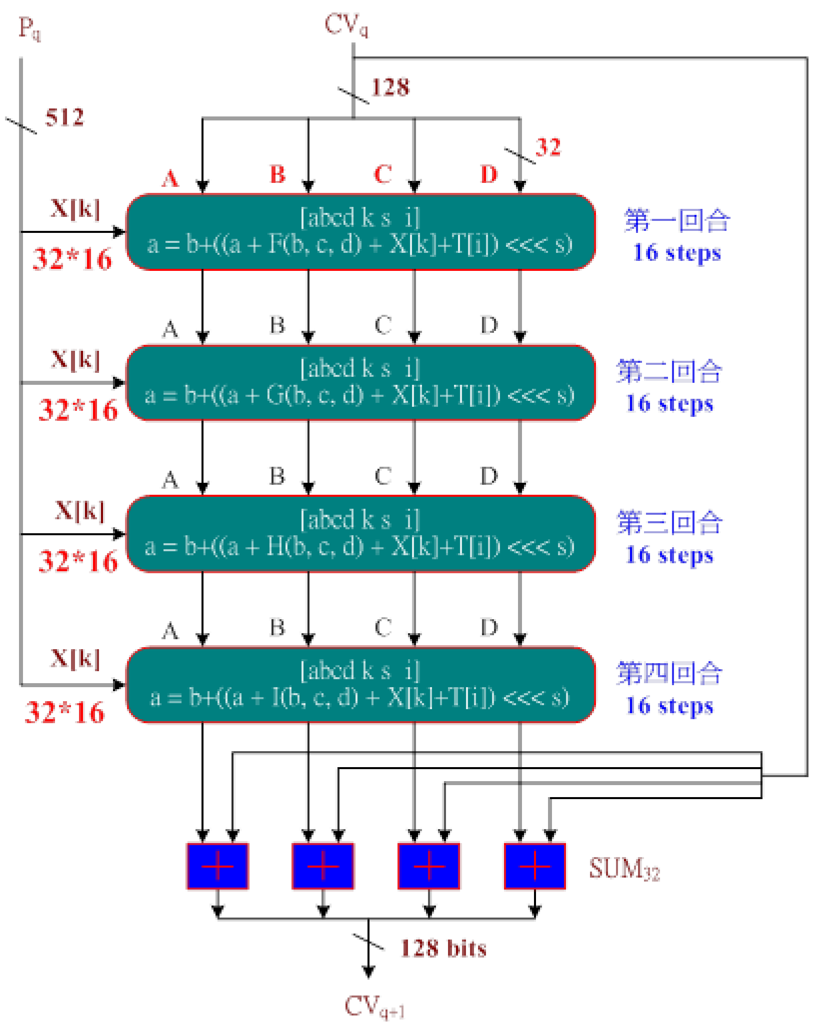

圖 4-4 MD5 演算法的功能圖

- **步驟 3：**進入第一回合運算，將 A、B、C 與 D 等四個暫存器與明文輸入的 X[k]，一起輸入到一個稱之為『壓縮函數』（容後介紹）內處理，此壓縮會使用到 sine 函數所建立的參數 T[i]

（容後介紹，圖 5-5）。每次處理的壓縮函數的輸入參數為 [abcd, k, s, i]，其中 abcd 表示上一執行次數的四個暫存器、k 表示輸入明文的區段 X[k]、i 表示 sine 函數的參考值 T[i] 與 s 表示向左迴旋的次數。輸出時，bcd 三個暫存器的內容不變，但 a 暫存器修改成 a = b+((a+F(b, c, d)+X[k]+T[i]) <<<s)，其中 F 函數每個回合都不同（容後介紹）。本回合需連續執行 16 次，各次的輸入參數如下：

[ABCD, 0, 7, 1]	[DABC, 1, 12, 2]	[CDAB, 2, 17, 3]	[BCDA, 3, 22, 4]
[ABCD, 4, 7, 5]	[DABC, 5, 12, 6]	[CDAB, 6, 17, 7]	[BCDA, 7, 22, 8]
[ABCD, 8, 7, 9]	[DABC, 9, 12, 10]	[CDAB, 10, 17, 11]	[BCDA, 11, 22, 12]
[ABCD, 12, 7, 13]	[DABC, 13, 12, 14]	[CDAB, 14, 17, 15]	[BCDA, 15, 22, 16]

以第二次執行為例，其輸入參數為 [DABC, 1, 12, 2]（k=1, s =12, i=2），表示將上一次運算的 D、A、B、C 暫存器分別填入本次的 A、B、C 與 D 暫存器中，再計算 a = b+((a+ F(b, c, d)+ X[1] + T[12]) <<< 2)，然而 B、C 與 D 暫存器的內容不變。

- **步驟 4：**進入第二回合運算。同樣執行 16 次，但變更 a 的函數為 G，則 a = b+((a+G(b, c, d)+X[k]+T[i]) <<<s)；每次輸入的參數如下（[abcd, k, s, i]）：

[ABCD, 1, 5, 17]	[DABC, 6, 9, 18]	[CDAB, 11, 14, 19]	[BCDA, 0, 20, 20]
[ABCD, 5, 5, 21]	[DABC, 10, 9, 22]	[CDAB, 15, 14, 23]	[BCDA, 4, 20, 24]
[ABCD, 9, 5, 25]	[DABC, 14, 9, 26]	[CDAB, 3, 14, 27]	[BCDA, 8, 20, 28]
[ABCD, 13, 5, 29]	[DABC, 2, 9, 30]	[CDAB, 7, 14, 31]	[BCDA, 12, 20, 32]

- **步驟 5：**進入第三回合運算。同樣執行 16 次，改變 a 的函數為 H，則 a = b+ ((a+ H(b, c, d)+ X[k]+T[i]) <<<s)；每次輸入的參數如下（[abcd, k, s, i]）：

[ABCD, 5, 4, 33]	[DABC, 8, 11, 34]	[CDAB, 11, 16, 35]	[BCDA, 14, 23, 36]
[ABCD, 1, 4, 37]	[DABC, 4, 11, 38]	[CDAB, 7, 16, 39]	[BCDA, 10, 23, 40]
[ABCD, 13, 4, 41]	[DABC, 0, 11, 42]	[CDAB, 3, 16, 43]	[BCDA, 6, 23, 44]
[ABCD, 9, 4, 45]	[DABC, 12, 11, 46]	[CDAB, 15, 16, 47]	[BCDA, 2, 23, 48]

- **步驟 6：**進入第四回合運算。同樣執行 16 次，改變 a 的函數為 I，則 a = b+ ((a+ H(b, c, d)+ X[k]+T[i]) <<<s)；每次輸入的參數如下（[abcd, k, s, i]）：

[ABCD, 0, 6, 49]	[DABC, 7, 10, 50]	[CDAB, 14, 15, 51]	[BCDA, 5, 21, 52]
[ABCD, 12, 6, 53]	[DABC, 3, 10, 54]	[CDAB, 10, 15, 55]	[BCDA, 1, 21, 56]
[ABCD, 8, 6, 57]	[DABC, 15, 10, 58]	[CDAB, 6, 15, 59]	[BCDA, 13, 21, 60]
[ABCD, 4, 6, 61]	[DABC, 11, 10, 62]	[CDAB, 2, 15, 63]	[BCDA, 9, 21, 64]

- **步驟 7：**輸出訊息摘要。將第四回合最後一次的運算結果，與原來輸入區段（CV_q）的相對應暫存器（A、B、C 與 D）相加後，得到本區段的輸出（CV_{q+1}）。兩個暫存器（32 bits）相加時，如有進位則將之捨棄（mod 2^{32}）。

很顯然的，MD5 演算法是經過 64 次的重複計算，類似攪拌機的功能，完全將 512 bits 資料的關聯攪碎。光攪拌還是無法消除位元之間的連帶性，因此，每一次攪拌時再加入一些『鹽』（Salt），使其更加混擾，類似『醃製法』（Salt Value）的功能。至於採用何種『鹽』會比較沒有關聯性，就 MD5 而言，它使用 Sine 函數的非線性數值特性，計算方式是 T[i] = 2^{32} × (abs(sin(i))，其中 i 表示弧度（Radians），i 由 1 到 64；亦即每一次運算取一個鹽 T[i]，共64 次運算，剛好由 T[1] 取

到 T[64]，T[i] 的數值如圖 4-5 所示。另外，每一次運算時，會輸入 32 個位元的明文（X[k]），原區塊的明文倍分割為 16 的段落（X[0], X[1], …, X[15]），也會在這 64 次計算中重複被輸入。

T[1] = D76AA478	T[17] = F61E2562	T[33] = FFFA3942	T[49] = F4292244
T[2] = E8C7B756	T[18] = C040B340	T[34] = 8771F681	T[50] = 432AFF97
T[3] = 242070DB	T[19] = 165E5A51	T[35] = 699D6122	T[51] = AB9423A7
T[4] = C1BDCEEE	T[20] = E9B6C7AA	T[36] = FDE5380C	T[52] = FC93A039
T[5] = F57COFAF	T[21] = D62F105D	T[37] = A4BEEA44	T[53] = 655B59C3
T[6] = 4787C62A	T[22] = 02441453	T[38] = 4BDECFA9	T[54] = 8F0CCC92
T[7] = A8304613	T[23] = D8A1E681	T[39] = F6BB4B60	T[55] = FFEEF47D
T[8] = FD469501	T[24] = E7D3FBC8	T[40] = BEBFBC70	T[56] = 85845DD1
T[9] = 698098D8	T[25] = 21E1CDE6	T[41] = 289B7EC6	T[57] = 6FA87E4F
T[10] = 8B44F7AF	T[26] = C33707D6	T[42] = EAA127FA	T[58] = FE2CE6E0
T[11] = FFFF5BB1	T[27] = F4D50D87	T[43] = D4EF3085	T[59] = A3014314
T[12] = 895CD7B1	T[28] = 455A14ED	T[44] = 04881D05	T[60] = 4E0811A1
T[13] = 6B901122	T[29] = A9E3E905	T[45] = D9D4D039	T[61] = F7537E82
T[14] = FD987193	T[30] = FCEFA3F8	T[46] = E6DB99E5	T[62] = BD3AF235
T[15] = A679438B	T[31] = 676F02D9	T[47] = 1FA27CF8	T[63] = 2AD7D2BB
T[16] = 49B40821	T[32] = 8D2A4C8A	T[48] = C4AC5665	T[64] = EB86D391

圖 4-5 sin(x) 函數所建立的『鹽』

4-3-3 MD5 壓縮函數

每一回合的運算器稱之為**『壓縮函數』（Compression Function）**，是 MD5 演算法的核心。它的功能是將 512 個位元的區塊，攪拌及壓縮成 128 個位元，其運算程序如圖 4-6 所示，各暫存器的運算式為：

$$a = b + ((a + g(b, c, d) + X[k] + T[i]) <<< s)$$

$$b = b,\ \ c = c,\ \ d = d$$

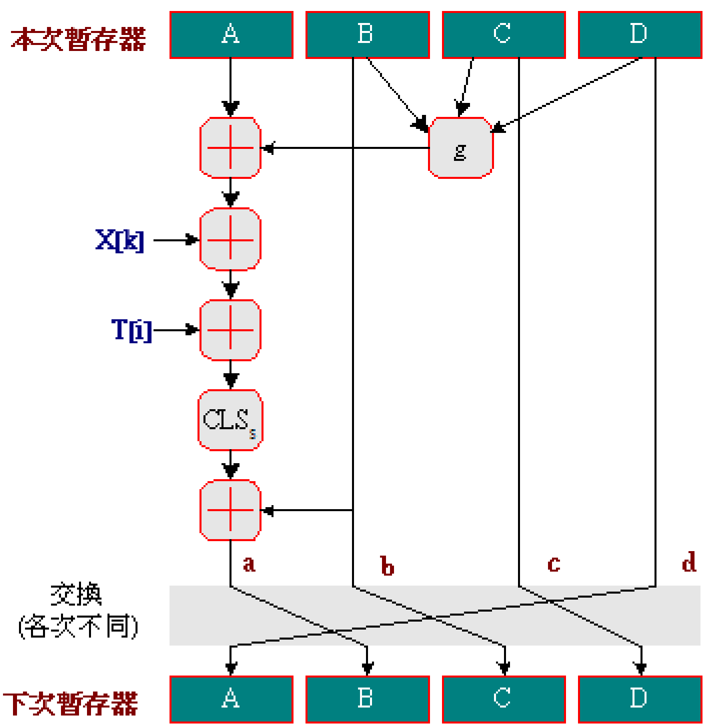

圖 4-6 MD5 壓縮函數的功能圖

其中：

- **a, b, c, d**：為四個 32 位元的暫存器。
- **g：**每一回合有不同的基本函數，分別是 F、G、H 與 I。
- **<<< s**：表示向左迴旋 s 個位元。
- **X[k]**：表示區塊明文中的第 k 個 32 位元字元。
- **T[i]**：表示圖 5-5 中的 Sine 函數值。
- **+** ：取 32 位元的同餘加法（mod 2^{32}）。

上述的意思是，每次（每回合有 16 次）只針對暫存器 a 的內容做運算，而 b、c 與 d 的內容不變；計算後的 a、b、c、d 暫存器分別填入下次的 B、C、D、A（各次皆不相同），做為下一次計算的暫存器內容。且每一回合（計有四個回合）的基本函數 g 亦不相同。MD5 利用 4 個邏輯運算子，來實現基本函數，分別是 AND（符號：$\wedge$）、OR（$\vee$）、NOT（$\neg$）與 XOR（$\oplus$），各個回合的基本函數歸類如下：

- 第一回合：F(b, c, d) = $(b \wedge c) \vee (\bar{b} \wedge d)$ = bc + $\bar{b}$ d
- 第二回合：G(b, c, d) = $(b \wedge d) \vee (c \wedge \bar{d})$ = db + $\bar{d}c$
- 第三回合：H(b, c, d) = b⊕c⊕d
- 第四回合：I(b, c, d) = c⊕ $(b \wedge \bar{d})$ = c⊕ $b\bar{d}$ = $c(b+\bar{d}) + \bar{c}(\bar{b}+d)$

函數 F 是一個條件函數：If b then c else d；同樣的，G 也是條件函數：If d then b else c；函數 H 是互斥或產生一個同位位元（Parity bit）；函數 I 是：If c then (b or not d) else (not b or d)。

對 MD5 而言，每四個回合都有一個基本函數（F、G、H 與 I），並且每一回合都計算 16 次；基本上，以 128 bits 的雜湊值而言，它是非常安全的。但對生日攻擊法而言，只要搜尋 2^{64} 的偽造訊息，仍然可以找出相同的雜湊值，就目前電腦速度來講，其安全性已漸堪慮。

4-4 SHA-1 演算法

『安全雜湊演算法』（Secure Hash Algorithm, SHA）是美國 NIST 所制定的標準，於 1993 年與 1995 年分別發表 FIPS PUB 180 與 FIPS PUB 180-1，目前都通稱後面的版本為 SHA-1。SHA 是以 MD4 為基礎發展出來的，其設計方式與 MD5 非常相似。首先我們列出 SHA-1 的特性，再來推演它的演算法，特性如下：

- **可以輸入不定長度的訊息**，但不可以超過 2^{64} 個位元。經過附加位元後必須是 512 位元的整數倍，但還餘有 448 個位元。填補方式與 MD5 一樣，需加入 64 位元的長度欄位。
- **附加位元後的訊息，以 512 位元為單位**，分割成若干個區塊；雖然，演算區塊的長度為 512 個位元；但每一區塊經過擴充字元，填入 32 位元的 W[t] 紀錄器，計有 80 個紀錄器（t =0, 1, 2, …, 79）。
- **每個步驟計算與最後運算的結果，皆得到 160 個位元的雜湊值。**
- SHA-1 區塊之間的演算程序和 MD5 一樣，如圖 5-3 所示。
- 利用 5 個 32 位元的暫存器（A、B、C、D 與 E），來儲存演算中的 160 位元的雜湊值。
- 演算步驟共有 4 個回合，每回合執行 20 次，共計 80 次的運算。
- 每回合含一個基本邏輯函數，計有 4 個邏輯函數（f_1、f_2、f_3 與 f_4）；並於每回合加入一個固定常數（$K_{1\sim4}$，或稱為『鹽』）。
- 每個步驟於演算基本邏輯函數時，會輸入相對應的常數（$K_{1\sim4}$）與訊息區段（W[t], t = 0, 1, 2, …, 79）。

SHA-1 演算法亦屬區塊運算方式，但較特殊的地方，是將512 個位元區塊擴充成 80 個 32 個位元的小區塊，每個執行步驟輸入一個小區塊。圖 4-7 為 SHA-1 演算法的功能圖，以下將說明它的運作程序。

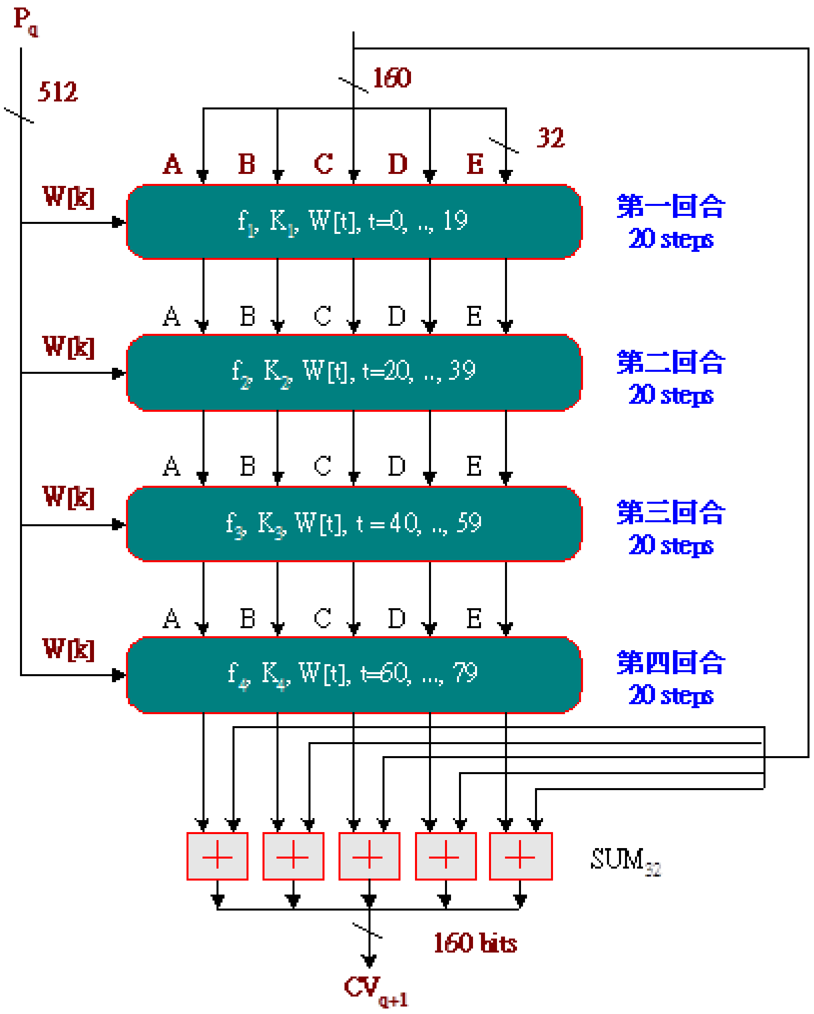

圖 4-7 SHA-1 演算法的功能圖

4-4-1 暫存器起始值

共計有 5 個 32 位元的暫存器，還未輸入計算之前，給予的起始值如下：（以 16 進位表示）

A：67 45 23 01
B：EF CD AB 89
C：98 BA DC FE
D：10 32 54 76
E ：C3 D2 E1 F0

前面 4 個暫存器與 MD5 相同，但 SHA-1 是採用『較高位元結尾』（Big-endian）的格式儲存（不同於 MD5）。

4-4-2 輸入常數

每一運算回合都會分別給一個固定常數（**類似『鹽』的功能**），四個回合共有四個常數（$K_{1\sim4}$）；如以步驟來計算，共有 80 個步驟（t），每個步驟的常數如下：（以 16 進位表示）

回合	步驟編號	輸入常數	取值方式 (整數)
第一回合	$0 \leq t \leq 19$	K_1 = 5A82799	$[2^{30} \times 2]$
第二回合	$20 \leq t \leq 39$	K_2 = 6ED9EBA1	$[2^{30} \times 3]$
第三回合	$40 \leq t \leq 59$	K_3 = 8F1BBCDC	$[2^{30} \times 5]$
	$60 \leq t \leq 79$		

第四回合	K_4 = CA62C1D6	$[2^{30} \times 10]$

4-4-3 訊息擴充

為了打亂訊息內資料的關聯性、或訊息的格式，SHA-1 採用了訊息擴充的方法，將訊息（512 個位元）擴充成 80 個 32 位元的小區塊（W[k], k=0, 1, …, 79）。其方法是，首先將訊息以每 32 位元為單位，分割為 16 個小區塊，分別存入 W[0]、W[1]、…、W[15] 之中，而第 16 個以後的小區塊，分別以下面公式計算填入：

$$W[t] = S^1(W[t-16] \oplus W[t-14] \oplus W[t-8] \oplus W[t-3]),\ t = 16, 17, \ldots, 79$$

譬如：

$$W[16] = S^1(W[0] \oplus W[2] \oplus W[8] \oplus W[13])$$
$$W[17] = S^1(W[1] \oplus W[3] \oplus W[9] \oplus W[14])$$
$$\ldots\ldots,$$
$$W[79] = S^1(W[63] \oplus W[65] \oplus W[71] \oplus W[76])$$

其中 S^1 表示向左迴旋一個位元的意思。由此可見，SHA-1 的訊息擴充方法是，前面 16 個小區塊是由原訊息分割得來；而第 16 個小區塊以後，是由前面某4 個小區塊之間執行 XOR 運算之後，再向左迴旋一個位元。如此說來，應該可以完全攪碎訊息的關聯性。

SHA-1 演算法中，每執行一個步驟（共計 80 個步驟），便輸入一個訊息的小區塊。譬如，第一個步驟（t=0），則輸入 W[0]；第二個步驟（t=1），則輸入 W[1]；依此類推，到了第 80 個步驟，剛好使用完最後的小區塊 W[79]。

4-4-4 壓縮函數

接下來介紹 SHA-1 的重頭戲：**『壓縮函數』（Compression Function）**。其功能是將上一步驟所計算的結果（CV_q），再重複計算一次，得到本次的計算結果（CV_{q+1}）；演算當中會加入參數 $k_{1\sim4}$ 與訊息小區塊 W[t]；總共計算了 80 次，其中又分為 4 個回合。SHA-1 壓縮函數的功能如圖 4-8 所示，運算過程如下：（以 5 個暫存器 A、B、C、D、E 為計算對象）

$$A = E + f_{1\text{-}4}(B, C, D) + S^5(A) + W[t] + K_{1\text{-}4}$$
$$B = A$$
$$C = S^{30}(B)$$
$$D = C$$
$$E = D$$

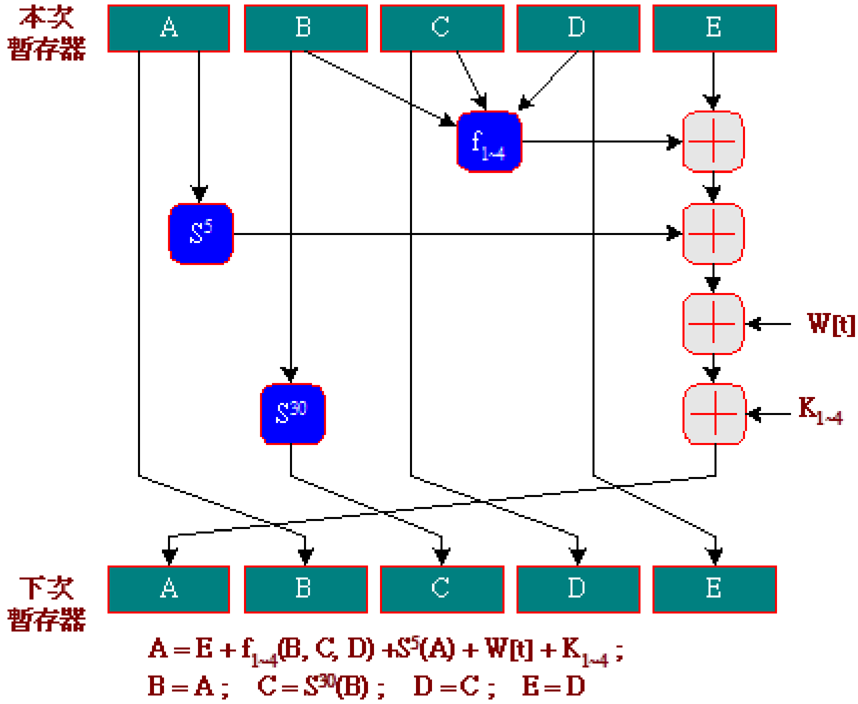

圖 4-8 SHA-1 壓縮函數的功能圖

其中，$S^n(A)$ 表示將暫存器 A 向左迴旋 n 位元的意思；『+』表示取 2^{32} 同餘（mod 2^{32}）的加法。第一回合（$0 \leqq t \leqq 19$）會輸入常數 k_1；第二回合（$20 \leqq t \leqq 39$）會輸入常數 k_2；依此類推，第四回合輸入參數為 k_4。

SHA-1 設計了 4 個基本邏輯函數（$f_{1\sim4}$），分別使用於每一回合的計算中。各個基本邏輯函數與使用時機如下：

回合	步驟編號	基本邏輯函數	簡式
1	$0 \leqq t \leqq 19$	$f_1 = (B \wedge C) \vee (\overline{B} \wedge D)$	$BC + \overline{B}D$
2	$20 \leqq t \leqq 39$	$f_2 = B \oplus C \oplus D$	$B \oplus C \oplus D$
3	$40 \leqq t \leqq 59$	$f_3 = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$	$BC+BD+CD$
4	$60 \leqq t \leqq 79$	$f_4 = B \oplus C \oplus D$	$B \oplus C \oplus D$

其中，AND、OR、NOT 與 XOR 分別以 、∨ 、 ¯ 與 ⊕ 符號表示。

4-5 破解雜湊函數

基本上，雜湊函數是將原來訊息打亂，得到另一個亂無章法的雜湊值，而且是越亂越好。很不幸的，我們發現大部分的傳輸訊息都有一定的格式，譬如，信件一開始就有 “Dear”，結束時又出現 “Your Best Regards”。或者資料檔案都有一定的格式，攻擊者既然知道了明文，就可以依照這些標準格式去修改內容，以達到相同的雜湊值。我們用一個例子來說明，假設春嬌希望約志明明天見面（See you tomorrow），寫好了信件，並建立了雜湊值（也許經過加密），她想應該很安全就交由世雄去發送；世雄原來就愛慕著春嬌，看了這封信當然很不高興，就利用他的資訊安全專長，找出 {See you yesterday}、{See you upset}、{don’t see you again} 等等，只要能符合相同的雜湊值即可，再將偽造信與春嬌簽署的簽署碼（加密的雜湊值）一起發送給志明，如此一來，就達到破壞他們之間的感情目的。接下來，我們來看看世雄到底用什麼技巧來攻擊雜湊演算法。

4-5-1 生日攻擊法

從數學的觀點來看，訊息經由雜湊演算法計算後，所產生的雜湊碼越長，則可能遺失的訊息就越少；如果採用較短的雜湊碼，可能遺失的訊息相對較多，不同訊息之間產生相同雜湊碼的機會也相對較大，這種觀念如同加密演算法，雜湊碼的位元數越長就越安全。攻擊者嘗試以不同的訊息來產生相同的雜湊碼，像是『暴力攻擊法』，但在雜湊演算法常稱為**『生日攻擊法』（Birthday Attack）**；首先就**『生日迷失』（Birthday Paradox）**的特性來觀察 [58]。

『生日迷失』是機率的問題（這裡僅簡略說明，請參考 [136]）。話說某位教授於課堂上向學生提出一個問題，問到需要多少位學生才能找出生日相同的機率大於 1/2。這個問題許多學生可能會利用一般機率的配對方式來計算，在 n 個學生之間的配對組合是 n(n-1)/2；而可能出現相同生日的機會是 1/365（一年 365 天）；如果欲得到的機率是 1/2，則 n 的最小值應該不難算出。許多學生相繼算出n 值，其中幾乎都超過 100。然而正確答案只有 23，也就是說，只要在 23 位學生之中，發生在任意兩個學生是相同生日的機率就會超過 1/2。這種觀念換成雜湊演算法就是，在 n 的訊息當中，需要嘗試過多少種訊息才可以找出相同的雜湊碼？一般雜湊碼的長度皆以二進位的長度來度量，譬如，某一雜湊演算法所產生的雜湊碼為 64 個位元，可能出現的組合是 2^{64} 種雜湊碼，但依照生日迷失的計算，只要嘗試 2^{32} 個訊息，就可以找到相同的雜湊碼。

生日攻擊的構想是由 Yuval [136] 所提出，其攻擊策略如下：（假設雜湊碼為 64 個位元）

- 攻擊者已得到一份經由簽署者簽署過的文件，簽署方式是明文後面加上已加密的雜湊值（使用簽署者的私有鑰匙）。
- 攻擊者利用明文與已知的雜湊演算法，計算出原明文的雜湊值。
- 接著，再依照該明文的格式修改其內容，譬如，保留原來空白鍵、使用較接近的文字，製造出其它偽造明文；並經過雜湊演算法計算出是否與原明文相同的雜湊值。
- 攻擊者針對此明文產生 2^{32} 個不同的變形，一定可以找出相同雜湊值的偽造明文。
- 攻擊者將偽造的明文與原來的簽署碼（原雜湊值經過加密或簽署的）結合起來，一併送出。
- 接收者收到文件，利用簽署者的公開鑰匙驗證該簽署碼無誤之後，則判定該文件的確是由簽署者所發沒錯；如此一來，攻擊者便達到目的了。

簽署者沒有想到利用私有鑰匙簽署過的文件，還是會被攻擊者冒充，主要關鍵在於雜湊演算法的複雜度不夠，與加密演算法（或簽署演算法）的強度無關，所以64位元的雜湊碼仍然過於脆弱，

唯有增加長度才可確保安全。

4-5-2 中途相遇攻擊法

『中途相遇攻擊法』（Meet-in-the-Middle Attack）的先決條件是得到一份明文與發送者所簽署的數位簽章。一般來講，雜湊演算法都是公開的，攻擊者可依此計算出明文的雜湊值；接著，依照雜湊值的長度，將明文分割為若干個區塊（如圖 5-3 所示，容後介紹），再將偽造的明文區分為同樣大小的區塊，使用每一區塊所產生的雜湊值來比較，如果不相同，則改變偽造明文使其中間值相同，依此類推，便可找到相同雜湊碼的偽造文。其步驟如下：（假設雜湊值長度為64位元）

- 根據已知的明文，產生雜湊值。
- 將明文（P）依照雜湊值的長度（64 個位元），區分為若干個區塊（$P_1, P_2, \ldots, P_n$）。
- 攻擊者製造另一個偽造文（Q），也依照 64 個位元的雜湊值長度，區分為若干個區塊（$Q_1, Q_2, \ldots, Q_n$）。
- 將明文與偽造文的相對區塊，送給雜湊函數計算，如得到 $H(P_j) = H(Q_j)$，則接下一區塊；否則修改偽造文，使其達到目的（故稱為中途相遇攻擊法），最多在 2^{32} 個不同的 Q_j 便可找到相同的雜湊值。搜尋完畢之後，再比較最後的雜湊值是否與原文的雜湊值相同。
- 找出相同的雜湊值之後，將此偽造文連同原文傳送給接收端。

由上述的推演，無論使用何種加密演算法，還是會被破解掉；其中最主要關鍵在於攻擊者根本不用理會加密演算（或加密鑰匙），只要找出相同雜湊值，便可達到攻擊目的；根本解決之道，除了不斷加強雜湊演算法的複雜度之外，增加雜湊值的長度方能達到安全效果。

4-6 亂數產生器

『亂數』（Random Number，或 Nonce）[32, 36, 83]在資訊安全上常常扮演非常重要的角色，我們將其應用歸類如下：

- **產生通訊鑰匙：**通訊一方或許會選用一個亂數作為會議鑰匙（秘密鑰匙演算法使用），再用加密方式傳送給對方。
- **演算法參數：**許多演算法都需要亂數來增加它的神秘性。譬如，RSA 演算法或 Diffie-Hellman 演算法都需要選擇亂數來作為製造鑰匙的材料。
- **計算器：**許多通訊協定（如 IPSec）都會選擇一個亂數計數器做為封包的序號，並作為防禦重複攻擊的辨識。
- **交叉確認**：如雙方需要確認所持有的鑰匙是否相同時，某一方會選用一個亂數再加密後，傳送給對方；對方解密後，再將亂數加一（或其他演算），加密後回傳給發送者；如此便可以確定雙方的鑰匙是否相同。

其實，亂數的應用不祇侷限於此，還有許多地方會應用到。如果我們回想一下，為何使用亂數？主要原因在於其『隨機性』並且是『不可預測的』。利用這兩因素所構成的通訊關鍵，別人就無法預估出來我們的通訊方式，如此便能達到資訊安全的『神秘性』功能。但話說回來，電腦是一個死板的計算工具，僅僅會依照所既定的程序（或程式）計算出所期望的數值。因此，無論採用何

種計算程序所算出來的亂數，還是有蛛絲馬跡可以尋找出亂數的出現規律；攻擊者只要能預測出可能使用的亂數，依然能輕而易舉的破解安全系統。況且任何一套系統都有各自的亂數產生器，並且產生一個不可預測的亂數，必須經過極複雜的運算程序，相對地會增加系統的負擔。因此，亂數的複雜度視其應用範圍而定。譬如，樂透彩卷是利用電腦（假如是）計算出六個幸運號碼，不論它的運算程序有多複雜，總是有人會不計成本去分析，因為只要能找出數字出現的規律，就有機會變成億萬富翁！如此說來，該電腦裡的亂數產生器非經過特殊設計不可。其實，許多系統利用物理現象來計算亂數，譬如，採用鍵盤鍵入時間間隔、游離電容量、或週邊電磁感應量等等，雖然這些因素會時改變，但欲製作它確實不易，除非是較專屬的系統，否則甚少使用。

既然亂數是利用某一演算法所計算出來，因此又稱為**『虛擬亂數』（Pseudo-random Number, PRN）**，以下介紹幾種與資訊安全有關的亂數產生函數。

4-6-1 典型亂數產生器

到目前為止，最常見的典型亂數產生器是**『線性同餘法』**，此演算法的計算公式為：

$$X_{n+1} = (a\,X_n + c) \bmod m$$

所產生的亂數序列為 $\{X_n\}$；它的計算方式是由前一個亂數，再加入其它參數之後，以同餘計算出這一次的亂數。同餘計算是擷取除法運算之後的餘數，如果所加入的參數與所取用的除數都很適當的話，所產生的亂數應該符合『隨機性』與『不可預測性』。接下來，說明相關的四個參數：

- **m 模數（Modulus）**：$m > 0$；最好是夠大的質數。
- **a 乘數（Multiplier）**：$0 \leqq a < m$。
- **c 增量（Increment）**：$0 \leqq c < m$。
- **X_0 起始值，或稱種子（Seed）**：$0 \leqq X_0 < m$。

如果：m、a、c 與 X_0 都選用整數的話，則會產生一序列的整數 X_n，其中為 $0 \leqq X_n < m$。

我們用較簡單的參數來測試看看，假設：$m = 7$、$a=2$、$c=1$、$X_0=1$，則所產生的亂數序列為：$X_1 = 3$、$X_2 = 0$、$X_3 = 1$、$X_4 = 3$、...；由此可以看出所產生的亂數序列為 {0, 1, 3, 0, 1, 3, .., }，這樣的選擇方式可能只出現 3 種數字（亦即週期只有 3）。如希望所產生的亂數夠亂的話，就必須將所有可能出現的週期加大（加大m，並挑選更佳的a、c）；如此一來，便可以在較長的序列中隨機尋找一個數字，也比較不容易被猜出來。增加序列週期最基本方法是增加 m 的數值（當然 a 與 c 也會有所關聯），一般我們都會將該電腦所能計算的能力，來作為週期出現率；譬如，以 32 位元處理機為例，

同餘模式計算就採用 2^{32} -1（mod 2^{32}-1，即是 m = 2^{32} -1）；而此函數又稱為**『全週期』（Full-period）**函數，公式如下：

$$\mathbf{X_{n+1} = (a\ X_n + c)\ mod\ 2^{32}-1}$$

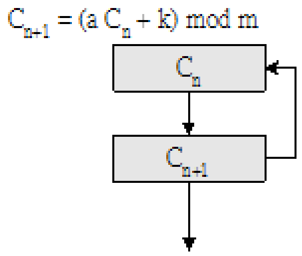

圖 4-8-1

接下來的重點是如何選用 a、c 與 X_0，其中 X_0 與 c 只是隨機參數而已，依照統計分析的結果，選擇任何值對安全性來講並不是影響很大，然而 a 的內容就另當別論了。至少我們期望上一次的值（X_n）與 a 相乘以後，儘可能會大於 m，如此所產生的亂數變化才會較大；但 a 的值越大的話，處理機的計算負荷相對越高；譬如 IBM 360 系列採用 a = 7^5 = 16807，所產生的亂數應該夠複雜了。

線性同餘法也是可能會遭受破解的，譬如，大多知道必然會採用全週期函數（m = 2^{32}-1），只要能找出 X_0、a 與 c，即可猜測出下一個亂數可能出現的數字。假設攻擊者收集到 X_1、X_2 與 X_3，則可排列出以下式子：

$$X_1 = (aX_0 + c) \bmod m$$
$$X_2 = (aX_1 + c) \bmod m$$
$$X_3 = (aX_2 + c) \bmod m$$

利用這三個方程式就可容易找出 a 與 c（也可以找出 m）。由此可見，只要得到這三次以上的亂數，便可猜測出下一次可能出現的亂數。由此可見，採用線性同餘法仍需週期性的改變相關參數才行。

4-6-2 DES 亂數產生

雖然，採用線性同餘法所製造出的亂數是可以預測的，但在許多地方還是需要它這種特性。譬如，串流加密法或無線網路傳輸（[4] 第十五章），都需要雙方能同時產生相同的『虛擬亂數』；此時，只要雙方協議好 m、a、c 與 X_0 四個參數的話，便能夠同時產生相同的虛擬亂數。但要如何來增加它的安全性呢？我們可以利用密碼學的加密方法，來打亂所產生的亂數序列。圖 4-9 為全週期亂數再經過加密的運作程序。其功能是：**首先利用線性同餘法產生全週期亂數之後，再經過某一加密**

演算法（如 DES 或 AES 演算法）處理，處理出來便將原來亂數的格式再打亂一次；如果通訊雙方都握有相同的鑰匙，也使用相同的演算法，如此一來，雙方應該可以產生相同的虛擬亂數。要注意的是，系統必須特別加強保護這把專門用來產生亂數的鑰匙（又稱為主鑰匙），要是被盜走的話，所產生的亂數便有可能被猜算出來，系統的安全就會出大問題了。

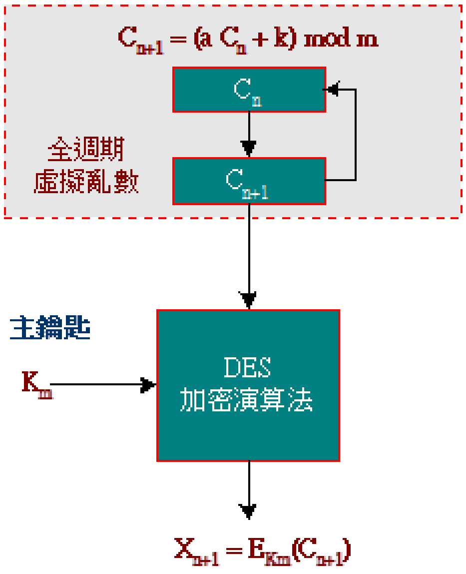

圖 4-9 利用 DES 加密器產生虛擬亂數

4-6-3 ANSI 亂數產生

ANSI（America National Standard Institute）所制定的 ANSI X9.17 亂數產生器，可以說是目前最可靠的資訊安全技術之一；許多安全性系統（如 PGP）也都採用這種技術。ANSI X9.17 是採用 Triple-DES（3DES） 加密法，所需計算的參數是採用隨時變動的日期與時間，當然也會用到一般亂數所產生的初始值，如圖 4-10 所示；將其特性歸類如下：

- **輸入參數：**使用兩個輸入參數。一者為隨時變動的日期與時間，並以 64 位元來表示；每計算一次，下一次計算的參數就會被更新。另一者為初始值，第二次以後計算，也會製造另一個亂數種子（Seed），做為下一次計算的初始值；第一次的初始值或許會採用一般系統的亂數。
- **加密鑰匙：**本演算法係採用 Triple-DES 加密法，並使用兩把鑰匙的演算法，因此鑰匙長度為 112（=56 × 2）。此鑰匙必須保護好，而且只能使用於虛擬亂數產生上。
- **亂數輸出：**亂數輸出也是 64 個位元（符合 3DES 演算法）；另外輸出一個 64 位元的種子（Seed），作為下一次計算虛擬亂數的參數。

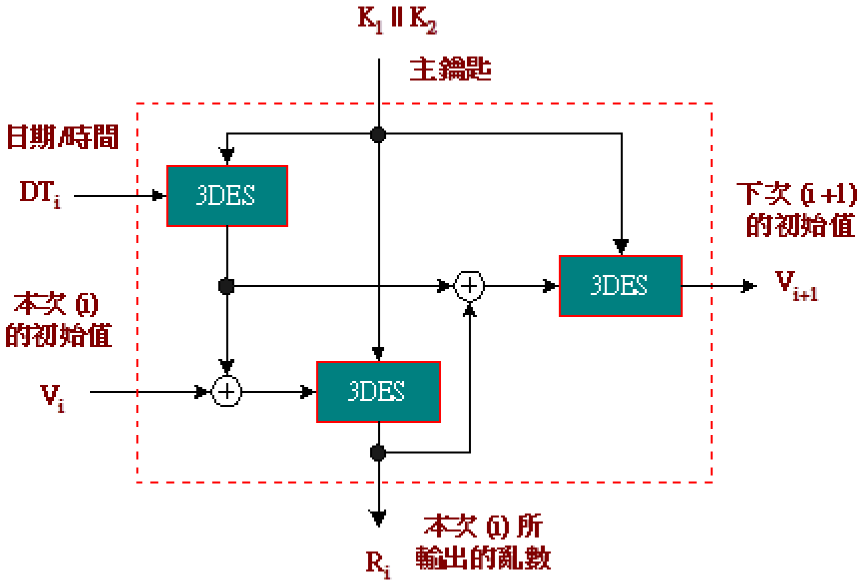

圖 4-10 ANSI X9.17 亂數產生器

我們依照圖 4-10 來定義下列數值：

- DT_i ：產生第 i 個亂數時，所輸入的日期與時間。
- V_i ：產生第 i 個亂數時，所輸入的種子值（或初始值）。
- R_i ：所產生出來的第 i 個亂數。
- $K_1 \| K_2$：產生亂數的主鑰匙。

演算法的計算輸出為：

- $R_i = 3DES_{K1\|K2}\,[Vi \oplus 3DES_{K1\|K2}\,[DT_i]]$
- $V_{i+1} = 3DES_{K1\|K2}\,[R_i \oplus 3DES_{K1\|K2}\,[DT_i]]$

如此所計算出來的虛擬亂數（R_i），其中包含了 3 個 3DES 加密器、隨時改變的日期與時間、以及循環計算出的種子（V_{i+1}，Seed）；再說，鑰匙長度又高達 112 個位元，要破解它的確是不容易的。就算攻擊者得到了當次的亂數（R_i），也無法計算出下一次種子的值（V_{i+1}）。