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

91-2623-7-009-005

執 行 期 間 ： 91年01月01日 至 91年12月31日

計畫執行單位： 國立交通大學
資訊工程學系
計 畫 主 持 人： 葉義雄 教授
計畫共同主持人： 林祝興 副教授

日 期 ： 中華民國九十一年十二月二十日

目錄

圖目錄

1. 簡介

隨著網際網路的普及與電子商務的發展，網路安全這個議題已經越來越受到廣泛的重視與關切。人們在進行網路通訊與電子商務同時，總是會擔心自己的通訊資料、信用卡帳號是否會被不懷好意者所監聽與竊取，因此一些保護這些通訊的安全機制也應運而生，例如：Secure Socket Layer (SSL)/Transport Layer Security (TLS) 協定、Internet Protocol Security Protocol (IPSec) 、Secure Electronic Transaction (SET) 協定，其中在用戶端、伺服端的架構機制上又以SSL/TLS與IPSec為兩大主流。

雖然IPSec是目前虛擬私有網路(VPN)的主要實作機制，但是由於目前虛擬私有網路的一些相容問題與昂貴的設備成本，再加上IPSec是實作在OSI模型的網路層，因此在存取控制(Access Control)上，它只能以封包過濾的方式來當作存取控制的依據，相較之下SSL/TLS由於是實作在OSI模型下的表現層之上與應用層之下的協定，因此它可以依據使用者身份、所使用的應用程式與服務、加密演算法、時間或是日期來當作存取控制的依據，再加上SSL/TLS普遍使用在諸如網頁伺服器的安全通訊標準上(HTTPS)，以及瀏覽器的支援與內建SSL/TLS功能，使得SSL/TLS廣為使用與接受。

目前發展SSL/TLS函式庫最為著名的就屬OpenSSL計畫。OpenSSL計畫是承接澳洲程式設計師Eric Young所發展的SSLeay函式庫而來，是由一群網路上有熱誠的志願者所組成的團隊，自願性地維

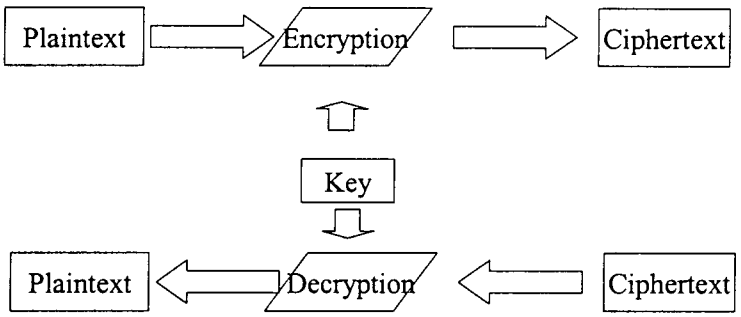
護與發展SSL/TLS函式庫，並且將開發的程式碼與發展成果公開給網路上每一個使用者，任何人皆可以下載函式庫與原始程式碼供作任何用途，不論是商業或是非商業。由於有這些公開組織發展函式庫供任何人使用，因此現今網路上以用戶端、伺服端為架構的通訊協定，諸如：TELNET、FTP、HTTP、NNTP、IMAP、POP、SMTP、IRC、LDAP等通訊協定，皆有支援SSL/TLS版本的開發與使用，因此SSL/TLS的穩定性與普遍性是無庸置疑的。

本計畫將引進SSL/TLS的觀念設計一具備身份確認的遠端連線機制，以保護透過遠端連機伺服器的通訊安全。

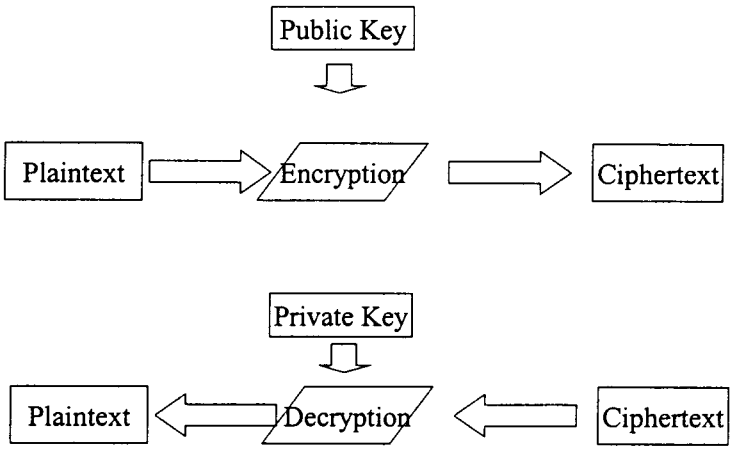
2. 相關技術

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：



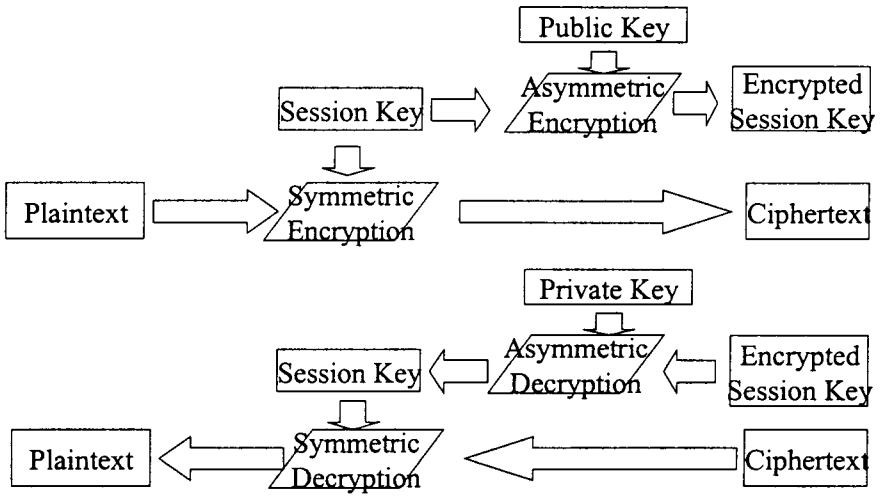
1. 對稱型密碼系統



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：



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

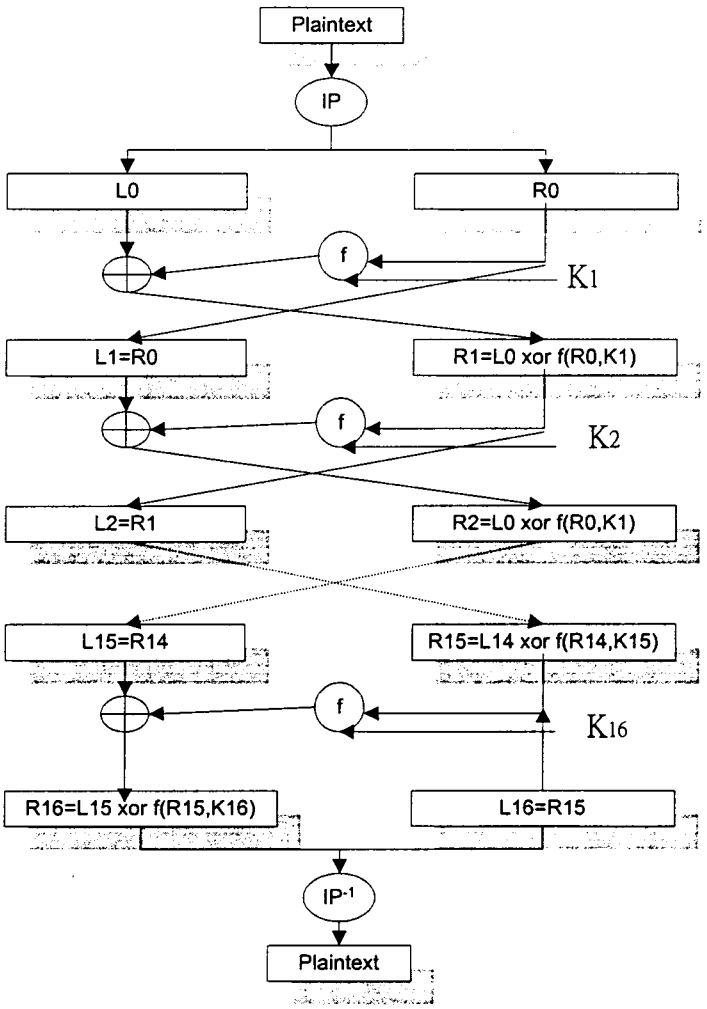
IP(M) = (L_0, R_0) = (32 bits, 32 bits)

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

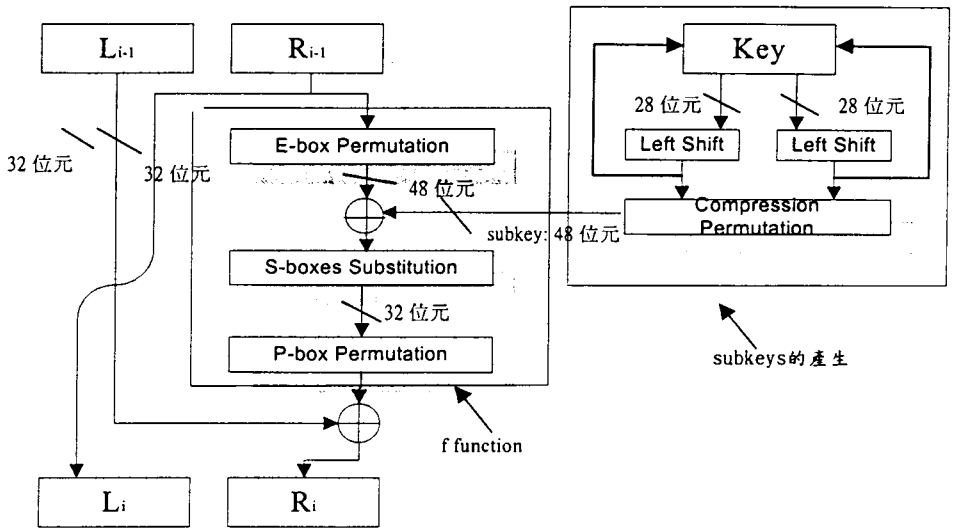
$(R_{16}, L_{16}) = (L_{15} \oplus F(R_{15}, K_{16}), R_{15})$

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

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



4. The Flow Chart of DES



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)

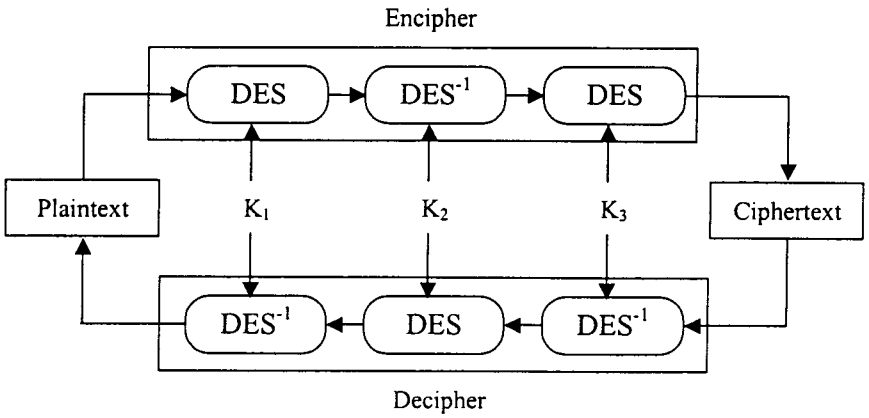
PC-1 (K) = $\mathbf{(A_0, B_0)}$ **=(28 bits, 28 bits)**

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

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

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

2.1.1.2. Triple DES (or DES^3)



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

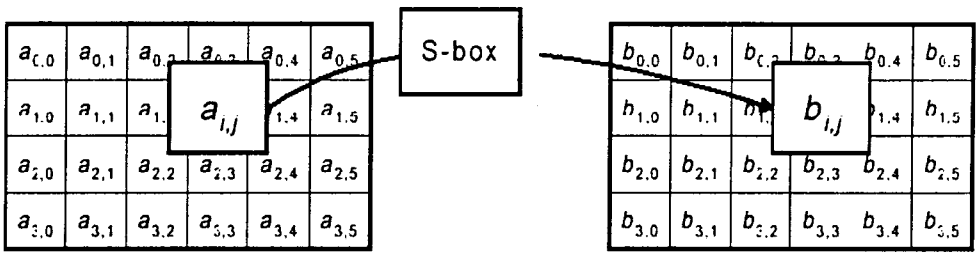
```
Round(State, RoundKey){
ByteSub(State);
ShiftRow(State);
MixColumn(State);
```

```
AddRoundKey(State, RoundKey);
}
```

最後一個 round 的轉換演算法

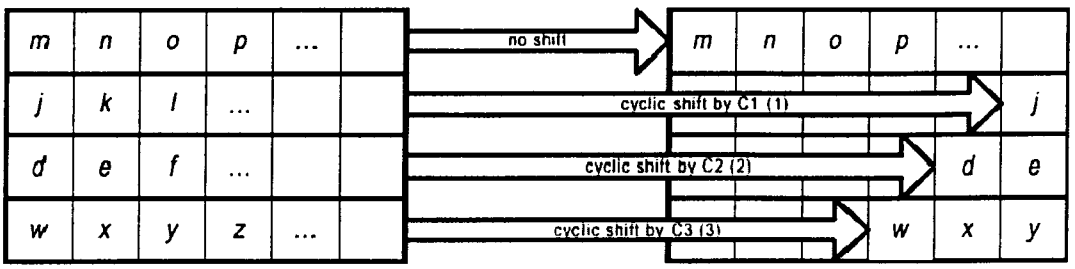
```
FinalRound(State, RoundKey){
ByteSub(State) ;
ShiftRow(State) ;
AddRoundKey(State, RoundKey);
}
```

***ByteSub* 的轉換**



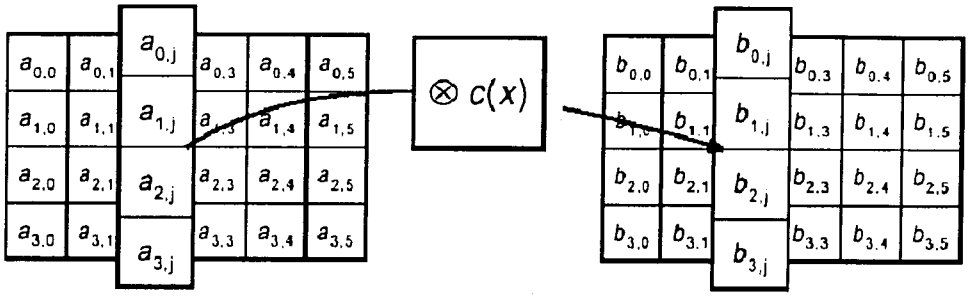
7. ByteSub Transformation of AES

ShiftRow 的轉換



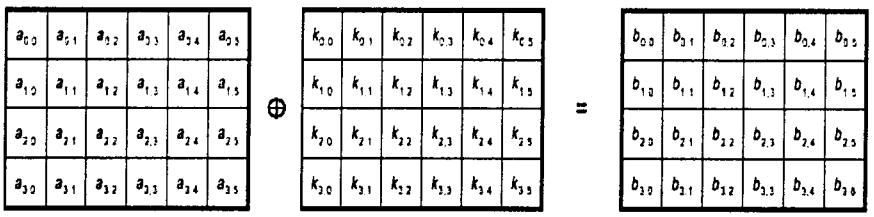
8. ShiftRow Transformation of AES

MixColumn 的轉換



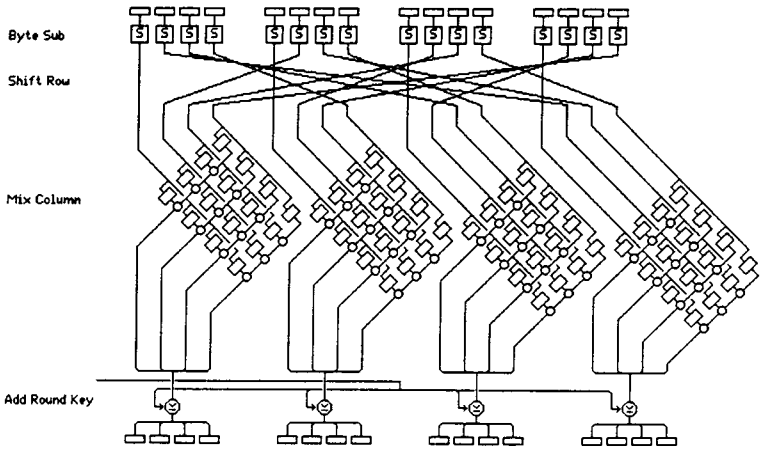
9. MixColumn Transformation of AES

AddRoundKey



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



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)= Φ(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 $64 \mid |p|$

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)

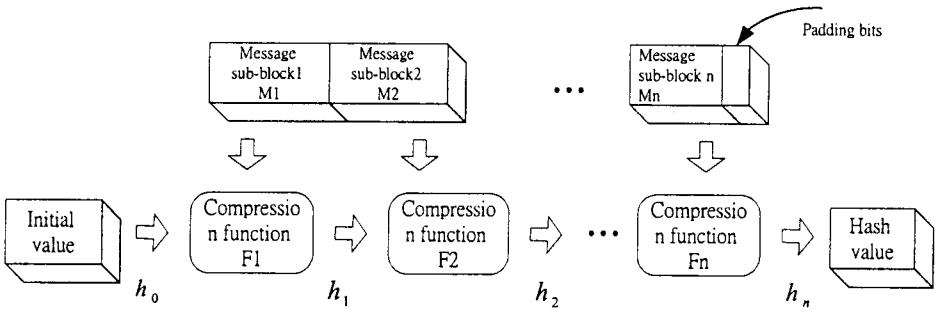
一個單向的雜湊函數H可以處理有任意的長度訊息M，而輸出一個有固定的長度的雜湊值h，即H= h。它有五個特性：

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。

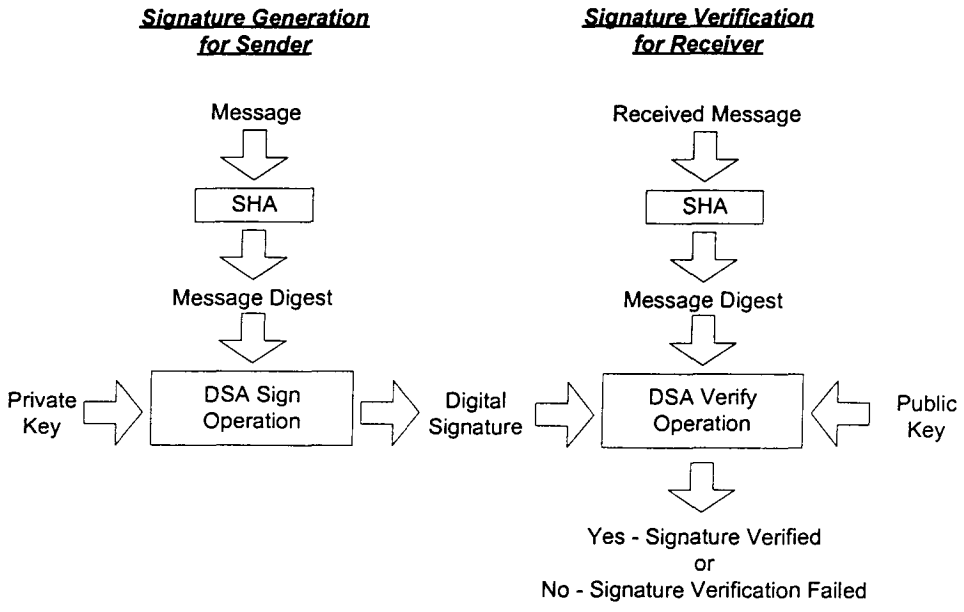


12. 反覆的壓縮函數之結構

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。



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：

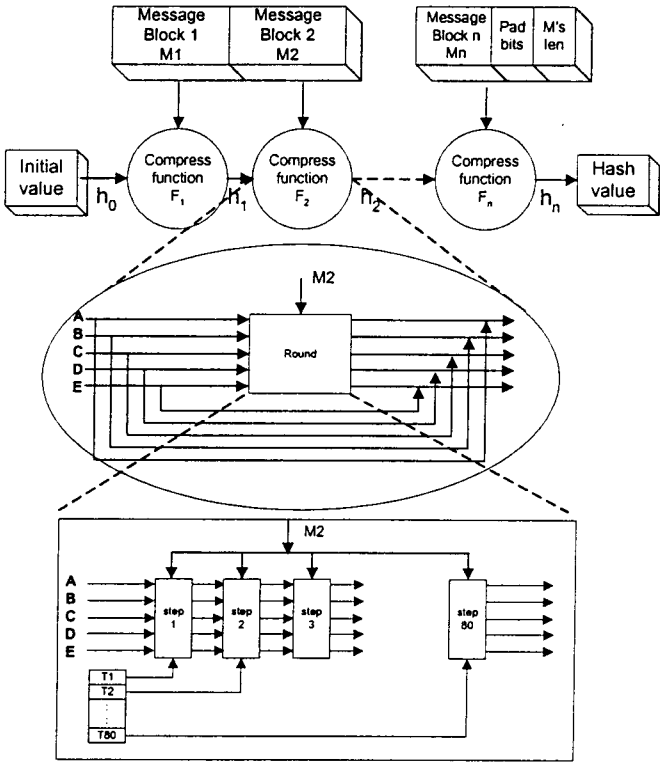
Round(a,b,c,d,k,s,i){a=b+((a +Z(b,c,d)+X[k]+Ti)<<s); }

Z = F， 0≦i≦19

Z = G，20≦i≦39

Z = H，40≦i≦59

Z = I，60≦i≦79



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 $= 2^{1/2}/4 = 0x5a827999$， $0 \leqq i \leqq 19$

$= 3^{1/}2/4 = 0x6ed9eba1$，$20 \leqq i \leqq 39$

$= 51/2/4 = 0x8f1bbcdc$，$40 \leqq i \leqq 59$

$= 101^{/2}/4 = 0xca62c1d6$，$60 \leqq i \leqq 79$

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];

else X[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：

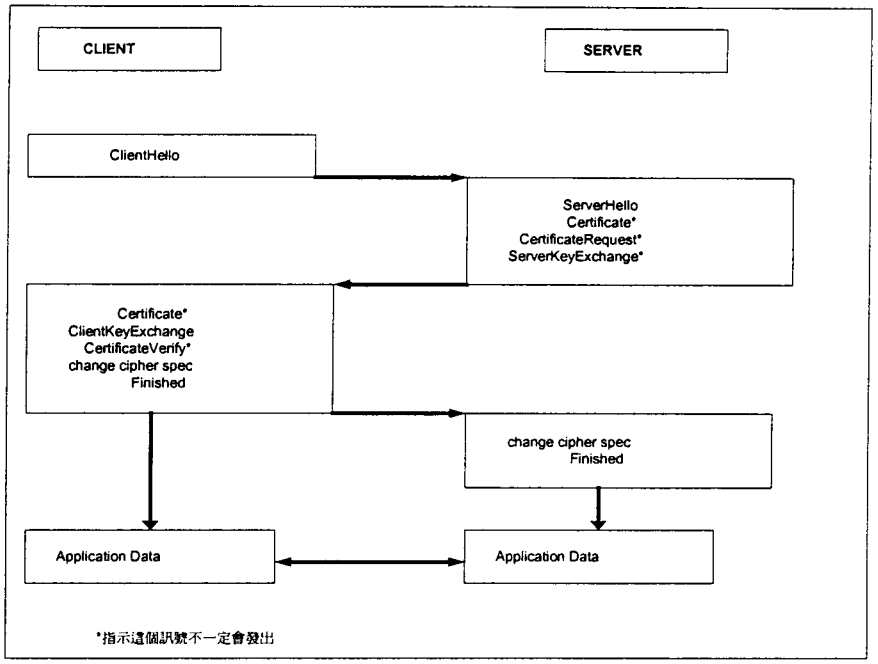
client 端送出一個「Client hello」的訊息，接收到的server 端會回應一個「Server hello」的訊息，否則就會發生一個錯誤，這個連結就失敗了。這個「Client hello」和「Server hello」訊息是用來建立client和server 間安全的限度，「Client hello」和「Server hello」訊息建立下面幾種屬性：通訊協定的版本、session ID、所使用的密碼演算法和壓縮的方法，並且，兩個隨取亂數是用來產生和交換key的。

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。

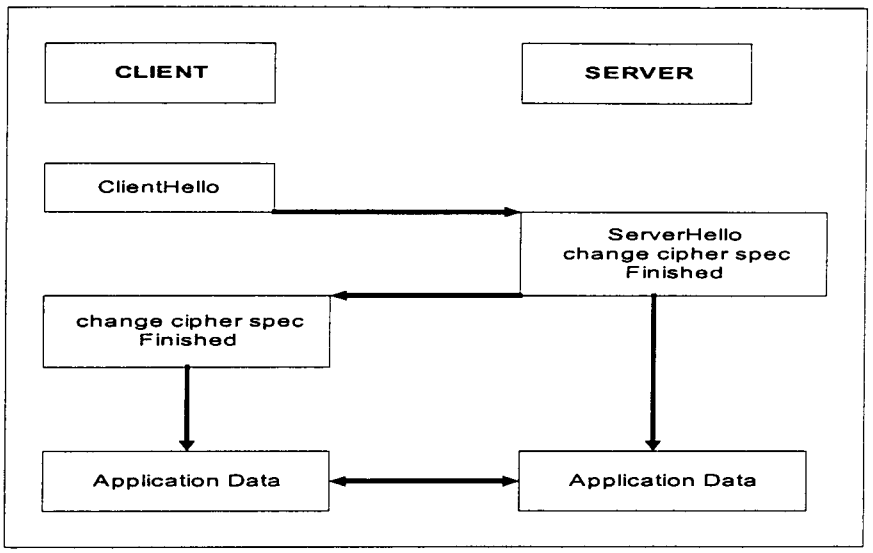


15. SSL 互握流程圖

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：



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

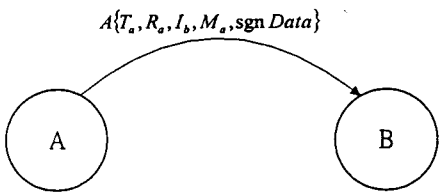
T->CGarble([some message]secret-key)

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

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。



17. X.509單向驗證程序

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

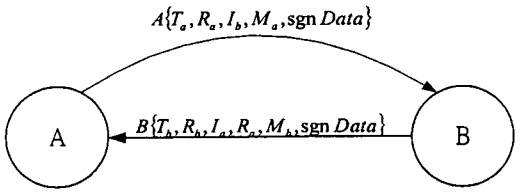
Tom現在就麻煩了，雖然他依然可以竄改傳送中的訊息，不過MAC的計算就使他現形了，Cary 或是Diana 發現不正確的MAC 值時，就可以馬上中斷通訊。

2.3. 身份驗證與存取控制

2.3.1. 身份驗證

2.3.1.2. 雙向驗證

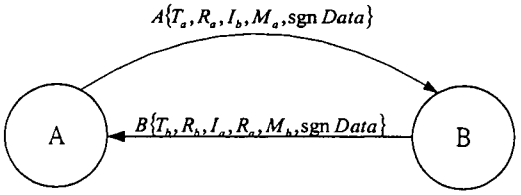
雙向驗證除了單向驗證由A傳送給B的資訊，還多了B回覆給A的資訊，其中包含了T_b，B的timestamp、R_b，B所產生的nonce、R_a，A之前所傳送的nonce 、I_a，表示這個資訊是要傳送給A、M_b，B要傳送給A的秘密資訊，為了保密緣故，這個秘密資訊會用A的公開金鑰作加密、及sgnData，將傳送的所有訊息利用B的私有金鑰所作的簽章，A收到B的驗證步驟完全跟單向驗證B所作的一樣，只不過多了確認送回來的R_a，是否是當初A所送出的R_a。雙向驗證是為了讓雙方可以互相驗證彼此的身份，並且彼此交換要給對方的秘密資訊。



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。

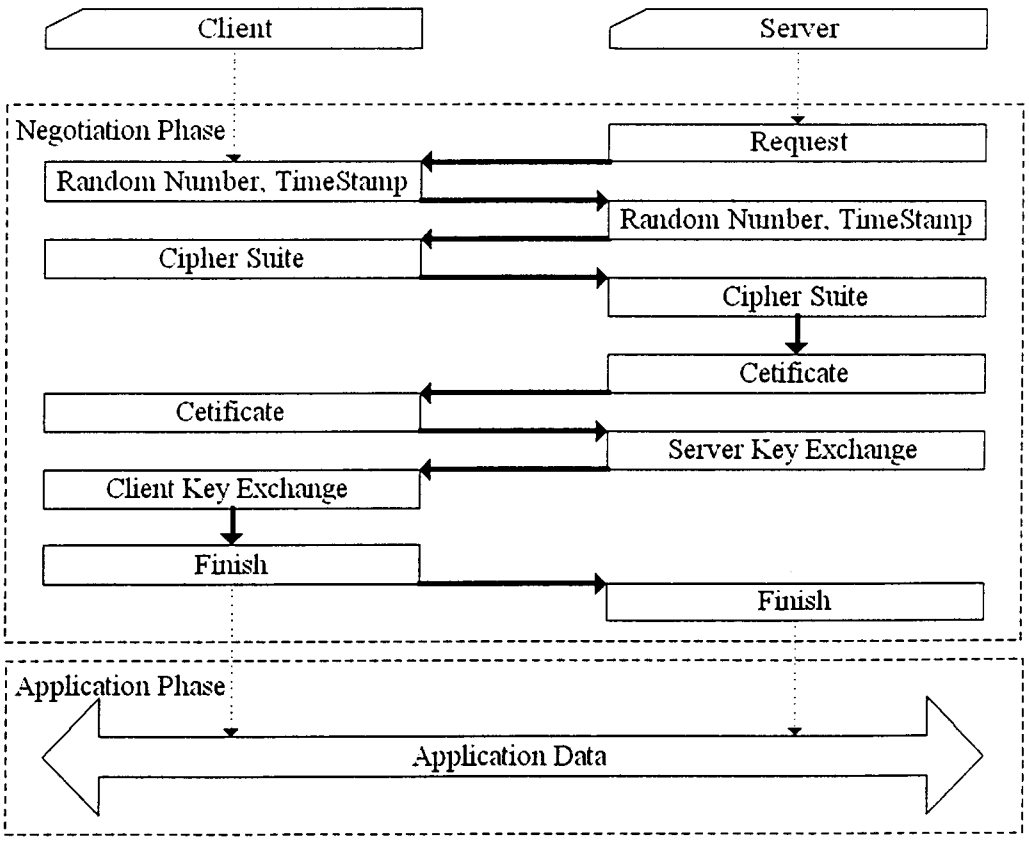


19. X.509三向驗證程序

ClientCryptographicParameters送出一個random(clienthello.random)，並且由Server端在ServerCryptographicParameters送出一個radom(serverhello.random)，並沒有包括A的Timestamp和B的Timestamp，無法防止重送攻擊，並不具有雙向認證的功能。

3.1.2. 具雙向認證的 SSL互握通訊協定

為了加強SSL缺乏雙向認證的缺失，本計畫將修正SSL協定並提出另一socket層的傳輸協定。其協定結構如下圖：



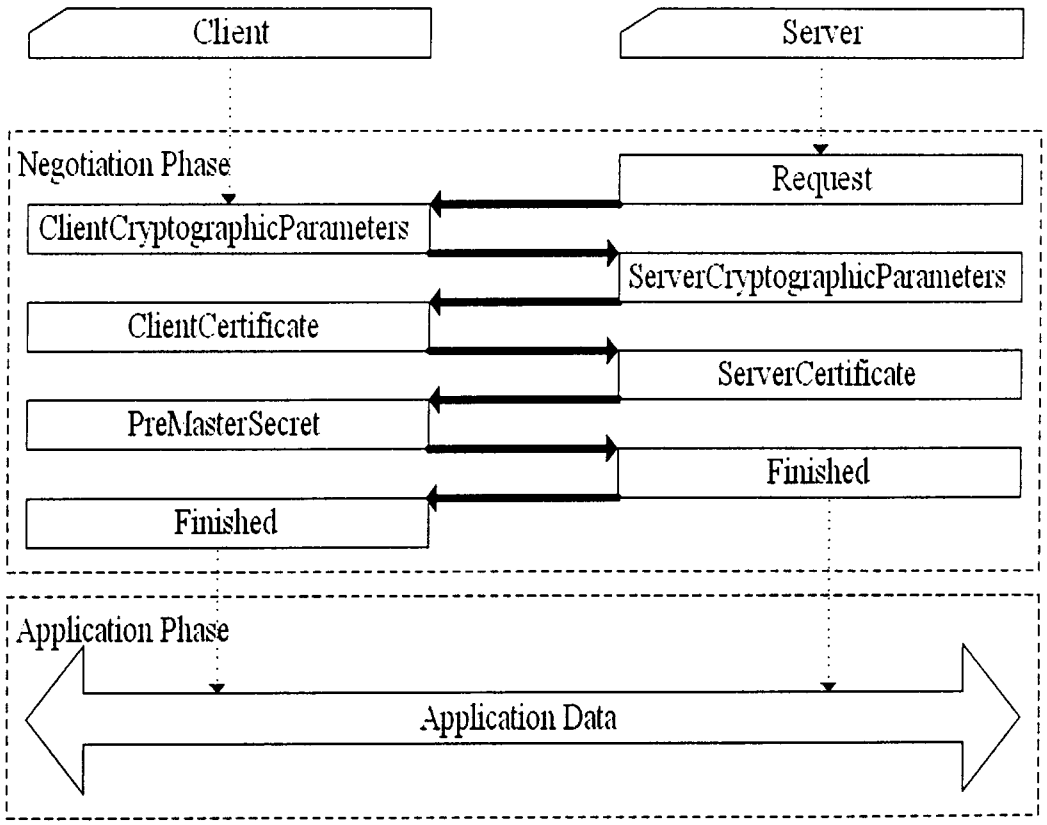
21. 具雙向認證的SSL互握通訊協定

3. Authenticated Telnet

3.1. 改良的 SSL 互握通訊協定

3.1.1. SSL 互握通訊協定的缺點

在建立本系統時，我們必須透過相關網路安全協定以達到資料安全傳輸的目的，SSL就是一個最廣為使用的協定，其協定結構如下圖：



20. SSL互握通訊協定

但是從 2.3.1.中所提到的 X.509 憑證驗證方法中，我們可以知道在 SSL 互握通訊協定中，僅由 Client 端在

與原本SSL互握通訊協定最大的不同點在於在這個改良的協定當中，我們在一開始 Client 送給 Server 的訊息當中加入了Random Number和 Timestamp，而 Server 回給 Client 的訊息當中也加入了 Random Number和Timestamp，而藉由此種做法來完成互相驗證彼此的身分，並且彼此交換要給對方的秘密資訊，以達到雙向驗證的，本計畫將此協定來實做Authenticated TELNET系統。

3.2. 改良的SSL MAC—NMAC

3.2.1. MAC的分析

當我們使用這些有名的密碼方法像DES、GOST、SAFER-K64和雜湊函數MD5、SHA和RIPE-MD時，Roe 計算出在軟體實際完成這些 MAC 組合的性能表現，如下表所示：

	DES/MD5	DES/SHA	DES/ RIPE-MD	GOST/MD5	GOST/SHA	GOST/ RIPE-MD	SAFER/ MD5	SAFER/ SHA	SAFER/ RIPE-MD
CBC	565	565	565	362	362	362	136	136	136
HMAC	18	25	22	18	25	22	18	25	22
NMAC	18	25	22	18	25	22	18	25	22
$E_k(H(X))$	18	25	22	18	25	22	18	25	22
$E_k(H_{k'}(X))$	18	25	22	18	25	22	18	25	22
$H(E_k(X))$	583	590	587	380	387	384	154	161	158
$H_{k'}(E_k(X))$	583	590	587	380	387	384	154	161	158

22. 各種MAC的比較表(處理一個 1 Mbit 的訊息.作業平台是DEC 3000; 單位是1/1000秒)

如同Figure 22所顯示的，在一個雜湊之後再做加密運算的 Cipher-MAC 和 Hash-Cihper-MACs 比其他的方法慢很多。

在一個加密運算後再做雜湊的 Hash-MACs 和 Hash-Cipher-MACs 是比較有效率的(對處理較長的訊息而言)。儘管 Hash-MACs 的安全是建立在某些基本的雜湊函數的假設，我們對建立在某些基本的雜湊函數和加密兩者的有效率的 Hash-Cipher-MACs 更感興趣。參考$E_k(H(X))$ 和 $E_{k1}(H_{k2}(X))$，我們將在下一節建立一個新的 Hash-Cipher-MAC 組合。

3.2.2. Non-deterministic MAC

其中一個方法是在雜湊之前在訊息中加入一個亂數(Random Number)，以 R 表示，因此同樣的訊息在不同時候會因為所選擇的 R 不同而不一樣。為了能夠讓雜湊的值可以被確認， R 會被加到從他產生出雜湊的值當作後繼的加密演算法的輸入，在我們的方法中一個訊息的MAC是 non-deterministic，或者簡短的說 NDMAC，而它可以以公式$NDMAC_k(X)=E_k(R \| H(R \| X))$來表示，而R是一個固定長度的亂數。在NDMAC中的 $R \| H(R \| X)$ 也可以算是X的中間雜湊函數值。

很顯然地接收者可以從秘密金鑰k和(MAC,X)來確認$MAC=NDMAC_k(X)$，經由以下的敘述：

1).對MAC解密得到中間的雜湊函數值 $R \| H(R \| X)$

2).把亂數 R 和雜湊函數值 h 從中間的雜湊函數值中取出

3).接收者計算 $H(R \| X)$和檢查是否$H(R \| X)=h$來確認正確

3.2.3. 安全分析

使用區塊加密演算法，我們提出的方法有下列幾個特性：

令$MAC(X)=E_k(Y)$

$Y= R \| H(R \| X)$

r：R的長度

k：k

h：$H(R \| X)$

1).給定 X 和 MAC(X)，如果 r 夠大，不管是從 X 或 MAC(X)，如果沒有秘密金鑰 k ，入侵者很難算得出中間的雜湊函數值Y。Y的預測值的數目是$MAX\{2^{-r}, 2^{-k}\}$。這是說入侵者在這個區塊加密演算法下很難蒐集到plaintext-ciphertext對。

2).給定 X，雖然入侵者可以造出一個合法的中間函數值$Y'=R' \| H(R' \| X)$，但是他因為他不知道秘密金鑰 k ，所以並不能夠產生一個合法的X的MAC值。

3).只有當兩個訊息有同樣的中間雜湊函數值時，他們才有一樣MAC值。因此，給定 X 和 MAC(X)，如果使用的雜湊函數是抗碰撞的，對入侵者來說是很難找到另一個訊息X'可以使$H(R' \| X)=H(R \| X)$和$MAC(X')=MAC(X)$。事實上，如果雜湊函數隨機程度夠強，MAC(X)是一個合法的X'的MAC的機率是X^{-h}。

4).即便雜湊函數並不是抗碰撞的，入侵者仍然很難偽造一個訊息，舉例來說，令X_1 和 X_2 是兩個不同的訊息但是在同一個雜湊函數H( )下卻有同樣的雜湊函數值。$H(R \| X_1)$不一定會等於$H(R \| X_2)$。因此，X_1的 MAC 不一定是 X_2合法的MACs。

5).即使雜湊函數是單向的，也就是說，給定一個雜湊函數值Y，找到一個明文X'使得H(X')=Y是有可能的，但是對於一個入侵

者來說給定 X 和 MAC(X) 來偽造一個訊息仍然很困難。理由是中間的雜湊函數值是隱藏的。

如果入侵者不知道區塊加密演算法的秘密金鑰，這個所提出的 MAC 方法是安全的。然而入侵者還是可以試著去破解。入侵者很難用 known/chosen text 攻擊去破解這個區塊加密演算法，因為他不知道這個加密演算法的輸入值(中間的雜湊函數值)。另一方面，如果這個加密演算法在做所有金鑰搜尋時是易破的，入侵者有可能找到秘密金鑰。這種入侵法叫做中途攻擊(meet-in-the-middle-attack)：入侵者首先計算並儲存對於一個給定的訊息 X 所有2^r個可能的中間雜湊函數值 $R \| H(R \| X)$。然後他用所有2^k個可能的金鑰解相對應的 MAC來和之前得到中間雜湊函數值做比對。如果比對一樣，這個解密的金鑰很有可能就是原本的秘密金鑰，因此，為了防範中途攻擊，不是選擇一個做所有金鑰搜尋時是不易破的加密演算法，就是長度夠長的亂數位元(random bits)。

簡單的比較我們的方法和 $E_k(H(X))$和$E_k(H_{k'}(X))$ 的MAC(兩種都是有效率)：對安全度來說，$E_k(H(X))$在H( )有碰撞問題時是易破的，它比我們的方法要弱很多。$E_k(H_{k'}(X))$和我們的方法有相似的特性，但是它必須要保留兩把金鑰，金鑰的管理負荷增加。對效能來說，我們的方法會因為加入一個亂數值到MAC而慢一點，這需要一個額外的亂數產生器和可能的雜湊壓縮函數和區塊加密演算法。在我們的方法中，亂數產生器會比雜湊壓縮函數要花較少的時間，因此，在訊息長的時候增加的虛耗是很小的，我們的方法仍然是有效率的。

MAC產生的步驟如下：

Step1. 亂數產生器產生兩個亂數R_1, R_2

Step2. R_1, R_2 與訊息M接合成 $R_1 \| M \| R_2$

Step3. $R_1 \| M \| R_2$ 通過SHA-1壓縮

Step4. 取出前 $l_b/2$ 位元為ΔSHA-1($R_1 \| M \| R_2$)

Step5. 與R_1, R_2 接合成 $R_1 \| \Delta\text{SHA-1}(R_1 \| M \| R_2) \| R_2$

Step6. $R_1 \| \Delta\text{SHA-1}(R_1 \| M \| R_2) \| R_2$ 以 k 通過加密演算法後得到MAC

註: l_b：加密演算法輸入的區塊長度，事前確定

MAC驗證的步驟如下：

Step1. MAC 以 k 通過解密演算法之後得到

$R_1 \| \Delta\text{SHA-1}(R_1 \| M \| R_2) \| R_2$

Step2. $R_1 \| \Delta\text{SHA-1}(R_1 \| M \| R_2) \| R_2$ 通過語法分析得到R_1、R_2與ΔSHA-1($R_1 \| M \| R_2$)

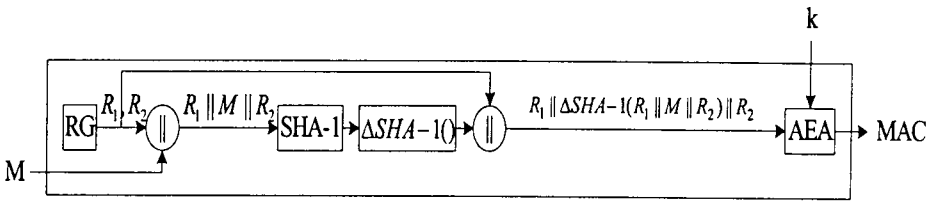
Step3. R_1、R_2與M接合成$R_1 \| M \| R_2$

Step4. $R_1 \| M \| R_2$經過SHA-1與ΔSHA-1()後得到

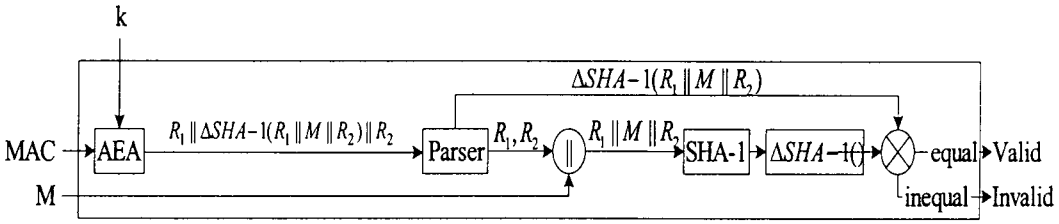
$\Delta\text{SHA-1}(R_1 \| M \| R_2)$

Step5. 將步驟2與步驟4得到的ΔSHA-1($R_1 \| M \| R_2$)做一個比對即可驗證MAC的正確性

MAC的產生與驗證的結構如下圖：



l_b: the block size of AEA, pre-determined.
$\Delta SHA-1()$: to get the first $l_b/2$ bits of the SHA-1's result.
RG: random number generator, to generate random numbers with length equal to $l_b/4$.
AEA: to encrypt message with pre-determined block size and the shared key.



l_b: the block size of AEA, pre-determined.
$\Delta SHA-1()$: to get the first $l_b/2$ bits of the SHA-1's result.
Parser: to parse the data with format: $R_1 \| \Delta SHA-1(R_1 \| M \| R_2) \| R_2$
AEA: to encrypt message with pre-determined block size and the shared key.

23. NMAC結構

4. SSLTelnet 使用手冊

4.1. SSLTelnet Server

4.1.1. 簡介

SSLTelnet Server 是在UNIX 工作站上，以C語言所寫成之伺服器。可使用inetd來呼叫，或是以standalone daemon方式來啟動。

4.1.1.1. 安裝步驟

作業系統版本： FreeBSD 4.7-RC

Step0. 需具有 root 權限

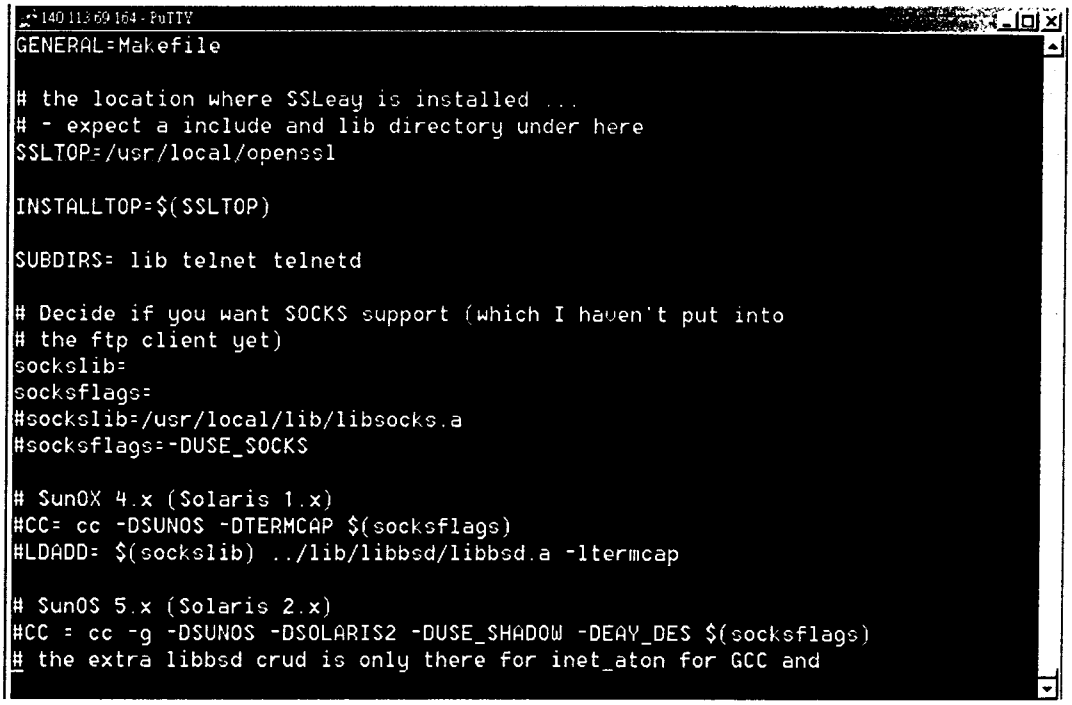
Step1. 安裝OpenSSL Library.

Step2. 將 SSLTelnet.tar.gz 在欲安裝的目錄下解開

#tar xvfz SSLTelnet.tar.gz

Step3. 修改 Makefile

將 SSLTOP 參數改成 OpenSSL Library 的位置（如：/usr/local/openssl)

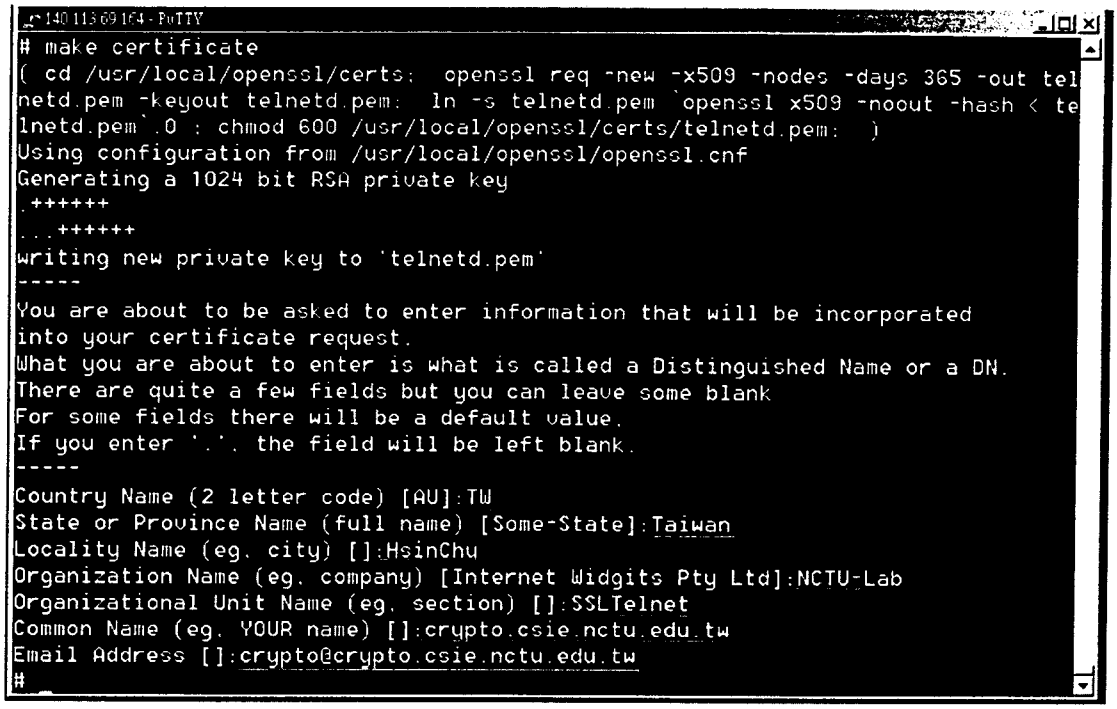


Step4. 解開目錄後，安裝方式：

#make ; make install;

Step5. 產生憑證

在make install 的最後步驟，會要求使用者產生憑證，必須手動鍵入憑證所需資料，此處會以問答方式呈現，依需求填入即可。憑證預設會放於 /etc/ssl/certs/telnetd.pem。如不想使用系統預設憑證，也可將自己的憑證置於 /etc/ssl/certs/ 底下，並將憑證改名為telnetd.pem



Step6. 檢視執行檔

以上步驟所產生的伺服器端執行檔 telnetd，將其複製到系統目錄，/usr/local/libexec/ 底下

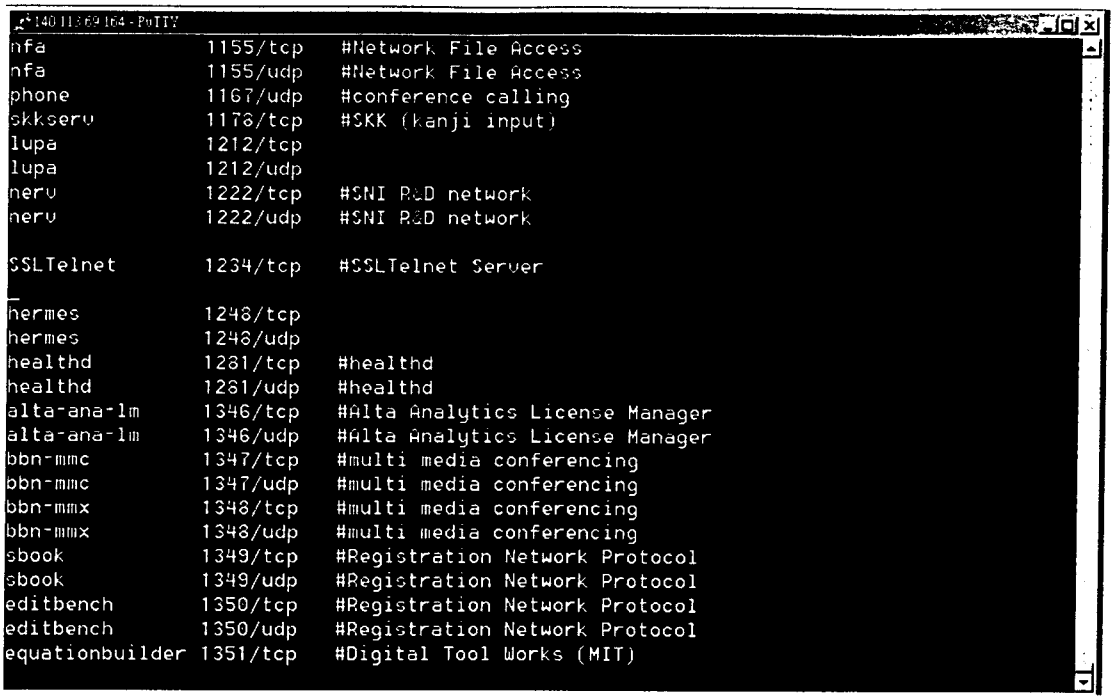
Step7. 啟動伺服器：

1).以 inetd 來呼叫：

a).修改 /etc/services，加入一行

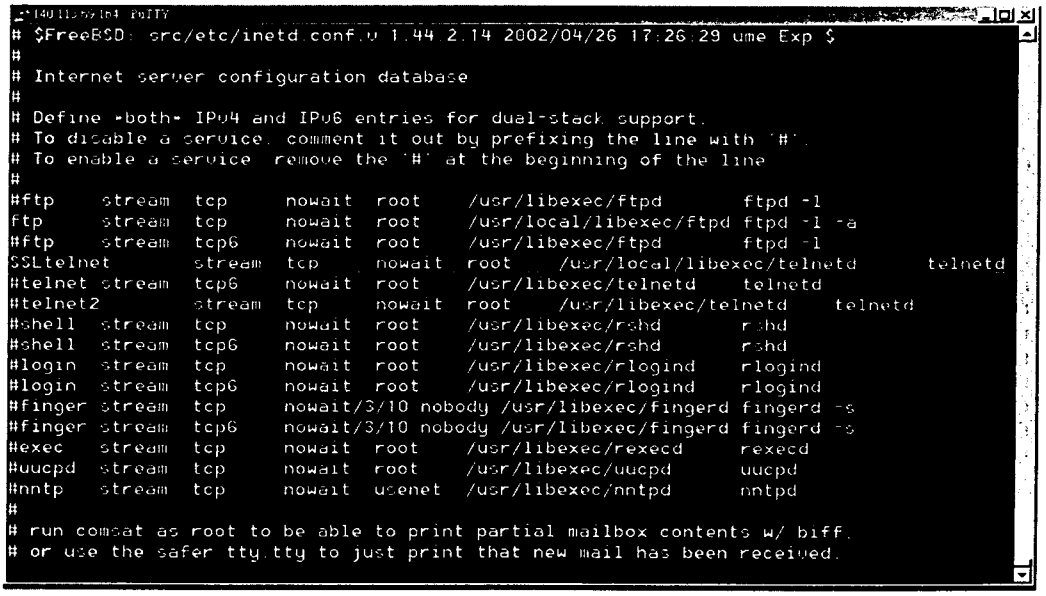
SSLTelnet 1234/tcp #SSLTelnet Server

其中 1234 是 port 號碼，可改成其他數字。



b).修改 /etc/inetd.conf，加入一行

SSLTelnet stream tcp nowait root /usr/local/libexec/telnetd\ telnetd



c).重新啟動 inetd

killall –1 inetd

2).以 stand alone 方式來啟動：

/usr/local/libexec/telnetd

4.1.1.2. 參數說明

除了以上所提到的一些SSLTelnet Server的參數外，還有一些額外的的參數可以使用，列出於下：

-z ssl

指定SSLTelnet Server只使用SSL protocol和Client溝通。

-z secure

當SSLTelnet Server和Client做SSL protocol handshake失敗時，立即停止連線而不會繼續執行。

-z verify=level

指定SSLTelnet Server確認Client的certificate的程度：

0 SSLTelnet Server不會要求Client的certificate。

1 SSLTelnet Server要求Client的certificate並且做認證，若certificate認證失敗，連線尚可繼續下去。

2 SSLTelnet Server要求Client的certificate並且做認證，若certificate認證失敗，連線則中斷。

-z cert=certfile

指定Server的certificate檔案。

-z key=keyfile

指定Server的RSA private key檔案。

-z cipher

指定SSLTelnet Server SSL protocol cipher suite的優先順序。

4.2. SSLTelnet Client

4.2.1. 簡介

SSLTelnet Client是用MS Visual C++ 6.0利用WIN32 API所compile而成，可於Win95、Win98、WinNT、Win2000和WinXP上執行。

4.2.2. 安裝步驟

Step1：

在硬碟中建立SSLTelnet Client的目錄（以C:\Program Files\SSLTelnet為例），將SSL_Telnet.zip解開放於此目錄中。

SSLTelnet.zip包含以下檔案：

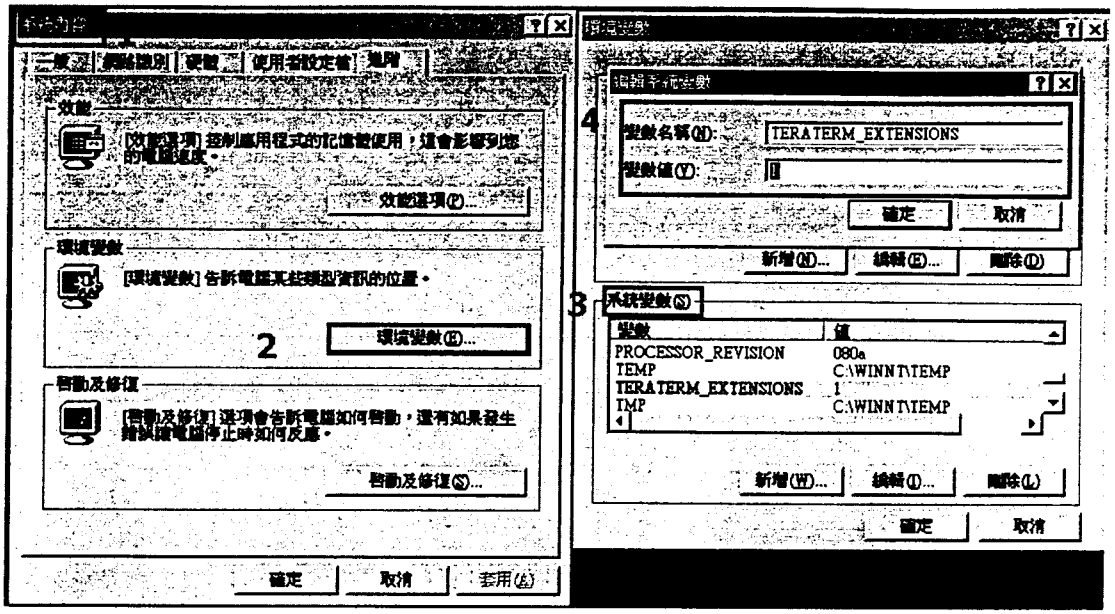
ssltelnet.exe	// SSLTelnet主程式
SSLTELNET.ini	// SSLTelnet設定檔
ttpcmn.dll	// SSLTelnet common routine檔
ttpdlg.dll	// SSLTelnet dialog功能檔
ttpfile.dll	// SSLTelnet file功能檔
ttpset.dll	// SSLTelnet setup功能檔
libleay32.dll	// Cryptographic algorithm library
ssleay32.dll	// SSLeay library
KEYBOARD	//鍵盤對應檔

Step2：

將所有檔案放入目錄中後（缺一不可），接著設定一環境變數TERATERM_EXTENSIONS為1。底下以Win2000為例：

我的電腦(按右鍵)→內容→進階→環境變數下

在系統變數按新增，設定變數名稱：TERATERM_EXTENSIONS，變數：1

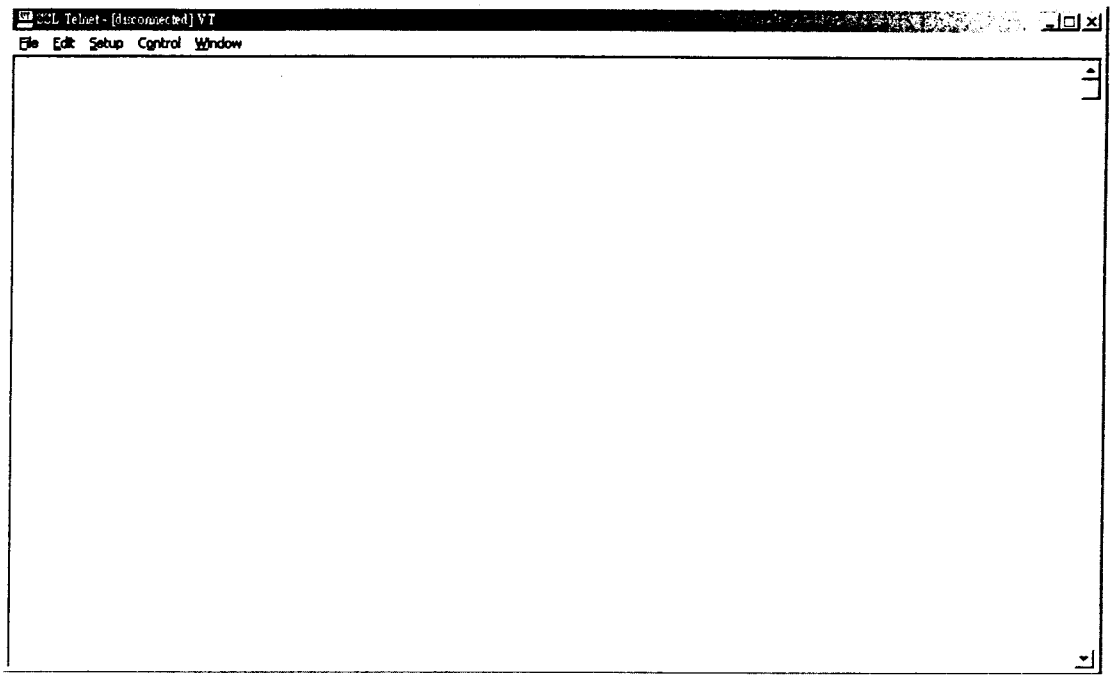


Step3：

啟動ssltelnet.exe，ssltelnet.exe即為Client用來和Server端連線的應用程式。

4.2.3. 使用說明

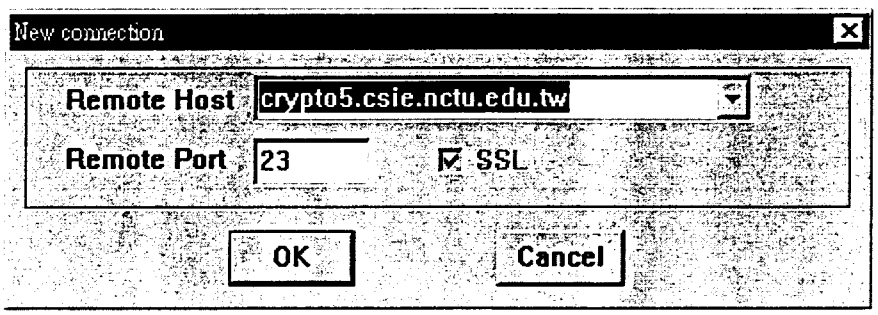
下圖為SSLTelnet的主要視窗，所有由Server端送出的ASCII文字接會顯示在這個視窗的client area中。client area右邊有一個scroll bar，可在INI檔案中（稍後會說明INI檔的內容）設定上捲的行數，並利用此scroll bar可顯示已上捲的資料。



此視窗包含了一個主選單，它們的功能和作用我們將一一解釋：

[File] -> [New Connection] or Alt+N

連線到一個新的Server（可為SSL或非SSL Telnet Server）。選了這個選項後會出現下面這個視窗，可讓使用者選擇遠端的主機名稱或是IP Address和Port。有一個標記為SSL但無法選擇的小框框，預設為SSL，如果Server端有啟動SSL功能，則為SSL telnet連線；反之則為一般的telnet連線。



[File] -> [Disconnect]

此選項表示要和Server端中斷連線。若已在連線中，會出現要使用者確認的Message Box。

[Edit] -> [Copy] or Alt+C

將主視窗中client area中用滑鼠左鍵標記出的文字Copy到剪貼簿中。剪貼簿中的資料可Paste於其它的應用程式中，或者利用此連線程式 [Edit] -> [Paste] 功能來送出資料。

[Edit] -> [Paste] or Alt+V

若有利用 [Edit] -> [Copy] 將某段文自Copy至剪貼簿中，或是別的應用程式有將文字資料Copy至剪貼簿中，此功能選項將把剪貼簿的內容，當做是使用者輸入的文字送到Server端。

[Edit] -> [Clear screen]

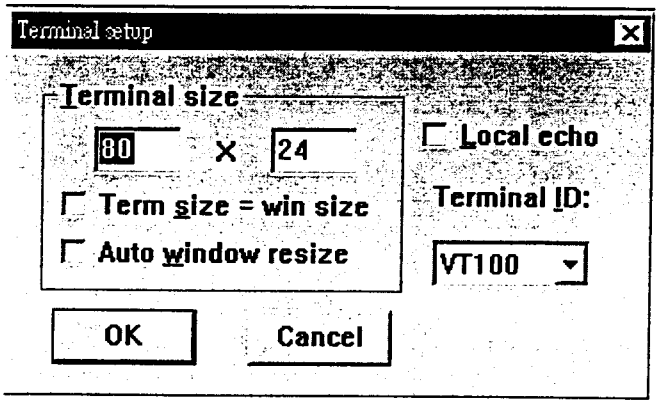
此功能會將主視窗的client area顯示區清除，並將游標移至client area左上方。

[Edit] -> [Clear buffer]

此功能將scroll bar的內容清除。並將游標移至client area左上方。

[Setup] -> [Terminal Setup]

按下此選單之後，會出現下面這個視窗：



在這個Terminal Setup視窗中可選和主視窗顯示有關的功能：

- Terminal size

可選擇主視窗顯示文字的行數和列數。當 "Term size = win size" 選下時，當我們改變主視窗大小時，會自動將顯示的行數和列數計算出來。當 "Auto window resize"選下時，當我們改變主視窗大小時，視窗行數和列數顯示大小並不會變更，但主視窗顯示區會依照須要出現水平的scroll bar，我們可利用垂直和水平的scroll bar來看到完整的顯示資料。

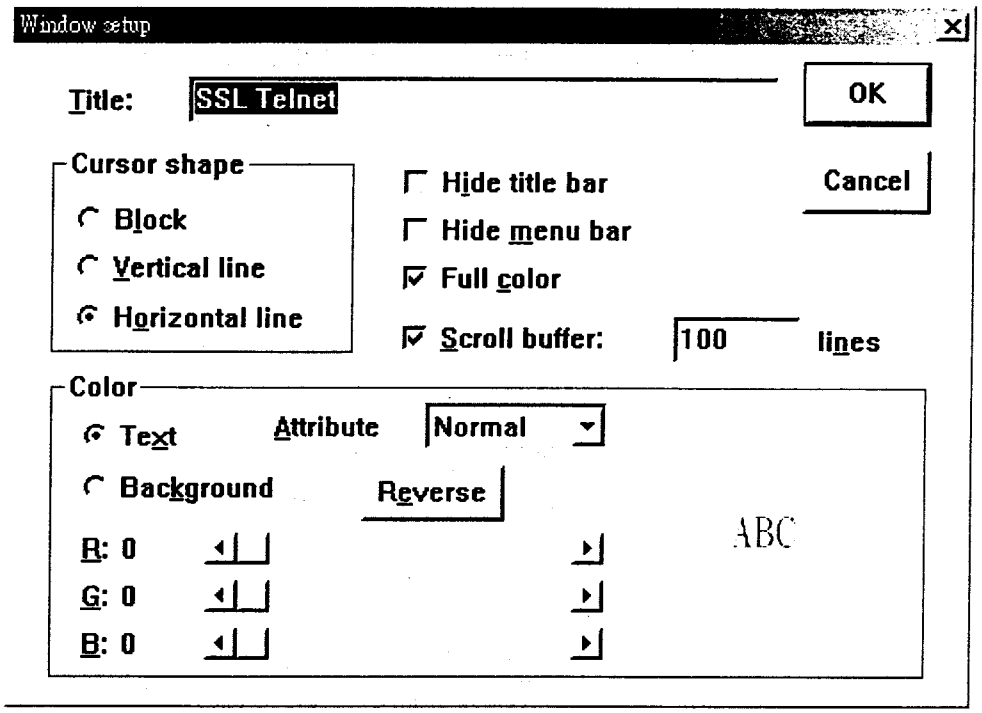
- Local echo

當local echo被選擇後，使用者所輸入的文字在Client端亦會顯示出來；也就是說，有可能一個ASCII code會在主視窗client area中顯示兩次（一次是Client自己印出來的，另一為Server所傳送回來的。）

- Terminal ID

使用者可依需要用SSLTelnet Client來模擬終端機型態。（有VT100, VT101, VT102, VT282, VT320, VT320這些可選。）

[Setup] -> [Window Setup]



Windows Setup主要是給使用者選擇主視窗顯示的一些屬性：

- Title

可將主視窗的標題(caption)改成使用者所喜歡的文字。

- Cursor shape

這個功能可用來改變主視窗顯示游標的形狀。有方塊狀，橫線和直線三種可選擇。

- Hide title bar

將主視窗的標題列隱藏起來。

- Hide menu bar

將主視窗的選單列隱藏起來。

- Full color

有時使用者會連線到BBS等利用ASNI code來顯示的Server端時，將full color這個選項選下後，ANSI code

就可正常顯示，使得顯示畫面更出色更生動。

- Scroll buffer

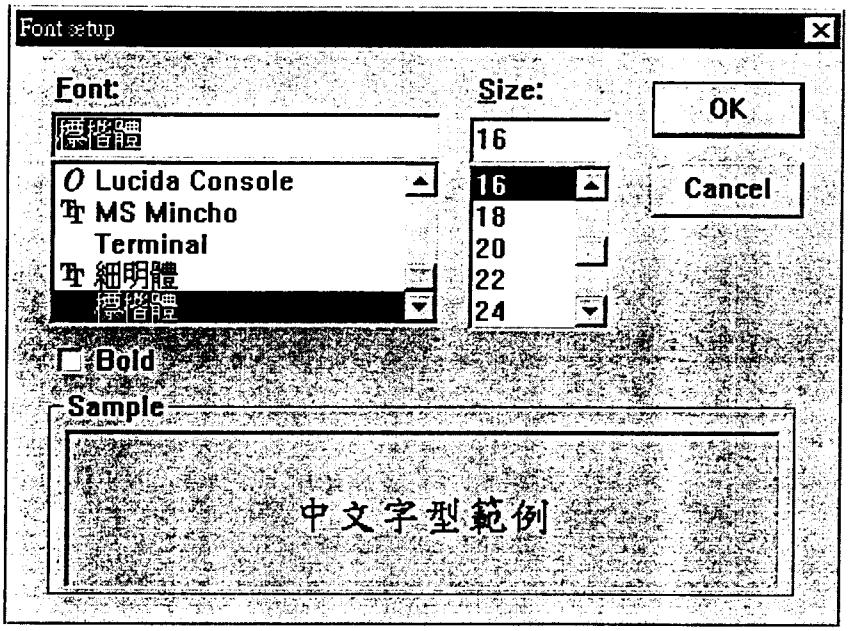
改變垂直scroll bar的scroll buffer大小。範圍限制在terminal vertical size(通常為24) 到10000行之間。

- Color

可改變主視窗顯示文字和背景的顏色。一般文字，一般背景，粗體文字，粗體背景，閃動文字，閃動背景等六種不同的屬性皆可分別指定其RGB顏色值。

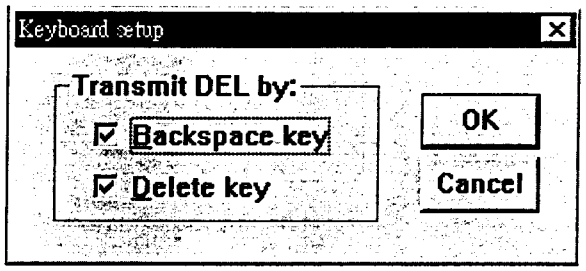
[Setup] -> [Font Setup]

此功能用來改變主視窗顯示的字型和大小或是粗體。

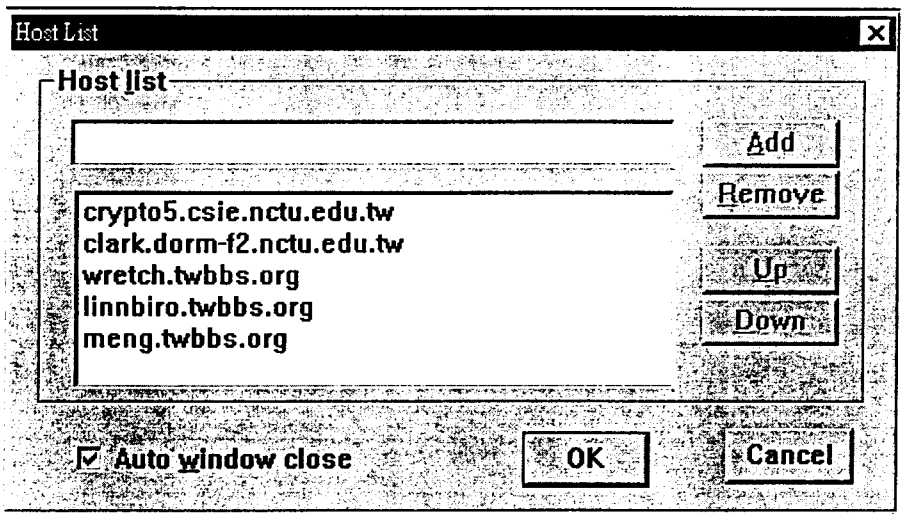


[Setup] -> [Keyboard Setup]

一般而言，當使用者按下鍵盤上的Backspace時，會送出^H(CTRL+H)到Server端，表示要倒退一個字元。但是不同的Server，不同的service port所定義的倒退字元並不相同。為了考慮這種差異性，當使用者發現無法以鍵盤上送出Server端所定義的倒退字元時，利用這個選項可選擇用Backspace或Delete來送出正確的倒退字元。



[Setup] -> [Host List]



Host list視窗讓使用者來編輯遠端機器列表，列表中的機器名稱可為host name或是IP Address。使用者也可以右邊的上下功能鍵來重新整理列表中機器名稱的順序。Auto window close這個選項若打勾，則當使用者和遠端機器連線中止後，主視窗便會關閉

；反之，若這選項若沒選，主視窗不會因為連線中止而關起來，使用者還可利用這個視窗和其它機器溝通。

[Setup] -> [Save Setup]

以上所有 [Setup] 的選項若有更動，必須要用Save Setup來儲存更改後的設定，以便下一次程式再執行時這些Option還能生效(儲存在SSLTELNET.INI檔)。

[Setup] -> [Restore Setup]

有時不同的使用者有他們自己不同的喜好設定，他們可能會有他們自己的設定檔。利用Restore Setup可以讓程式去讀取不同的設定檔。

[Control] -> [Reset Terminal]

這個功能會將游標重新設定到最左上角，但並不會清除主視窗顯示區域。

[Control] -> [Are You There?]

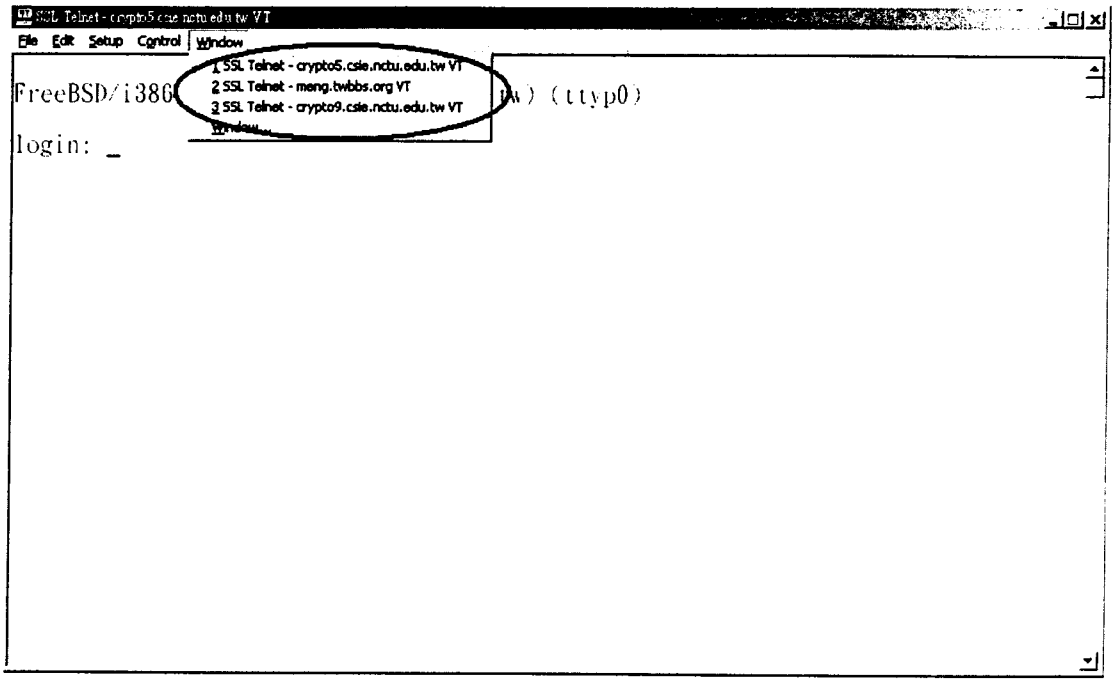
送出Telnet Protocol所定義的Are You There命令字串。

[Control] -> [Send Break]

送出Telnet Protocol所定義的Break命令字串。

[Window]

這個選項會列出所有SSLTelnet Client的視窗，如下圖：



使用者可利用這個功能切換到其它的SSLTelnet Client視窗。

使用者手動編輯INI設定檔(SSLTELNET.INI)

上面各個選項改變儲存後會將改變的值寫入SSLTELNET.INI這個檔案中，每當應用程式一開始執行，會讀取此INI檔做為程式內一些選項變數的初始值。除了用上面的方法可以改變設定值外，使用者也可手動來更改。檔案內各欄位說明如下：

```
VTPos=xx,yy                    // 視窗程式的起始位置
TerminalSize=xx,yy             // 視窗程式的行數列數大小
TermIsWin=on or off            // 同 [Setup] -> [Terminal Setup] 中的term size = win size選項
AutoWinResize=on or off        // 同 [Setup] -> [Terminal Setup] 中的auto window rezie選項
LocalEcho=on or off            // 同 [Setup] -> [Terminal Setup] 中的local echo選項
TerminalID=terminalID          // 同 [Setup] -> [Terminal Setup] 中的terminal ID選項
Title=string                   // 同 [Setup] -> [Window Setup] 中的title選項
CursorShape=block or vertical or horizontal
                               // 同 [Setup] -> [Window Setup] 中的cursor shape選項
FullColor=on or off            // 同 [Setup] -> [Window Setup] 中的full color選項
```

EnableScrollBuff=on or off // 同 [Setup] -> [Window Setup] 中的scroll buffer選項

ScrollBuffSize=number // 同 [Setup] -> [Window Setup] 中的scroll bufferlines選項

VTColor=200,200,200,0,0,0

VTBoldColor=255,255,255,0,0,0

VTBlinkColor=255,0,0,0,0,0 // 同 [Setup] -> [Window Setup] 中的color選項

VTFont=8514fix,0,-20,136 // 同 [Setup] -> [Font Setup] 中的font選項

EnableBold=on or off // 同 [Setup] -> [Font Setup] 中的bold選項

BSKey=DEL // 同 [Setup] -> [Keyboard Setup] 中的del選項

DeleteKey=on // 同 [Setup] -> [Keyboard Setup] 中的del選項

AutoWinClose=on or off // 同 [Setup] -> [Host List] 中的auto win close選項

Historylist=on or off // 是否自動更新host list的順序和內容

Beep=on or off // 視窗是否會發出beep聲

BeepOnConnect=on or off // 連線成功後是否發出beep聲

ConfirmDisconnect=onor off // 是否確認結束連線

DelimList=$20!"#$24%&'()*+,-./:;<=>?@[\]^`{|}~

// 字元與字元的分介符號

DelimDBCS=on or off // 字元與字元的分介符號是否包含DBCS

MaxBuffSize=10000 // scroll buf的最大行數值

NonblinkingCursor=on or off // 游標是否閃爍

ScrollThreshold=12

SelectOnActivate=on or off

TelBin=on or off // Telnet option binary初始值

TelEcho=on or off // Telnet option echo初始值

WindowMenu=on or off // 是否在選單出現Window這個選項供使用者切換至

// 其它SSLTelnet Client視窗

SSLFlag=on or off // 是否使用SSL來連線

SSLCipher="ssl_cipher" // SSL所使用的密碼方法優先權列表

SSLCipher可用的字串:

"DES", "RC4", "RC2", "IDEA", "MD5", "SHA", "SSLv2", "SSLv3", "ALL",

"NULL-MD5", "NULL-SHA", "EXP-RC4-MD5", "RC4-MD5", "RC4-SHA",

"EXP-RC2-CBC-MD5", "IDEA-CBC-SHA", "EXP-DES-CBC-SHA", "DES-CBC-SHA",

"DES-CBC3-SHA",....等。

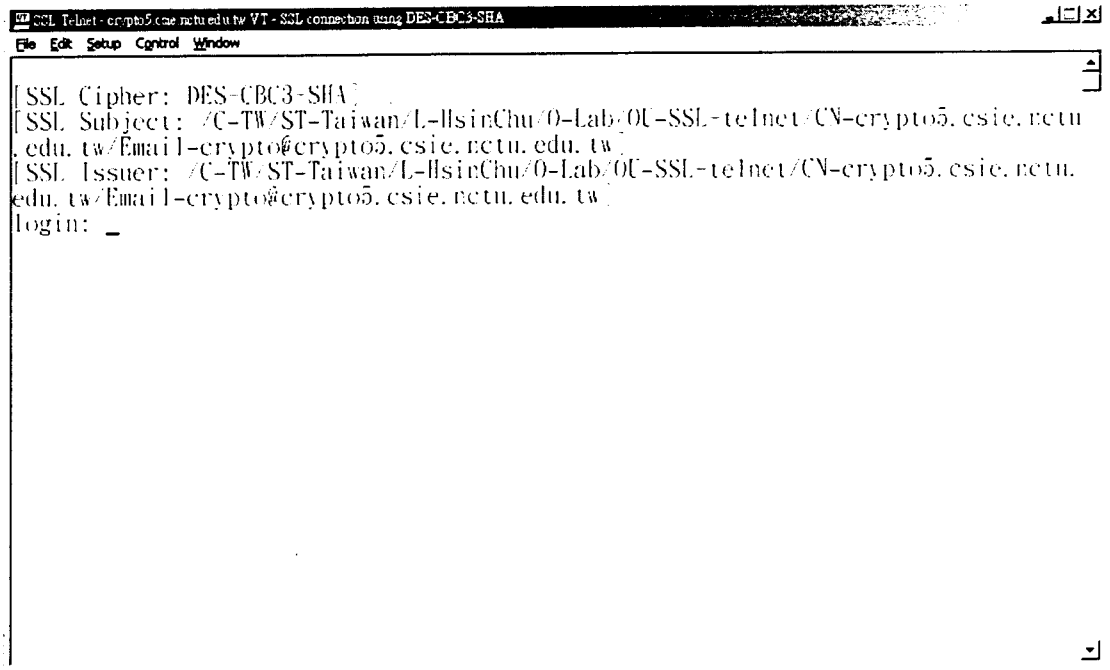
每一字串之間用 ":" 區隔，如 "RC2:RC4-MD5:DES-CBC3-SHA"

Host1=host name or IP address // 遠端機器列表

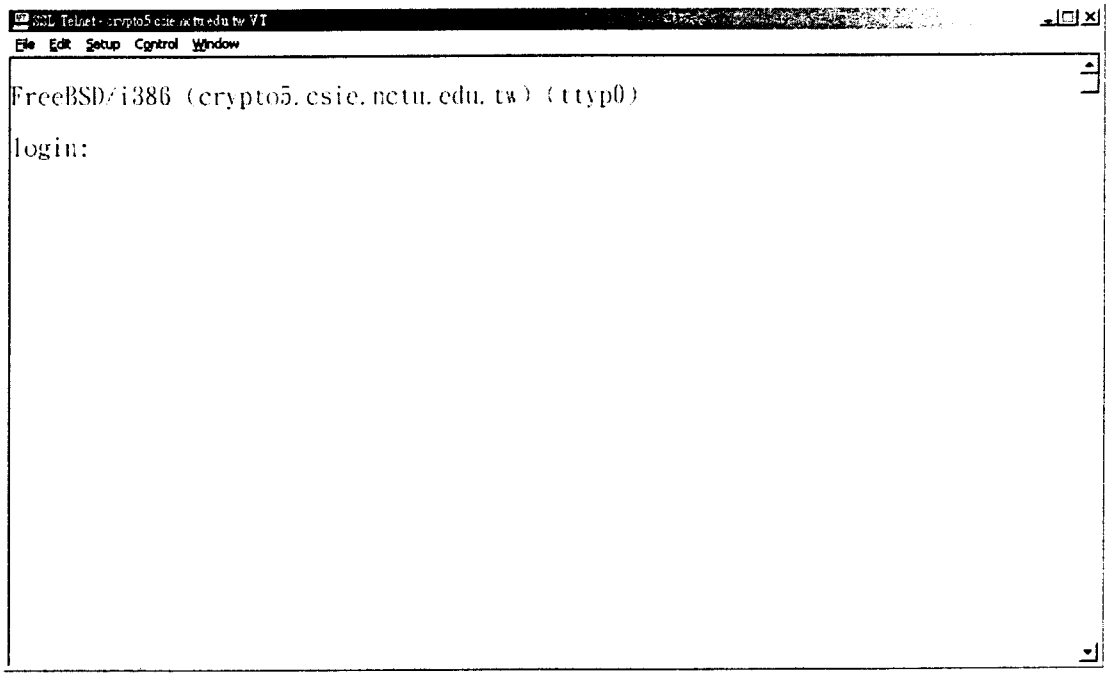
Host2=host name or IP address

Host3=host name or IP address

連線至SSLTelnet Server主機



[Server支援SSL功能]



[Server不支援SSL功能]

上面這兩張圖中上面這張是利用SSLTelnet Client連線到SSLTelnet Server時的畫面，下面這張是SSLTelnet Client連線到一般不支援SSL功能的Server的畫面。從視窗的標題列可以分辨出Server是否支援SSL的功能。連線到SSLTelnet Server且設定啟動SSL功能時，標題列字串中最後會出現 "SSL connection"；反之，若連線至一般Server或設定不啟動SSL功能時，標題列則不顯示任何字串。

當連線至SSLTelnet Server成功後，視窗顯示區會馬上出現SSL所用的CipherSpec，SSLTelnet Server的certificate的subject和issuer。在這個例子中，這三項分別為：

- [SSL cipher: DES-CBC3-SHA]
- [SSL Subject: /C=TW/ST=Taiwan/L=HsinChu/O=Lab/
  0U=SSSL-telnet/CN=crypto5.csie.nctu.edu.tw/
  Email=crypto5.csie.nctu.edu.tw]
- [SSL issuer: /C=TW/ST=Taiwan/L=HsinChu/O=Lab/
  0U=SSSL-telnet/CN=crypto5.csie.nctu.edu.tw/
  Email=crypto5.csie.nctu.edu.tw]

當這些information出現以後，表示SSL protocol的handshake已完成，在這以後所有Server和Client所傳輸的資料皆會經過安全處理（包括使用者的名字和密碼），任何網路上的第三者跟本無法知到這些傳輸的資料。

4.2.4. 程式架構

Source Code：src.zip(Src.zip內含source和visualc兩個目錄)

此程式開發環境為MS Visual C++6.0。Workspace內主要包含了五個projects，其中一個為執行檔的project，另外四個為自定的工具函數動態連結檔(DLL)。把一個程式分為多個project的好處是可以縮短執行檔compile的時間，一些已寫好的功具函數設計成DLL型態可簡少執行檔在系統中佔記憶體的大小，因為並不是所有的功具函數皆會同時用到，只要需要時再load進來就好了。

這五個主要的project所包含的範圍為下：

ttermpro應用程式函數，視窗程式函數。

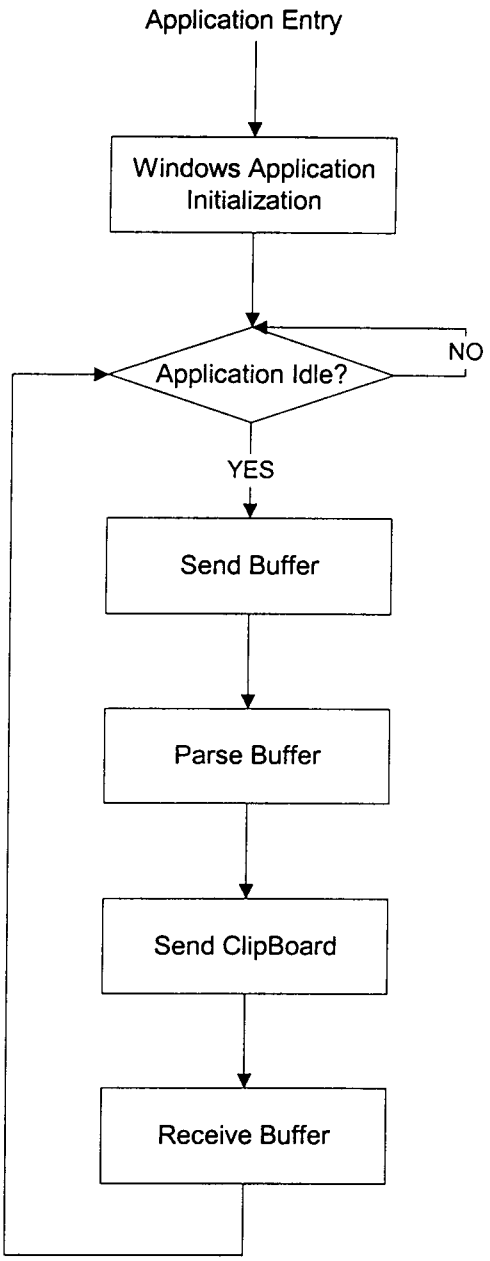
ttpcmn資料傳輸介面函數。

ttpdlg對話方塊函數。

ttpfile檔案讀寫函數

tpset應用程式setup函數。

SSLTelnet Client主要流程圖：



24. SSLtelnet Client 程式主要流程圖

當程式一執行後，在應用程式初使化的動作完成後，會跳入一迴圈中來處理所有的網路輸入輸出和輸入資料的分析。這迴圈會判斷應用程式是否處於idle狀態，若是，則先送出輸出buffer(使用者用鍵盤key in的資料)至Server，然後處理分析判斷輸入buffer內的資料，是一

般文字還是Telnet protocol的command sequence？接著，若使用者有將clip board內的資料送出，會執行送出clip board的程序。此迴圈最後一步驟是讀取從Server端送出的資料，讀取後將資料放入輸入buffer中。至此，迴圈內四個程序皆執行後，此迴圈又周而復始地無限執行，直到應用程式結束為止。

由於Visual C++ MFC視窗程式的架構，應用程式和視窗程式是兩個有關係但不同的部份，剛剛所談到的流程圖是應用程式方面的討論，至於視窗程式的架構就複雜多了。視窗程式主要用來處理使用者鍵盤輸入，滑鼠移動，視窗顯示區域重繪和選單選項等有關主要視窗的處理部份。由於MFC對Windows採event-driven的型態，每當使用者對視窗有所動作，在視窗程式接收到系統送來的message後，根據程式中對每個message處理方式來執行其對應的code。BEGIN_MESSAGE_MAP和END_MESSAGE_MAP中列出了視窗程式會處理的message。

```
BEGIN_MESSAGE_MAP(CVTWindow, CFrameWnd)
//{{AFX_MSG_MAP(CVTWindow)
ON_WM_ACTIVATE()
ON_WM_CHAR()
ON_WM_CLOSE()
ON_WM_DESTROY()
ON_WM_DROPFILES()
ON_WM_GETMINMAXINFO()
ON_WM_HSCROLL()
ON_WM_INITMENUPOPUP()
ON_WM_KEYDOWN()
ON_WM_KEYUP()
ON_WM_KILLFOCUS()
ON_WM_LBUTTONDBLCLK()
```

```
ON_WM_LBUTTONDOWN()
ON_WM_LBUTTONUP()
ON_WM_MBUTTONDOWN()
ON_WM_MBUTTONUP()
ON_WM_MOUSEACTIVATE()
ON_WM_MOUSEMOVE()
ON_WM_MOVE()
ON_WM_NCLBUTTONDBLCLK()
ON_WM_NCRBUTTONDOWN()
ON_WM_PAINT()
ON_WM_RBUTTONDOWN()
ON_WM_RBUTTONUP()
ON_WM_SETFOCUS()
ON_WM_SIZE()
ON_WM_SYSCHAR()
ON_WM_SYSCOLORCHANGE()
ON_WM_SYSCOMMAND()
ON_WM_SYSKEYDOWN()
ON_WM_SYSKEYUP()
ON_WM_TIMER()
ON_WM_VSCROLL()
ON_MESSAGE(WM_IME_COMPOSITION,OnIMEComposition)
ON_MESSAGE(WM_USER_ACCELCOMMAND, OnAccelCommand)
ON_MESSAGE(WM_USER_CHANGEMENU,OnChangeMenu)
ON_MESSAGE(WM_USER_CHANGETBAR,OnChangeTBar)
ON_MESSAGE(WM_USER_COMMNOTIFY,OnCommNotify)
ON_MESSAGE(WM_USER_COMMOPEN,OnCommOpen)
ON_MESSAGE(WM_USER_COMMSTART,OnCommStart)
ON_COMMAND(ID_FILE_NEWCONNECTION, OnFileNewConnection)
ON_COMMAND(ID_FILE_DISCONNECT, OnFileDisconnect)
ON_COMMAND(ID_EDIT_COPY2, OnEditCopy)
ON_COMMAND(ID_EDIT_PASTE2, OnEditPaste)
ON_COMMAND(ID_EDIT_CLEARSCREEN, OnEditClearScreen)
ON_COMMAND(ID_EDIT_CLEARBUFFER, OnEditClearBuffer)
ON_COMMAND(ID_SETUP_TERMINAL, OnSetupTerminal)
ON_COMMAND(ID_SETUP_WINDOW, OnSetupWindow)
ON_COMMAND(ID_SETUP_FONT, OnSetupFont)
ON_COMMAND(ID_SETUP_KEYBOARD, OnSetupKeyboard)
ON_COMMAND(ID_SETUP_TCPIP, OnSetupTCPIP)
```

```
ON_COMMAND(ID_SETUP_SAVE, OnSetupSave)
ON_COMMAND(ID_SETUP_RESTORE, OnSetupRestore)
ON_COMMAND(ID_CONTROL_RESETTERMINAL, OnControlResetTerminal)
ON_COMMAND(ID_CONTROL_AREYOUTHERE, OnControlAreYouThere)
ON_COMMAND(ID_CONTROL_SENDBREAK, OnControlSendBreak)
ON_COMMAND(ID_WINDOW_WINDOW, OnWindowWindow)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

在每個處理message的的函式裏，又會呼叫一些自訂的更低層的函數。這些自訂的函數主要是處理更複雜的工作。以ON_WM_CHAR() 為例，當使用者按下鍵盤上一個按鍵時，會去執行OnChar() 這個我們用MFC定義的函數，但在OnChar() 中，又會去呼叫底層負責網路傳輸的函數CommTextOut()，如果INI檔內設定local echo，則接著會執行CommTextEcho() 秀到視窗顯示區上。我們用scroll bar當另一個例子來說明，當使用者拉了垂直的scroll bar，會執行OnVScroll()，函數內會依照它的第一個參數nSBCode（用來分辨scroll bar是位於最高，最低，或者是下一頁等這些狀態），來呼叫DispVScroll()，這又牽扯到我們定義視窗的暫存buffer和螢幕輸出等功能函數。

這些功能函數的檔案如下：

```
buffer.c        // 處理scroll buffer的功能函數檔
clipboar.c      // 處理clip board的功能函數檔
commlib.c       // 處理網路 communication的功能函數檔
keyboard.c      // 處理keyboard的功能函數檔
ssl.c           // 自訂 ssl的功能函數檔
```

telnet.c　　// 分析從Server端接收到的自串功能函數檔

ttdialog.c　　// 處理對話方塊的功能函數檔

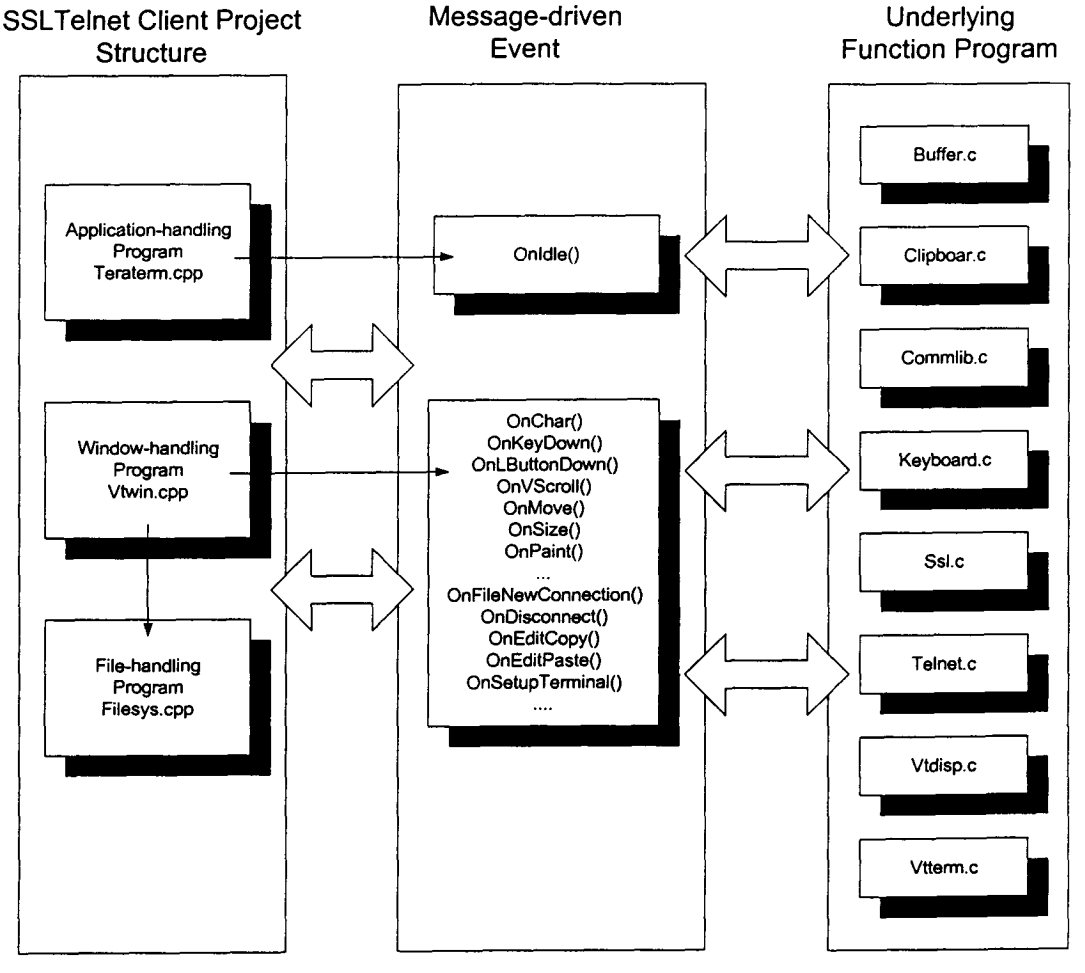
ttwinmain.c　　// 視窗 variable和 flag檔

vtdisp.c　　// 螢幕輸出處理函數檔

vtterm.c　　// VT終端機模擬功能函數檔

等source files。

最後，我們將整個SSLTelnet Client的程式架構包含之前所有提到的觀念用一張圖來描繪它們之間的關係：



25. SSLtelnet Client 架構圖

很清處的可以看出，當我們要修改程式的功能時，我們必須先要修改project中的teraterm.cpp和vtwin.cpp，然後再去找出它們裏面的message-driven event function，看看那一個是我們想要修改的部份。有些event function會呼叫underlying tool function，所以依照我們所欲達到的目標而適度修改underlying tool function也有其必要。

5. 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，相關Source code 請參考所附之CD。本計劃也實作出有安全機制之遠端連線(telnet)通訊協定之安全通訊系統，並以 Sniffer Pro網路封包擷取軟體攔截封包監看並測試，確認其安全性，期能以此強化國防體系的安全的連線機制。

6. 參考文獻

[1] 附上之 CD 光碟片(含原始碼)

[2] 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)。

[3] 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)。

[4] 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)。

[5] 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)。

[6] 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)。

[7] 美國國防部可信賴計算機系統評估準則 (橘皮書)。

[8] 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)。

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

[10] 各分公司於86/3/31—86/4/15提供研究所彙整、及分析。

[11] 電腦應用系統稽核 (工研院，85/3)。

[12] 資訊工業策進會 (EDI簡訊)。

[13] 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)。

[14] 公文處理現代專題研討, 84年 行政院研考會。

[15] 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。

[16] 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。

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

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

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

[20] ITU Rec. X.509(1993), 1995。

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[37] W. Richard Stevens, "TCP/IP Illustrated, Volume1", Addison Wesley,1999

[38] 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

[39] 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

[40] 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

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

[42] 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

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

[44] B.Preneel, personal communication, 1995

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

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

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

[48] ANSI X3.106,"American National Standard for Information Systems-Data Link Encryption," American National Standards Institute, 1983.

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

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

[51] 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.

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

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

[54] The SSL 3.0 Protocol,

http://home.netscape.com/newsref/std/SSL.html

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

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

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

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

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

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

[61] NIST FIPS PUB 180-1,"Secure Hash Standard," National Institute of Standards and Technology, U.S. Department of Commerce, DRAFT, 32 May 1994.

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

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

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

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

[66] 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

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

[68] 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.

[69] 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.

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

[71] 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.

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

[73] ISO DIS 8732, "Banking-Key Management (Wholesale)," Association for Payment Clearing Services, London, Dec 1987.

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

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

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

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

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

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

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

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

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

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

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

[85] 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.

[86] 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.

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

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

[89] 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.

附錄A. Telnet與SSLTelnet之分析與比較

本實驗將運用 Sniffer Pro 來分析Telnet 與 SSLTelnet 之差異。Sniffer Pro 是一套功能非常強大的軟體，可以用來分析網路的流量與過濾、監看封包等等。

實驗環境

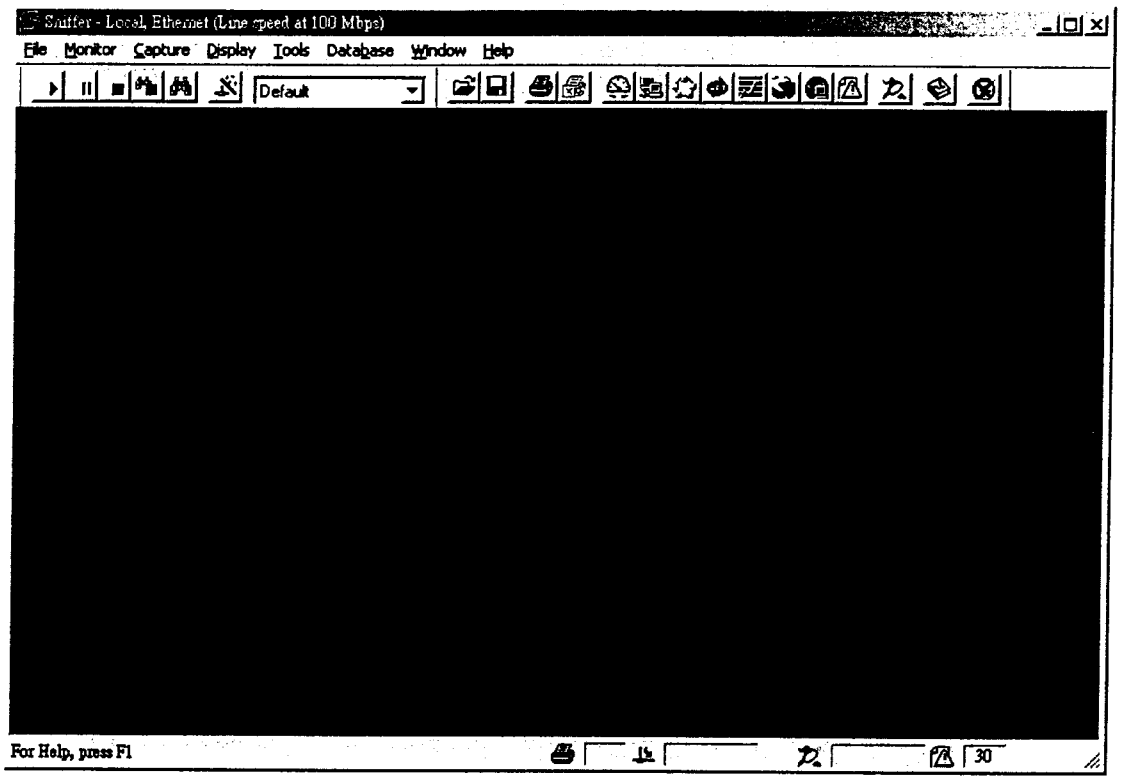
	Server 端	Client端
CPU	Pentium-166	PentiumIII-1G
記憶體	128M	256M
安裝程式所需硬碟空間	10M	3M
作業系統	FreeBSD-4.7-RELEASE	Windows 2000 Professional
網路卡	100M Ethernet	100M Ethernet
IP	140.113.208.125	140.113.208.142

所需軟體

Server 端	Client端
Openssl-0.9.6g	src.zip
SSLTelnet.tar.gz	SSLTelnet.zip
gcc30-3.0.4	Microsoft Visual C++6.0

在以下的例子中我們將在 IP 分別為140.113.208.142 與140.113.208.125 這兩台機器之間分別用 telnet 和SSLtelnet 來傳送封包。然後利用 Sniffer Pro 來作為我們監看傳送的封包內容的工具

，分析並比較所傳送封包的內容。



實驗甲.用Sniffer Pro 監看 telnet 傳送的封包

Step1. 設定要擷取哪些封包

[Capture]-> [Define Filter...]-> [Address]

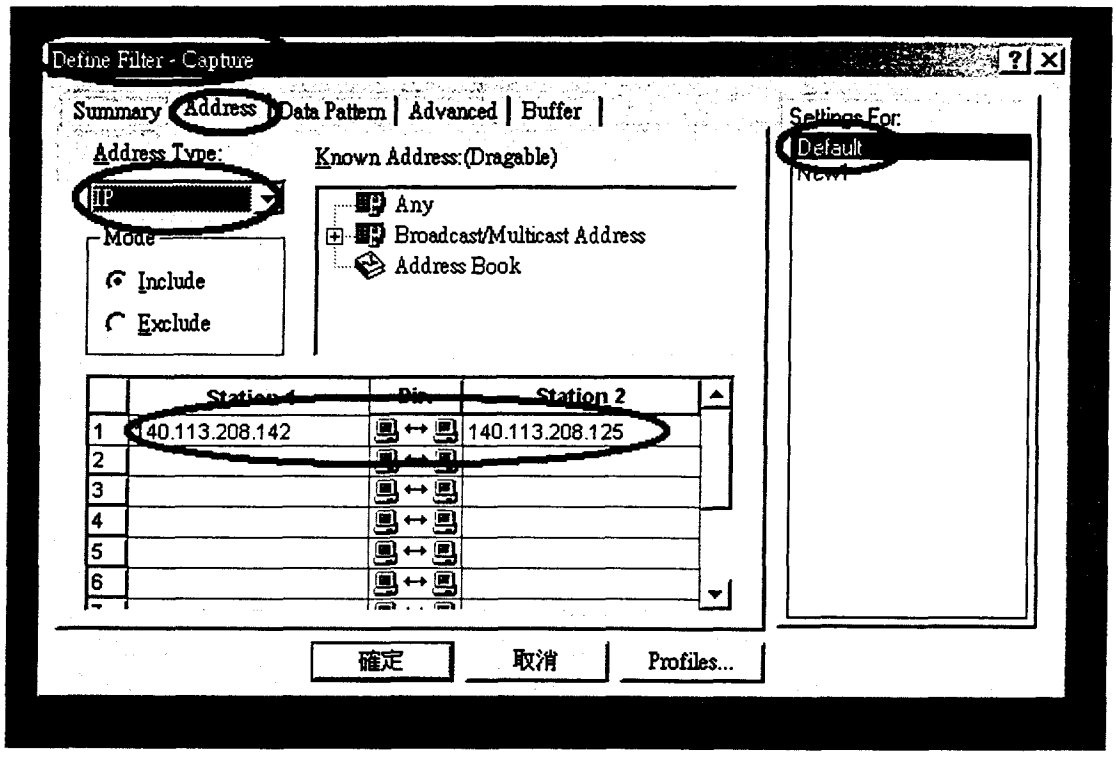
Set for 設為 Default

Address Type 設為 IP

Station1 設為 140.113.208.142 (source)

Station2 設為 140.113.208.125(destination)

Dir. 設為 <-> (雙向)



Step2. 設定要顯示哪些封包

[Display]-> [Define Filter…]-> [Address]

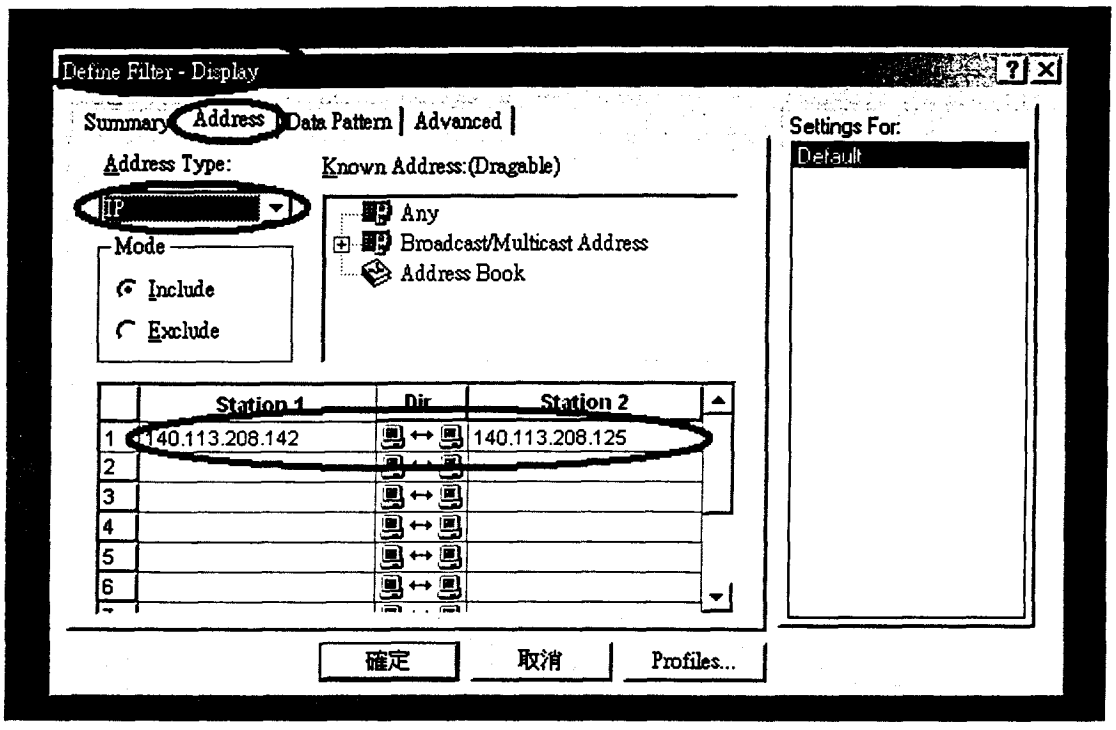
Set for 設為 Default

Address Type 設為 IP

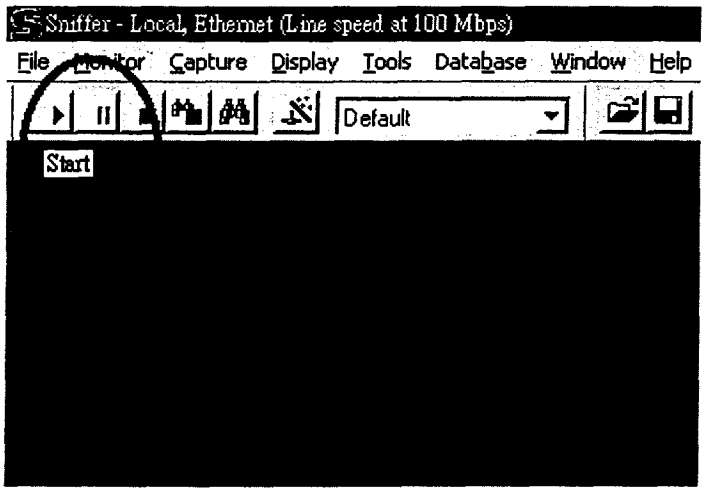
Station1 設為 140.113.208.142 (source)

Station2 設為 140.113.208.125(destination)

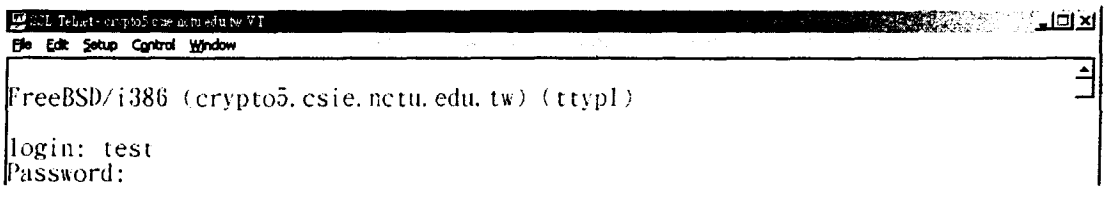
Dir. 設為 <-> (雙向)



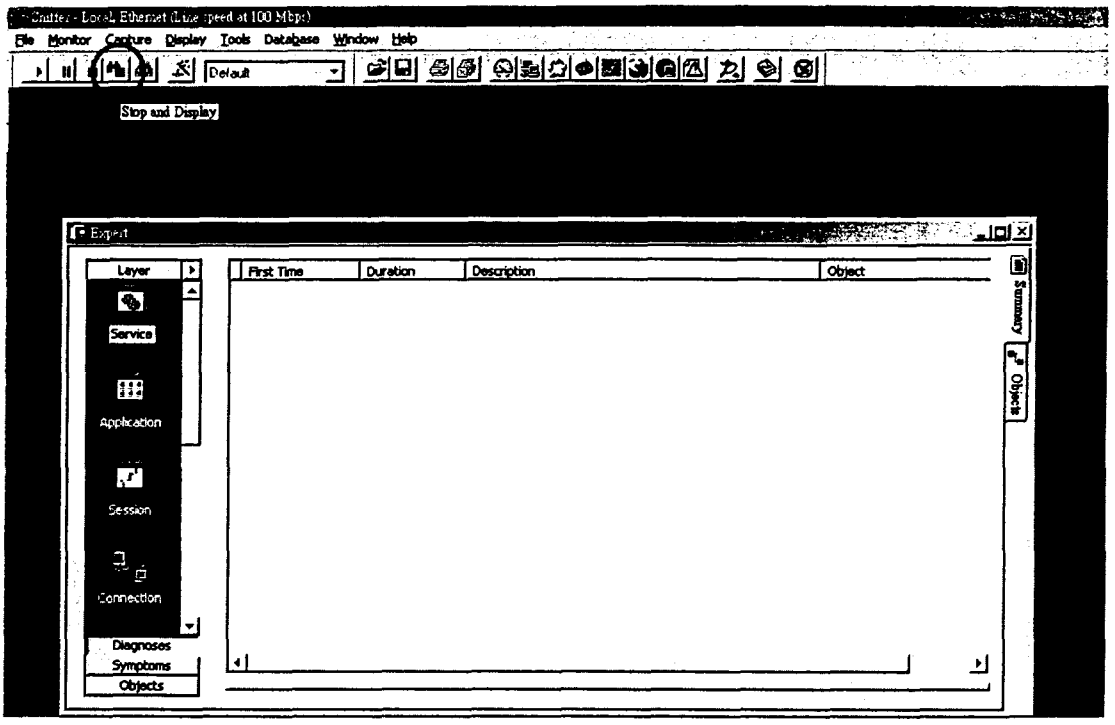
Step3.開始監聽並抓取封包



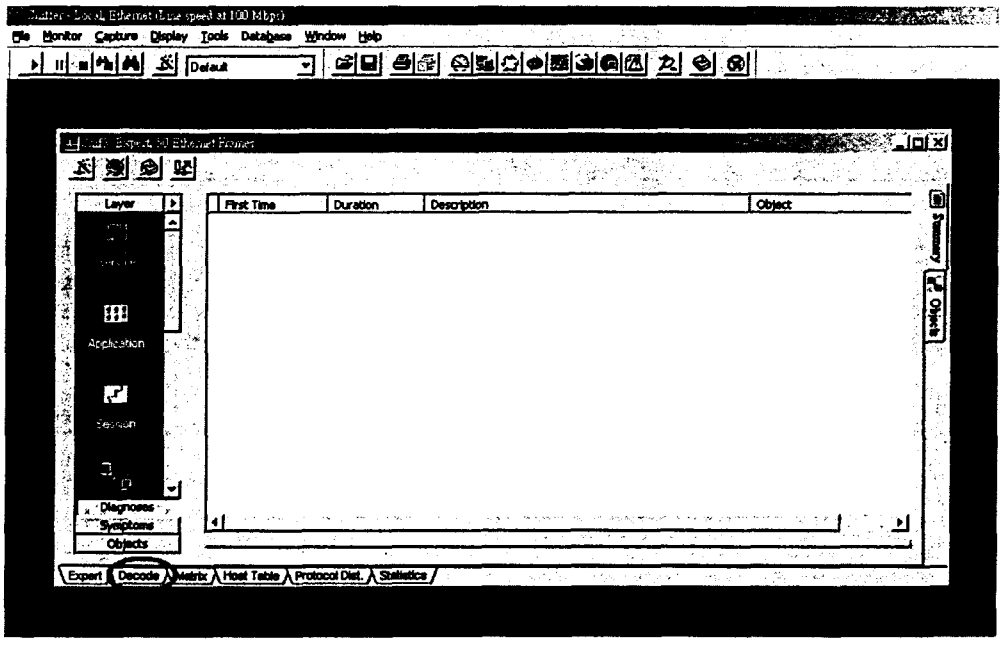
Step4.在140.113.208.142使用telnet 連線到 140.113.208.125的port 23(140.113.208.125 在port 23 以普通的telnetd來listen)並打入login帳號為test，密碼為12345作為測試 ，顯示的畫面如下



Step5. 停止抓取並顯示封包內容

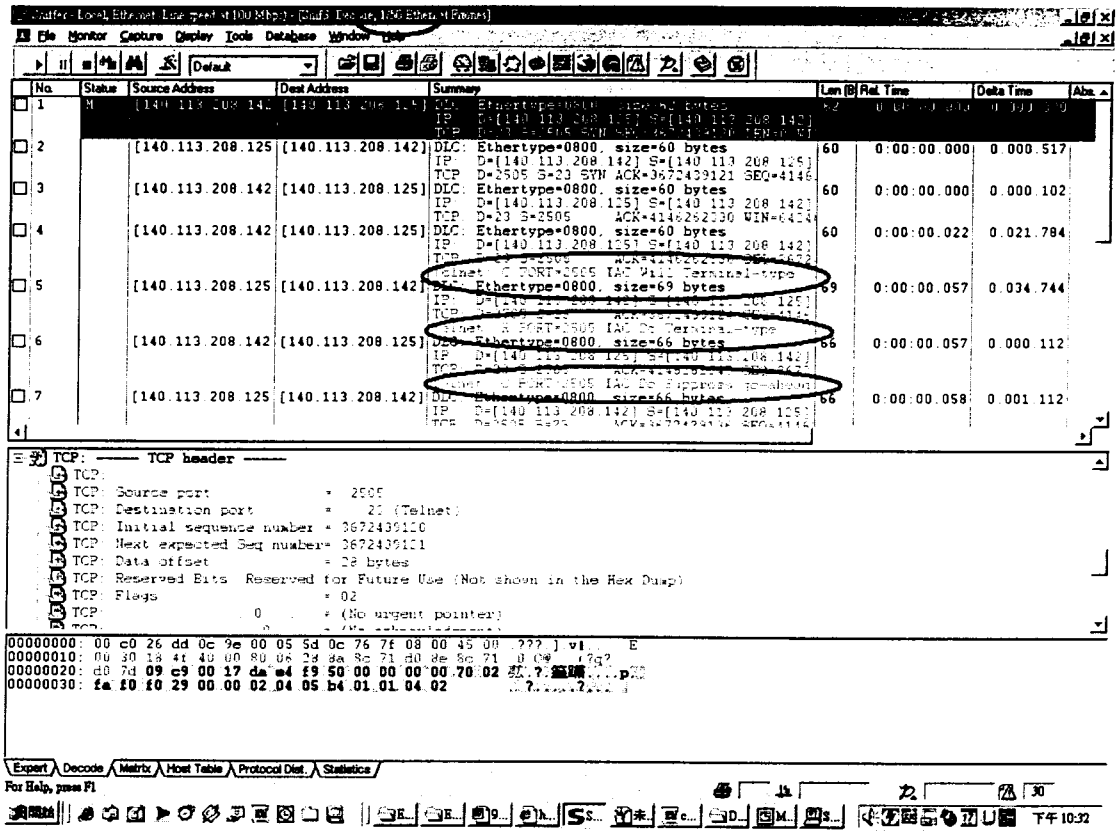


Step6. 選取 Decode

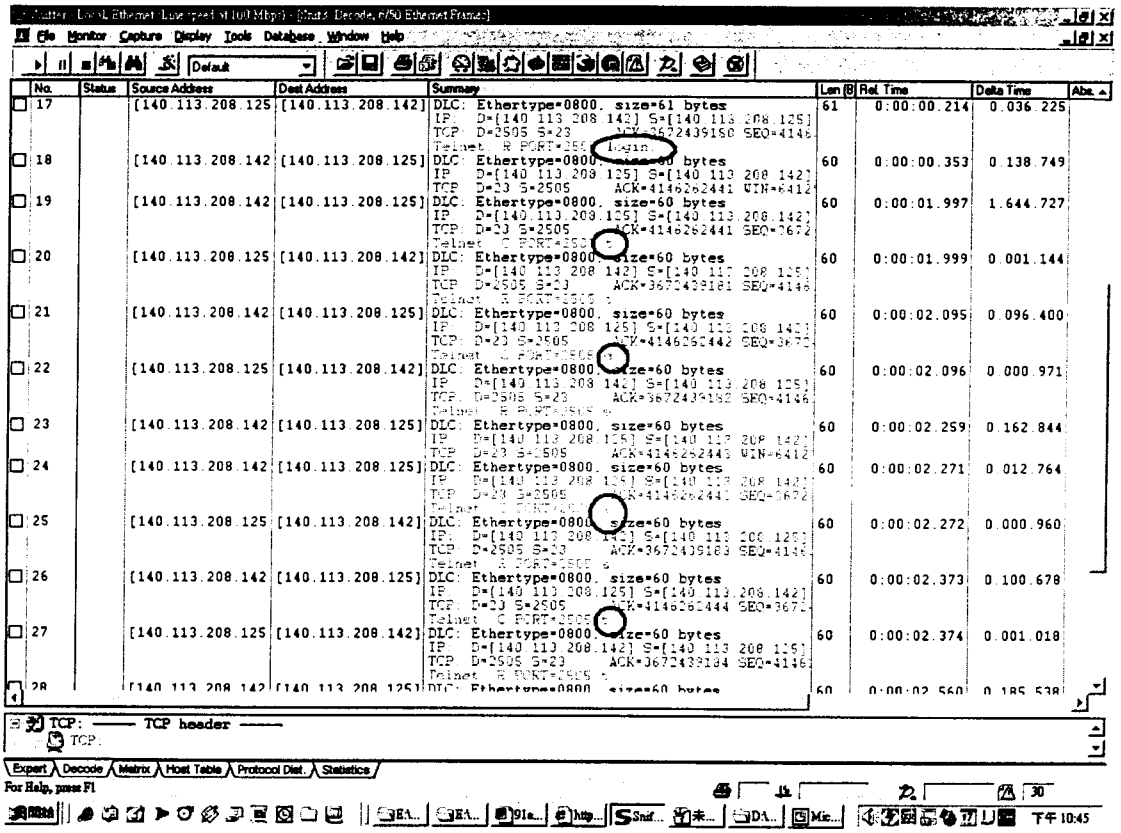


Step7. 顯示的結果

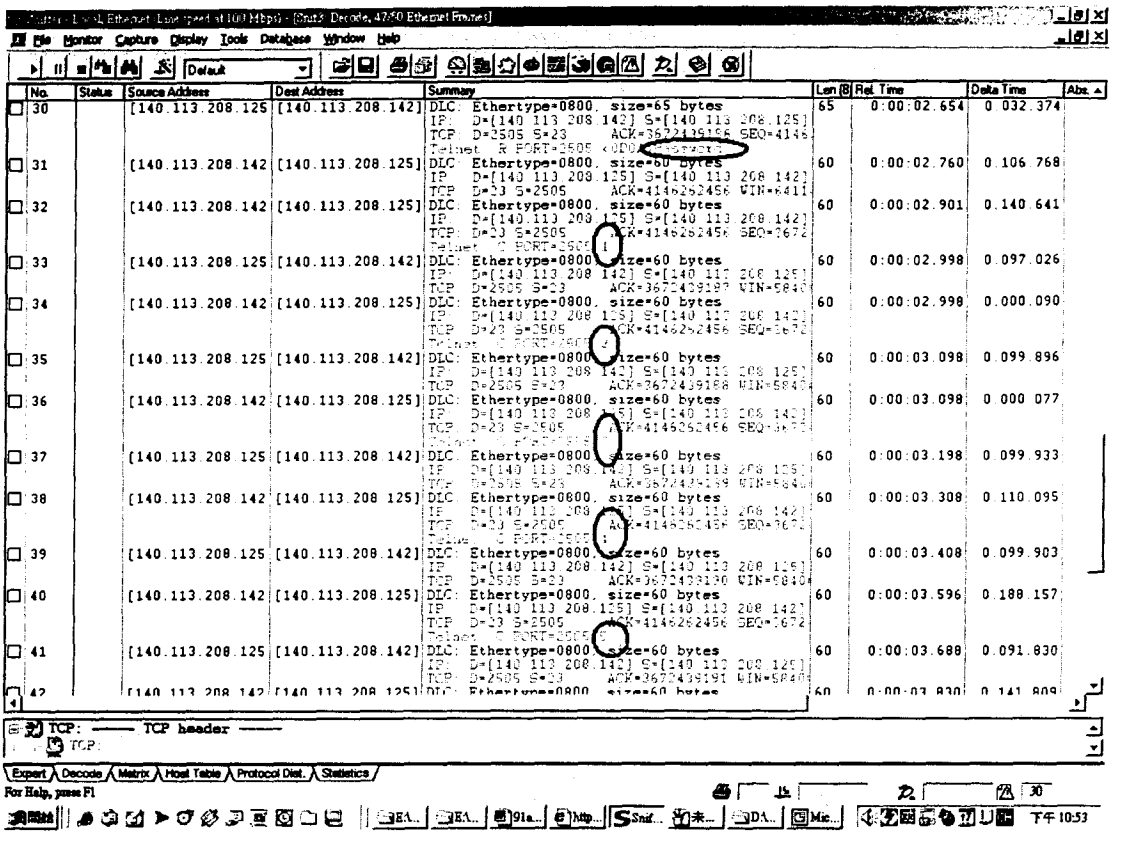
來來回回共收到50個封包，出現Telnet IAC的control message。



出現 login 之後出現帳號 test 的字樣



在140.113.208.125送Password:之後抓到140.113.208.142 所送的12345

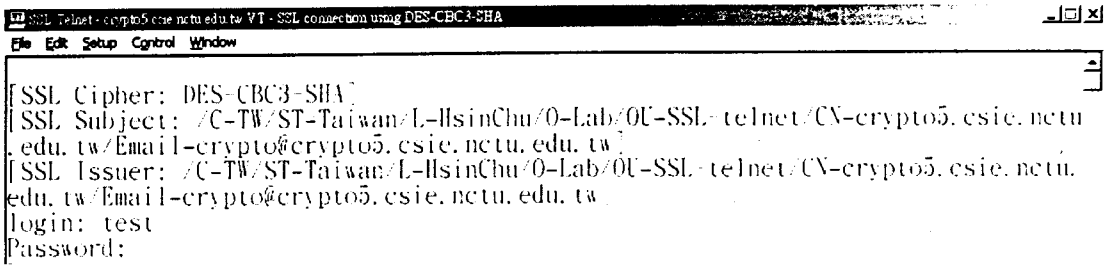


由此可見，telnet 是十分不安全的，可以輕易的抓到所送的帳號密碼。

實驗乙.用Sniffer Pro 監看 SSLtelnet 傳送的封包

Step1~3 與甲.敘述的步驟一樣

Step4.在140.113.208.142使用SSLtelnet 連線到 140.113.208.125 的port 1234(140.113.208.125 在port 1234 以SSLtelnetd來listen)並打入login帳號為test，密碼為12345作為測試



Step5、Step6 與 敘述的步驟一樣

Step7. 顯示的結果

在第10個封包中發現140.113.208.125(crypto5.csie.nctu.edu.tw)的憑證，在陸續幾個封包中也看到 SSL handshake 的步驟，卻找不到所打的login 與 password 和 test 與 12345，可見此SSLtelnet 連線確實可以達到認證與安全的效果