網路安全的理論與實務

楊中皇 著

第四章 單向雜湊函數

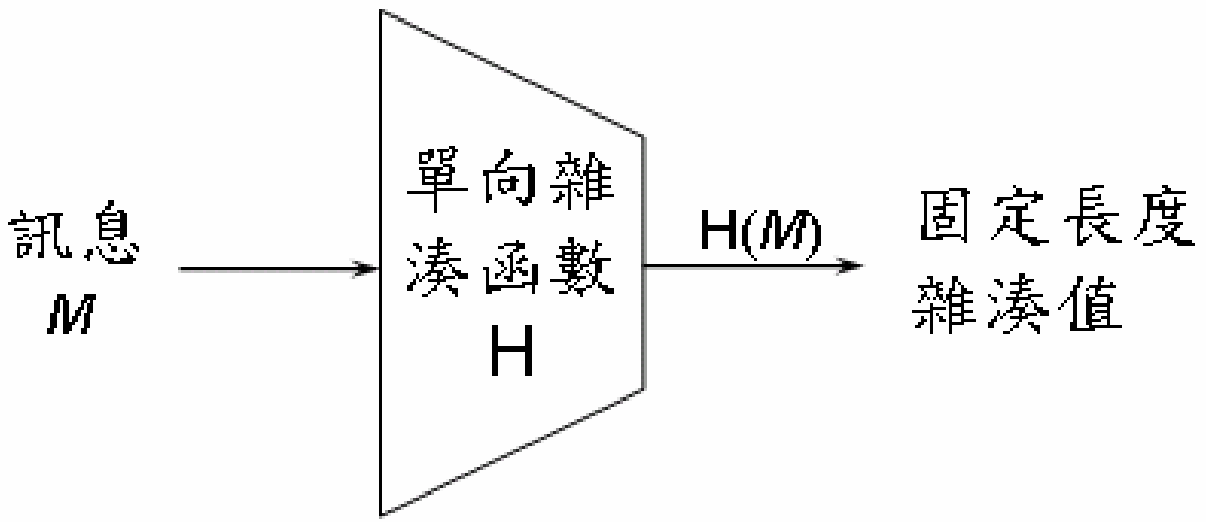
http://crypto.nknu.edu.tw/textbook/

第四章 單向雜湊函數

- 單向雜湊函數
- MD5雜湊函數
- SHA-2雜湊函數
- HMAC金鑰雜湊訊息確認碼

單向雜湊函數(one-way hash functions)

- 單向雜湊函數(one-way hash function)又被稱爲訊息指紋(message fingerprint)演算法或訊息摘要(message digest)演算法
- 任意長度的輸入訊息透過單向雜湊函數的計算可求得一個固定長度的訊息雜湊值



碰撞抵抗性(collision resistance)

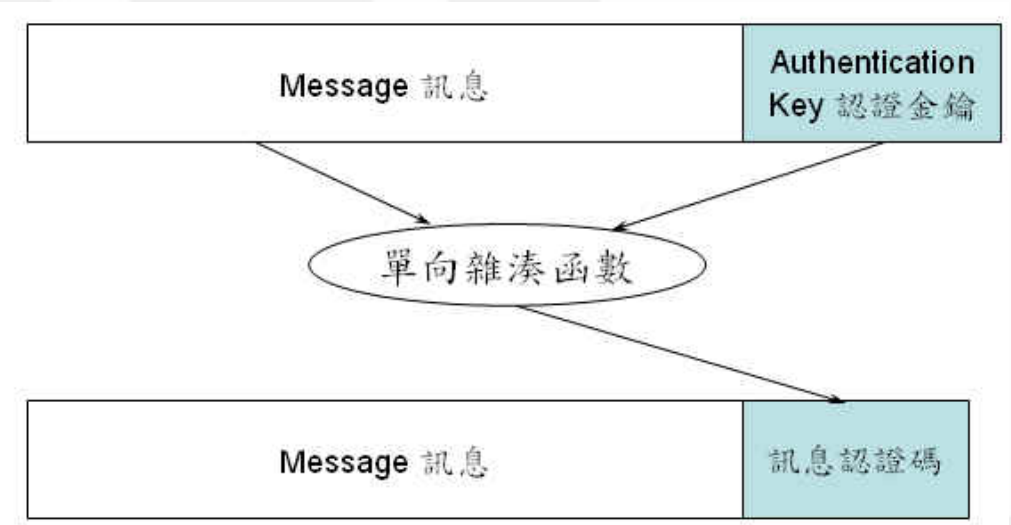
- 找不到兩個相異的訊息M_1與M_2會被計算出相同的輸出訊息指紋
- 若我們以符號H代表單向雜湊函數，則要從單向雜湊函數的輸出$y = \mathrm{H}(x)$找出x，在計算上是不可能的，這也就是爲何稱爲單向的原因。我們希望在計算上做到　　若$M_1 \neq M_2$，則$\mathrm{H}(M_1) \neq \mathrm{H}(M_2)$
- 如果訊息摘要長度爲n位元，則有強碰撞抵抗性的單向雜湊函數的安全性約爲(利用所謂的生日攻擊法評估)　　$\sqrt{2^n} = 2^{n/2}$

常見單向雜湊函數的比較

	MD5	SHA-1	SHA-256	SHA-384	SHA-512
摘要長度(位元組)	16	20	32	48	64
安全性	不安全	有疑慮	2^{128}	2^{192}	2^{256}
最大訊息長度(位元)	2^{64}-1	2^{64}-1	2^{64}-1	2^{128}-1	2^{128}-1
標準公佈年份	1992	1995	2002	2002	2002

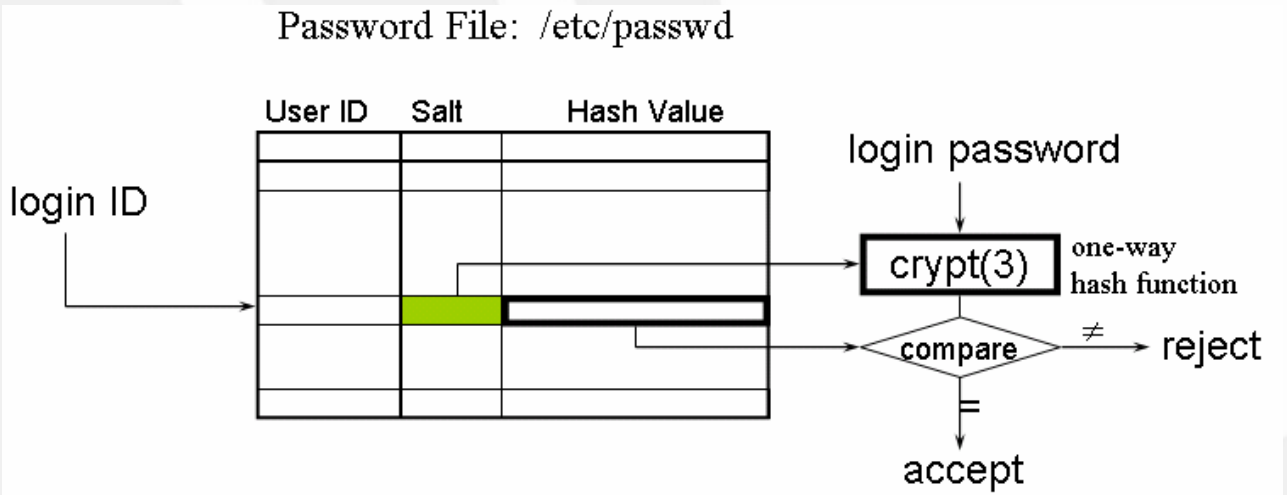
單向雜湊函數用於訊息防偽

- 訊息發送者先將訊息與訊息認證用的金鑰作爲單向雜湊函數的輸入，計算訊息摘要，再取訊息摘要的某些位元附加在原有訊息當作所謂的訊息認證碼(message authentication code)。
- 訊息與訊息認證碼將傳送給接收者，而兩者皆可能在傳送過程中被竄改，然而接收者可以重新計算收到之訊息與秘密金鑰的雜湊函數值，再取雜湊函數值的某些位元與收到的訊息認證碼做比較，相同時才接受，不同時則代表資料或認證碼被竄改



單向雜湊函數用於驗證通行碼

- 每個使用者有帳號(ID)與通行碼，兩者輸入正確時才允許登入。第一次設定使用者帳號時，我們隨機挑一個12位元的亂數Salt與使用者原先設定的通行碼合併，然後進行雜湊函數的計算，訊息摘要值儲存於/etc/passwd檔案對應到使用者ID帳號的欄位。下一次有人想要登錄該帳號時，若該帳號存在，就從檔案/etc/passwd將新輸入ID的資料取出。其中12位元的亂數Salt與目前想使用者輸入之通行碼合併，然後進行雜湊函數的計算，再與存放於/etc/passwd檔案內剛取出的該ID使用者的雜湊函數做比較，錯誤的話代表通行碼錯誤



MD5雜湊函數

- MD5雜湊函數是麻省理工學院教授R. Rivest設計
- 源自於更早的MD4雜湊函數加以改良而成
- RFC 1321標準文件不只提供MD5程式，也含有測試程式，容易瞭解
- MD5演算法的輸入是一個不超過2^{64}個位元長度的訊息，而固定產生輸出爲128位元的訊息雜湊值
- MD5是到目前爲止最被廣泛使用的雜湊演算法
- MD5已被中國山東大學的王小雲教授等學者於2004年所破解

MD5雜湊函數步驟

- 下列四步驟組成：
- 步驟一：添加附加位元(padding bits)
- 步驟二：儲存訊息長度
- 步驟三：初始化暫存器
- 步驟四：以512位元(16字元組)爲單位，處理訊息

MD5步驟一：添加附加位元(padding bits)

- MD5首先需進行添加位元，使總長度與448位元在mod 512中同餘
- 附加的方式是先固定加個位元1，如果需要再用位元0補到所需的長度。我們一定要先添加一個“1”，然後再添加0到511個“0”
- 假設訊息爲三個英文小寫字母"abc"，共24位元，則先添加位元"1"，成爲25位元，接著需再添加223個"0"，使得長度成爲448位元。但是如果原始長度剛好爲512位元的倍數，則須添加一個"1"與447個"0"

423個0　　64位元表示原始長度爲 24個位元

$$\underbrace{01100001}_{a}\ \underbrace{01100010}_{b}\ \underbrace{01100011}_{c}\ 1\ \overbrace{00...00}\ \overbrace{00...011000}$$

MD5步驟二三

- 步驟二：儲存訊息長度
  - 將真正明文長度(沒有添加位元之前)以64位元的表示方法附加於前面已增添過的明文後端。此時，整個新明文長度正巧是512位元的倍數。
- 步驟三：初始化暫存器
  - 設定4個32位元(1字元組)的暫存器，這些暫存器在步驟四將與輸入的訊息進行計算，每次與512位元運算產生新值，最後輸出4個暫存器的內容128位元(32x4=128)

state[0] = 67452301
state[1] = EFCDAB89
state[2] = 98BADCFE
state[3] = 10325476

MD5步驟四: 以512位元爲單位處理訊息

- 每次處理512位元，先複製到變數X[0], X[1], …, X[15]，每個X[i]爲16字元組(word)，1字元組等於32 位元
- 處理的流程可分爲四個回合，每回合用到MD5專有的邏輯函數

MD5的512位元訊息區段演算過程

```
    a = state[0], b = state[1], c = state[2], d =
state[3]
/* MD5 回合 1 */
    FF (a, b, c, d, x[ 0], 7, 0xd76aa478); /* 1 */
    FF (d, a, b, c, x[ 1], 12, 0xe8c7b756); /* 2 */
    FF (c, d, a, b, x[ 2], 17, 0x242070db); /* 3 */
    FF (b, c, d, a, x[ 3], 22, 0xc1bdceee); /* 4 */
    FF (a, b, c, d, x[ 4], 7, 0xf57c0faf); /* 5 */
    FF (d, a, b, c, x[ 5], 12, 0x4787c62a); /* 6 */
    FF (c, d, a, b, x[ 6], 17, 0xa8304613); /* 7 */
    FF (b, c, d, a, x[ 7], 22, 0xfd469501); /* 8 */
    FF (a, b, c, d, x[ 8], 7, 0x698098d8); /* 9 */
    FF (d, a, b, c, x[ 9], 12, 0x8b44f7af); /* 10 */
    FF (c, d, a, b, x[10], 17, 0xffff5bb1); /* 11 */
    FF (b, c, d, a, x[11], 22, 0x895cd7be); /* 12 */
    FF (a, b, c, d, x[12], 7, 0x6b901122); /* 13 */
    FF (d, a, b, c, x[13], 12, 0xfd987193); /* 14 */
    FF (c, d, a, b, x[14], 17, 0xa679438e); /* 15 */
    FF (b, c, d, a, x[15], 22, 0x49b40821); /* 16 */
```

MD5函數

MD5演算法的四個邏輯函數FF, GG, HH, II

FF (a, b, c, d, x, s, ac): $a = ROTATE\_LEFT\ (b + ((a + F(b,c,d) + x + ac), s)$

GG (a, b, c, d, x, s, ac): $a = ROTATE\_LEFT\ (b + ((a + G(b,c,d) + x + ac), s)$

HH (a, b, c, d, x, s, ac): $a = ROTATE\_LEFT\ (b + ((a + H(b,c,d) + x + ac), s)$

II (a, b, c, d, x, s, ac): $a = ROTATE\_LEFT\ (b + ((a + I(b,c,d) + x + ac), s)$

MD5 演算法的字元組處理函數

$ROTATE\_LEFT(x, s) = (x << s) \vee (x >> 32 - s)$, 字元組$x$向左旋轉$s$個位元

$$F(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$$

$$G(x, y, z) = (x \wedge z) \vee (y \wedge \neg z)$$

$$H(x, y, z) = x \oplus y \oplus z$$

$$I(x, y, z) = y \oplus (x \vee \neg z)$$

MD5 演算法的運算子

- $\wedge$: 位元*AND*運算
- $\vee$: 位元*OR*運算
- $\oplus$: 位元*XOR*運算
- $\neg$: 位元補數運算
- $+$: 取2^{32}的同餘加法運算
- $<<$: 位元左移運算，$x << s$就是x向左移s個位元，右邊補s個0
- $>>$: 位元右移運算，$x >> s$就是x向右移s個位元，左邊補s個0

OpenSSL進行MD5運算

- 假設將三位元組”abc”放於檔案”foo”，那麼可用下列指令來計算檔案”foo”的MD5雜湊函數值

  openssl dgst –md5 foo

命令提示字元

```
C:\openssl\OUT32>type foo
abc
C:\openssl\OUT32>openssl dgst -md5 foo
MD5(foo)= 900150983cd24fb0d6963f7d28e17f72
```

OpenSSL測試MD5效率

- openssl speed md5

命令提示字元

```
C:\openssl>cd out32

C:\openssl\OUT32>openssl speed md5
To get the most accurate results, try to run this
program when this computer is idle.
First we calculate the approximate speed ...
Doing md5 10485760 times on 16 size blocks: 10485760 md5's in 25.16s
Doing md5 10485760 times on 64 size blocks: 10485760 md5's in 31.73s
Doing md5 2621440 times on 256 size blocks: 2621440 md5's in 11.96s
Doing md5 655360 times on 1024 size blocks: 655360 md5's in 7.40s
Doing md5 81920 times on 8192 size blocks: 81920 md5's in 6.01s
OpenSSL 0.9.8 05 Jul 2005
built on: Mon Sep 19 08:36:38 2005
options:bn(64,32) md2(int) rc4(idx,int) des(ptr,cisc,4,long) aes(partial) idea(i
nt) blowfish(idx)
compiler: bcc32 -DWIN32_LEAN_AND_MEAN -q -w-aus -w-par -w-inl  -c -tWC -tWM -DOP
ENSSL_SYSNAME_WIN32 -DL_ENDIAN -DDSO_WIN32 -D_stricmp=stricmp -O2 -ff -fp -DOPEN
SSL_NO_RC5 -DOPENSSL_NO_MDC2 -DOPENSSL_NO_KRB5
available timing options: TIMEB HZ=1000
timing function used: ftime
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
md5               6669.01k    21153.31k    56125.17k    90675.40k   111680.59k
```

SHA-2雜湊函數

- SHA-2雜湊函數包括三種演算法，分別是SHA-256、SHA-384及SHA-512
  - 我們僅說明安全性爲2^{128}的SHA-256
- SHA-256演算法輸入的訊息長度不能超過2^{64}個位元，處理區段爲512個位元，在實際進行計算之前的前置處理有以下三個步驟：
  - 步驟一：訊息附加位元(與MD5相同)
  - 步驟二：儲存訊息長度(64個位元)
  - 步驟三：設定初始的雜湊值(8個32位元暫存器)

SHA-256的初值表

a = 6a09e667
b = bb67ae85
c = 3c6ef372
d = a54ff53a
e = 510e527f
f = 9b05688c
g = 1f83d9ab
h = 5be0cd19

SHA-256步驟四:512個位元爲區段進行處理

//根據第 i 個輸入區段 $M^{(i)}$導出 64 個 32 位元的字元組，以 Wt 表示個別的字元組

$$Wt = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

//設定前個訊息區段計算完的雜湊值到暫存器的 8 個字元組變數

$a = H_0^{(i-1)}$

$b = H_1^{(i-1)}$

$c = H_2^{(i-1)}$

$d = H_3^{(i-1)}$

$e = H_4^{(i-1)}$

$f = H_5^{(i-1)}$

$g = H_6^{(i-1)}$

$h = H_7^{(i-1)}$

//每個訊息區段，要經過 64 個基本運算

For t = 0 to 63:

{

$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e, f, g) + K_t^{\{256\}} + W_t$

$T_2 = \sum_0^{\{256\}}(a) + Maj(a, b, c)$

$h = g$

$g = f$

$f = e$

$e = d + T_1$

$a, b, c, ..., h$：MD 緩衝區的 8 個字元組變數。

$H^{(i)}$：第 i 個雜湊值，由 $a,b,c,...,h$ 所組成。

$H^{(0)}$表示初始值，$H^{(N)}$表示訊息摘要結果。

$H_j^{(i)}$：第 i 個雜湊值的第 j 個字元組。

K_t：第 t 個計算回合所用的常數。

M：前置處理後的訊息。

$M^{(i)}$：第 i 個訊息區段，每個訊息區段為512位元

$M_j^{(i)}$：第 i 個訊息區段的第 j 個字元組。

n：字元的旋轉位數或移位位數。

N：M 所分解的區段數。

T：雜湊計算過程中暫存的字元組。

W_t：根據目前輸入區段導出的字元組。

SHA-256的函數

SHA-256 演算法的六個邏輯函數

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$

$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$$

$$\sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$$

$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18} \oplus SHR^3(x)$$

$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19} \oplus SHR^{10}(x)$$

SHA-256 演算法的字元處理函數

$SHR^n(x) = x >> n$, 字元組x向右移n個位元

$ROTR^n(x) = (x >> n) \vee (x << 32 - n)$, 字元組$x$向右旋轉$n$個位元

SHA-256的常數K_t

SHA-256 的常數 K_t

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

SHA-256 演算法的運算子

∧：位元*AND*運算

∨：位元*OR*運算

⊕：位元*XOR*運算

¬：位元補數運算

+：取2^{32}的同餘加法運算

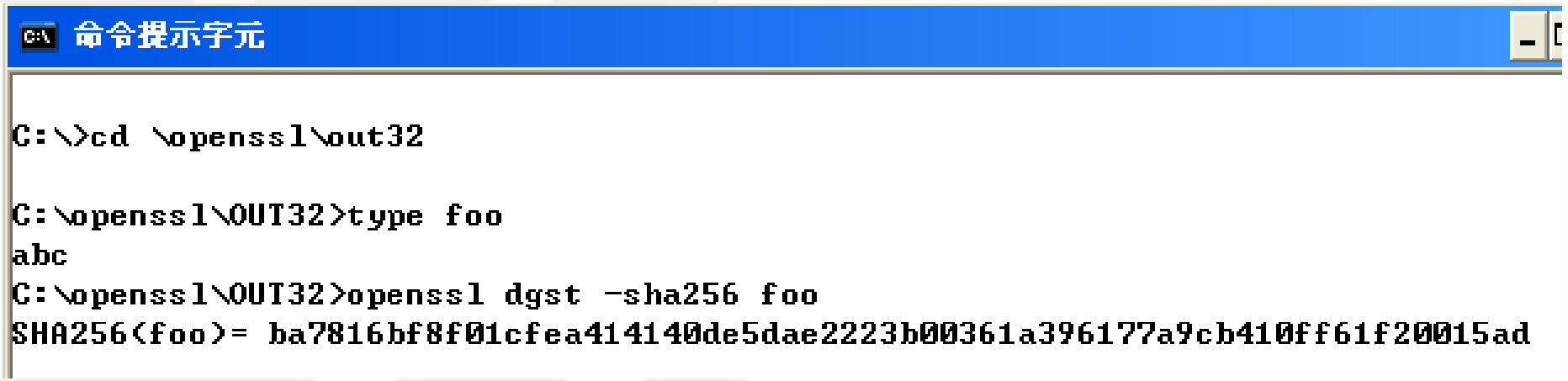
<<：位元左移運算，$x << n$就是x向左移n個位元，右邊補n個0

>>：位元右移運算，$x >> n$就是x向右移n個位元，左邊補n個0

OpenSSL進行SHA-256運算

- 假設將三位元組”abc”放於檔案”foo”，那麼可用下列指令來計算檔案”foo”的SHA-256雜湊函數值

  openssl dgst –sha256 foo



OpenSSL測試SHA-256效率

- openssl speed sha256

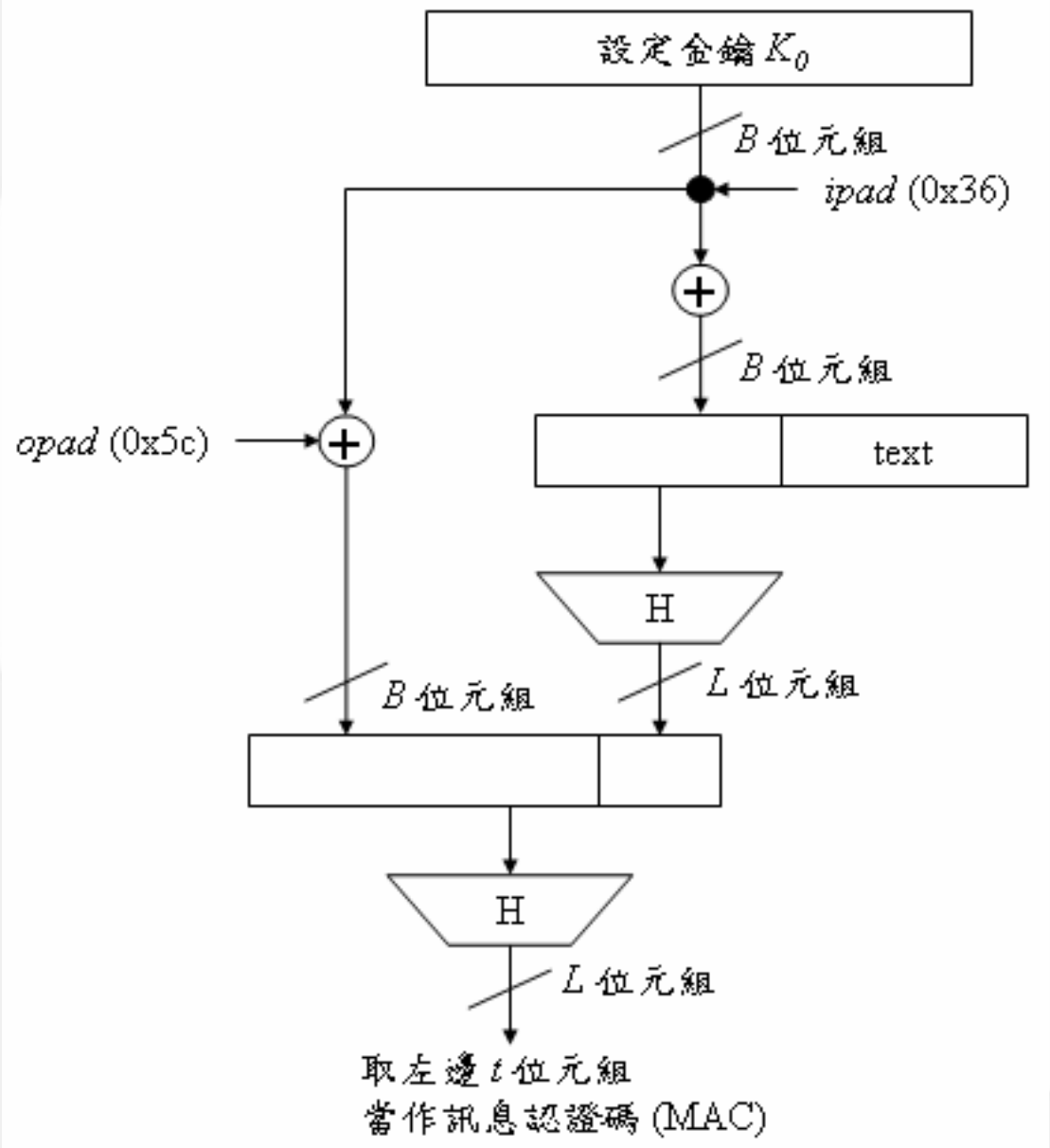
命令提示字元

```
C:\openssl\OUT32>openssl speed sha256
To get the most accurate results, try to run this
program when this computer is idle.
First we calculate the approximate speed ...
Doing sha256 10485760 times on 16 size blocks: 10485760 sha256's in 40.83s
Doing sha256 10485760 times on 64 size blocks: 10485760 sha256's in 65.53s
Doing sha256 2621440 times on 256 size blocks: 2621440 sha256's in 37.88s
Doing sha256 655360 times on 1024 size blocks: 655360 sha256's in 30.45s
Doing sha256 81920 times on 8192 size blocks: 81920 sha256's in 27.22s
OpenSSL 0.9.8 05 Jul 2005
built on: Mon Sep 19 08:36:38 2005
options:bn(64,32) md2(int) rc4(idx,int) des(ptr,cisc,4,long) aes(partial) idea(i
nt) blowfish(idx)
compiler: bcc32 -DWIN32_LEAN_AND_MEAN -q -w-aus -w-par -w-inl  -c -tWC -tWM -DOP
ENSSL_SYSNAME_WIN32 -DL_ENDIAN -DDSO_WIN32 -D_stricmp=stricmp -O2 -ff -fp -DOPEN
SSL_NO_RC5 -DOPENSSL_NO_MDC2 -DOPENSSL_NO_KRB5
available timing options: TIMEB HZ=1000
timing function used: ftime
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
sha256            4109.14k    10240.31k    17713.84k    22036.14k    24655.15k
```

金鑰雜湊訊息確認碼 (HMAC)

- 金鑰雜湊訊息確認碼(Keyed-Hash Message Authentication Code，HMAC)
- HMAC是一種利用單向雜湊函數提供私密金鑰訊息防偽
- 訊息發送者與訊息接收者有共同訊息防偽用秘密金鑰，而希望能偵測訊息在傳送過程是否被竄改
- 設計的概念是發送者先將私密金鑰與訊息透過某種計算過程算出所謂的訊息認證碼(message authentication code，MAC)後附加在訊息中
- 訊息與訊息認證碼將傳送給接收者，而兩者皆可能在傳送過程中被竄改，然而接收者可以重新計算訊息認證碼及與收到訊息認證碼做比較，符合時才接受，不符合時則代表至少訊息或認證碼之一被竄改

HMAC (FIPS 198)



HMAC (FIPS 198) 演算法運算流程

步驟	說明
1	如果訊息認証金鑰K的長度等於B，設$K_0=K$，直接跳到步驟4。
2	如果K的長度大於B，則對K作雜湊函數運算，再將其輸出結果附加B-L個0x00字元成為K_0。
3	如果K的長度小於B，則將K附加B-K個0x00字元組成為K_0。
4	將K_0與$ipad$作⊕運算，產生一串長度為B的訊息：$K_0 \oplus ipad$。
5	將步驟4所產生的結果接續原始輸入訊息成為：$(K_0 \oplus ipad) \| text$。
6	將步驟5的結果執行雜湊函數運算：$H((K_0 \oplus ipad) \| text)$。
7	將K_0與$opad$作⊕運算，產生另串長度為B的訊息：$K_0 \oplus opad$。
8	將步驟6的結果接續到步驟7的後面：$K_0 \oplus opad \| H((K_0 \oplus ipad) \| text)$。
9	將步驟8的結果執行雜湊函數運算：$H((K_0 \oplus opad \| H((K_0 \oplus ipad) \| text))$。
10	依據步驟9的結果，輸出最左邊的t個位元組作為最終輸出的雜湊訊息確認碼HMAC。

HMAC (FIPS 198)參數說明

B：雜湊函數每個處理區段的位元組數，例如 SHA-256 的 *B*=64。
H：選用的雜湊函數演算法。
L：雜湊函數 *H* 輸出的雜湊值長度，單位為位元組。例如 SHA-256 為 32。
K：原始的訊息認證金鑰。
K_0：經過前置處理後的金鑰，其大小應等於 B 個位元組。
0x*N*：表示 16 進位值為 *N* 的位元組。
ipad： *B* 個 16 進位值為 0x36 的位元組成的資料。
opad： *B* 個 16 進位值為 0x5C 的位元組成的資料。
t：輸出結果的 *HMAC* 碼長度。
text：未經任何處理的原始輸入訊息。
||：將欲處理的訊息接續(concatenation)。
⊕：執行 XOR 位元運算。

- *ipad* 是由*B*個0x36 (00110110二元)位元組組成，$K_0 \oplus ipad$會將一半K_0位元反相改變
- *opad* 是由*B*個0x5c (01011100二元)位元組組成，$K_0 \oplus opad$也會將一半K_0位元反相改變。
- 以HMAC演算法計算訊息確認碼的計算方程式可以表示如下：

$$MAC(text)_t = HMAC(K, text)_t = H((K_0 \oplus opad) || H((K_0 \oplus ipad) || text))$$

- 通常認證金鑰長度爲128位元，或16位元組。若雜湊函數採用SHA-256，則B=64 (位元組)，所以我們將進行步驟3，附加48位元組0x00。

參考資料

1 A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press Series on Discrete Mathematics and Its Applications, 1996. http://www.cacr.math.uwaterloo.ca/hac/

2 NIST, “The Keyed-Hash Message Authentication Code (HMAC),” FIPS PUB 198, http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf, March 6, 2002.

3 NIST, “Secure Hash Standard,” FIPS PUB 180-2, http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

4 R. Rivest, “The MD5 Message-Digest Algorithm,” IETF RFC 1321, April 1992, http://www.ietf.org/rfc/rfc1321.txt.

5 X.Y. Wang and H.B. Yu, "How to Break MD5 and Other Hash Functions," *Advances in Cryptology–Eurocrypt’05*, pp.19-35, Springer-Verlag, May 2005.