附　　件

國防科技學術合作協調小組研究計畫成果報告

有安全機制之遠端連線(telnet)通訊協定之設計與製作
The Design of Authenticated TELNET Protocol

計畫編號：90-CS-7-009-003

執行期間：　90 年　 1 月　 1 日至　90 年　12 月　31 日

計畫主持人：葉義雄 副教授

共同主持人：林祝興 副教授

執行單位：交通大學資訊工程系

中華民國　　90 年　　12 月　　31 日

目錄

圖目錄

1. 簡介

近年來，網際網路（Internet）不僅在政府部門、國防單位、學術機構、工商業界造成一股使用熱潮，甚至連一般民間團體亦如雨後春筍般的趕搭網際網路的時髦列車，而網際網路無遠弗屆之特性及取之不盡、用之不竭的網路資源正是造成這股流行的主要原因。

網際網路的確也帶給人們無比的方便，就在人們為了能參與此虛擬世界之盛會而雀躍不已時，一直困擾每一位電腦使用者之電腦病毒程式也藉網際網路之便，悄悄進入電腦系統，輕則某些辛辛苦苦所建立之寶貴資料付之一炬，重則整個電腦系統遭到徹底破壞。

幾年前，美國國防部電腦中心曾經遭人透過網路侵入，康乃爾大學研究生亦曾透過網路將其設計之害蟲(WORM)程式散佈美國各地，雖然後來發現這些人都是出於惡作劇之心態，所造成的損失可能並不怎麼嚴重，但卻透露了一些非常嚴重的訊息；在我們所認為資源無限的網際網路中潛藏著無限的犯罪動機，而整個作業環境亦似乎暴露出太多的破綻，讓這些人有機可趁。試想，國際間為了政治的因素而透過網際網路竊取軍事機密、工商業界同行中為了競爭而透過網際網路竊取商業機密、電腦駭客(HACKER)為了達成其某種程度的成就感而竭盡所能的尋找破綻，以便入侵別人之電腦設備進行惡作劇，在加上一些專為破壞他人電腦系統為樂的電腦犯罪，網際網路的安全性(Internet Security)問題已經到了不得不加以正視的地步。

為了達到這個目的我們必須藉由密碼學、網路身份憑證與網路安全協定等相關的技術來抵抗諸如竊聽、擅改、假造和冒用身份的攻擊。我們所設計的概念是以協商的方式來溝通彼此通訊所使用的相關密

碼學方法與相關參數。另外我們加強網路訊息碼(Message Authentication Code)的機制以強化系統的安全性。並以此設計一Authenticated TELNET通訊協定並將此一協定運用於電子閘門系統。

2. 相關技術

2.1. 密碼學之研究

密碼學上可分為兩大類：對稱型加密系統與非對稱型加密系統，下圖為示意圖：

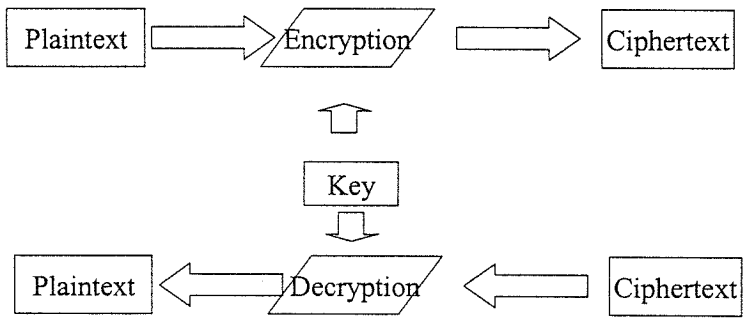

Figure 1. 對稱型密碼系統

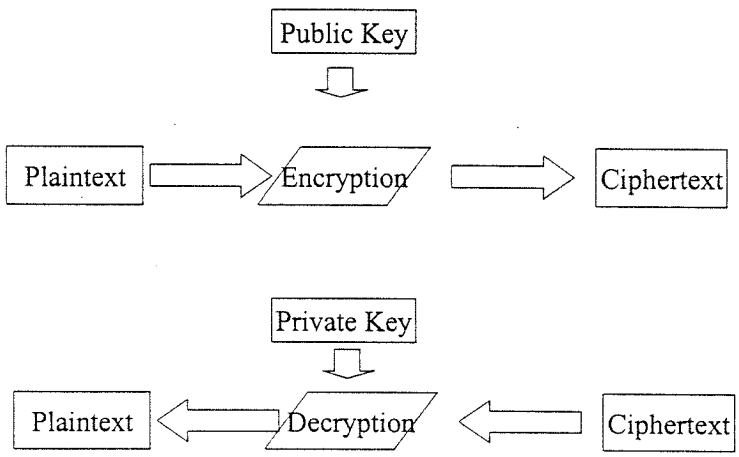

Figure 2. 非對稱型密碼系統

目前常被採用的對稱型加密系統有美國國家標準局所提出的DES（Data Encryption Standard）以及由Xuejia Lai 及 James Massey 所提出的 IDEA （International Data Encryption Algorithm）還有最近所提出的 AES (Advanced Encryption Standard)；非對稱型加密系統有由 Ron Rivest 、Adi Shamir 及Leonard Adleman 所共同提出RSA，而且廣為業界採用，另外一個非對稱型加密系統是由美國國家標準局所提出的 DSA（Digital Signature Algorithm）；對稱型加密速度快但是難於達到數位簽章的效果，而非對稱型加密系統雖可達到數位簽章的效能但是速度太慢，所以一般都是利用結合的方式來取得平橫點。如下圖所示：

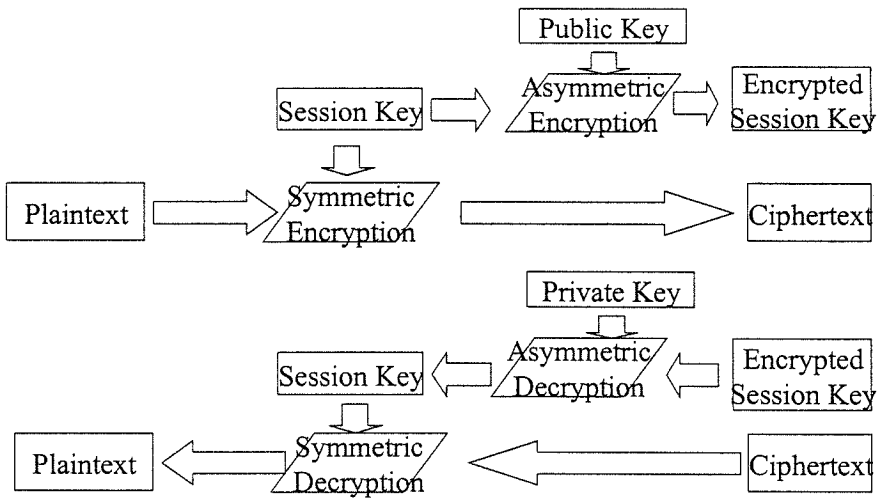

Figure 3.　對稱與非對稱型的結合密碼系統

2.1.1. 對稱型加密系統

2.1.1.1. DES (Data Encryption Standard)

Plaintext M(64 bits)

Ciphertext C(64 bits)

Key K(56 bits or 64 bits)

Round Function F(X, Y) : X(32 bits) and Y(48 bits)

There are 16 round in DES.

IP: initial permutation

FP: final permutation such that IP.FP = FP.IP = Identity

$IP(M) = (L_0, R_0) = (32 \text{ bits}, 32 \text{ bits})$

$(L_j, R_j) = (R_{j-1}, L_{j-1} \oplus F(R_{j-1}, K_j)), j = 1, \ldots, 15$

$(R_{16}, L_{16}) = (L_{15} \oplus F(R_{15}, K_{16}), R_{15})$

$F(X, Y) = P(S(E(X) \oplus Y))$

$C = FP(R_{16}, L_{16})$

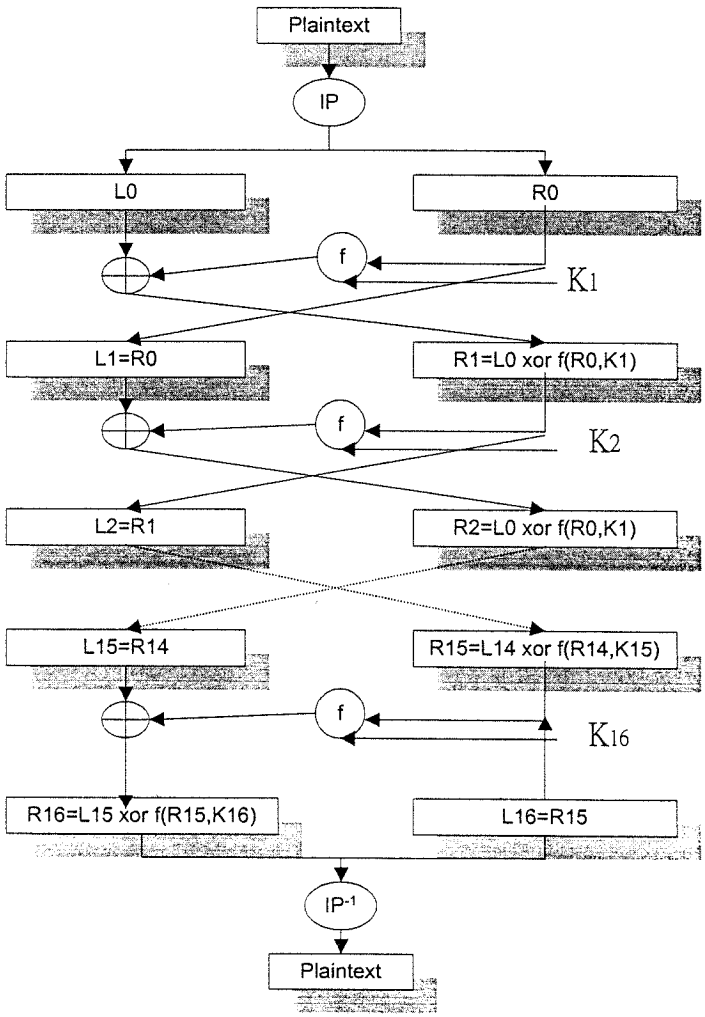

Figure 4. The Flow Chart of DES

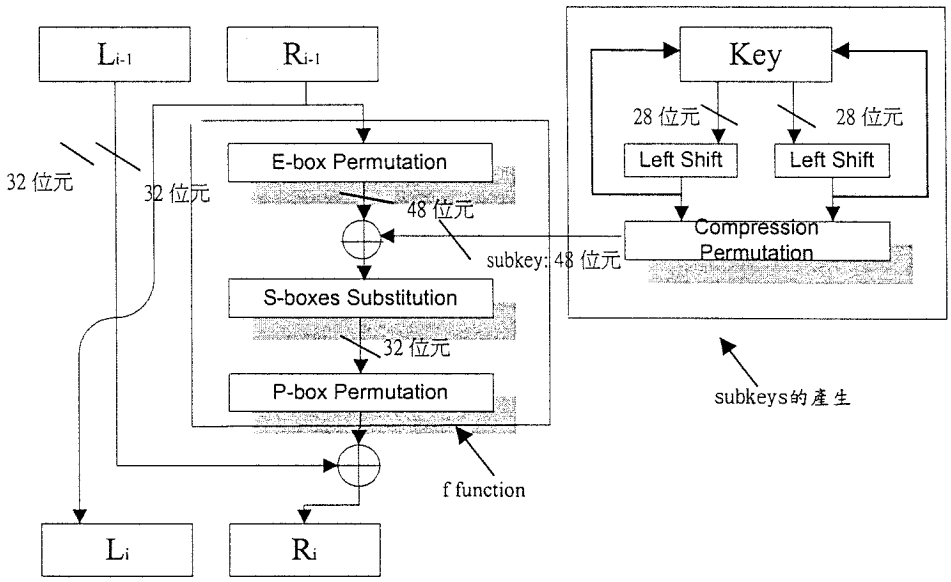

Figure 5. The Description of Round in detail

One round of DES

$\mathbf{LS_1 = 1}$

$\mathbf{LS_j}$ **= jth round Left rotation**

Key K(56 bits)

PC-1 (K) = $\mathbf{(A_0, B_0)}$ **=(28 bits, 28 bits)**

$\mathbf{A_j = LS_j (A_{j-1}), j = 1, \ldots, 16}$

$\mathbf{B_j = LS_j (B_{j-1}), j = 1, \ldots, 16}$

$\mathbf{K_j}$ **= PC-2** $\mathbf{(A_j B_j), j = 1, \ldots, 16}$

2.1.1.2. Triple DES (or DES^3)

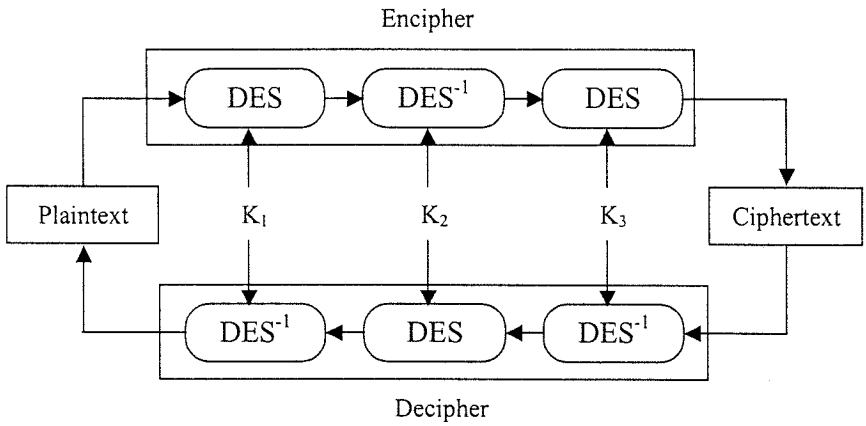

Figure 6. The Flow of Triple DES

2.1.1.3. Advanced Encryption Standard (AES)

2.1.1.3.1. AES的導論

加密演算法是資料安全的核心技術之一。通常使用FIPS(聯邦的資料處理標準)46 -- DES(資料加密標準) 在 1977由U.S政府所宣佈。DES是一個秘密-鑰匙區塊密碼。它的明文和密文都是64-位元，且秘密鑰匙的長度是 56個位元。 NIST(標準和技術的全國學會)每隔五年評估DES判斷DES是否仍然足夠安全被採用為標準。最新的評估於1993。NITS預期單一資料加密標準在1998會被逐步淘汰且在 2000會有一個新的標準會被提出。DES為什麼被淘汰的理由是由於最近20年之密碼方法和電腦硬體的發展。

因此 NIST 在 1997 中公開地提出AES[4]。大體上的目標是要在進入下個世紀之內發展一個能夠把敏感的政府資料保護得很好的具體定義之加密演算法的 FIPS。AES可發展成取代DES，然而 NIST預期Triple DES在未來中仍然是被眾所認知的演算法(為美國政府使用)。可被接受的最小需求和評估標準之草圖是：

1. AES 將是被公開的定義。

2. AES 將是一個對稱的區塊密碼。

3. AES 將被設計以使得鑰匙長度可依照需要來增加。

4. AES 將是可在硬體和軟體中被設計。

5. AES 將或是a) 免費可得的 或 b)在美國國家標準協會 (ANSI) 專利權政策一貫的術語之下可得的。

符合上述的需求之演算法將基於下列各項因素被判斷：

1. 安全（亦即，密碼分析所必須的成果），

2. 計算的效率，

3. 記憶體的需求，

4. 硬體和軟體之適合度，

5. 簡單,(即演算法可容易地被了解)

6. 靈活性（適應性），

7. 授權（許可）需求。

在2000 年 10月2日，NIST 宣佈它選擇 Rijndael 為 AES。

2.1.1.3.2. Rijndael 的設計詳述

每個 round 的轉換

前面 Nr-1 個 round 的轉換演算法

```
Round(State, RoundKey){
ByteSub(State);
ShiftRow(State);
MixColumn(State);
AddRoundKey(State, RoundKey);
```

```
}
```

最後一個 round 的轉換演算法

```
FinalRound(State,RoundKey){
ByteSub(State) ;
ShiftRow(State) ;
AddRoundKey(State,RoundKey);
}
```

***ByteSub* 的轉換**

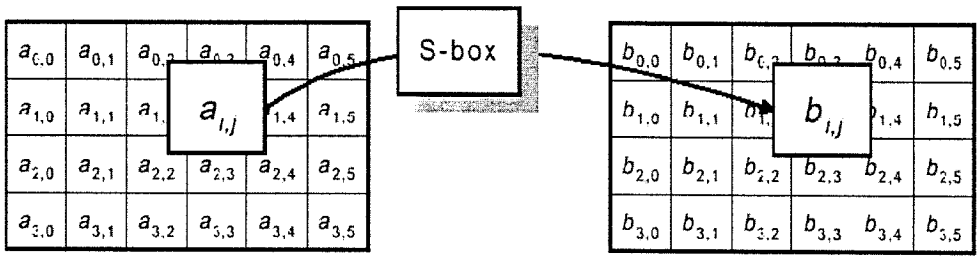

Figure 7. ByteSub Transformation of AES

ShiftRow 的轉換

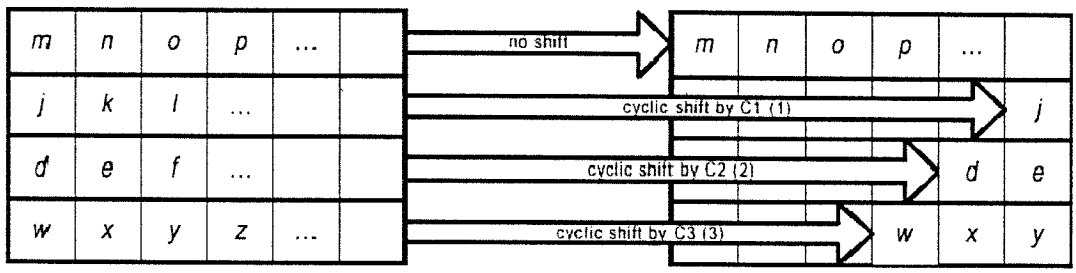

Figure 8. ShiftRow Transformation of AES

MixColumn 的轉換

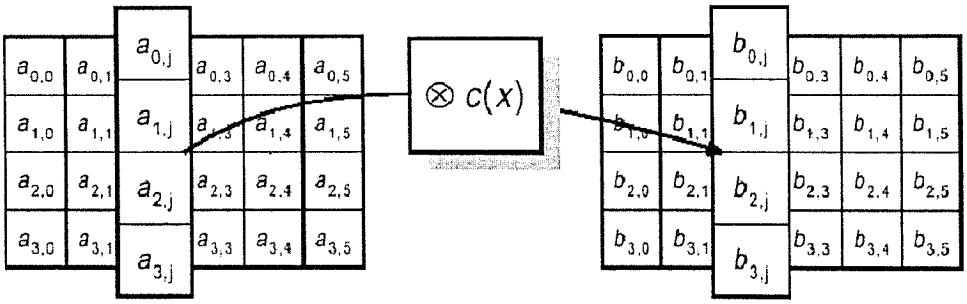

Figure 9. MixColumn Transformation of AES

AddRoundKey

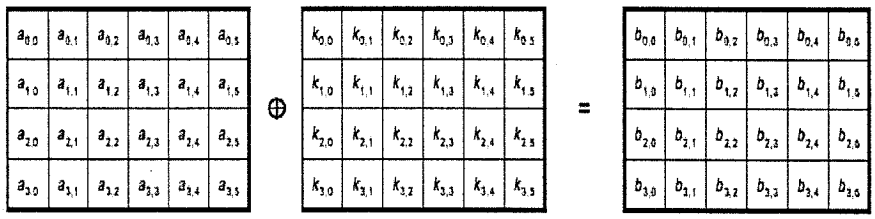

Figure 10. AddRoundKey Operation of AES

總結來說，每一個 round 都含有四個部份，ByteSub, ShiftRow, MixColumn , AddRoundKey, 如此我們可以得到一個大概的觀念如下圖

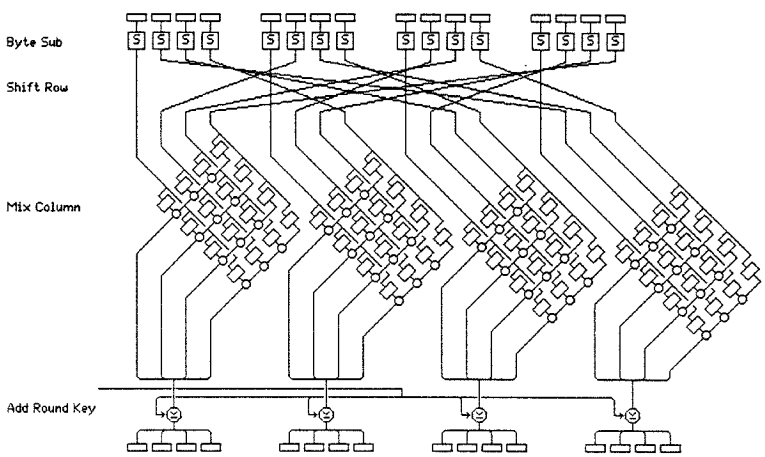

Figure 11. The data flow of AES

2.1.2. 非對稱型加密系統

2.1.2.1. RSA (R. L. Rivest, A. Shamir, L. M. Adleman)

(a) $n = p * q$ p 和 q 是兩個大質數.

隨機選取一個數 e 滿足 gcd(e, (p-1)(q-1)) = 1, (p-1)(q-1)= $\Phi(n)$, 取一個正整數 d , $d < n$而且 $e*d \equiv 1 \pmod{\Phi(n)}$.

(c) Private keys (私密金鑰): d (decryption key)

(d) Public keys (公開金鑰): e (encryption key), n.

(e) Encryption (加密): $E(y) = y^e \pmod n$.

(f) Decryption(解密) $D(y) = y^d \pmod n$.

2.1.2.2. Digital Signature Standard(DSS) 和Digital Signature Algorithm(DSA)

NIST, 在1991年提出 DSA.

底下是DSA的描述

(DSA 是Schnorr and ElGamal 兩種簽章法的變型)

p = a prime number, $512 \le |p| \le 1024$ and $64 \mid\mid p \mid$

q = a prime factor of p-1, $|q| = 160$

$g = h^{(p-1)/q} \bmod p$, where h is a random number < p-1 s.t. $h^{(p-1)/q} \bmod p > 1$

x = a random number < q

$y = g^x \bmod p$

public key : p, q, f, y

private key : x

signing : (To sign a message M).

k : a random number < q

signature (r, s) where

$r = (g^k \bmod p) \bmod q$

$s = k^{-1}(H(M) + xr) \bmod q$

verifying

$w = s^{-1} \bmod q$

$u_1=(H(M) *w) \bmod q$

$u_2=(rw) \bmod q$

$v=((g^{u1}*y^{u2}) \bmod p) \bmod q$

if v=r, then the signature is verified.

Proof:

$w=s^{-1} \bmod q=k(H(M)+xr)^{-1} \bmod q$

$u_1=(H(M)*w) \bmod q$

u2=(rw) mod q

v=((gu1*yu2) mod p) mod q

=((gu1*(gx)u2) mod p) mod q

=(gu1+x(u2) mod p) mod q

=(g(H(M)*w) mod q + (xrw) mod q mod p) mod q

=(g(H(M)+xr)*w mod q mod p) mod q

=(gk mod q mod p) mod q

=r (=(gk mod p) mod q)

(gk mod q mod p) mod q

=(gk+lq mod p) mod q

=(gkglq mod p) mod q

=(gk(h(p-1)/q)lq mod p) mod q

=(gk(hp-1)l mod p)mod q

=(gk mod p) mod q

2.1.3. 雜湊函數 (hash function)

2.1.3.1. 單向雜湊函數 (one-way hash function)

一個單向的雜湊函數H可以處理有任意的長度訊息M，而輸出一個有固定的長度的雜湊值h，即H= h。它有五個特性：

1. 輸出長度為常數：對任何的訊息M，h的長度是固定的。
2. 容易計算：對任何訊息M，可容易的計算h。
3. 很難倒推的運算：對任何的h，很難找到M使得H(M) = h。
4. 不充分的衝突：對任何的訊息 M_1，很難找到 M_2 使得 $H(M_1)=H(M_2)$。
5. 嚴謹地衝突：很難找到一對 M_1 和 M_2 使得 $H(M_1)=H(M_2)$。

當一個單向的雜湊函數滿足上述一到四的特性，它被歸類為"不充分的的"。且若一個單向的雜湊函數能滿足上述所有的五個特性，它被分類為"嚴謹的"。

我們可把單向雜湊函數看成有能力作壓縮的函數。要轉換任意長度的訊息到固定長度的雜湊值，圖2顯示出使用反覆的壓縮技術。一開始，填補位元在訊息M的尾巴。然後把訊息劃分成 n個子區塊。每個壓縮函數 F_i，只有2個輸入：第i個子區塊 M_i，和上一個壓縮函數的輸出 h_{i-1}。（即 $F_i(M_i,h_{i-1})= h_i$）。然後一個接一個地，壓縮函數被執行直到得到M的數位特點 h_n，我們把它稱為Message Digest hereafter。

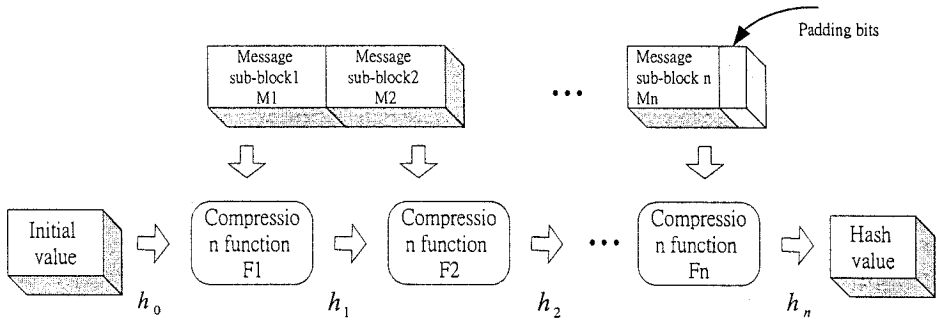

Figure 12. 反覆的壓縮函數之結構

2.1.3.2. SHA (Secure Hash Algorithm)

SHA也是專為32位元的機器所設計的演算法，由MD4演變而來，某些步驟也與MD5相似。它可以將任意長度的訊息，轉換為一個長度固定為160位元的Message Digest。SHA本來是應用在美國NIST所提出的數位簽章標準（DSA, Digital Signature Standard）中的一部份，如圖形1，也可單獨用在任何需要雜序值的環境中。輸入長度為m的任意訊息M（m≧0），執行步驟如下：

Append k padding bits：同MD5的步驟一。

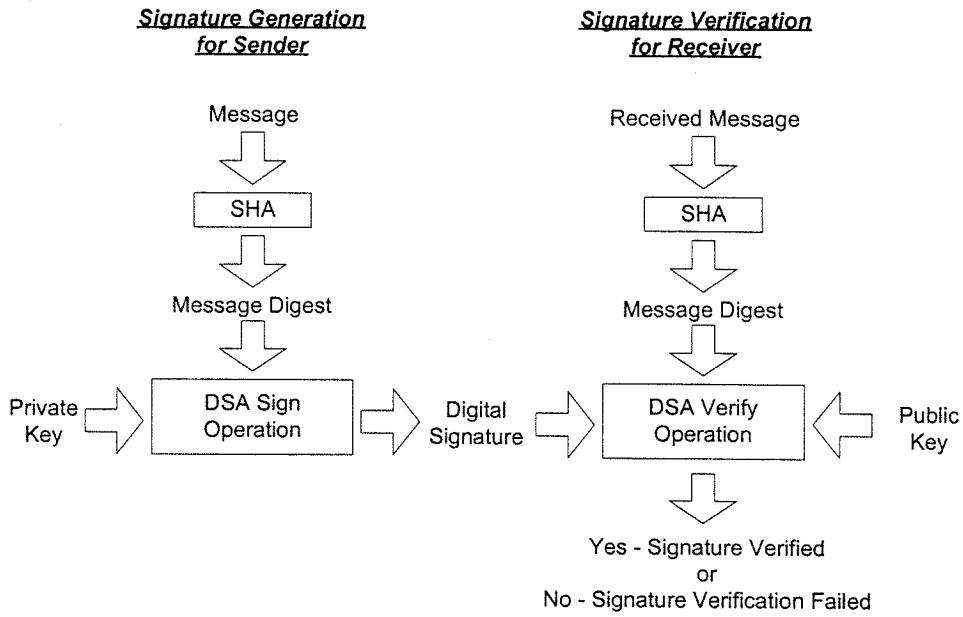

Figure 13. 圖形1 在DSA中採用SHA的作法

Append the length of M：同MD5的步驟二。

Definition and Initialization：

Definition：

1. Basic functions：

F(X,Y,Z) = (X AND Y) OR ((¬X) AND Z)

G(X,Y,Z) = X⊕Y⊕Z

H(X,Y,Z) = (X AND Y) OR (X AND Z) OR (Y AND Z)

I(X,Y,Z) = X⊕Y⊕Z

2. Round functions：

Round(a,b,c,d,k,s,i){a=b+((a +Z(b,c,d)+X[k]+Ti)<<s); }

Z = F， 0≦i≦19

Z = G，20≦i≦39

Z = H，40≦i≦59

Z = I，60≦i≦79

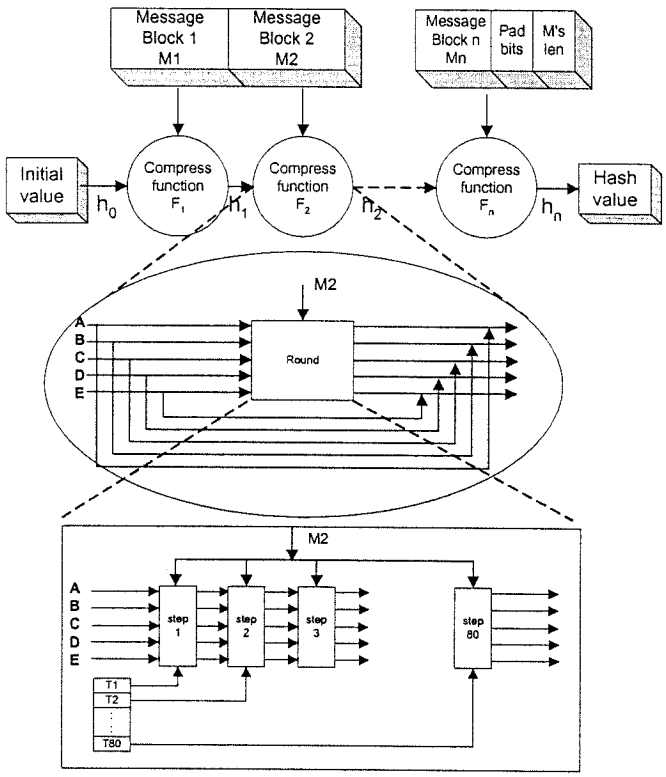

Figure 14. 圖形2 SHA流程架構

Initialization：

1. **Initialize Chaining Variables：五個連鎖變數A, B, C, D, E 初值：**

   $word_{32}$ A :67452301

   $word_{32}$ B :efcdab89

   $word_{32}$ C :98badcfe

   $word_{32}$ D :10325476

   $word_{32}$ E :c3d2e1f0

2. **Parameters：**

Ti = $2^{1/2}/4$ = 0x5a827999， 0≦i≦19

= $3^{1/}$2/4 = 0x6ed9eba1，20≦i≦39

= 51/2/4 = 0x8f1bbcdc，40≦i≦59

= $101^{/2}$/4= 0xca62c1d6，60≦i≦79

3. Duplication：

WordP=A, Q=B, R=C, S=D, V=E; //貯存初值

BigBlockX; // X[0]，…X[79]代表了自低到高位的Words

Algorithm:

For (int i=0; i<=n-1; i++) { //重複對每個子區塊處理，共80步驟

for (int j=0; j<=79; j++)

if (j<=15)X[j] = SubBlock i[j];

elseX[j]=(X[j-3]⊕X[j-8]⊕X[j-14]⊕X[j-16])

for (int j=0; j<=79; j++) {

int TEMP = (A << 5) + Round(a, b, c, d, k, s, i) + E + X[j] + Tj;

E=D; D=C;C=B<<30; B=A; A=TEMP;

}

} // i迴圈結尾

4. Output message digest：

把最後輸出的A, B, C, D, E五個32位元值與最初的設定的值相加（A = A + P; B = B + Q; C = C + R; D = D + S; E = E +V;），再將五個Word32值連接在一起，成為M的160位元Message Digest。

2.1.3.3. SHA-1

SHA-1是替舊版SHA在填充X[16..79]內容時，加上<<1的shift動作，也就是X[j]=(X[j-3]⊕X[j-8]⊕X[j-14]⊕X[j-16]) << 1。

2.1.4. Message Authentication Code (MAC)

2.1.4.1. MAC的簡介

在開放的計算和通訊之實際世界，在傳輸過程中對不可信任的媒介產生一個檢查資訊的完整性是很重要的。訊息認證碼（MAC）產生這樣的完整檢查機制。典型地，在兩個群組間共同分享著一個安全鑰匙且此兩個群組可使用此安全鑰匙藉著附加在MAC的資訊送給接收者完整的資料來認證彼此傳輸的資訊。MAC在網路訊息傳輸中被廣泛的使用來認證使用者的檔案。在此，我們不能處理傳輸訊息的秘密，即完整的訊息被直接的傳輸。

一般來講，有三種方法可被應用來架構MAC機制：Cipher-MAC，Hash-MAC和Hash-Cipher-MAC。

2.1.4.2. Cipher-MAC

Cipher-MAC在一些加密模式中以區塊密碼演算法加密訊息。以CBC模式加密的DES在FIPS公布成為一個MAC產生器使用Cipher-MAC標準的例子。

2.1.4.3. Hash-MAC

Hash-Mac使用單向雜湊函數和一個秘密鑰匙來壓縮一個訊息且產生組合的雜湊值作為訊息的MAC。舉例來說，如果Alice想送Bob

一個訊息M的MAC且他們共同使用秘密鑰匙k。Alice將接著M後，且計算連接後的雜湊值*H(K,M)*，*當作訊息M的MAC*。然後Alice送MAC和訊息*M*給Bob。因為Bob知道秘密鑰匙*k*，他可在產生MAC。而不知k的第三人Mallory則不能產生MAC。

一個Hash-MAC的例子為Bellare提出的HMAC，以 $H(k_1 \| H(k_2 \| M))$ 計算訊息X的MAC：*H()*是一個任意的雜湊函數， $k_i = k \oplus C_i$ 在此k為秘密鑰匙和兩個常數 C_i，而符號||表示concatenation.。相似於Cipher-MAC，產生一個有效的MAC需有一個確切的鑰匙。除此之外，有另一個Hash-MAC方案叫做NMAC，它的鑰匙是透過他們初始變數基本的雜湊函數，即NMAC(X)= $H_{k_1}(H_{k_2}(M))$ 。

2.1.4.4. Hash-Cipher-MAC

Hash-Cipher-MAC組合雜湊函數和密碼來建構MAC。令 $E_k()$ 為一使用秘密要匙k的對稱加密演算法。 $H_k()$ 定義使用鑰匙的雜湊函數。一個訊息的MAC可有以下四種形式：

1. $E_k(H(X))$
2. $E_{k1}(H_{k2}(X))$
3. $H(E_k(X))$
4. $H_{k2}(E_{k1}(X))$

2.2. SSL通訊協定

2.2.1. SSL（Secure Sockets Layer）網路安全協定

SSL通訊協定是一個在網路上提供通訊安全的通訊協定，這個通

訊協定允許Client/Server 的應用程式用一種設計過的方式通訊，以避免被偷聽、擅改、及偽造傳輸中的訊息。

SSL通訊協定最主要的目標就是在兩端通訊應用程式時提供私人性和可靠性，這個通訊協定由兩層組成，SSL Record通訊協定是在最低層，這層是用在處理各種較高層的通訊協定。一個這樣的通訊協定（稱SSL互握通訊協定）允許client 端和server 端互相認證而不必管其間的編碼方法和密碼學中的key，SSL的一個好處是它可以是對任一應用程式的通訊協定是獨立的，也就是說它可以架在不同的應用程式的通訊上，一個較高層的通訊協定可以架在SSL通訊協定的上端。

SSL通訊協定提供連結的安全性,並有以下三種特性：

1. 連結是私人的，編碼是使用在初始的互握，目的在定義一個secret key，對稱型的密碼學方法做為資料的編碼，如DES、RC4等等。
2. 端點可以用非對稱型、public key或密碼學來認證，如RSA或DSS等等。
3. 連結是可信賴的。訊息都是完全用一MAC key 包裝後傳送，安全的雜湊函數可用來做為MAC的運算。

SSL 通訊協定v3.0則有以下的四個目標：

1. 密碼學提供的安全性：SSL應該被用來建立兩端的安全連結。
2. 繼承性：不一樣的程式設計師可以用SSL v 3.0來發展應用程式，並且可以成功地交換密碼參數而不用其他程式碼的知識援助。
3. 延伸性：SSL尋求一種方法去提供一個架構進入新的public key，並且使得加密方法在需要時加以組合。這將會完成兩

個次要的目標：避免建立一個新的通訊協定和不用實做出一整個新的安全資料庫。

4. 相關的有效性：密碼方法的運作趨向於較快速的CPU要求，特別是public key 的運算。基於這個原因，SSL通訊協定引進一些方案來減少重新建立連結時所需的連結次數。

SSL屬於層級的通訊協定，在每一階層，訊息中包含有每小片段的長度、描述和內容。SSL帶著訊息來傳遞，將資料分割為可處理的小包裝，並且壓縮資料，做成一MAC，最後編碼後再將結果送出。收到傳送過來的資料後，要將之解密、確認正確性、解壓縮，然後再傳送到更高階層的client端。

一個SSL的段落是非常安全的。SSL互握通訊協定的責任是來協調client 端和server端的狀態，允許每個通訊協定的狀態可以一致，而不管這狀態是否平行的。就邏輯上來講，這個狀態會被表現兩次，一次當作現在正在運作的狀態，第二次可能當作即將來臨的狀態，並且要保持讀和寫是分開進行的。當一個client 端或server端接收到一個「Change cipher spec」的訊息時，它就會將「即將來臨讀出的狀態」複製到「正在讀出的狀態」；當client 端或server端送出一個「Change cipher spec」訊息時，它會將「即將來臨寫入狀態」複製到「現在寫入的狀態」。當互握的通訊協定完成後，client 端和server 端將會交換「Change cipher spec」的訊息,然後用一個新且彼此同意的方法來通訊。

一個SSL session可以包含幾個安全的連結，互通的兩端可能有多個同步的session。每個session 的狀態有包含下列元素：

- session的認證者：

  server 端可以挑選一個隨意的位元組序列來認證一個可動

作的或是重開的session 狀態。

- 端點認證：

  X509.v3[X509]可以認證端點，這個狀態的元素有可能是空的（null）。

- 壓縮方法：

  用哪一種演算法來壓縮資料，特別是編碼過的資料。

- cipher spec

  分辨出資料的編碼演算法(如DES，null等等)或是MAC的演算法（如MD5或SHA）。它並且定義密碼方面的屬性，如hash_size。

- Master secret

  client 端和server端來分享48個位元組。

- 是否可再使用

  用一個旗幟來指示一個session是否可以用來重新啟動一個連結。

而這些連結的狀態包含下列元素：

- client 端和server端的隨取亂數
- client 端和server端可以選定位元組的序列來做為連結使用。
- server write MAC secret

  server 端用來做為資料寫入的secret 是用MAC 運算得來。

- client write MAC secret

  client 端用來做為資料寫入的secret 是用MAC運算得來。

- server write key

server 端用來做為資料加密的key，亦是client用來解密的key。

●client write key

client 端用來為資料加密的key，亦是server端用來解密的key。

●初始的向量

當一個用在CBC的block cipher被用到，一個初始向量（IV）會被每一個key保存。這個片段會首先被SSL通訊協定來重新啟動，最後每個段落（record）的密文段（block）會被保留，以讓下一個接著來的段落使用。

●序列號碼

每個通訊的伙伴會保存各別的序列號碼來從每個連結中接收或送出資訊。當一方送出或接收到一個「Change cipher spec」訊息，正確的序列號碼會被設為零，序列號碼是一種64位元組並且不超過二的六十四次方減一的號碼。[7,8]

2.2.2. SSL 互握通訊協定總覽

SSL 互握通訊協定是在通訊的兩端正式傳送訊息之前所做的協調，以便讓通訊期間，使用者的保密與資料的傳輸有依據及保障。

session 狀態的密碼參數是SSL互握通訊協定所製造，而SSL互握通訊協定是在SSL Record的上層運作。當一個SSL client 端和 server 端第一次通訊，他們彼此要先同意一個通訊協定的版本，還要選擇密碼的演算法、互相認證和利用public-key的編碼技巧來產生共同分享

的機密。這個過程是在互握通訊協定中運作的，下面是簡要的介紹：

client 端送出一個「Client hello」的訊息，接收到的server 端會回應一個「Server hello」的訊息，否則就會發生一個錯誤，這個連結就失敗了。這個「Client hello」和「Server hello」訊息是用來建立client和server 間安全的限度，「Client hello」和「Server hello」訊息建立下面幾種屬性：通訊協定的版本、session ID、所使用的密碼演算法和壓縮的方法，並且，兩個隨取亂數是用來產生和交換key的。

Hello訊息的送出後，如果認證通過的話server 端會接著送出認證要求。並且，如果需要的話，「Server key exchange」訊息會接著送出。假如server 端已經認證成功，它將會要求client 端送出一個認證訊息。不過前提是已經選擇了適當的密碼方法。

現在server 端回送出一個「Server hello done」的訊息來指示互握協定中的Hello訊息階段已經圓滿完成了。再來server 端就會等待client 端的回應。假如server端已經送出一個「Certificate request」的訊號，server 端會回應一個「Certificate」訊號或是一個「No certificate」警告。「Client key exchange」訊號在這個時候送出，這個訊號的內容會根據「Client hello」和「Server hello」訊息所選擇的public-key演算法而改變。假如server端已經發出一個帶有簽章能力的訊號，一個數位簽章的「Certificate verify」訊息會被送出以明白地檢定此認證是否成功。

在此時，client端送出一個「change cipher spec」的訊息，並且將下一個Cipher Spec複製到現在的Cipher Spec 中。然後client端在新的演算法、key和秘密下送出「Finished」訊息。在回應方面，server 端會送出它自己的「Change cipher spec」訊息，轉換下一個編碼方法成為現在的編碼方法，並在新的密碼方法下送出自己的「Finished」訊

息。如此，互握過程已經結束，client 端和server 端即可以開始交換應用程式的資料。請看下面圖表。

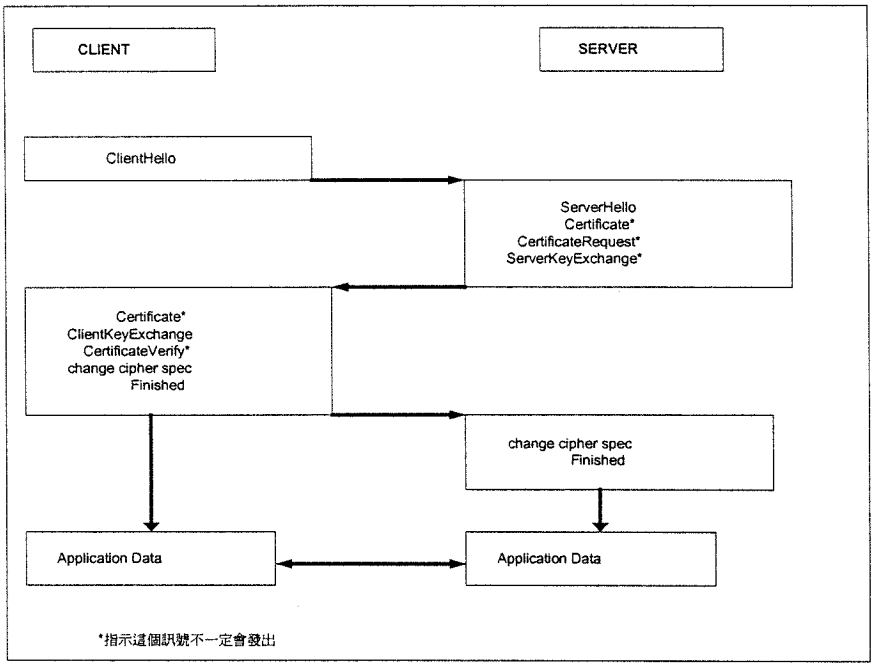

Figure 15. SSL 互握流程圖

當client 端和server 端決定重開一個先前的session 或再複製一個已存在的session，訊息傳遞的過程如下：

server 端用session ID來送出一個「Client Hello」訊息，來重新啟動。然後server 端會檢查此session ID是否相符，如果相符的話，server 端將會在這個session 的狀態下去重新建立一個連結，並會用相同session ID的值送出一個「Server Hello」訊息。在此時，client 端和server 端都必須送出一個「Change cipher spec」訊息，然後接著送出「Finished」訊息。一旦重新建立連結的過程已經完成，client 端和server 端就可以開始交換應用程式的資料，假如session ID沒有相符的，表示這是一個新的session，server 端就會產生一個新的session ID，並且這個SSL的client 端和server

端就要進行一個完整的互握通訊協定過程。重建連結的互握過程如下圖所示：

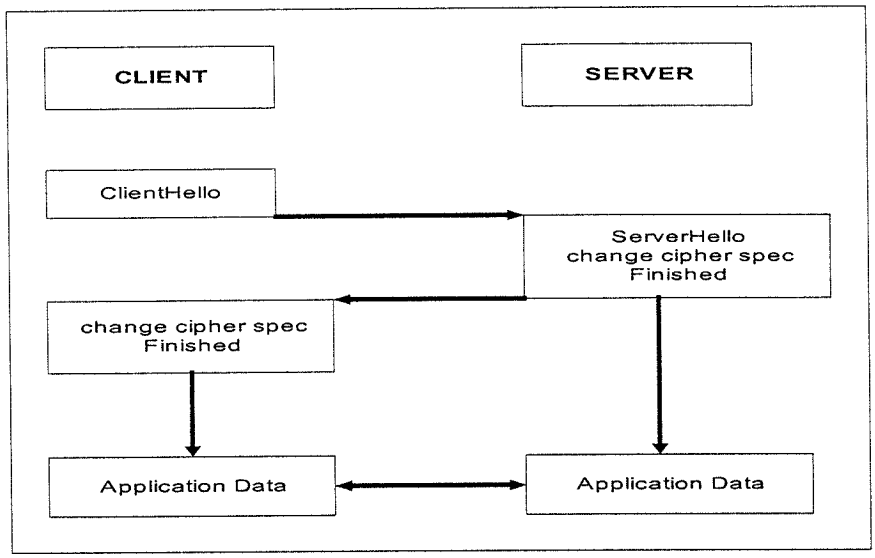

Figure 16. SSL 重新連接流程圖

SSL互握通訊協定所需交換之訊息

在上一節我們提到過互握通訊協定的過程，現在我們來深入探討這互握過程中所交換的訊息：

(1)Client hello 訊息：

- client_version
- random (clienthello random)
- session_id
- cipher_suites
- compression_methods

(2)Server hello 訊息：

- server_version
- random (serverhello.random)

- session_id
- cipher_suite
- compression_method

(3)**Server certificate 訊息：**

- certificate_list

(4)**Certificate Request 訊息：**

- certificate_types
- certificate_authorities

(5)**ServerKeyExchange 訊息：**

- params
- signed_params
- md5_hash
- sha_hash

(6)**Client Certificate 訊息：**

- certificate_list

(7)**ClientKeyExchange 訊息：**

- exchange_keys

(8)**CertificateVerify 訊息：**

- signature

(9)**Finished 訊息：**

- md5_hash
- sha_hash

這個SSL互握的通訊協定是已經定義好的較高階層SSL Record通訊協定其中之一，這個通訊協定用來溝通每個session 的安全屬性。互握訊息是用來支援SSL Record階層。互握通訊協定要送出的訊息，

依序為「Hello_requests」、「Client_hello」、「Server_hello」、「Certificate」、「Server_key_exchange」、「Certificate_request」、「Server_hello_done」、「Certificate_verify」、「Client_key_exchange」、「Finished」。而這些互握通訊協定所傳送的訊息必須依照一定的訓序，用不同的順序傳送會導致不可預知的錯誤。

下面我們就簡略看一下這些訊息的意義：

(1) **Hello 訊息**

Hello 訊息是用來交換client 端和server 端的安全能力，當一個新的session 開始，編碼方法、壓縮方法、和演算法都設為零，現在的CipherSpec演算方法是用來重新審查訊息。Hello 訊息有分client 端的Hello訊息和server 端的Hello訊息，我們分別來看看：

Client 端的Hello訊息包含了以下幾種物件：

●client_version

client 端想要用哪一版本的通訊協定來通訊。這將是client端所能支援的最新版本通訊協定，現在使用3.0這個最新的版本。

●Random

一個由client 端產生出來的亂數資料結構。

●Session_id

Client 端想要使用那個session ID來做為通訊的連結，如果沒有session_id找的到或想重新產生出新的安全參數的話，這欄位會被設為空的。

●Cipher_suites

這是client 端所能支援的密碼方法的列表，

而這些的排列方式取決於client 端的偏好，假如session_id這個欄位不是空的話(表示提出重新建立連結的要求)，這個向量就至少須包含原來session的密碼方法。

●Compression_methods

這是client 端所能提供的壓縮方法，同樣也是經過排序的。假如session_id這一欄位不是空的，這個向量一定要包含原來session 的壓縮方法。

Server 端的Hello訊息包含了以下幾種物件：

●server_version

這個欄位將會存有client 端在「client hello」訊息中所支援的最低和server 端所能提供的最高版本。現在通常使用3.0的版本。

●Random

這個亂數資料結構是server 端所產生的，並且一定要不同於client 端所產生出來亂數的值。

●Session_id

這是這個連結session 的確認。假如ClientHello.session_id不是空的，server 端將會從這個session 的session ID列表中挑選出一個相符的。假如找到相符的，server 端將會重新建立一新的連結，並且使用這個詳細的session 狀態，server 端還會回應一個跟client 端所提供相同的值。這指示了一個重新使用的session還有控制連結的兩端將會持續通訊直到「finished」訊息

為止。相反的，這個欄位會存放一個不一樣的值來指示這是個新連結。Server 端會回應一個空的session_id來指示這個小節不能被重新使用。

●Cipher_suite

Server 端會從ClientHello.cipher_suites中挑選一個密碼方法來使用，當要重建一個session時，這個欄位會保存被置換session 的狀態值。

●Compression_method

Server 端會從ClientHello.compression_method中挑選一個壓縮方法來使用，當要重建一個session時，這個欄位會保存被置換session 的狀態值。

(2) **Server certificate訊息**

假如一個server 端需要認證，此server 端就會在送出「Server hello」訊息後立刻送出它的「Certificate」訊息，這個「Certificate」訊息形式必須是選定的密碼方法可接受的。同樣的形式用在當client 端要回應一個「Certificate request」訊息時。

這個訊息包含了以下的物件：

●certificate_list

這是一個X.509.v3認證中的一個序列，它是以發出認證身份證的順序來排列，最末端會是一個認證中心（CA）。

(3) **Server key exchange訊息**

假如沒有認證的話，或是只有簽章認證（如DSS 認證），及用Fortezza/DMS 來做為交換金鑰的演算法，server 端就會送出「Server key exchange」的訊息。

這個訊息包含了以下的物件：

●Params

Server 端金鑰交換的參數。

●Signed_params

一個根據相關params的hash值。並且是合乎這個hash 使用的簽章。

●Md5_hash

MD5 (ClientHello.random + ServerHello.random + ServerParams) 。

●Sha_hash

SHA (ClientHello.random + ServerHello.random + ServerParams)。

(4) Certificate request訊息

如果適合選定的密碼演算法，一個非不知名的server 端可以要求client 端送出一個認證。

這個訊息包含了以下的物件：

●certificate_types

這個欄位是存一認證所需的型態集的序列，它是依照server 端的出現順序排列的。

●Certificate_authorities

這是認證中心（CA）可接受的名字序列。

(5) Server hello done訊息

Server 端會送出的個「Server hello done」訊息來指示「Server hello」和相關訊息的結束，在送出此訊息後，server 端就會等待client端的回應。接受到「Server hello done」訊息後，client 端需要確認server 端的證明是否有效，並且確認「Server hello」訊息是可接受的。

(6) **Client certificate訊息**

這是在接收到server 端送來的「Server hello done」訊息後，client端最先送出的訊息，這訊息只有在server端要求一個認證時才會送出，假如沒有適合的證明，client端應該送出一個「No certificate」的訊息來警告。這個錯誤只是個警告，假如必須要client 端的認證時，server 端可以回應一個「Fatal handshake failure」警告。

(7) **Client key exchange訊息**

這個訊息的選擇依據已經選好的public-key演算法來決定，這些資訊是存放在下一個session 的狀態中，由它可以來選擇適合的record資料結構。

(8) **Certificate verify 訊息**

這個訊息是用來鑑定client certificate的正確性，如果這client certificate 有簽章的功能，此訊息就會發出以表確認。

(9) **Finished 訊息**

這個訊息通常會跟隨著「Change cipher specs」之後送出，它可以用來確認key exchange及認證的過程是否成功，而這個訊息是第一個由所溝通過的演算法、key和秘密所包裝的封包，在送完此訊息後，通訊的兩端就會開始交換實

際的資料，接收到此訊息則必須確認內容的正確性。

SSL key 之傳送與資料包裝

我們用RSA keyExchange 做例子，觀察SSL通訊協定中，key 之傳送以及資料之包裝。

●ClientHello與ServerHello

現在回想SSL互握通訊協定，client 端會送出一個「ClientHello」訊息來初始互握通訊協定，而server 端也會回送一「ServerHello」訊息，server 端及client 端都會產生一亂數。

```
struct{
    Uint32 gmt_unix_time;
    Opaque random_byte[28];
}Random;
```

gmt_unix_time - 送出此結構的日期及時間，用標準UNIX 32位元格式表示

Random_bytes - 安全亂數產生器產生的28位元組亂數

```
struct{
    Random random;
    ...
}ClientHello;

struct{
    Random random;
    ...
}ServerHello;
```

●Client key exchange

若是用RSA public key來做為KeyExchange 的演算法，Client 端就會產生一亂數(PreMasterSecret)，加密後送出。

```
struct{
    ProtocolVersion client_version;
    Opaque random[46];
}PreMasterSecret;
```

client_version - client 端所支援的最新版本

random - 46位元組的變數

```
struce{
    public_key_encrypted          PreMasterSecret
    pre_master_secret;
}EncryptedPreMasterSecret;
```

●MasterSecret 之計算

client 端將pre_master_secret用server 端的public key編碼後送給server 端，server 端收到後用它的private key 解碼，現在兩端都知道pre_master_secert，可以開始master_secert 的計算。

```
Master_secert=
    Md5(pre_master_secret
        +SHA( 'A' +pre_master_secret
            +ClientHello.random+ServerHello.random)
```

```
)+
Md5(pre_master_secret
    +SHA( 'BB' +pre_master_secret
        +ClientHello.random+ServerHello.random)
)+
Md5(pre_master_secret
    +SHA( 'CCC' +pre_master_secret
        +ClientHello.random+ServerHello.random)
); //48位元組
```

//上式中的"+"表示串連的意思

●MAC 的產生

有了master_secret 後，我們就可以用它來產生所須的key_block，方法如下：

```
key_block=
    MD5(master_secret+SHA( 'A' +master_secret
        +ServerHello.random+ClientHello.random))
    +MD5(master_secret+SHA( 'BB' +master_secret
        +ServerHello.random+ClientHello.random))
    +MD5(master_secret+SHA( 'CCC' +master_secret
        +ServerHello.random+ClientHello.random))
        +[……];
```

//直到夠用為止

Key_block - 會被依續分為下面幾個部份，值得注意的是，

key_block 會產生夠用的位元組以滿足下面分割的需求：

client_MAC_write_secret[CipherSpec.hash_size]
//MD5 需16位元組

server_MAC_write_secret[CipherSpec.hash_size]
//MD5 需16位元組

client_write_key[CipherSpec.key_material] //DES 需8位元組

server_write_key[CipherSpec.key_material] //DES 需8位元組

client_write_IV[CipherSpec.IV_size] //DES 需 8 位元組

server_write_IV[CipherSpec.IV_size] //DES 需 8 位元祖

有了上述secret 後，client 端及server 端就可以知道如何包裝MAC：

MAC= hash(MAC_write_secret + pad_2
+hash(MAC_write_secret + pad_1 + seq_num
+ legnth + content));

pad_1 - 若使用MD5，其為十六進位之’36’ 重覆48次。

pad_2 - 若使用MD5，其為十六進位之’5c’ 重覆48次。

seq_num - 這段訊息的續列號碼。

length - 經壓縮內容的長度（以位元組為單位）。

●製作實際傳輸於網路上的資料

有了MAC後，如果是用stream-cipher 來傳送，其內容如下：

```
stream-cipher struct{
    opaque content[SSLCompressed.length];
    opaque MAC[CipherSpec.hash_size];
} GenericStreamCipher;
```

依照上列順序，一段由SSL加密的文件於是完成。

```
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    select (CipherSpec.cipher_type) {
        case stream: GenericStreamCipher;
        case block: GenericBlockCipher;
    } fragment;
} SSLCiphertext;
```

type - 依照SSLCompressed.type而定。

versoin - 依照SSLCompressed.version而定。

length - 依照SSLCiphertext.fragment而定。

SSL安全性討論

在這一節我們用RSA public key 的編碼方法來探討SSL通訊協定，看它如何確保認證及資料傳輸的安全。在RSA public key的密碼方法中，用public key編碼的資料可以用 private key來解開，而用private key編碼的資料也只能用public key 來解碼，下面我們就來看看SSL的

運作：

●用public key 的密碼方法來認證

在這我們引用兩人Cary和Diana，[something]key 表示用key 來加解密something。假如Cary 想要認證Diana，Diana 手上有public key 及private key，她將public key 送給Cary（詳細方法描述於後），然後Cary 發出一個訊息給Diana：

C->D random-message

Diana 收到後，用自己的private key將此訊息加密再送給Cary。

D->C [random-message] Diana' s-private-key

Cary 收到後，用Diana 之前送給她的public key 將訊息解開，再跟之前送給Diana 的訊息比較，如果相同的話，就可以知道她是與Diana通訊，由於他人無法知道Diana 的private key，所以無法用它來加密Cary 給的訊息，然後回送給Cary。

●數位簽章

其實就上面所說，如果有人想要假冒Diana，他可以截取Diana 送給Cary的訊息，往後就發出此訊息來假扮Diana。所以Diana最好是計算出一摘要訊息，編碼後再送給Cary，因為此摘要訊息不容易被還原成原訊息，所以就算截取到摘要訊息，假冒者也無法得到原訊息的內容，所以用這技巧，Diana就能有效保護自己，而這也是數位簽章的意思。

不過Diana 在Cary 送來的訊息中簽名，好像沒有比原來直接加密Cary的訊息安全，因為假扮者仍可以只拿此訊息來跟Cary 通訊，所以Diana 需要自己發出一段訊息，簽名加密再送出才能保證安全，所以我們再來看認證的過程：

C->DHi, are you Diana?

D->CCary, this is Diana

[digest(Cary, this is Diana)]Diana' s-private-key

在上面通訊協定中，Diana 首先送出「Cary, this is Diana」訊息，緊接著再送出加密過且已簽名的訊息，Cary收到後可以輕易地辨別是否與Diana 通訊，而Diana亦只在這個訊息上簽名，並不會使得假冒者有機可趁。

●送出public key 的方法

Diana 如何可靠地送出自己的public key 呢?我們再來看看下面的通訊過程：

C->DHi

D->CHi, I am Diana, Diana' s-public-key

C->Dprove it

D->CHi, this is Diana

[digest(Hi, this is Diana)]Diana' s-private-key

仔細看不難發現，在上面過程中，每人都可以假扮自己是Diana，只要你有public key 和private key，你可以向Cary說你是Diana，並將自己的public key 送出，然後依照通訊協定進行，Cary 並不會發現你是假冒的。

為了解決這問題，SSL引進了認證（CA Certificate Authority）的觀念，一個認證證明（Certificate）有下面幾個成分：

■發出此認證證明的單位名稱

■此認證證明是為誰發出

■發給對象的public key

■時間效力

這個認證證明是用發行者的private key來加密，而每個人都曉得此發行者的public key，所以運用此認證證明，就可以確認誰是真正的Diana，如果Diana好好保存自己的private key，通訊的過程就能保證安全了：

C->DHi

D->CHi, I am Diana, Diana' s-certificate

C->Dprove it

D->CHi, this is Diana

[digest(Hi, this is Diana)]Diana' s-private-key

現在Cary 收到訊息後，可以先檢查認證證明，確認簽章及認證對象是否為Diana，確認後，她便相信這個public key 是Diana所擁有，然後要求Diana 表明身份，Diana 就會如上所述，做一訊息摘要，加密送給Cary，Cary再利用Diana的public key 解開來對照Diana的原訊息，如此就可以讓Cary了解對方確實為Diana。

●交換彼此的秘密

一旦Cary 已經確認了Diana，她就可以送出一段訊息，而這訊息只有Diana 可以解開：

C->D[secret]Diana' s-public-key

如果要知道Cary 送什麼給Diana，除非有Diana 的private key，才能將此訊息解碼，所以就算有人截取到傳送中的訊息，亦不能知道Cary 送了什麼給Diana，所以交換彼此的秘密也是public key 密碼方法的強大功能，如果Cary 是傳送另一把key 給Diana，那這把key就可以當做對稱型加密方法（如DES、RC4、IDEA）的key了。

因為Cary 產生這個秘密送給Diana，所以她知道此秘密，而Diana擁有private key 可以將收到的訊息解密，亦可知道這個秘密。兩端

都知道這個秘密後，就可以開始對稱型密碼演算法，並且用這方法來做為資料傳輸，下面就是修正過後的通訊協定：

C->DHi

D->CHi, I am Diana, Diana' s-certificate

C->Dprove it

D->CHi, this is Diana

[digest(Hi, this is Diana)]Diana' s-private-key

C->Dok, Diana, here is a secret

[secret]Diana' s-public-key

D->C[some message]secret-key

至於secret-key怎麼定義，那就看通訊協定如何定義它，亦可以將原來的秘密直接當做secret-key。

●MAC的使用

雖然假冒者已經無法竊取通訊中的資訊，不過他卻可以破壞他人的通訊。例如他坐在Cary 和Diana 中間當做傳聲筒，他可以傳遞真實的訊息給對方，但是保留某些特定的部份，製造通訊的障礙。假設此人為Tom，我們看下面例子：

C->THi

T->DHi

D->THi, I am Diana, Diana' s-certificate

T->CHi, I am Diana, Diana' s-certificate

C->Tprove it

T->Dprove it

D->THi, this is Diana

[digest(Hi, this is Diana)]Diana' s-private-key

T->CHi, this is Diana

[digest(Hi, this is Diana)]Diana' s-private-key

C->Tok, Diana, here is a secret

[secret]Diana' s-public-key

T->Dok, Diana, here is a secret

[secret]Diana' s-public-key

D->T[some message]secret-key

T->CGarble([some message]secret-key)

Tom 傳遞先前的訊息，直到通訊的兩端開始傳遞秘密再從中破壞，因為此時Cary 已經信任Diana，所以她相信收到的訊息，而不知道已遭竄改，雖然Tom 不知道實際傳遞的訊息，也無法送出任意訊息給Cary，但是依然可以破壞她們間的通訊。

為了避免這樣的情形發生，Cary 和Diana 必須引進一個訊息認證碼（MAC Message Authentication Code），MAC 是一串資料，它由傳輸的資料及秘密所計算出來，如下：

MAC:= Digest(some message, secret)

因為Tom 不知道傳輸間的secret，所以他無法計算出正確的Digest值，甚至Tom 任意的竄改訊息，成功的機會也是很低的，因為Digest 的資料非常龐大。用MD5 來做個例子，Cary 和Diana 傳送資料時包含一個128位元的MAC值，Tom 猜中正確MAC的機會是微乎其微

。再看下面的例子：

C->DHi

D->CHi, I am Diana, Diana' s-certificate

C->Dprove it

D->CHi, this is Diana

[digest(Hi, this is Diana)]Diana' s-private-key

C->Dok, Diana, here is a secret

[secret]Diana' s-public-key

[some message, MAC]secret-key

Tom現在就麻煩了，雖然他依然可以竄改傳送中的訊息，不過MAC的計算就使他現形了，Cary 或是Diana 發現不正確的MAC 值時，就可以馬上中斷通訊。

2.3. 身份驗證與存取控制

2.3.1. 身份驗證

SSL轉接系統使用到身份驗證服務的功能模組為接收模組與控制模組，接收模組的身份驗證服務是用來確認使用者身份，而控制模組的身份驗證服務則是用來確認初始者、管理者與操作者的身份。至於身份驗證服務種類，目前有使用在SSL轉接系統的有Smart Card與憑證驗證服務兩種方式。憑證驗證服務在X.509裡共定義三種驗證程序，分述如下。

2.3.1.1. 單向驗證

在單向驗證程序中，A會單向將資料送給B驗證，其中資料包含著T_a，A的timestamp、R_a，A所產生的nonce、I_b，表示這個資訊是要傳送給B、M_a，A要傳送給B的秘密資訊，為了保密緣故，這個秘密資訊會用B的公開金鑰作加密、及sgnData，將傳送的訊息利用A的私有金鑰所作的簽章。其中T_a裡面記載著產生timestamp的時間與過期的時間，R_a是A隨機產生的數字，而且這個數字在R_a的有效期間內都是唯一的。接收端B會先驗證A憑證的真實性與sgnData的正確性，皆下來是藉由I_b確認資料是送給自己的，檢查T_a的時間，確認是在T_a合法時間內，檢查R_a的值，防止重送攻擊，並且用自己私有金鑰來解開M_a，獲得A所要傳送給B的資訊。

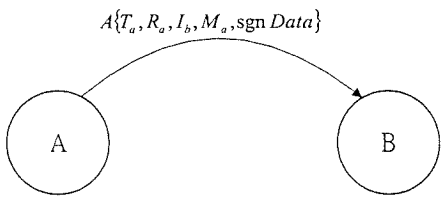

Figure 17.　X.509單向驗證程序

2.3.1.2. 雙向驗證

雙向驗證除了單向驗證由A傳送給B的資訊，還多了B回覆給A的資訊，其中包含了T_b，B的timestamp、R_b，B所產生的nonce、R_a，A之前所傳送的nonce 、I_a，表示這個資訊是要傳送給A、M_b，B要傳送給A的秘密資訊，為了保密緣故，這個秘密資訊會用A的公開金鑰作加密、及sgnData，將傳送的所有訊息利用B的私有金鑰所作的簽章，A收到B的驗證步驟完全跟單向驗證B所作的一樣，只不過多了確認送回來的R_a，是否是當初A所送出的R_a。雙向驗證是為了讓雙方可以

互相驗證彼此的身份，並且彼此交換要給對方的秘密資訊。

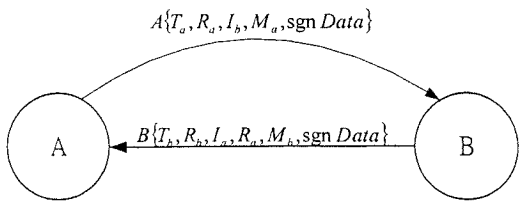

Figure 18. X.509雙向驗證程序

2.3.1.3. 三向驗證

三向驗證除了雙向驗證的傳送資訊外，還多了A回送給B的資訊，這個資訊裡包含著B所送過來的R_b，可以讓B檢驗R_b是否與當初B所傳送的R_b相同，這樣的機制可以用在兩造之間時間沒有同步化的情形下。

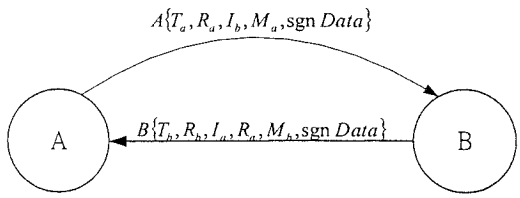

Figure 19. X.509三向驗證程序

3. SSL轉接系統

3.1. SSL轉接系統模型

SSL轉接系統可以用在企業內部網路(Intranet)通往網際網路(Internet)的通道閘口，將內部網路各種不同應用程式(Telnet、FTP、

HTTP等)及應用伺服器，通往網際網路的通訊明文資料(plaintext)加密成為密文資料(ciphertext)，之後再轉送至目的地轉接系統。或是將網際網路上另一台轉接系統所傳送進來的密文資料(ciphertext)解密成為明文資料，之後再轉送到內部網路的特定應用伺服器或是應用程式端。我們可以用下圖表示此一點對點模型。

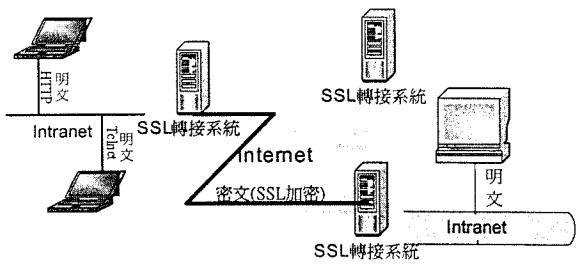

Figure 20.　SSL轉接系統之點對點模型

SSL轉接系統也可以執行成為個人電腦內的常駐服務，對內聆聽在本地端網路位址(127.0.0.1)，對外使用個人電腦在網際網路上的合法網路位址，將所有欲通往網際網路上轉接系統的對外明文資料送至本地端網路位址(127.0.0.1)上的特定阜號，經過加密處理後再由網路上的合法位址送出至目的轉接系統。同樣地從網際網路上另一台轉接系統送進來的密文資料，在經由個人電腦合法位址進入之後，經過加密處理後便轉送回本地端網路位址(127.0.0.1)上的特定阜號，如此一來所有暴露在網際網路上的通訊資料流，皆可以獲得轉接系統的保護。下圖是用來表示此一端對點、端對端模型。

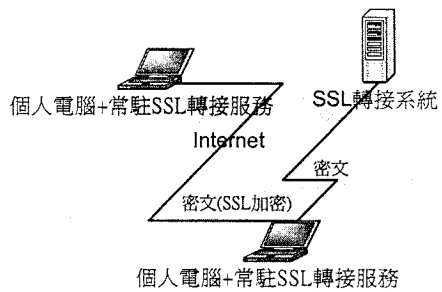

Figure 21.　SSL轉接系統之點對端模型

SSL轉接系統可以直接與瀏覽器(Browser)以SSL/TLS方式溝通，因此不管使用者在網路上的任何一個角落，只要有瀏覽器的存在皆可以直接在瀏覽器與轉接系統間建立一條SSL/TLS加密的通道，可以用來提供遠端控制轉接系統或是由轉接系統轉送至目的網頁提供伺服器。下圖是用來表示此一模型。

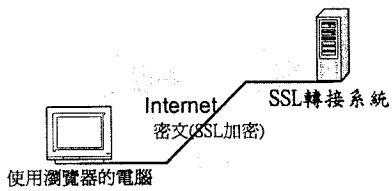

Figure 22. SSL轉接系統接駁瀏覽器之模型

3.2. SSL轉接系統特色

3.2.1. 提供一個安全的通訊環境

SSL轉接系統不僅讓用戶端以應用程式透通(transparency)方式安全轉接，更讓遠處位於轉接系統後的應用程式伺服端安全地接收與發送通訊資料，能夠有效地防止機密資料(sensitive information)在傳送過程中外洩(disclosure)、防止非法人員對傳送資料篡改，以確保資料內容的完整性(integrity)、防止網路上傳遞的資料被重播(replay)、刪除(deleting)、與遺失(loss)，以確保資料內容次序的完整性(sequence integrity)、及防止非法人員冒名傳送假資料，以確保訊息來源身份的辨識(message origin authentication)。

3.2.2. 提供便宜、簡單的解決方案

現今的VPN產品在價格上視其整合性、效能與其他附加功能，從數十萬到數百萬以上比比皆是，且VPN在金鑰管理上的多樣性與複雜性，反而影響到不同品牌，甚至同品牌不同類型VPN產品間的相容性。反觀SSL轉接系統，使用SSL/TLS的機制，不僅金鑰交換、密鑰產生、與資料加解密皆是由SSL/TLS協定完成，更重要的是網路上已經有現成的SSL/TLS函式庫，不僅公開程式碼、自由下載並且免費提供給任何人作任何用途，因此SSL轉接系統比任何VPN產品皆有價格上的競爭優勢。

3.2.3. 採用多階層的管理機制

SSL轉接系統採用多階層的管理機制，依權限大小授與管理階層三種不同身份：初始者(Initiator)、管理者(Manager)、操作者(Operator)。不同的管理身份有著不同的管理權限與責任，層層地管理轉接系統的機密資訊與繞路管理，能夠有效地防止人員不小心或是蓄意地洩漏轉接系統重要機密資訊，達成使用、操作、管理層層把關，安全滴水不漏的局面。

3.2.4. 採用強化的安全機制

SSL轉接系統特地針對SSL/TLS協定上的弱點[9]作防範與加強，例如避免使用有安全漏洞的SSL Version 2.0、及不安全的金鑰交換機制(Anonymous Diffie-Hellman)，還有使用1024 位元的公開金鑰、使用Triple DES與IDEA等安全度較高的對稱式密碼演算法，並且將系統中最重要的私有金鑰寫入硬體中。這些方法皆能有效地增加系統對內對外的安全度，確保系統運作安全無虞。

3.2.5. 相容於瀏覽器密碼機制

一般瀏覽器(Browser)皆內建SSL/TLS功能，可以與具備SSL/TLS功能的網頁提供伺服器建立起加密通道，使用者更可以將自己的憑證匯入瀏覽器中，作為遠端網頁提供伺服器確認使用者身份之用。這樣的機制同樣可以轉移到瀏覽器與SSL轉接系統，兩者之間不僅可以建立起SSL/TLS加密通道，且SSL轉接系統可以要求用戶端提供辨別身份的憑證，以確認使用者的真實身份，在確認使用者身份後，再提供使用者安全通訊轉接的功能。

3.3. 系統架構

3.3.1. 系統概觀

整個SSL轉接架構，依功能來分類共可分為五大模組與兩大服務，分別是控制模組(Control Module)、資料處理模組(Message Process Module)、密碼模組(Crypto Module)、接收模組(Received Module)、與傳送模組(Sent Module)，另外還有身份驗證服務(Authentication Services)、及存取控制服務(Access Control Services)。在身份驗證服務方面，我們可以採取Smart Card、Challenge-Response、Certificate、LDAP authBind、NIS(Network Information System)之中的任一種方式。存取控制服務方面，我們可以採取Role-based Access Control、Access Control List之中的任一種方式。完整架構圖如下圖所示。

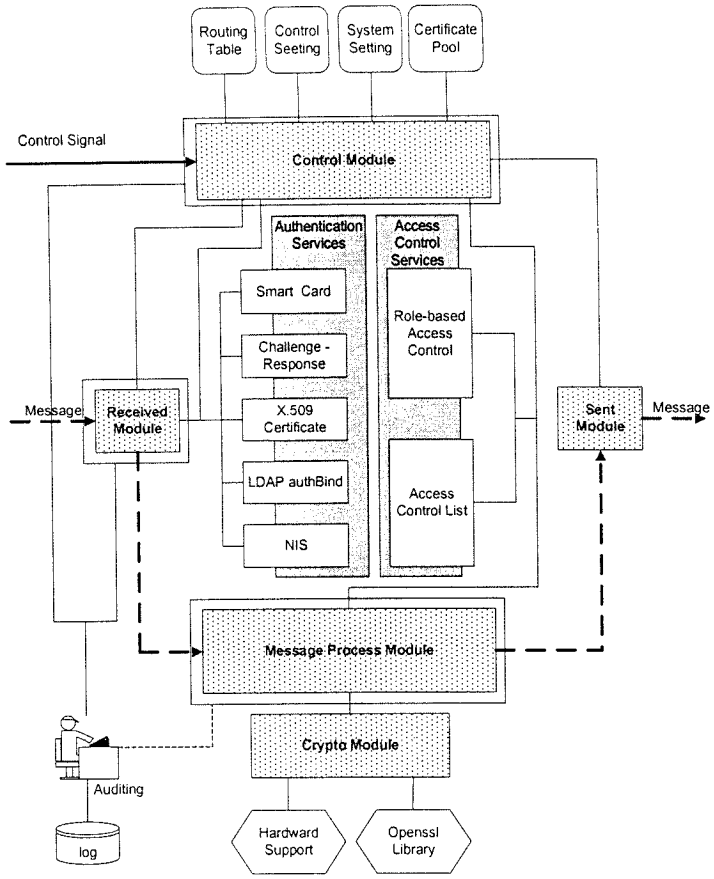

Figure 23. SSL轉接系統架構

轉接的資料流是由接收模組進入，經由資料處理模組，最後由傳送模組送出。管理者的控制訊號則由控制模組進入，對於繞路設定(Routing Table)、控制設定(Control Setting)、系統設定(System setting)、以及憑證資料群(Certificate pool)作讀取、修改、新增、與刪除等控制處理。

控制模組會將繞路設定、系統設定、與憑證資料群傳送到接收模組與傳送模組，參與這兩個模組的處理與運作。而身分驗證服務則是提供控制模組驗證管理人員的身份與接收模組驗證使用者的身份。存取控制服務是提供控制模組在驗證身份後進行存取控制與資料處理模組在處理轉接資料流時作存取控制。

密碼模組則可以藉由硬體支援，將私有金鑰儲存在硬體設備中，

所有需要私有金鑰處理的資料皆轉由硬體來處理，再丟回資料處理模組，其餘的密碼功能則可由Openssl函式庫來提供。接收模組、控制模組、與資料處理模組皆受到Auditing的監控，所有的存取紀錄以log形式儲存在資料庫中，以供系統管理人員檢視、分析之用。

3.4. 多階層管理機制

SSL轉接系統採用多階層管理機制，將所有人員分為初始者、管理者、操作者、及使用者四種身份，每種身份各司其職，維持著系統從安裝、設定、運作、與使用各個階段的需求。使用者使用轉接服務與管理者、及操作者對於管理模組的修改、刪除、新增都將受Auditing的監視。

3.4.1. 初始者

初始者是指負責系統設定、系統安裝、及初始化的人員，另外還負責啟始化周邊支援設備，諸如身份驗證服務與存取控制服務，以及新增、刪除、修改、與讀取管理者、操作者、與使用者的身份與權限。初始者擁有管理架構裡的最高權限，是SSL轉接系統建立之後，就會存在的管理人員身份，之後再由初始者去設定管理者、操作者與使用者的資料，而這些資料皆會送到身份驗證與存取控制兩項服務的資料庫中，供之後在驗證身份與存取控制時使用。在系統初始化完成後，整個系統的維運便交由管理者來接管。

3.4.2. 管理者

管理者的權限是次於初始者而在操作者與使用者之上，其工作是負責系統初始化後的運作管理，包含憑證管理、金鑰管理、繞路管理、及使用者管理共四大管理。管理者在登入系統時，必須先通過身份驗證服務的身份確認，之後對於四大管理的權限便藉由存取控制服務而取得，以下就四大管理作深入的描述。

3.4.2.1. 憑證管理

憑證管理是採憑證匯入系統方式。整個SSL轉接系統共需要兩種憑證，分別是發行CA憑證與伺服端憑證。發行CA憑證是用來驗證外來憑證時的依據，當兩台SSL轉接系統要互相建立SSL/TLS通道時，發出建立連線的那一端稱為轉接用戶端，接收連線的那一端稱為轉接伺服端，在SSL/TLS協定裡規定，轉接用戶端一定會收到轉接伺服端送來的憑證，此時轉接用戶端便需要藉由這些發行CA憑證的資訊來確認轉接伺服端的身份真實性，假若驗證失敗，管理者可以事先選擇失敗後是否要允許連線或是切斷連線。至於轉接用戶端的憑證，假若轉接伺服端有設定要求轉接用戶端提供憑證的話，便會在轉接伺服端送出憑證之後伴隨憑證要求訊息給轉接用戶端，轉接用戶端收到後，就送出自己的憑證給轉接伺服端，再交由轉接伺服端根據該系統收集的發行CA資訊，來確認這張憑證是否是受到信任，進而確認轉接用戶端的身份。在此需要說明的是由於轉接系統有可能是轉接用戶或是轉接伺服的身份，因此其憑證皆是採用系統的伺服端憑證。這些發行CA的憑證與伺服端憑證是由管理者匯入系統，設定系統參數後，讓SSL/TLS handshake時，能夠藉由讀取參數而送出憑證或是取得憑證驗證的依據，維持SSL/TLS運作的正常。

3.4.2.2. 金鑰管理

金鑰管理也是採匯入系統方式。系統匯入伺服端憑證時，同樣會要求對應的私有金鑰匯入，管理者同時可以檢驗憑證與私有金鑰是否互相吻合，以避免SSL/TLS Handshake 之金鑰不符的錯誤發生。管理者對於金鑰使用有一定週期的規範，當超過該週期時伺服端憑證與私有金鑰就應作更新的動作，而這段期限的訂定可以是發行該數位憑證之CA在憑證有效期上作強制性的規範，或是管理者自發性地更換新的伺服端憑證與私有金鑰。更換憑證與私有金鑰的用意是在於防止惡意人士對於憑證裡的公開金鑰在有效時間內從事反解私有金鑰運算成功，以RSA演算法為例，在1999年公布的RSA-155(512位元)挑戰需要5.2個月的時間才有辦法解出，並且使用了將近三百台工作站與個人電腦，在特殊的演算法的環境下才能達成。但伴隨著電腦運算速度的倍數成長，以現有的機器設備勢必能在更短的時間內反解出512位元的RSA私有金鑰，因此金鑰的定期更新是必需的。

金鑰除了在有效時間內被反解的風險外，另外還有洩漏的風險，在SSL轉接系統中，對於密碼模組下的硬體支援，是採用將私有金鑰寫入硬體加速器中，藉由修改Openssl函式庫，將所有用到私有金鑰計算的部分，皆導由加速器來運算。這樣的好處是私有金鑰一旦安裝在硬體中，所有惡意人員要藉由偷窺、及洩漏系統私有金鑰而進行的惡意企圖都將被瓦解，因為私有金鑰已經寫入硬體中，硬體只對通過確認的資料流進行私有金鑰運算，私有金鑰完全不會外露在危險當中，以確保系統的安全性。

3.4.2.3. 繞路管理

繞路管理是SSL轉接系統的運作核心，藉由繞路管理，可以設定在轉接伺服端的特定接收阜開啟轉接服務，轉接到另一系統之轉接伺服端的特定接收阜或是區域網路內的某台應用伺服器的特定接收阜。另外還有指定轉接用戶端與轉接伺服端所使用的密碼機制、是否使用SSL/TLS加密通道、是否重用session、是否允許不合法憑證作SSL/TLS連結、使用SSL/TLS的版本、繞路模式、使用的憑證、與憑證驗證模式等設定。

SSL轉接系統密碼機制的設計，是採要建立SSL/TLS通道的轉接用戶端與轉接伺服端之間所選定的密碼機制當中複雜度最高的一種來當作雙方的共通密碼機制，而所有可供選擇的密碼機制又以3DES_SHA為最高等級。是否使用SSL/TLS加密通道，簡而言之便是轉接系統的進入與出去的資料流是否要作SSL/TLS加密保護。是否重用session是指是否要開啟SSL/TLS協定當中的session重用機制，假若開啟的話，當固定的轉接伺服端與轉接用戶端之間存在不止一個SSL/TLS連線時，先開啟連線所協商出的密碼原則，可以被同樣兩端的相同、及不同繞路的其他連線利用SSL/TLS session重用機制，在SSL/TLS Handshake時，不經複雜的協商過程，直接使用先前連線所協商的密碼原則，馬上進入資料加密保護傳輸程序。成功開啟session重用機制最多可減少將近50%的存取時間。

是否允許不合法憑證作SSL/TLS連結選項裡所指的不合法憑證，一般是指該張憑證的發行CA憑證並沒有儲存在系統的身份驗證服務資料庫裡，因此系統無法信任此張憑證的真實性，假若我們允許不合法的憑證作連結，則我們會在驗證失敗後仍允許SSL/TLS通道的建立，當然這樣的選擇會減低通訊的安全度，在正常情況下我們是不啟用此項設定，開啟這項設定大部分時機是用在SSL轉接系統會將加密

資料流轉接到某應用伺服器，而此應用伺服器並非系統範圍的一部份，且發行該應用伺服器憑證的CA憑證並沒有安裝到本地轉接系統的身份驗證服務資料庫裡。

繞路模式是指繞路所要轉接的應用程式協定，在系統的設定裡有TCP模式、FTP模式、HTTP模式，管理者視繞路所要轉接的應用程式協定而更改此項設定。使用的憑證是指繞路送出與接進來的連線，在使用SSL/TLS時，所要使用到的憑證。憑證驗證模式有兩種選擇：只驗伺服端或是用戶端與伺服端皆驗證，在SSL/TLS的協定裡，我們可以選擇驗證用戶端與伺服端，或是只驗伺服端，驗證的過程是交由身份驗證服務來作驗證。

3.4.2.4. 使用者管理

使用者是指使用SSL轉接服務的人員。在使用者管理上，SSL轉接系統是採群組管理方式，每一個群組包含一個或多個使用者，每個使用者屬於一個或多個群組，而群組可以擁有多個繞路權限與獨立憑證。群組管理的好處是為了簡化使用者管理上的繁雜，因為伴隨著使用者的增加，管理者若沒有將使用者分成群組來處理，每一個使用者都要設定允許的繞路權限與使用的憑證，這些步驟將會成為管理者的沈重負擔，而且也會造成日後管理上的不便，至於個別使用者的設定，我們是以使用者本地端網路位址或是使用者個人憑證作為身份驗證依據，每個使用者只要加入群組，即具有該群組的繞路權限。

3.4.3. 操作者

操作者的權限是在管理者之下，其工作是負責SSL轉接系統的一

般運作。操作者與管理者不同的地方在於操作者的權限範圍只限於繞路管理與使用者管理，憑證管理與金鑰管理則是操作者無法觸及的項目。操作者有管理繞路與使用者的兩項權力，在SSL轉接系統的正常運作下，單純管理這兩個項目，已經足夠讓系統在正常情形下順利運作，因此操作者可以分擔管理者管理系統的負擔，並且在不觸及系統安全的重要資訊(金鑰與憑證)下，讓系統能夠正常地運作。

3.4.4. 使用者

使用者是指使用SSL轉接系統服務的人員。使用者對於SSL轉接系統只有使用服務的角色，由於並不具有管理的角色，因此使用者是無法登入管理模組，參與系統的管理與維護。SSL轉接系統對於使用者的身份確認是由使用者的來源位址、使用者的憑證或是其他身份驗證服務機制來確認身份，只要通過SSL轉接系統的身份驗證，轉接系統便會處理使用者應用程式所送出的資料流，將資料流安全地送至另一台轉接系統或是其後端的應用程式伺服器，假若沒有通過身份驗證的話，系統會中斷此一使用者連線，不會替該使用者的資料流作轉接的動作。

3.4.5. 存取控制

SSL轉接系統使用存取控制來對通過身份驗證的使用者、初始者、管理者與操作者的權限與行為作規範。SSL轉接系統所使用的存取控制服務是ACL(Access Control List)，以使用者、群組與繞路三者關係為例，藉由檢驗繞路的ACL，可以很方便地瞭解那個群組、及那些使用者是屬於這個繞路的範圍，例如使用者α是屬於群組A，群組A

屬於繞路甲，因此使用者α可以使用繞路甲的繞路權限，如下圖所示。

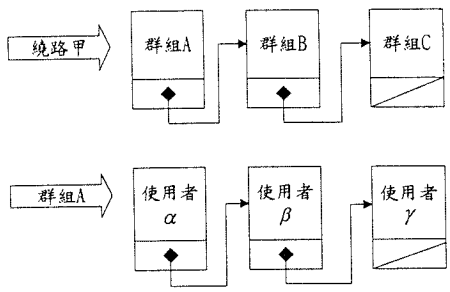

Figure 24. 存取控制範例一

在管理階層方面，初始者、管理者與操作者的權限同樣也是以ACL來對金鑰管理、憑證管理、繞路管理、及使用者管理作存取控制。以下圖為例，大雄擁有對金鑰管理的讀取權限與使用者管理的新增、修改、刪除、及讀取的權限，技安擁有對金鑰管理的新增、修改、刪除、及讀取的權限與使用者管理的新增、修改、刪除、及讀取的權限，而阿福則是具有對金鑰管理新增、及讀取的權限與使用者管理的讀取權限。

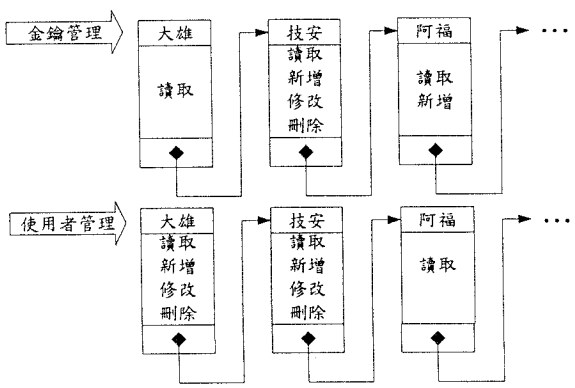

Figure 25. 存取控制範例二

藉由身份驗證服務，能對轉接系統的初始者、管理者、操作者與使用者的身份作確認，之後再藉由存取控制服務，對初始者、管理者、操作者與使用者作權限控制。而這些服務的資料設定則是依據多階層管理架構，初始者有新增、刪除、修改、及讀取管理者、操作者與使用者的權限，管理者有新增、刪除、修改與讀取使用者的權限，操作者有新增、刪除、修改、及讀取使用者的權限，所有設定動作，皆受到Auditing的監視，記錄在資料庫中備查。

3.5. 機密性與完整性

SSL轉接系統的資料機密性與完整性是由密碼模組(Crypto Module)所達成，配合系統憑證管理與金鑰管理輔助，以SSL協定機制來作憑證交換、密碼機制(Cipher Suite)協商、及計算通訊密鑰，將由接收模組送至資料處理模組的資料流交由通訊密鑰來加密保護。

SSL轉接系統目前所支援的密碼機制有3DES_SHA、IDEA_128_SHA、RC4_128_SHA、RC4_128_MD5、DES_56_SHA與RC4_56_SHA共六種，這些密碼機制皆名列在Eric Murray報告中所提出的高強度密碼機制中。兩個SSL轉接系統在SSL/TLS Handshake時，系統會以安全度較高的密碼機制為雙方協商的優先考量，以加強資料傳輸的機密性。而這些密碼機制是由Openssl函式庫所實作，因此隨著Openssl的發展與更新，這些密碼機制將隨著更新與升級，以防範在有效時間內的攻擊法。

SSL 轉接系統特地針對Eric Murray報告中所提出的SSL/TLS弱點作防範與加強，除了使用高強度密碼機制外，更避免使用SSL Version 2.0協定與不安全的金鑰交換機制(Anonymous Diffie-Hellman)，且使用1024位元公開金鑰來當作轉接伺服端金鑰，能有效地防止金

鑰在有效時間內被反解出的危機。竊取系統通訊資料的常見方式除了以特殊演算法與暴力攻擊法計算通訊密鑰或是反解出私有金鑰以破解SSL/TLS Handshake時所傳送的機密資訊外，另一個方式便是直接偷取系統的私有金鑰。SSL轉接系統使用硬體加速器來減低私有金鑰被洩漏的可能，硬體加速器在寫入私有金鑰同時，也會將存取該把私有金鑰的密鑰寫入硬體，而所有需要私有金鑰加解密的資料流在轉入硬體運算前，需要有相對密鑰的輸入，才能讓硬體加速器作私有金鑰運算，在不暴露私有金鑰的情形下，能有效地防止系統管理人員有心或是無意地將私有金鑰洩漏出去的風險。

4. 結論

在本計畫執行的過程中，我們對網路安全的相關課題作一深入的探討所包含的相關主題有密碼學中的對稱行加密系統、非對稱型加密系統、雜湊函數與訊息確認碼以及網路安全協定中的SSL、TLS與Telnet等相關的網路安全協定，並以此實做出電子閘門系統的安全通訊系統，以期能強化國防體系的安全機制。

5. 參考文獻

[1] 電信總局暨所屬機構電腦作業安全管理實施要點(交通部電信總局印發)。

[2] 電信總局暨所屬各機構個人電腦軟體管理與管制暫行實施要點(交通部電信總局印發)。

[3] 交通部電信總局暨所屬各機構推行國家機密保護辦法實施細則(交通部電信總局印發82/3)。

[4] 電腦處理個人資料保護法施行細則(法務部法律事務司，法規編號:行政09-03-017)。

[5] 行政機關電子資料流通實施要點(行政院研究發展考核委員會資訊管理處)。

[6] 美國國防部可信賴計算機系統評估準則 (橘皮書)。

[7] 行政院研究發展考核委員會專題研究計畫成果報告(健全政府機關電子公文交換作業安全專題研究計畫)。

[8] 中華電信企業資訊網路(Intranet)規劃書(資訊處85/12)。

[9] 各分公司於86/3/31—86/4/15提供研究所彙整、及分析。

[10] 電腦應用系統稽核 (工研院，85/3)。

[11] 資訊工業策進會 (EDI簡訊)。

[12] 電信網路上資訊安全技術之研究與規劃期末報告 (中華電信資訊處，85/6/30)。

[13] 公文處理現代專題研討, 84年 行政院研考會。

[14] 楊中皇，*資訊安全簡介*，電信研究所基本科技研究室，1995年9月。

[15] 賴溪松、韓亮、張真誠著，*近代密碼學及其應用*，松崗電腦圖書資料股份有限公司，1995年9月。

[16] 連秀婉譯，*Internet安全性技術實務*，博碩顧問有限公司，1996年6月。

[17] Secure Electronic Transaction(SET) Specification Book 2, 1996。

[18] Public-Key Crytography Standards, RSA Data Security, Inc。

[19] ITU Rec. X.509(1993), 1995。

[20] Data Encryption Standard, FIPS 46, 1977。

[21] Applied Cryptography, Wiley&Sons, Inc., 1994。

[22] Internet Firewall and Network Security, New Riders Publishing, 1995。

[23] Request for Comments 1421, 1422, 1423 - Privacy Enhancement for Internet Electronic Mail, 1993。

[24] A. Frier, P. Karlton and P. Kocher, "The SSL 3.0 Protocol ", Netscape Communication Corporation, Nov. 1996

[25] T. Dierks and C. Allen, "The TLS Protocol Version 1.0", RFC2246, Jan. 1999

[26] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol," RFC 2401, Nov. 1998

[27] Visa International and MasterCard. Secure Electronic Transaction (SET), Version 1.0. 31 May 1997

[28] "Extranet Security and Management: The Difference Between Extranets and VPN's", Aventail Technical Paper, MAR 1999

[29] R.S Sandhu, P. Samarati, "Access control: principle and practice",IEEE Communication, vol.32, Sept. 1994

[30] OpenSSL Project's URL : http://www.openssl.org/.

[31] H. Krawczyk, M. Bellare and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC2104, Feb. 1997

[32] Eric Murray, "SSL Server Security Survey", can be found : http://www.lne.com/ericm/papers/ssl_servers.html, Jul. 2000

[33] RSA-Challenge, Factorization of RSA-155: http://www.rsasecurity.com/rsalabs/challenges/factoring/rsa155.html

[34] A. Goldberg, R. Buff, A. Schmitt, "Secure Web Server Performance Dramatically Improved by Caching SSL Session Keys", June 1998

[35] W. Stallings, "Cryptography and Network Security: Principles and Practice", 2nd Ed. Prentice Hall International, Inc

[36] W. Richard Stevens, "TCP/IP Illustrated, Volume1", Addison

Wesley,1999

[37] H. Tanaka, "Identity-Based Noninteractive Common-key Generation and Its Application to Cryptosystems, " Transactions of the Institute of Electronics, Information, and Communication Engineers, V.J75A,n.4 , Apr 1992

[38] J. Tardo and K. Alagappan, "SPX : Global Authentication Using Public Key Certificates, "Proceedings of the 1991 IEEE Computer Society Symposium on Security and Privacy, 1991

[39] A. Tardy-Corfdir and H. Gillbert, " A Known Plaintext Attack of FEAL-4 and FEAL-6 ," Advances in Cryptography –Crypto' 91 Proceedings, Springer-Verlag,1992

[40] M.-J Toussaint, " Verification of Cryptographic Protocols ," ph.D. dissertation, Universite de Liege 1991

[41] Y.W Tsai and T.Hwang , "ID Based Public Key Cryptosystem Based on Okamoto and Thanaka's Based One-way Communication Scheme,"Electronics Letters, v.26,n.10,1 May 1990

[42] G.Tsudik," Message Authentication with One–way hash Functions,"ACM Computer Communications Review, v.22,n.5 1992

[43] B.Preneel, personal communication, 1995

[44] G.P.Purdy , " A High-Security Log-in Procedure, " Communications of the ACM,v.17,n.8,Aug 1974

[45] J.-J. Quisquater ," Announcing the Smart Card with RSA Capability ," Proceedings of the Conference: IC Cards and Applications, Today and Tomorrow , Amsterdam , 1989

[46] G.J. Popek and C.S. Kline,"Encryption and Secure Computer Networks," ACM Computing Networks,"ACM Computing Surveys,v.11,n.4,Dec 1979

[47] ANSI X3.106,"American National Standard for Information Systems-Data Link Encryption," American National Standards

Institute, 1983.

[48] NIST FIPS PUB 186, "Digital Signature Standard," National Institute of Standards and Technology, U.S. Department of Commerce, 18 May 1994.

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

[50] R. 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.

[51] ISI for DARPA, RFC 793: Transport Control Protocol, September 1981.

[52] SSH-Cryptography A-Z – Cryptographic Algorithms http://www.ssh.fi/tech/crypto/algorithms.html

[53] The SSL 3.0 Protocol, http://home.netscape.com/newsref/std/SSL.html

[54] Information on SSLRef, a reference implementation available, http://home.netscape.com/newsref/std/sslref.html

[55] CCITT Recommendation X.509 - The Directory: Authentication Framework, 1993

[56] The SSL Protocol Version 3.0, Internet Draft, http://home.netscape.com/eng/ssl3/draft302.txt

[57] How SSL Works, http://developer.netscape.com/tech/security/ssl/howitworks.html

[58] RFC-959, http://www.cis.ohio-state.edu/htbin/rfc/rfc959.html

[59] R.L. Rivest, "The RC4 Encryption Algorithm," RSA Data Security, Inc., Mar 1992.

[60] NIST FIPS PUB 180-1,"Secure Hash Standard," National Institute of Standards and Technology, U.S. Department of Commerce, DRAFT,

32 May 1994.

[61] R.Rivest. RFC 1321: The MD5 Message Digest Algorithm, April 1992.

[62] T. C. Wu and W. H. He, "A geometric approach for sharing secrets," *Computers & Security*, v. 14, n. 2, 1995

[63] A. Shamir, "How to share a secret," *Communications of the ACM*, v. 22, n. 11, 1979

[64] B. Schneier, "Applied Cryptography," 2nd Ed., John Wiley & Sons, 1996.

[65] A. Herzberg, S. Jarecki, H. Krawczyk and M. Yung, "Proactive secret sharing or: how to cope with perpetual leakage," *Advances in Cryptology-CRYPTO'95 Proceedings*, Springer-Verlag, 1995

[66] D. R. Stinson, "Cryptography – theory and practice," CRC Press, 1995.

[67] R.L. Rivest, A. Shamir, and L.M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystem," *Communications of the ACM*, v. 21, n. 1, Feb 1978, pp. 120-126.

[68] R.L. Rivest, A. Shamir, and L.M. Adleman, "On Digital Signatures and Public-Key Cryptosystems," MIT Laboratory for Computer Science, Technical Report, MIT/LCS/TR-212, Jan 1979.

[69] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, v. 48, n. 177, 1987, pp. 203-209.

[70] V.S. Miller, "Use of Elliptic Curves in Cryptography," *Advances in Cryptology - CRYPTO '85, Lecture Notes in Computer Science*, 218(1986), Springer-Verlag, pp. 417-426.

[71] ANSI X9.17 (Revised), American National Standard for Financial Institution Key Management (Wholesale)," American Bankers Association, 1985.

[72] ISO DIS 8732, "Banking-Key Management (Wholesale),"

Association for Payment Clearing Services, London, Dec 1987.

[73] W. Tuchman, "Hellman Presents No Shortcut Solutions to DES," *IEEE Spectrum*, v. 16, n. 7, July 1979, pp. 40-41.

[74] R.L. Rivest, "The RC4 Encryption Algorithm," RSA Data Security, Inc., Mar 1992.

[75] R.L. Rivest, "The MD5 Message Digest Algorithm," RFC 1321, Apr 1992.

[76] "Proposed Revision of Federal Information Processing Standard (FIPS) 180, Secure Hash Standard," *Federal Register*, v. 59, n. 131, 11 Jul 1994, pp. 35317-35318.

[77] Research and Development in Advanced Communication Technologies in Europe, *RIPE Integrity Primitives: Final Report of RACE Integrity Primitives Evaluation (R1040)*, RACE, June 1992.

[78] M.J.B. Robshaw, "The Final Report of RACE 1040: A Technical Summary," Technical Report TR-9001, Version 1.0, RSA Laboratories, Jul 1993.

[79] J. Postel and J.Reynolds: Telnet Protocol specification, RFC 854, May 1983.

[80] J. Postel and J.Reynolds: Telnet option specifications, RFC 855, May 1983.

[81] Alan O. Freier, Philip Karlton, and Paul C. Kocher: The SSL Protocol, Netscape Communication Corp, ver 3.0, Mar. 1996.

[82] T. Dierks, C. Allen: The TLS Protocol Version 1.0, RFC 2246, Jan 1999.

[83] B. Schneier: Applied Cryptography, 2nd Ed. John Wiley & Sons, 1996.

[84] T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 10-18.

[85] X. Lai and J. Massey, "A Proposal for a New Block Encryption Standard," *Advances in Cryptology – EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 389-404.

[86] R.L. Rivest, "The MD4 Message Digest Algorithm," *Advances in Cryptology - CRYPTO '90: Proceedings*, Springer-Verlag, 1991, pp. 303-311.

[87] R. Srinivansan, Sun Microsystems, RFC-1832: XDR External Data Representation Standard, August 1995.

[88] Y.S. Yeh, C.C. Wang, "Construct Message Authentication Code with One-Way Hash Functions and Block Ciphers", *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, v. E82-A, No. 2, Feb. 1999, pp. 390-393.