第八章 安全性網頁系統

鈴、、鈴、『春嬌！我是志明！咱明晚去西門町看電影好麼？』『好啦！志明』.....;『按怎是你？世雄』『哎！罔市妳來做啥？』。(怎會這樣呢？)

8-1 安全網頁系統架構

8-1-1 網頁系統簡介

『全球資訊網』(World Wide Web, WWW)[3] 是目前電子商務最主要的應用系統，它是由『瀏覽器』(Brower)與『網頁伺服器』(Web Server)所構成。不同版本的瀏覽器(IE 或 Netscape)可以瀏覽各種系統底下的網頁系統(如 Apache 或 IIS)；也就是說，瀏覽器與網頁伺服器之間只要透過標準程序(如 HTML、HTTP)，便可達成之間的通訊行為，而不論兩者軟體是安裝何種系統(如 Win 2k 或 Unix/Linux)之上。因此，WWW 可以說是整合異質電腦之間的最佳利器。基本上，透過瀏覽器可以瀏覽網頁上所提供的資訊，近年來，更成為商業上交易的工作平台。客戶透過瀏覽器查閱廠商所提供的資訊，亦可直接在網站上購買所需的產品。如此一來，僅有文字的敘述文件，並不能滿足客戶需要；為了增加瀏覽的效果，瀏覽器整合了聲音、文字、影像、以及動畫等等，使其成為多媒體的應用環境。不僅如此，瀏覽器為了能整合各種應用，同時還崁入多個應用系統及通訊協定，如 FTP、News、BBS 與 Java、Java Script、Active X、Flash、Media Play、等等，因而使 WWW 系統更加多采多姿。

然而瀏覽器終究的目標，是要成為各種應用系統的工作平台，除了必須具有文字、聲音、影像、動畫等功能外，雙方性與多工性傳輸功能也是不可或缺的。亦是，致使瀏覽器可以隨時傳送訊息給不同的伺服端，並要求各種應用需求。然而多個伺服端也可同時回應訊息給同一個瀏覽器。總而言之，我們希望能將 WWW 應用環境儘量接近於人的實際環境，只要透過瀏覽器操作，如身歷其境般，從事所期望的應用。譬如，一個『人』利用瀏覽器的影音播放軟體選擇所欲收看影片，就如此『人』已經進入電影院看電影一樣，當這個『人』看完電影之後，也可以透過瀏覽器賞閱百貨公司產品，更進一步，可

以購買所喜愛的產品；此情此景尤如『人』看完電影接著去逛街一樣，上述種種僅需仰賴瀏覽器便可以達成。

我們知道可以嵌入不同應用系統的環境，這表示它具有相當高的相容性；不容置疑，相容性越高的系統，其安全防護能力相對越低。由此可見，瀏覽器非但僅是一個非常複雜的應用軟體，安全性是相當脆弱的，所以利用它來從事電子商務，是一件非常危險的工作。但話說回來，瀏覽器的方便性又讓我們愛不釋手，如何建構一個安全性的 Web 系統，將是一件既嚴肅且重要的事件。

8-1-2 網頁系統的安全考量

還未介紹 SSL 協定之前，我們先以一個範例說明電子商務上可能發生的問題，以及 SSL 協定提供何種功能來補救它。圖 8-1 中，假如 Alice 小姐透過瀏覽器瀏覽到某一家網路商店，並且向該網站（Bob 網站）訂購一個花瓶。正當 Alice 選定產品後，並輸入信用卡號碼(進入 https://)，且待 Bob 網站收到信用卡後會將它轉送到 Alice 所屬的銀行，接下來，銀行確定信用卡無誤後，再將該費用轉帳到 Bob 的帳戶，同時回應一個確認的訊號給 Bob 網站，Bob 網站再透過郵寄方式將 Alice 所訂購的產品寄給它；如此整個交易算完成了。

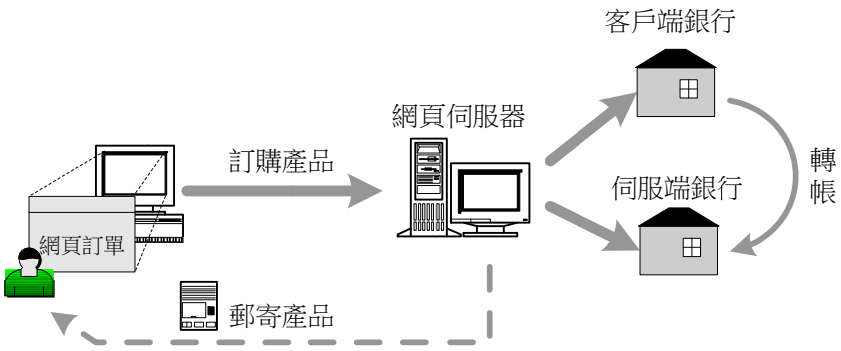


圖 8-1 電子商務範例

上述的交易行為看起來非常簡單，其實存在許多危險性行為，分述如下：

◈ 如何確認持卡人：客戶只要輸入信用卡便可以採購商品，這是一件非常有爭議的問題。譬如，Alice 的小孩利用它的信用卡在網站採購商品，然而 Alice 並不知情，待信用卡帳單寄到時，Alice 如不承認這筆交易，難免會發生嚴重的紛爭。再者，

如果 Alice 的信用卡號碼已被他人知道（這種情況應該很普遍，也很容易），別人利用此號碼到網站去購物，有誰能知道呢？甚至，駭客利用 Alice 信用卡號碼採購了許多東西，並寄到 Alice 家，Alice 又要如何來否認呢？

◈ 如何確認網站：Bob 網站是否是合法的？說不定 Bob 網站是虛設的，Alice 輸入信用卡號碼後，根本拿不到所採購的產品。再者，若駭客竄改 DNS 伺服器，將 Bob 網站（假設是一個著名的網站）轉址到另一個網站，Alice 的貨款已被駭客取走，但還認為是 Bob 網站欺騙了她。

◈ 訊息傳輸是否安全：電子商務大多是透過網路傳輸來達成各種交易行為，這些訊息在網路上傳輸，任何人都可以輕易由網路上截取資料。譬如，Alice 的信用卡號碼欲透過網路傳送給 Bob 網站，不幸若遭駭客盜取其信用卡號碼，則可能進一步盜取其帳款。

針對上述三種可能發生之不安全現象，SSL 也提供三種安全機制來克服，分述如下：

1. 客戶端認證（Client Authentication）：SSL 提供一種機制來讓伺服器認證所欲通訊對方的身分。透過客戶端認證機制，伺服器可以確定客戶的確實身分，以防他人假冒身分。譬如，銀行欲將客戶的存款資料下載給客戶時，必須確定連線中客戶的身分無誤。

2. 伺服端認證（Server Authentication）：SSL 必須提供一種機制，好讓客戶端確定連線中伺服端的真正身分。譬如，客戶端傳送信用卡號碼給伺服端時，必須確定伺服端真正的身分，以免遭受到詐騙。

3. 加密連線（Encrypted Connection）：在 SSL 連線中的傳輸資料都必須經過加密處理，作為達到資料隱密性功能，以防資料於網路傳輸中被盜取。

另外要強調一點的是，目前並非每一部電子商務網站都具有上述三個功能，但加密連線功能是不容或缺的。功能強一點的網站可能包含伺服端認證功能，最完善的網站則需再包含客戶端認證功能。隨著與日俱增的應用環境需求，除了可以選擇是否加入伺服端認證與客戶端認證之外，也可經由雙方協議出加密功能的強度如何。基本上，隱密

性強度愈強的加密處理，相對其加密與解密演算法的負荷就愈重，但其中也牽涉到通訊效益的問題。

8-1-3 Secure Web 的運作程序

所謂『安全性網頁系統』(Secure Web)，即是將 SSL/TLS 安全機制植入 Web 系統中，具有認證雙方身份，以及建立安全連線的功能。在電子商務環境裡，認證身份大多採用數位憑證。圖 8-2 即是 Secure Web 的運作程序，首先雙方交換憑證以確認身份 (訊號 (1) 與 (2))。客戶與伺服端之間互相確認身份之後，必須再互相交換鑰匙材料 (訊號 (3) 與 (4))。雙方再利用所交換的材料來製作出一把會議鑰匙，協議會議鑰匙的方法大多採用 Diffie-Hellman 演算法 (請參考第四章)，以後雙方便利用此鑰匙來通訊 (訊號 (5))。

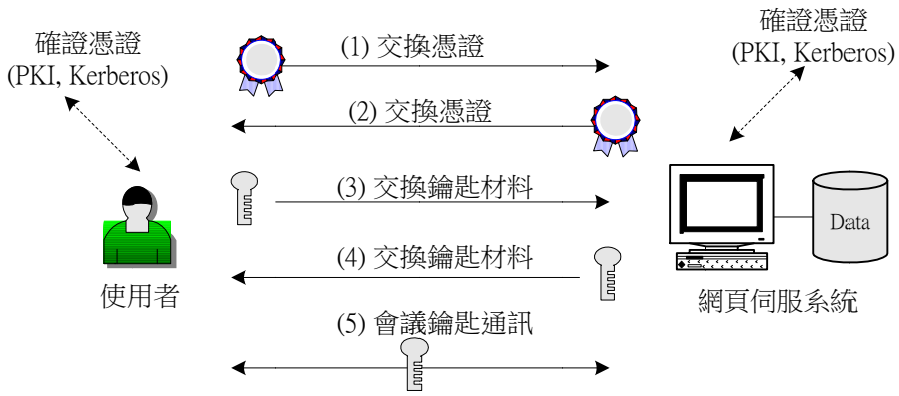


圖 8-2 安全性網頁系統的運作程序

雙方建立會議鑰匙之後，這把鑰匙便是雙方共享的秘密鑰匙，所採用的加密方法也是秘密鑰匙演算法 (如 DES)。接下來，如何將上述的動作整合一個完整的協定，讓客戶端與網頁伺服器兩者來共同遵循與運作，以達成『安全性網頁系統』(Secure Web System) 的環境。

基本上，SSL 必須架設在 TCP 協定之上，也就是說，僅能使用於連接導向的傳輸服務。但為了銜接各種應用系統，針對各系統有其專屬埠口，並以前置『安全性』(Secure) 來標示與原來系統之間的不同，如表 11-1 所示。

表 11-1 SSL 機制之傳輸埠口 (節錄)

應用系統協定	傳輸埠口	說　　明
https	443/tcp	具有 SSL 安全機制的 HTTP
ssmtp	465/tcp	具有 SSL 安全機制的 SMTP
snews	563/tcp	具有 SSL 安全機制的 NEWS
ssl-ldap	636/tcp	具有 SSL 安全機制的 LDAP
spop3	995/tcp	具有 SSL 安全機制的 POP3

8-2 SSL/TLS 安全協定

『安全性插座層』(Secure Socket Layer, SSL)是由網景(Netscape)公司所展出來，目前已修正到第三版 (SSL v3.0)。網景公司將 SSL 安裝於該公司的瀏覽器 (Netscape Browser) 與網頁伺服器上，使它們之間可以協議出雙方可接受的安全機制並傳輸加密後訊息，如此以達到安全性需求。

該公司起初免費提供支援 SSL 的瀏覽器，無形中已大大刺激伺服器的銷售量，致使 SSL 協定漸被其他瀏覽器與伺服器接受 (如 Microsoft IE)，支援 SSL 協定已成為網頁系統必備的條件，從此以後，其他應用系統也開始思考如何將 SSL 協定嵌入其中，使成為具有 SSL 防護的 Telnet、E-mail、FTP 或 LDAP 等等。如此一來，SSL 已不再是網頁系統的專屬安全機制，漸成為通用型網路應用的安全協定，更成為安全性電子商務中不可或缺的工具。

ITEF (Internet Engineering Task Force) 有感於 SSL 協定漸漸成為工業標準，且認為其安全性是可靠的，便於 1999 年發表『傳輸層安全』(Transport Layer Security, TLS) 協定，並由 RFC 2246 與 RFC 3546 發佈流傳。基本上，TLS 是以 SSL v3.0 為基礎，運作程序也大致相同，唯選用的身分證演算法與訊息格式稍有不同。SSL v3.0 規格的官方網站如下：

Netscape：http://wp.netscape.com/eng/ssl3/draft302.txt

然而，TLS 規格可由 RFC 2246 中取得；讀者可以發現兩份規格大致上相同，都是利用 C 語言來描述。本章先以介紹 SSL v3.0 的相關技術為主，然後再列出 SSL 與 TLS 之間的相異點，便可完整學習兩種協定。

8-3 SSL 安全協定

8-3-1 SSL 協定堆疊

SSL 是屬於『層次協定』(Layer Protocol)，擁有獨立的層次介面讓各種應用系統連接，其層次堆疊如圖 8-3 所示。SSL 協定包含兩個主要層次：『SSL 握手層』(SSL Handshake Layer) 與『SSL 記錄層』(SSL Record Layer)，其中 SSL 握手層包含『握手協定』(Handshake Protocol)、『變更密文規格協定』(Change Cipher Specification Protocol) 與『警告協定』(Alert Protocol) 等三個協定；SSL 記錄層只有一個『記錄協定』(Record Protocol)。

『握手協定』是被用來認證雙方身分與協商相關安全機制，協商完成之後，再利用『記錄協定』來傳輸加密後的訊息。在握手協定運作當中，如有特殊異常狀態發生時，則利用『警告協定』通知對方。一旦協議完成，便利用『密文規格變更協定』來通知對方改變加密規格。另外，當執行『握手協定』時，其訊息也是需要經過『記錄協定』的封裝，但協議完成之後，高層應用協定 (如 HTTP) 便可以直接透過『記錄協定』來傳輸訊息。

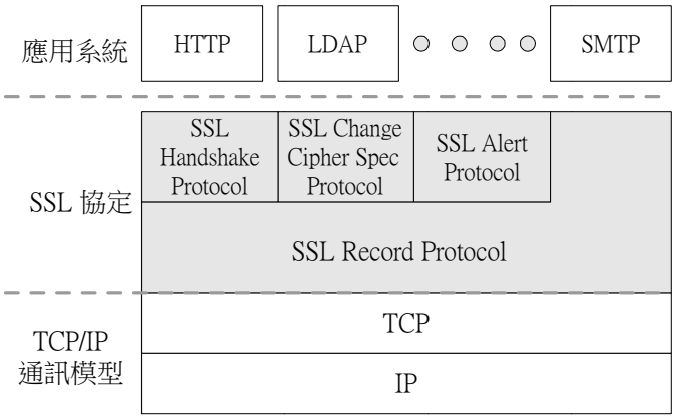


圖 8-3 SSL 協定堆疊

基本上，SSL 必須架設在 TCP 協定之上，也就是說，僅能使用於連接導向的傳輸服務。但為了銜接各種應用系統，針對各系統有其專屬埠口，並以前置『安全性』(Secure) 來標示與原來系統之間的不同，如表 11-1 所示。其中，HTTP over TLS 規範由 RFC 2818 制定；RFC 3207 制定 SSMTP over TLS 規格；RFC 2595 規範 POP 與 IMAP over TLS；RFC 2830 制定 LDAP over TLS；RFC 2712 將 SSL 嵌入於 Kerberos 系統

內。

8-3-2 SSL 握手層

圖 8-4 為 SSL 四種協定的封裝格式。SSL 握手層包含了變更密文規格、警告與握手等三種協定，其訊息封裝格式說明如下。

【(A) 變更密文規格協定封裝】

變更密文規格協定是用來通知對方改變安全套件。當雙方透過握手協定協議出安全套件之後，利用此訊息通知對方改變密文規格的時機，至於接收此訊息的另一方，則必須準備以新的密文規格和對方通訊。此訊息封裝包含下列兩個欄位 (圖 11-4 (a)):

- Content Type (內文型態，簡化 Type): 每一個訊息封裝上都有此欄位，用來表示該訊息的內文型態。本訊息型態為 20。
- Change Cipher Spec：一個位元組。內容為 1，表示通知對方的意思。

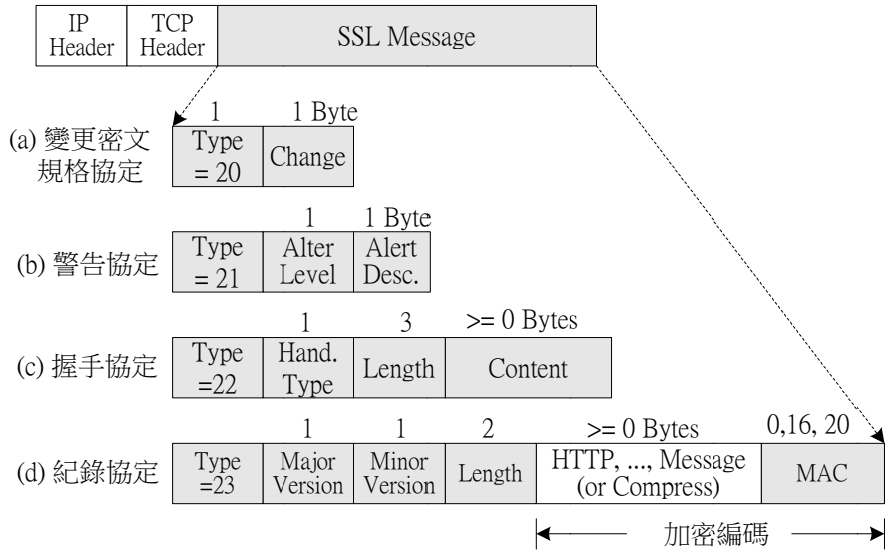


圖 8-4 SSL 四種協定的封包格式

【(B) 警告協定封裝】

雙方利用 SSL 協定通訊的情況下，當某一方發現有異常障礙時，便利用警告協定通知對方。譬如，利用握手協定來協商加密套件，當某一方不能接受對方所提出的加密規格時，可利用警告協定通知對方，並註明不能接受的原因。此訊息封裝包含三個欄位

（如圖 11-4 (b) 所示），如下：

◈ Content Type：警告協定的內文型態為 21。

◈ Alter Level：警告層次。此欄位表示異常狀態的嚴重性，可分為兩個層次：Warning（警報性，1）與 Fatal（致命性，2），前者為警示而已，後者表示發生嚴重異常狀態。

◈ Alter Description：警告描述。此欄位以數值描述發生警告的狀況，有下列發生原因。

【（C）握手協定封裝】

握手協定的功能是認證雙方憑證與協商安全套件，其封裝訊息包含一連串的客戶端與伺服端之間的交換訊息。圖 11-4 (c) 為握手協定的封包格式，包含三個欄位，各欄位功能如下：

◈ 內文型態（Content Type）：握手協定的內文型態為 22。

◈ 握手型態（Handshake Type）：表示該封包所承載的訊息格式，在 SSL 協定中，共定義十種訊息（或稱命令）格式，如表 11-2 所示。各種命令功能如下：

- hello_request：伺服端隨時都有可能發送此訊息給客戶端，此訊息為簡化的通知訊息，用來要求客戶端重新傳送 Hello 訊息，作為協議雙方的安全機制。
- client_hello：SSL 是屬於主從式架構，所有連線都是由客戶端主動提出，再由伺服端回應客戶端的要求。然而 client_hello 是啟動協商一個 Session 的命令，再依照安全機制的強度如何，client_hello 訊息內也許會包含不同的參數，譬如，交換鑰匙所需的亂數、加密套件、以及壓縮方法。一個加密套件裡可能包含所欲協商的加密演算法、訊息認證碼（MAC）等的演算法代號（IANA 指定）。
- server_hello：伺服端收到 Client_hello 之後，選擇可以接受的安全參數，再封裝在該訊息的承載內容，回應給客戶端。client_hello 與 server_hello 兩訊息除了啟動雙方協議一個 Session 外，也直接協議雙方可以接受的安全機制。
- certificate：X.509v3 憑證的內容。當對方要求憑證（certificate_request）作為

身分證明時，則必須傳送此訊息證明之。憑證的內容必須包含交換鑰匙的加密套件，以及相對應的交換鑰匙演算法。

- server_key_exchange：當伺服端傳送憑證給客戶端之後，必須立即發送此訊息來達成雙方的密鑰交換。交換鑰匙過程中的所有訊息都藉由憑證內指定的加密套件保護；經過雙方鑰匙交換（與 client_key_exchange）之後，會建立一個 SSL/TLS 通訊連線所需的主密鑰（Master Secret），而此主密鑰就是被利用於 SSL 記錄協定中加密鑰匙的依據。

- certificate_request：一個非匿名的伺服器可能會發送此訊息，作為要求客戶端傳送憑證以證明他自己的身分。如果選擇此類型的加密套件，在此訊息之後，必須緊接著鑰匙交換訊息來決定主秘鑰。

- server_hello_done：伺服端 Hello 訊息傳送完畢，接著會等待客戶端傳送 Hello 訊息。意味著，伺服端已經將相關安全機制的參數全部傳送給客戶端了。

- certificate_verify：此訊息被使用於證實客戶憑證的正確性，它必須接在憑證訊息之後傳送，並攜帶著一個數位簽章作為要求對方確認憑證的內容。

- client_key_exchange：客戶端鑰匙交換訊息，其中包含有協議建立主密鑰的相關參數，以及數位簽章。

- finished：協議完成訊息，一般都接在『變更密文協定』訊息之後傳送，來證實已成功的協議加密套件與認證完成。本訊息是協議完成演算法、主鑰匙、以及相關安全機制之後，通知對方的第一個訊息；首先客戶端將這些資料經由某一種演算法計算出一個雜湊值，並將其包裝在 finished 訊息內發送給伺服端，伺服端收到此訊息後，也使用同樣的演算法計算出一個雜湊值，如果兩者相同，則表示雙方所擁有的安全套件是一樣的。

◈ 長度（Length）：三個位元長度。表示後面訊息內容（Content）欄位的長度。

◈ 內容（Content）：此欄位存放所攜帶訊息，長度不定。各種訊息格式（命令）所需要的參數都不相同，如表 11-2 中參數欄位所示。

表 8-2 SSL/TLS 握手協定的訊息型態

訊息（命令）型態	參數（內容）
hello_request	無
client_hello	版本、亂數、會議 ID、加密套件、壓縮方法
server_hello	版本、亂數、會議 ID、加密套件、壓縮方法
certificate	一連串的 X.509v3 憑證內容
server_key_exchange	鑰匙材料、數位簽章
certificate_request	憑證型態、認證中心
server_done	無
certificate_verify	數位簽章
client_key_exchange	鑰匙材料、數位簽章
finished	雜湊值

8-3-3 SSL 記錄層

SSL 記錄層只有一個『記錄協定』(Record Protocol)，功能是當 SSL 通訊達成雙方身份認證與協商所欲採用的安全機制之後，則雙方可利用記錄協定來通訊，當然記錄協定所承載的訊息是經由雙方協議後的加密套件所加密。圖 8-4 (d) 為記錄協定的封裝格式，包含欄位如下：

- ◈ 內文型態（Content Type）：記錄協定的內文型態為 23。
- ◈ 主要版本（Major Version，8 位元）：SSL/TLS 版本，目前 SSLv3，此欄位為 3；TLS v1.0，此欄位也是 3。
- ◈ 次要版本（Minor Version，8 位元）：目前 SSLv3，此欄位為 0；TLS v1.0，此欄位為 1。SSL/TLS 協定是利用此欄位區分 SSL 與 TLS。
- ◈ 壓縮後長度（Compression Length，16 位元）：表示資料經過壓縮後的長度（或是沒壓縮處理的長度），以位元組為單位。資料的最大長度為 2^{14} + 1024。
- ◈ 訊息（Message）：此欄位存放上層協定的訊息，如 HTTP、SMTP 或 LDAP 協定等等。

◈ 訊息確認碼 (MAC): 本欄位存放上述所有欄位的內容與協議出的會議鑰匙計算出來的 MAC 碼，作為訊息完整性檢查與訊息確認功能使用。

8-3-4 紀錄層的封裝程序

通訊雙方透過 SSL 握手協定協議成功之後，除了協議出安全套件(如加密演算法、主密鑰、壓縮方法與 MAC 演算法) 之外，也必須協議出每一安全套件的相關參數，SSL 記錄協定再依照這些安全參數來包裝訊息。另外，SSL 記錄協定是將上層協定(如 HTTP 協定) 的資料，以某一固定單位 (TCP 傳輸單位) 分割成若干個封包，每一個封包依照安全參數(容後介紹產生方法)處理相關措施，最後再包裝成 SSL 協定封包，並載入 TCP 封包內，如圖 8-5 所示，其封裝步驟如下：

1. 分段處理：首先將上層資料 (如 HTTP) 分割為若干個區段，至於每一個分段到底多大，與壓縮方法有關。如果傳輸資料沒有經過壓縮，每一分段必須小於 16 KBytes。

2. 壓縮資料 : 依照雙方協議的壓縮方法壓縮分段資料，並參考 compression method 安全參數。在 TLS 規範中並沒有指定壓縮方法，而 SSL 協定中有壓縮方法的選擇項目。

3. 附加 MAC 欄位：依照雙方協議出的 client/server write MAC secret 安全參數計算出『訊息認證碼』(MAC) 碼，並將它附加在資料後面 ; 其中客戶端採用 client write MAC secret，伺服端使用 server write MAC secret 之密鑰。

4. 加密編法：將分段資料加密編碼。由雙方協議出來的加密系統加密，其中所使用的加密鑰匙是 client/server write key 參數所指定，這兩把鑰匙也是由握手協定所協議出來的。如果採用區塊加密法 (Block Cipher)，則需要協議出起始向量值 (client/server write IV)。

5. 附加 SSL 記錄協定標頭 : 每一分段必須附加記錄協定標頭，標頭的欄位如圖 8-3 (d) 所示。

6. TCP 封包封裝：最後將 SSL 封包植入 TCP 封包的訊息欄位上，由 TCP 連線承載傳送。

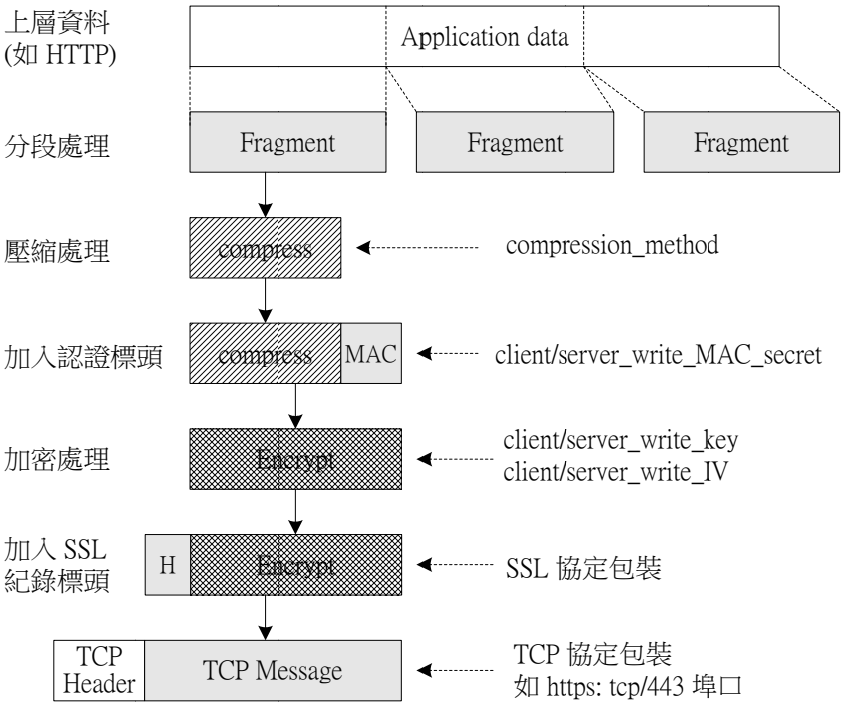


圖 8-5 記錄協定的封裝封包

8-3-5 SSL 協商連結識別

雙方利用 SSL 握手協定協議出某一個安全套件之後，便將此安全套件的相關參數登錄於一個稱之為『會議連結』(Session Connection) 記錄上，並給予唯一的『會議識別碼』(Session ID) 號碼。一個 Session 表示建立一個安全機制的描述內容，譬如，加密演算法、MAC 演算法等等，而且一個建立完成的 Session 可被重複使用。也就是說，SSL 透過握手協定雙方通訊的安全機制，此機制會被包裝成一個 Session。即使通訊完畢後，此 Session 並未消失，可作為下次通訊時重複使用，如此可減少雙方協議安全機制的負荷。另外，任何一個通訊連結(譬如 HTTP 連線)也可能會引用多條 Session 來描述其安全機制。一個 Session 的參數包含如下：

◈ 會議識別碼 (Session Identifier): 伺服端任意指定的一個位元序列的數值，作為標示每一 Session 獨一無二的識別碼。

◈ 對等憑證 (Peer Certificate): 通訊雙方之間的對等 X.509v3 憑證，此參數可能是空值。

◈ 壓縮方法 (Compression Method): 指定資料在加密編碼之前，如需經過壓縮，所欲採用的壓縮方法。

◈ 密文規格（Cipher Spec）：指定大量資料的加密演算法與 MAC 演算法，同時定義加密編碼時所需的參數，譬如雜湊亂數的大小。

◈ 主密鑰（Master Secret）：此參數定義伺服端與客戶端之間共享的秘密鑰匙，長度為 48 位元組。

◈ 是否可回復（Is Resumable）：此參數表示該 Session 是否允許被重複使用。

上述各項參數是被用來建立『SSL 記錄層』（SSL Record Layer）的安全參數，也就是說，記錄協定在將上層協定的資料分段包裝時（如圖 11-3 (d) 所示），其中壓縮、加密、以及計算 MAC 值的演算法，都是參照 Session 中各項參數來完成的。

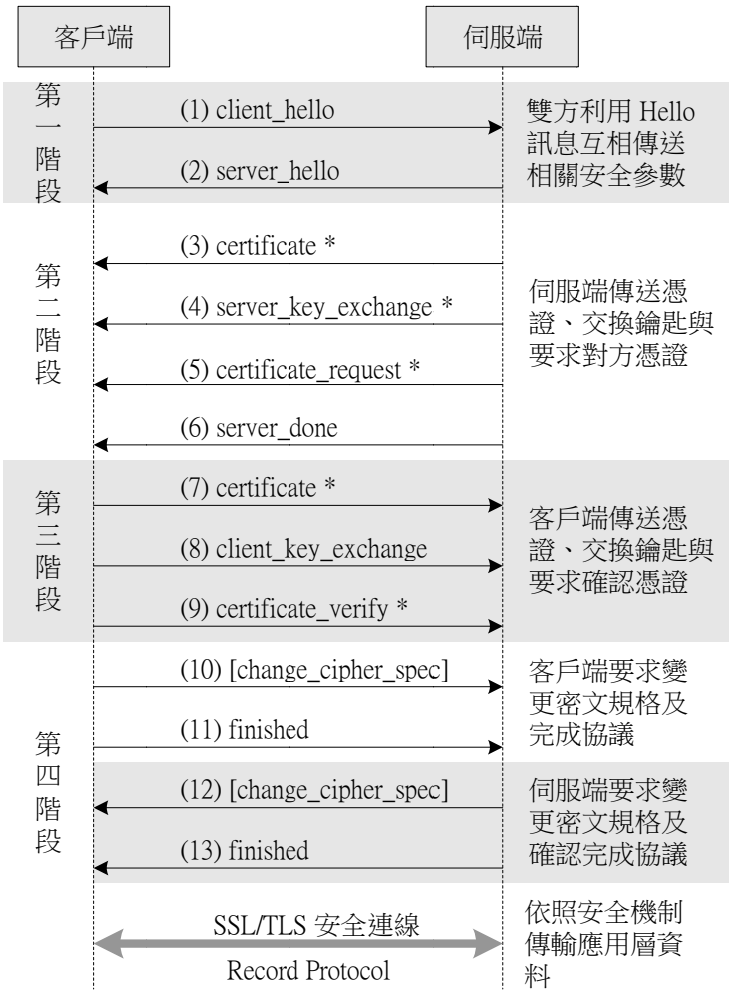
8-4 SSL 握手協定運作

『握手協定』（Handshake Protocol）是整個 SSL 協定的運作核心，客戶端與伺服器之間利用握手協定確認雙方身份，並協議出雙方可接受的安全機制，下列是握手協定的運作程序中，所必須完成工作的步驟：

1. 雙方交換 Hello 訊息，作為協議演算法、交換亂數數值使用，並檢查是否有 Session 可以重複使用。
2. 雙方交換所需的鑰匙材料，並利用這些鑰匙材料製作出『前置主密鑰』（Pre-master Secret）。
3. 雙方交換身份憑證並利用憑證上鑰匙加密訊息，來完成雙方認證對方身分。
4. 利用『前置主秘鑰』與交換亂數值，產生通訊所需的『主密鑰』（Master Secret），一般系統稱之為『會議鑰匙』（Session Key）。
5. 將所建立的安全參數登錄於『會議連結』上，並提供給 SSL 記錄協定執行壓縮、計算 MAC 值、以及加密處理的相關安全參數。
6. 允許客戶端與伺服端證實所建立安全參數是相同的，並且保證在協議當中未受到駭客偽裝攻擊。

圖 8-6 為 SSL 握手協定的運作程序，利用四個協商階段完成上述的工作，概略說明如下：由客戶端啟動第一階段協商，採用 Hello 訊息來協商所能接受的安全套件，

並建立『前置主密鑰』(Pre-master Secret) 作為保護爾後交換鑰匙的訊息；第二階段與第三階段的功能是，雙方互相認證對方憑證與交換鑰匙材料，並建立『主密鑰』以作為傳輸資料所用；第四階段的功能是，雙方確認所建立的安全套件 (主密鑰與 MAC 演算法) 無誤之後，依照安全套件與主密鑰計算出相關安全參數，然後啟動變更密文規格協定，並通知對方以完成協議。



備註：有星號 (*) 表示選項訊息

圖 8-6 SSL/TLS 握手協定的運作程序

值得注意的是，並非每一個 Session 的安全機制都如同圖 11-6 般的完善，也許會有不同的選擇。譬如，若伺服端不需要認證客戶端的憑證，便不用送出 certificate_request 訊號，客戶端也不會傳送憑證給伺服端。因此，在圖 11-6 中的訊號如果有加上星號 (*) 記號，表示該訊息是選項項目，需依照安全機制的強度選擇使用，以下分別敘述各階段的運作程序與命令格式。

8-4-1 第一階段：協商安全套件

第一階段裡包含兩個 Hello 訊息（圖 8-3 訊號 (1) 與訊號 (2)），主要用來交換亂數與協議雙方可以接受的安全套件，每一訊息（client_hello 與 server_hello）包含下列參數：

- ◈ 版本（Version）：為 SSL 協定版本，目前為 3。
- ◈ 亂數（Random）：資料結構為一個 32 位元的時戳與 28 位元的亂數，如 client_hello 訊息內的亂數（28 位元）稱之為 ClientHello.random，然而 server_hello 訊息則稱為 ServerHello.random。此亂數協議『前置主密鑰』使用，或作為交換鑰匙當中預防重送攻擊時使用。
- ◈ 會議識別碼（Session ID）：欲建立安全機制的會議識別碼。如果 client_hello 中此參數為零的話，則表示欲建立新的 Session；否則表示欲引用已建立完成 Session 的識別碼。伺服端收到 Client_hello 後，如果此參數為零，便指定一個號碼回應給客戶端（server_hello），表示欲建立新的 Session；如果不是零的話，依照該 Session ID 數值搜尋是否有此 Session 的安全機制，如有便回應同樣的 ID 給客戶端；則回應錯誤訊息。
- ◈ 加密套件（Cipher Suite）：客戶端（client_hello）以此參數作為建議所欲採用的交換鑰匙演算法與加密規格，伺服端（server_hello）也用此參數回應可以接受的安全機制。在 SSL/TLS 規格書內已將各種安全套件編號，協商時祇要指定套件號碼即可。但除了協商安全套件之外，還是需要協商該套件之下的加密規格，協商內容接下來就會介紹到。
- ◈ 壓縮方法（Compression Method）：列出可以使用的壓縮演算法。但 TLS 規格內並沒有指定壓縮方法（SSL 協定有列出一些方法可供選擇）。

經過第一階段協商之後，必須能協商出雙方可以接受的安全套件與壓縮方法。如果伺服端無法接受客戶端所建議的安全套件時，伺服端會發送 hello_request 訊息，要求客戶端重新建議新的方案。

在 SSL 規範裡，安全套件可以區分為：『鑰匙交換』（Key Exchange）與『加密規格』（Cipher Specification）兩部份，前者的功能是雙方以交換鑰匙材料方式製作共享的『前置主密鑰』（之後再計算成主密鑰）；後者是協商加密系統的相關參數。SSL 鑰匙

交換協定是以 RSA、Diffie-Hellman 與 Fortezza 等三種密碼系統為基礎，其中 Diffie-Hellman 密碼系統又可區分為三種模式；共計有五種協定可供協商，分別敘述如下：

- RSA：利用接收端的 RSA 公開鑰匙來對秘密鑰匙加密。首先客戶端產生一個『前置主密鑰』，並利用伺服端的公開鑰匙加密，然後再傳送給伺服端。
- 匿名的 Diffie-Hellman：利用 Diffie-Hellman 演算法進行交換鑰匙材料，唯計算前置主密鑰的參數是取自雙方自行產生的亂數（鑰匙材料），但容易受到中間人攻擊。
- 固定的 Diffie-Hellman：利用 Diffie-Hellman 演算法進行鑰匙交換，但計算前置主密鑰的參數是取自雙方公開鑰匙憑證（X.509v3）上所簽署的亂數來產生。
- 暫時的 Diffie-Hellman：建立只使用一次的秘密鑰匙，其建立方法是，首先傳送端利用私有 RSA 鑰匙或 DSS 鑰匙來交換與簽署 Diffie-Hellman 公開鑰匙；接收端則利用對應的公開鑰匙確認這個簽署，並且雙方可藉由憑證彼此確認公開鑰匙。雙方安全的交換參數後，再依照 Diffie_Hellman 驗算法計算出前置主密鑰，此方法也是三種方法中最安全的。
- Fortezza：Fortezza 密碼系統。本書未介紹，有興趣讀者請參考 SSL v3 協定規格。

除了協議鑰匙交換協定之外，在 Hello 訊息內也包含加密規格可供雙方協議使用，加密規格有下列協商項目：

- 加密演算法（Cipher Algorithm）：可選用 RC4、RC2、DES、3DES、DES40、IDEA 或 Fortezza 等加密系統。
- MAC 演算法（MAC Algorithm）：MD5 或 SHA-1 演算法。
- 密文型態（Cipher Type）：串流加密（Stream Cipher）或區塊加密（Block Cipher）。
- 可否出口（Is Exportable）：表示該套件是否允許出口；True 或 False。
- 雜湊碼大小（Hash Size）：0、16（MD5）或 20（SHA-1）個位元組。
- 鑰匙材料（Key Material）：一連串的二進位數值（亂數），用來產生鑰匙。
- 起始值大小（IV Size）：密文區段串接（CBC）加密演算法的起始值大小。

8-4-2 第二階段：伺服器確認與鑰匙交換

經過第一階段協商後，雙方已協議出鑰匙交換協定與加密套件，第二階段利用協議出的鑰匙交換協定，互相交換雙方的鑰匙材料，並從事伺服端身份的確認，運作程序如圖 8-3 中訊號 (3) ~ (6)。如果第一階段協議當中需要伺服端憑證時，伺服端會傳送自己的憑證資料（X.509v3）給客戶端。譬如，採用固定的 Diffie-Hellman 演算法時，便需要伺服端的憑證，這是因為憑證中有簽署過的鑰匙材料（亂數）。接著，是否需要傳送 server_key_exchange 訊息的條件，完全決定於第一階段所協議的安全套件，一般若無需交換訊息建立秘密鑰匙，就不需要此訊息，如匿名的 Diffie-Hellman 演算法，因為用來計算秘密鑰匙的亂數是固定數值，所以不需要雙方交換亂數；或者是採用 RSA 鑰匙交換。

接下來，certificate_request 訊息也是選項項目，如果雙方協議的安全套件需要認證客戶端，或者需要客戶憑證上所簽署的亂數時，可發送此訊息要求客戶端傳送憑證資料。最後伺服端以 server_done 表示已完成相關訊息的傳送，之後伺服端就進入等待狀態，預備接收客戶端的安全訊息。

8-4-3 第三階段：客戶端確認與鑰匙交換

當客戶端受到 sever_done 訊息後便進入第三階段的協商（圖 11-6 (7) ~ (9)）。如果安全套件裡需要客戶端的憑證時，則客戶端會傳送憑證相關訊息給伺服端(certificate)，接下來的『客戶鑰匙交換』(client_key_exchange) 訊息是不可以或缺的，但內容是隨鑰匙交換協定而異：

- RSA：客戶端產生一個 48 位元組的『前置主秘鑰』(Pre-master Secret)，並以伺服端的公開鑰匙加密，或是用 server_key_exchange 中的臨時 RSA 鑰匙來加密，再將加密後的前置主秘鑰包裝於 client_key_exchange 訊息內。
- 暫時或匿名的 Diffie_Hellman：將客戶端公開的 Diffie-Hellman 參數包裝於此訊息內。
- 固定的 Diffie-Hellman：客戶端公開的 Diffie-Hellman 參數被包裝於憑證之中，因此，此訊息並未承載任何資訊。

◈ Fortezza：此訊息攜帶客戶端的 Fortezza 參數。

最後，如果客戶端有傳送憑證給伺服端，則必須送出『憑證驗證』(certificate_verify) 訊息，來驗證客戶端憑證。此訊息內攜帶著客戶端憑證內資料的雜湊值，並經過客戶端私有鑰匙簽署過。伺服端利用相對應的公開鑰匙將此簽章解密，並驗證所接收憑證資料的雜湊值是否與 certificate_verify 內的雜湊值相同。

8-4-4 第四階段：完成

雙方協商完成之後，亦即已完成建立一個會議(Session)連結。如圖 11-6 (10) ~(13) 所示，客戶端送出『變更密文規格』訊息來要求伺服端正式改變安全協定，此訊息並非握手協定所有，而是利用密文變更規格協定來完成；相同的，伺服端也會發送此訊息給客戶端。收到變更密文規格訊號之後，雙方會將新的安全機制複製到現有的加密規格內。另外，『完成』(finished) 訊息就較為複雜，除了驗證雙方協議是否成功之外，還必須完成下列安全參數設定：(產生方式容後介紹)

◈ Client write MAC secret：客戶端產生 MAC 碼的密鑰。

◈ Server write MAC secret：伺服端產生 MAC 碼的密鑰。

◈ Client write key：客戶端加密傳輸訊息的鑰匙。

◈ Server write key：伺服端加密傳輸訊息的鑰匙。

◈ Client write IV：客戶端起始向量，僅使用於區塊加密法。

◈ Server write IV：伺服端起始向量，僅使用於區塊加密法。

值得注意的是，雖然客戶端與伺服端所使用的加密鑰匙、MAC 鑰匙與起始向量並不相同，但雙方都擁有這些安全參數，祇不過各自採用不同的安全參數來增加破解的困難度。例如，當伺服端收到客戶端所傳送的加密訊息，則利用客戶端的加密鑰匙來解密，反之亦然；如果攻擊者破解出客戶端的加密鑰匙，無法知曉伺服端的加密鑰匙。

8-4-5 簡化的協議方式

並非每一次 SSL 連線都需要經由上述四個階段的協商過程，始能建立雙方的安全機制。記得前面說過，一個已協商成功的安全機制稱之為『會議』(Session) 連結，並

且給予一個唯一的識別碼，稱為 Session ID，此外，當新的連線啟動時，也可以引用已協商完成的 Session。如圖 8-7 所示，客戶端啟動握手協定時，在 client_hello 訊息內指定使用哪一個 Session ID 的安全機制，只要伺服端同意，並將該 Session ID 置於 server_hello 訊息內，雙方便成功協議使用該 Session ID。接下來，便可直接用 change_cipher_spec 來啟動新的安全機制。

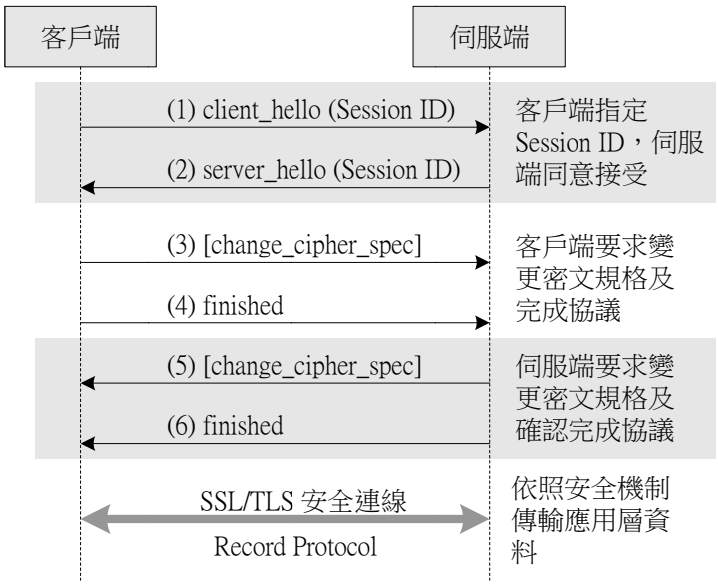


圖 8-7 簡化的握手協定

8-5 SSL 主密鑰產生

在 SSL 握手協定裡最主要有三種功能：認證身分、協議演算法（MAC 與加密編碼）與建立會議鑰匙。前兩項只要雙方協議由何種方式來達成即可，至於建立會議鑰匙就複雜許多。它是利用雙方所交換的鑰匙材料所建立，再利用此鑰匙加密，達成資料的隱密性功能（對稱加密法效率較高）。但 SSL/TLS 為了達到安全性更高，並不直接利用鑰匙材料產生會議鑰匙，而是先產生『前置主密鑰』（Pre-master Secret），再利用它產生『主密鑰』（Master Seeret），之間皆會經過一道複雜程序，如圖 8-8 所示。經過這些程序是為了提高複雜度，否則竊聽者擷取了鑰匙材料，說不定也可以製造出相同的會議鑰匙。產生了主密鑰之後，再利用它產生各種加密鑰匙，將於下節介紹（如圖 8-9 所示）。

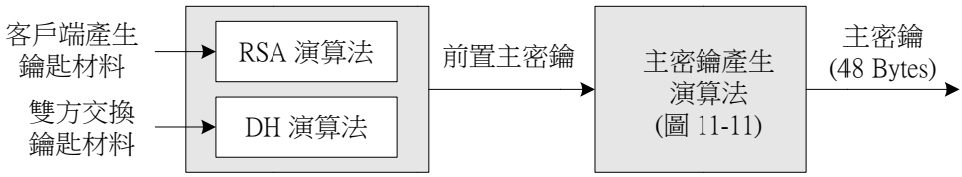


圖 8-8 主密鑰產生方法

基本上，主密鑰長度是 48 位元組，只使用一次，因此被破解的機率非常小。由圖 8-8 可以看出，主密鑰產生過程又可分為兩個階段：產生前置主密鑰與計算主密鑰，以下分別介紹之。

8-5-1 產生前置主密鑰

在 SSL 協定規範中，產生前置主密鑰有 RSA 與 Diffie_Hellman 等兩種演算法，前者是利用數位憑證內的公開鑰匙傳送，後者是利用雙方交換鑰匙材料所產生。

【(A) RSA 演算法】

RSA 演算法是由客戶端產生一個 48 位元組的『前置主密鑰』(Pre-master Secret)，並利用伺服端的公開鑰匙對此前置主密鑰加密，再傳送給伺服端。伺服端再利用自己的私有鑰匙解密此密文，得到與客戶端相同的前置主密鑰。我們以圖 8-6 為範例，來說明 RSA 演算法交換鑰匙材料。首先，雙方以 Hello 訊息互相交換一個亂數，並表明欲採用 RSA 演算法產生主密鑰 (訊號 (1) 與 (2))；接著，客戶端產生一個 48 位元組的亂數 (前置主秘鑰，Pre-S)，並利用伺服端公開鑰匙加密後，傳送給伺服端 (client_key_exchange，訊號 (3))；接下來，雙方利用此前置主密鑰與亂數計算出主密鑰 (容後介紹)，並通知對方已完成 (finished，訊號 (4) 與 (5))，往後雙方就可以利用主密鑰 (K) 來保護資料的傳輸。所產生的主密鑰為：

K = f(Pre-S, ClientHello.random, ServerHello.random)

其中，Pre-S 為前置主密鑰，ClientHello.random 與 ServerHello.random 是啟動訊號 (Hello) 上所攜帶的亂數。

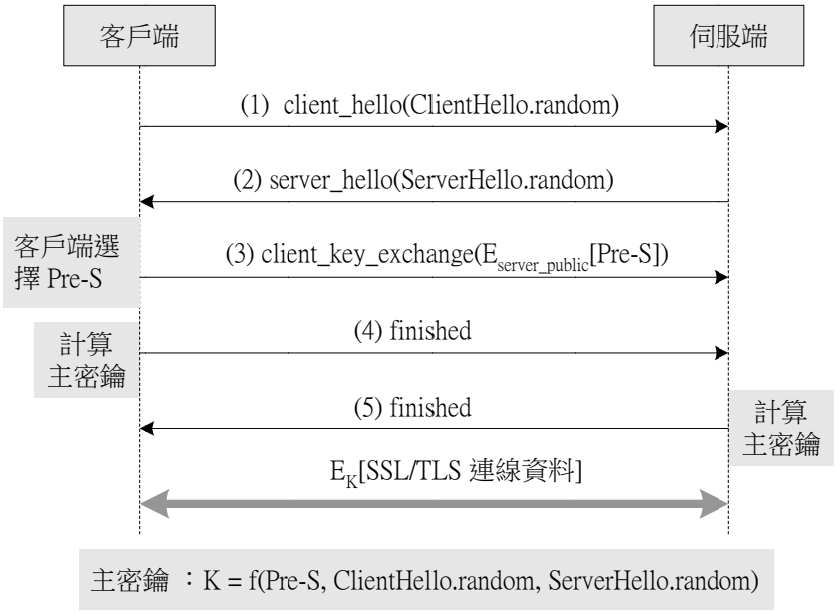


圖 8-9 RSA 演算法的主密鑰產生

【(B) Diffie_Hellman 演算法】

Diffie-Hellman 演算法是雙方互相交換一個數值 (鑰匙材料)，再各自利用所交換的鑰匙材料計算出一個相同數值(『前置主密鑰』)。圖 8-10 為雙方交換鑰匙材料的運作方式。首先雙方利用 Hello 命令交換亂數與協議選擇採用 D-H 演算法(訊號 (1) 與 (2))。接著，雙方再利用鑰匙交換命命傳送鑰匙材料給對方 (訊號 (3) 與 (4))。雙方收到鑰匙材料之後，依照 Diffie-Hellman 演算法計算出『前置主密鑰』(計算方法請參考本書 4-4 節介紹)，再利用此前置主密鑰計算出『主密鑰』(容後介紹)，完成後通知對方 (訊號 (5) 與 (6))。

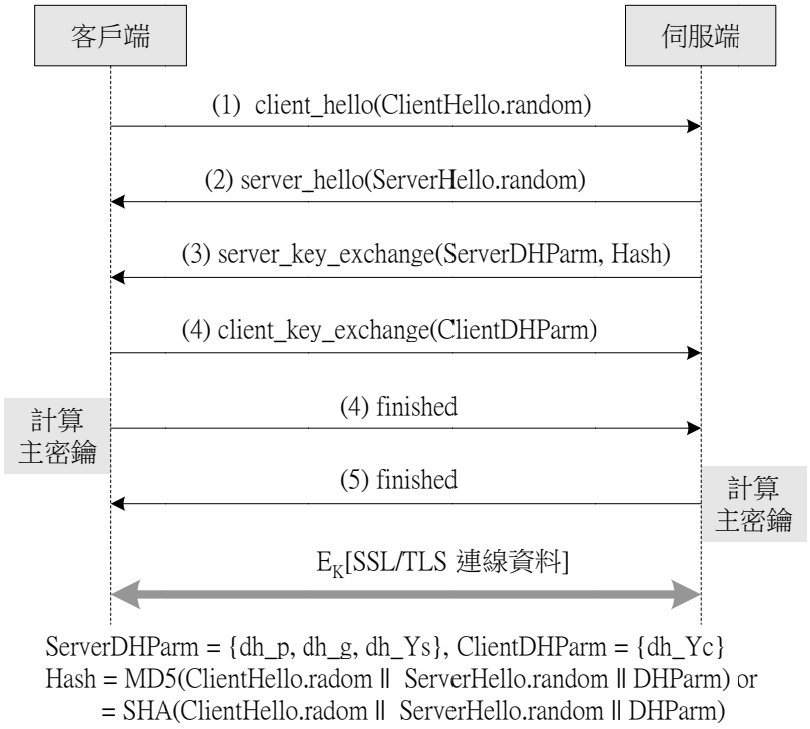


圖 8-10 Diffie-Hellman 的主密鑰產生

圖 8-10 中，雙方所交換的鑰匙材料如下：

$$\text{ServerDHParm} = \{dh\_p, dh\_g, dh\_Ys\} \fallingdotseq \{n, g, g^x \bmod n\}$$

$$\text{ClientDHParm} = \{dh\text{-}Yc\} \fallingdotseq \{g^y \bmod n\}$$

因此，所產生的前置主密鑰為：

$$\text{Pre-master Secret} = g^{xy} \bmod n$$

另外，鑰匙材料交換命令（key_exchange）裡也需要產生雜湊碼，作為確定訊息的完整性；SSL 協定允許選擇採用 MD5 或 SHA 演算法，計算參數如下：

MD5_Hash = MD5(ClientHello.radom || ServerHello.radom || ServerParm)
SHA_Hash = SHA(ClientHello.radom || ServerHello.radom || ServerParm)

8-5-2 主密鑰的計算

無論採用 RSA、Diffie-Hellman 或 FORTEZZA 演算法產生『前置主密鑰』之後，都使用同一種計算公式來產生『主密鑰』(Master Secret)，計算公式如下：

master_secret = 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))

圖 8-11 顯示出上述的計算程序。基本上，SSL 協定所欲產生主密鑰的長度是 48 位元組（384 bits），然而 MD5 演算法所產生的雜湊碼為 16 個位元組，因此需要計算三次再將其連結起來。SHA 所計算出來的雜湊碼（20 位元組），然後再經過 MD5 計算，但不再計算長度。計算公式中 A 表示起始向量（如 8-8-2 介紹，MD5(ClientHello.random || ServerHello.random)），BB 與 CCC 表示前一次所計算出來主密鑰（master_secret）的值，重複計算到長度為 48 位元組為止。

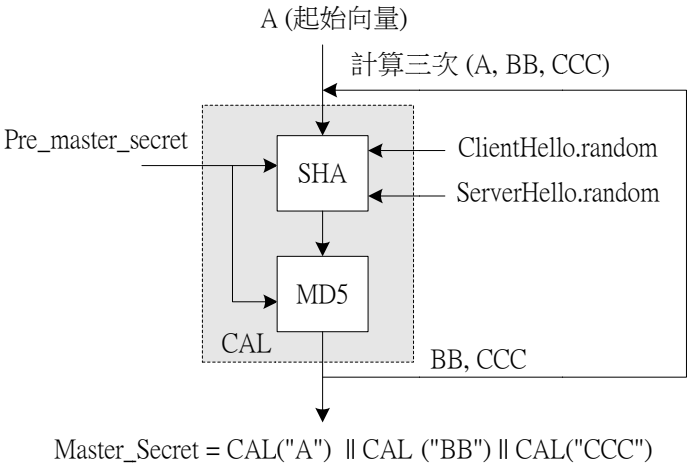


圖 8-11 主密鑰的計算產生

雖然，經過鑰匙交換程序產生了『主密鑰』，但並非直接使用它來從事加密或計算 MAC 碼，這必須視採用何種『加密套件』而定。但無論如何，主密鑰是一個 SSL 會議（SSL Session）安全機制的主要依據，由它來產生各種安全參數，譬如，加密鑰匙、MAC 鑰匙、、、等等；如何運用，下一節再介紹。

8-6 SSL 會議鑰匙的計算

經過公開鑰匙系統（如 RSA 或 Diffie-Hellman）協議及計算出『主密鑰』之後，接下來，必須利用它來計算出秘密鑰匙系統所需的會議鑰匙。圖 8-12 是利用主密鑰產生會議鑰匙的運作程序。首先由主密鑰產生一個鑰匙區塊，它的長度如何是加密套件規範而定，接著再依序由鑰匙區塊中取出各種鑰匙的『密鑰』（Secret），最後加入通訊連線中所產生的亂數，經由 MD5 計算後，產生完成的會議鑰匙，以下分別介紹各個程序。

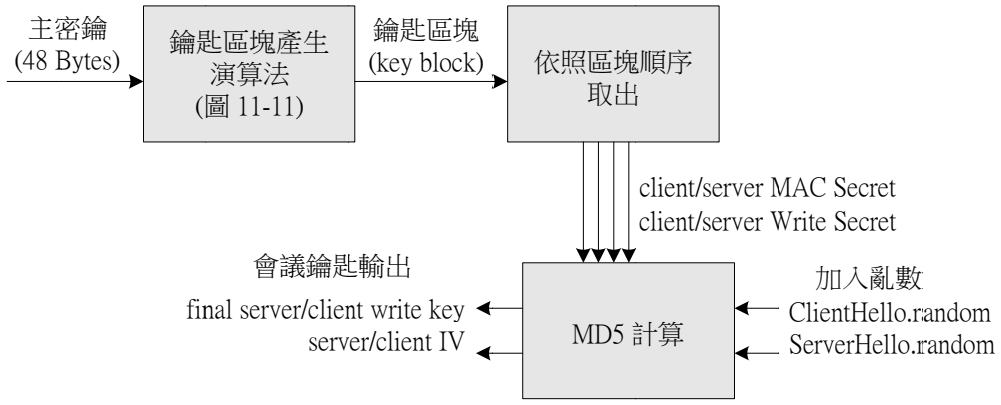


圖 8-12 SSL 鑰匙計算程序

8-6-1 鑰匙區塊產生

SSL 協定利用主密鑰產生一個稱之為『鑰匙區塊』(Key Block) 的鑰匙序列，再利用鑰匙區塊來產生各種鑰匙。鑰匙區塊的計算如下：

key_block = MD5(master_secret || SHA("A"|| master_secret || ClientHello.random || ServerHello.random)) ||
MD5(master_secret || SHA("BB"|| master_secret || ClientHello.random || ServerHello.random)) ||
MD5(pre_master_secret || SHA("CCC"|| pre_master_secret || ClientHello.random || ServerHello.random)) || .....

可以發現，鑰匙區塊的計算方式與主密鑰很相似，祇不過將 pre_master_secret 改成 master_secret 而已 (計算程序如圖 11-11 所示)。但主密鑰祇計算三次 (長度 48 位元組)，而鑰匙區塊到底需要計算幾次，這可依所協商的加密套件而定。依照加密套件所參考的參數如下：

- client_write_MAC_secret [CipherSpec.hash_size]：客戶端計算 MAC 鑰匙的長度。
- server_write_MAC_secret [CipherSpec.hash.size]：伺服端計算 MAC 鑰匙的長度。
- client_write_secret [CipherSpec.key_material]：客戶端加密訊息鑰匙的長度。
- server_write_secret [CipherSpec.key_material]：伺服端加密訊息鑰匙的長度。

如果所計算出來的鑰匙區塊超出所需長度，則拋棄後面多出來的位元。

8-6-2 計算相關鑰匙

基本上，美國允許出口的加密系統的鑰匙都較短，為了增加它的安全性，SSL 協定規定除了利用鑰匙區塊計算之外，再增加一次雜湊演算法計算，來增加它的安全性，計算方式如下：

final_client_write_key = MD5(client_write_key || ClientHello.random || ServerHello.random)

final_server_write_key = MD5(server_write_key || ServerHello.random || ClientHello.random)

另外，某些加密演算法需要『起始向量』(IV)，譬如 CBC 加密套件，它的產生方式如下：

client_write_IV = MD5(ClientHello.random || ServerHello.random)

server_write_IV = MD5(ServerHello.random || ClientHello.random)

值的注意的是，Client 與 Server 計算會議鑰匙與 IV 之間的亂數排列次序並不相同。經過 MD5 計算後，應該產生 128 bits (16 Bytes) 長度的雜湊值，隨著各種演算法擷取前面所需的位元，而拋棄到後面多出來的位元。譬如，採用 DES 演算法，則取用前面 56 bits，而拋棄後面剩下來的位元。

8-6-3 鑰匙產生範例

我們以 SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 安全套件，說明鑰匙區塊、加密鑰匙與 MAC 鑰匙的產生過程，該套件需要兩個 MAC 鑰匙與兩個加密鑰匙，分別分配給客戶端與伺服端持有，其中 MAC 鑰匙需要 16 Bytes、加密鑰匙需要 5 Bytes 的鑰匙材料來計算，因此，鑰匙區塊必須產生高於 42 Bytes 的鑰匙長度。鑰匙區塊每經過一次 MD5 計算，會產生 16 Bytes 的鑰匙材料，因此，必須計算 3 次產生 48 Byte，之後再取前面 42 Bytes 的鑰匙材料。選用鑰匙區塊的次序如下：(以位元組計算)

- client_write_MAC_secret = key_block[0,.., 15]
- server_write_MAC_secret = key_block[16, …, 31]
- client_write_key = key_block[32, …, 36]

- server_write_key = key_block[37, …, 41]

選擇完鑰匙區塊之後，接下來計算最後的加密鑰匙，其中會依照所需長度來擷取，如下：

- final_client_write_key = MD5(client_write_key || ClientHello.random || ServerHello.random) [0. …, 15]
- final_server_write_key = MD5(server_write_key || ServerHello.random || ClientHello.random) [0, …, 15]
- client_write_IV = MD5(ClientHello.random || ServerHello.random) [0, …, 7]
- server_write_IV = MD5(ServerHello.random || ClientHello.random) [0, …, 7]

我們可以看出，採用 RSA RC2 加密演算法需要 128 bits（16 Bytes）長度的鑰匙，HMAC-MD5 也是；另外，加密演算法的起始向量為 64 bits（8 Bytes）。