第十三章 用戶認證協定

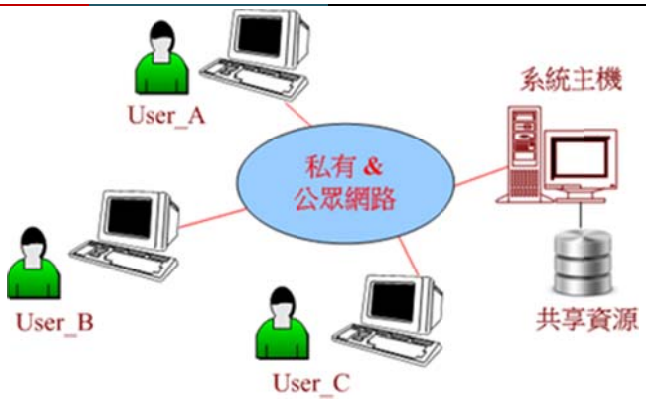
『叩』!!『叩』!!『您是誰？』『我是志明、春嬌快開門啦 !!』。喂！等一下！他真的是志明嗎？可以讓他進去嗎？（裡面真的是春嬌嗎？）

13-1 用戶認證

13-1-1 用戶認證簡介

『認證』(Authentication) 可由兩方面來討論，一者為『訊息認證』(Message Authentication，或稱確認)，主要確認訊息是否確實由某人所發送，並且確認訊息在傳輸當中並無遭受偽造或竄改。換句話說，所欲確認的對象是訊息，這方面已在第六章詳細介紹過。另一者為『用戶認證』(User Authentication)，所欲確認的對象是某一個『實體』，它可能是人、組織單位、或系統主機等等。基本構想是肯定對方的身份無誤之後，才允許雙方通訊，此為本章介紹的重點。就用戶認證的技巧而言，可以使用秘密鑰匙系統或公開鑰匙系統。基本上，屬於較封閉性系統（如電子化系統），多半採用秘密鑰匙系統；然而對於開放性系統（如電子商務系統），一般皆採用公開鑰匙系統比較方便。這兩種鑰匙系統的認證協定，本章都會分別說明。

用戶認證是確定通訊中的對方是否如所希望通訊的人，而不是冒牌的。如果網路上只使用到電子郵遞（或公文交換），使用數位簽署便足以確認發信人的真偽。但如應用到分散式資料庫存取時，所牽涉的認證問題就變得非常複雜。通常一個資料庫都會提供多人同時使用，並且每一個人的使用權限也不盡相同，只有明確的身份認證，才能避免使用者超越權限存取資料。另一方面，攻擊者若冒充合法使用者來偷竊資料，也有可能在合法使用者存取資料時，竊聽傳輸中的訊息。所以用戶認證技術不但要有複雜的密碼學基礎，還必須定義出雙方對認證處理的方式；也就是說，必須制定一個『認證協定』（Authentication Protocol），通訊雙方依照此協定共同來處理認證身份的問題。



在資料存取環境裡有兩個重要的課題很容易讓人混淆：『認證』(Authentication) 和『授權』(Authorization)。認證是確認通訊中是否確實其人 (或程序 process)；授權則是確認通訊中這個人的使用權限。例如，世雄到志明那裡索取『電腦網路與連結技術』書本時，志明就必須做下列的判斷：

1. 來訪者果真是世雄嗎？(認證)
2. 世雄有權利索取此書嗎？(授權)

志明需確定上述兩個條件都成立時，才可以將書交給世雄。第一個條件比較重要，因為需先確定對方身份無誤之後，才有必要進入第二個條件去審查其權限範圍，相較於第一個條件，第二個條件可就容易多了。或許我們可將兩個條件分開處理，譬如，可由第三者(春嬌)判斷來訪者是否為世雄，如果正確的話，她再通知志明此人身分無誤；接下來，由志明自己審查世雄的權限。如此一來，春嬌好比扮演著認證使用者身份的工作，我們可稱它為『認證系統』(Authentication System)，如本章欲介紹的主題 - Kerberos 系統。

13-1-2 用戶認證途徑

既然，認證協定主要著重於通訊雙方的身分確認，首先我們就一般的資訊系統而言，到底是如何表達某一個人的身份。基本上，可能有下列幾種途徑：

◈ 使用者帳號/密碼：利用使用者帳號與密碼辨識身份是最普遍的方法，但必須事先在系統上建立帳戶才行。目前在 Internet 網路上，許多系統為了允許陌生人進入，通常會設定一個特殊帳戶讓陌生人使用，譬如 Guest 帳戶。

◈ 網路位址/網域名稱：係利用 IP 封包標頭的來源位址、或者經由 IP 位址查詢其

網域名稱 (DNS 功能)，用來判斷發送此封包的身份。換句話說，利用 IP 位址判斷出該封包的出處，再決定是否給予通過、或接受請求等服務。

◈ 共享秘密鑰匙：利用特殊管道將秘密鑰匙傳送給另一方；如果能確認對方所持有的鑰匙與自己是相同的，便可以確定通訊者的身份。不過這種機制大多需要其他系統來輔助，譬如，以帳號/密碼產生共享鑰匙。

◈ 公開鑰匙：先將個人的公開鑰匙傳送給想要連絡的人，或是某一接收者已擁有多人的公開鑰匙。當接收者收到某一個利用私有鑰匙加密的資料時，並且可以利用其公開鑰匙解密的話，便可確定傳送資料者的身份。這種機制容易受他人欺騙，攻擊者或許可偽裝另一人，進而傳送偽裝的公開鑰匙來瞞騙接收者。

◈ 數位憑證：利用標準化數位憑證可說是目前最妥當的方法，因為憑證內可紀錄使用者名稱、郵遞地址、公開鑰匙等個人訊息；此外，憑證是經由權威單位所簽署保證，並已提供認證該憑證的方法。

◈ 智慧卡：智慧卡或 IC 卡內可以儲存個人數位憑證、一次密碼、公開鑰匙等等。何況智慧卡使用時，必須輸入密碼，因此有如提款卡一般，有雙重確認的功能 (卡片與密碼)。

上述眾多身份的表示方法，總是離不開如何去認證它，本章重點主要在於介紹一些確認的方法，以辨識其真偽。

13-2 使用者密碼

認證使用者身份最簡單的方法，當推眾人皆知的『帳號/密碼』(Account/Password) 系統。當使用者輸入帳號名稱之後，系統會要求使用者再輸入密碼，以確定使用者身份。但話說回來，僅就密碼去認證使用者身份，恐怕不是十分安全的做法；譬如，銀行系統，客戶除了輸入密碼來確認身份之外，還要有『提款卡』才允許從提款機領錢。因此，許多系統不僅要求客戶輸入密碼之外，還需要如 IC 卡的東西來確認身份。但無論如何，輸入密碼是最基本的確認途徑；然而，若將密碼僅作為系統判斷客戶身份的工具，那未免又太小看密碼的功能了。對一般系統而言，為了簡化客戶端進入系統的方便性，通常

僅要求使用者輸入密碼便完成認證的工作，但當密碼輸入之後，可能會衍生許多後續的問題，以下分別介紹這些後續問題。

13-2-1 共享密鑰建立

我們可以回想一下，使用者可以在任何一部電腦上，以輸入帳號與密碼的方式登入系統，而該部電腦可能就在私有網路內，也可能位於全球任何角落。基本上，登入的電腦不應該受地理位置的限制，而且密碼也不應該以明文方式傳送，因為明文方式傳送的密碼，一旦被攻擊者攔截到之後，便可輕易入侵系統。相對地，系統也有保存一份使用者的『帳號/密碼』資料，以作為使用者登入系統的密碼比對；相同的道理，儲存在系統之內的密碼也絕不可以明文方式儲存，否則一旦入侵者進入系統盜取密碼檔案，整個系統的安全性必將陷入崩潰。如此說來，處理密碼需具備下列三個條件：

- 密碼不可以明文方式傳輸；
- 密碼也不可以明文方式儲存；
- 加密（或雜湊演算）後的密碼也不可以在網路上傳輸。

由前面兩個條件，如何達成密碼的確認，說起來的確是一件有趣的問題，第三個條件更為懸疑，我們按部就班的介紹如何克服這些問題。一開始簡單的構想是，密碼經過加密後再傳送給系統；系統也將密碼加密後再儲存於密碼檔案裡。既然負責將客戶端輸入的密碼加密的是客戶端所登入的工作站，因此，工作站與系統之間必須存在某一個協定，來處理有關客戶與系統之間的身份確認的工作，此協定便稱為『認證協定』（Authentication Protocol）。由此可見，從事於身份認證的主要成員包含使用者、工作站與主機系統等三個。如圖 13-1 所示，使用者於工作站輸入帳號與密碼之後，工作站就輸入的帳號與密碼，和系統主機之間以某一種『認證協定』來確定使用者的身份。

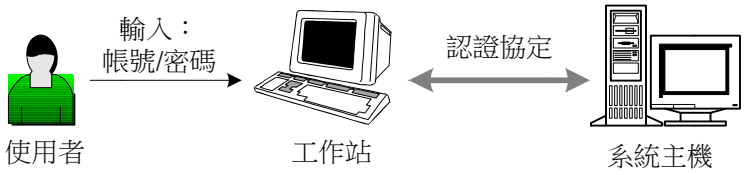


圖 13-1 身份認證的運作

既然密碼不可以明文傳送與儲存，意味著工作站與主機之間必須協議出一種方法來將密碼隱藏起來。最簡單的方法是將密碼加密成為一個亂無章法的一串亂碼，或是經由雜湊演算法計算出一個雜湊值；但無論經由何種演算法來隱藏密碼還是有被破解的危機。譬如，攻擊者可以由工作站與系統之間攔截多個加密後的密碼，從中找出加密鑰匙，如此便可以找出使用者的密碼。因此，為了增加密碼的隱藏性，通常會加入某一數值與密碼混合加密（或雜湊演算），該數值一般稱為『鹽』（Salt），這種處理技巧通稱為『醃製法』（Salt Value）。經由醃製法處理過的密碼則稱為『共享密鑰』（Shared Secret），其處理過程如下：

- 密碼（Password）：P。
- 個人的參數值（鹽）：S_m，m = 1, 2, …, n。
- 演算法：f()，可能是加密演算法（DES）或雜湊演算法（MD5、RC4）。
- 個人的共享密鑰：$K_T = f(P, S_m)$。

並非所有認證協定都是透過醃製法製造共享密鑰，許多系統還是直接將密碼經過演算法計算出來而已，譬如，Unix/Linux、Windows NT 等系統。但新的協定為了增加密碼的安全性，已大多加入醃製法功能。基本上，我們希望每一個使用者所加入的『鹽』不要一樣，因此，在系統裡必須儲存每一個使用者的『鹽』。

有了上述醃製法的概念之後，系統管理員除了針對每一個使用者建立帳號之外，也必須針對每一個帳戶選擇一個亂數作為『鹽』，並允許使用者輸入或更改密碼來建立共享密鑰。建立後的密碼檔案格式就如圖 13-2 所示，檔案內每一個帳戶的紀錄包含：帳戶名稱、鹽、以及共享密鑰（有些系統沒有鹽的欄位）。

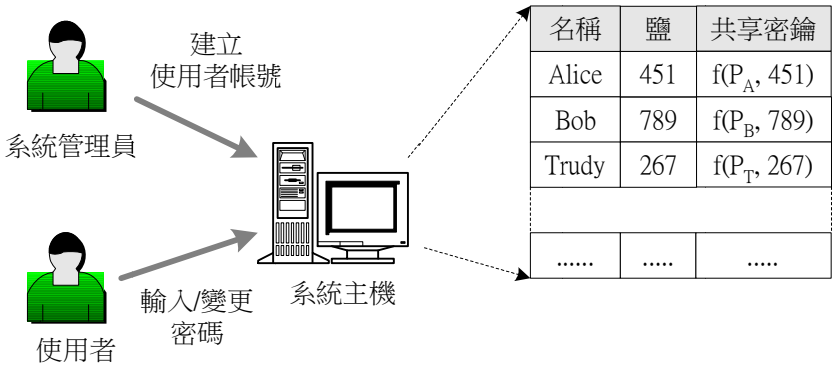


圖 13-2　建立共享密鑰

13-2-2 驗證雙方共享密鑰

我們用一個簡單的運作程序，來說明雙方確認密碼的方法。如圖 13-3 所示，使用者在工作站上輸入帳號之後（訊號 (1)），工作站便將帳號名稱傳送給系統主機（訊號 (2)）；系統主機收到帳號名稱之後，便由密碼檔案裡搜尋出該帳號的『鹽』，並回傳給工作站（訊號 (3)）；此時，工作站便要求使用者輸入密碼（訊號 (4)）；再利用所收到的鹽與密碼，一起經由某一種演算法計算出雜湊值，再將此雜湊值回傳給系統主機（訊號 (5)），系統比對所收到的雜湊值是否與密碼檔案所儲存的相同，如果相同的話，則可以確定對方的身份無誤。

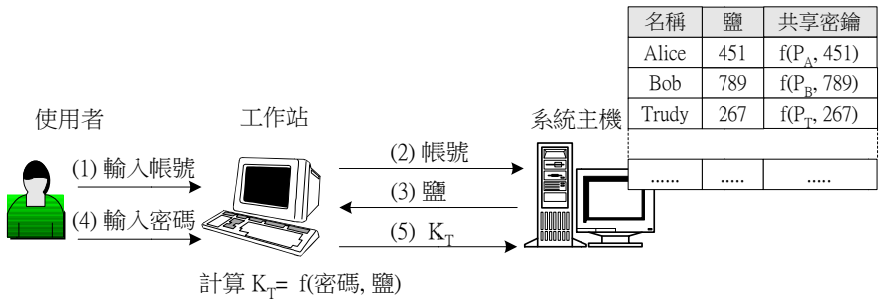


圖 13-3 簡單的密碼確認程序

圖 13-3 做法是將加密後的密碼（f(密碼, 鹽)），直接在網路上傳輸，所以攻擊者只要攔截到該訊息（訊號 (5)），便可以直接登入系統，完全不需要去考慮密碼是經由何種演算法計算而得。為了安全，必須合乎先前所介紹的第三個條件：『加密後的密碼也不可以在網路上傳輸』。如此說來，驗證密碼必須另尋它法才行；簡單的構想是利用秘密鑰匙演算法，只要能證實雙方所擁有的鑰匙是相同的，便能證實對方的身份。因此，我們可將加密後的密碼當作雙方共享的秘密鑰匙，如果能證實雙方所持有的鑰匙是相同的，便能確認對方的密碼無誤。

驗證雙方所持有的鑰匙是否相同的基本方法是『盤問/回應』（Challenge/ Response），如圖 13-4 所示。圖 13-4 與圖 13-3 兩者之間的運作程序非常類似；在圖 13-4 中，系統主機收到工作站所傳送過來的帳號之後，除了傳送『鹽』的亂數之外，也附加了一個亂數 R_1，其作為盤問的訊息（訊號 (3)）。接下來，工作站收到 R_1 與『鹽』之後，即時要求使用者輸入密碼（訊號 (4)），並計算出共享密鑰（K_T），再利用 K_T 向 R_1 加密之後回傳給系統主機（訊號 (5)）；系統主機同樣利用密碼檔案裡所儲存的共享密鑰

向 R_1 加密，如果兩者加密後的訊息相同的話，便可以認證對方的加密鑰匙是相同的，也可以確認密碼無誤。

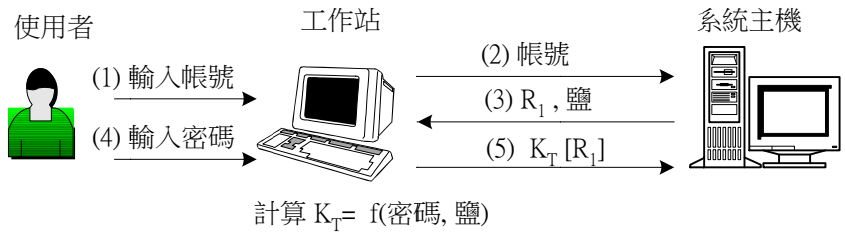


圖 13-4 盤問與回應的密碼確認

由此可見，我們係利用加密後的密碼作為雙方共享秘密，因此，將它稱之為『共享密鑰』（Shared Secret）。但也不一定要使用加密技巧來實現『盤問與回應』協定。我們主要目的是要驗證雙方所持有的鑰匙是否相同，並不是要針對亂數 R_1 做隱密性傳輸；因此，可以採用加密演算法（如 DES）或訊息確認方法（如 HMAC）。由此說來，加密後的密碼也不一定拿來作鑰匙使用，只能夠說是工作站與系統之間的共享秘密（Shared Secret）而已；我們為了突顯它的重要性，因此，將它翻譯成『共享密鑰』。

在一套應用系統之下，通常會有多個帳戶（或稱使用者），每一個帳戶與系統之間都有一把共享密鑰(圖 13-5)；系統為了安全地保管這些共享鑰匙，一般都儲存於密碼檔案裡。然而，針對使用者而言，個人共享密鑰是與系統溝通的最主要途徑，因此又將它稱為個人的『主密鑰』（Master Secret）。

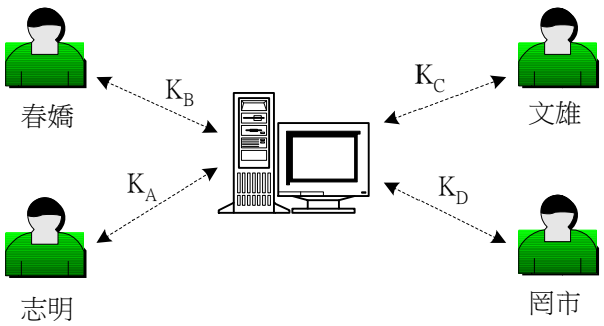


圖 13-5 每一使用者的主密鑰

13-2-3 密碼破解

密碼是進入系統最基本的檢視工具，無論採用電子數位憑證、或者其它實體的物件來認證身份（如生物檢測或 IC 卡），都需要輸入密碼來配合；並且密碼也是產生使用

者『主密鑰』的主要依據，由此可見密碼的重要性。一般密碼是由使用者隨意選出，當然不會輕易告訴他人，但是使用者在使用密碼時，常犯有下列幾點錯誤：

- ◈ 輸入密碼時，被它人窺視出密碼內容。
- ◈ 將密碼儲存於某一個檔案內。
- ◈ 密碼是由有意義的數字（如生日、結婚紀念日）所組成。
- ◈ 被他人詐取密碼。
- ◈ 怕自己忘記密碼，而將密碼書寫於紙上。
- ◈ 為了使用方便，不常變更密碼。

針對目前許多應用程式都是架設在網頁系統上，入侵者只要植入一個偽裝的程式在瀏覽器上，一旦使用者執行該程式時，入侵程式會出現要求輸入密碼，一般使用者會誤認為是應用程式所要求輸入的，便在不知覺的情況下輸入密碼；該入侵程式蒐集到密碼之後，也暗地裡傳回攻擊者手上，此攻擊法又稱為『社交攻擊法』（Social Attack）。類似社交攻擊法來取得密碼的技巧不勝枚舉，總而言之，防範密碼被破解的最根本方法，除了提高使用者自己的警覺心外，別無它法，終究不能完全依賴系統的防護。接下來，介紹幾種系統防範密碼被破解的方法。

【（A）線上密碼猜測】

除非安全性較高的系統，否則一般系統都不會刻意將使用者的『帳戶名稱』隱藏起來；再說，如果系統故意隱藏帳戶名稱，攻擊者欲盜取得該訊息也不是困難的事；由此可見，唯有密碼才是真正的防護線。攻擊者會試著利用帳戶名稱，再嘗試各種可能出現的密碼來進入系統；當然，攻擊者還未猜測密碼之前，會先蒐集使用者的各種訊息，譬如，出生日期、結婚日期、子女出生日期、子女名字、配偶生日、等等，再依照該使用者的個性猜測可能使用的密碼。雖然這種方法不是非常聰明，但它的成功率確是非常的高，這種方法又稱為『字典攻擊法』（Dictionary Attack）。

一般系統為了防範線上密碼猜測，都會限制密碼的輸入次數，也會限制密碼最短的字元數；如果使用者連續輸入幾次（一般都設定 3 次）錯誤的密碼，系統便會自動關閉該使用者（帳戶名稱）的登入。關閉之後，如果僅限制再登入的時間間隔，只要入侵

者有夠耐心的話，還是有可能入侵成功；因此，許多系統要求管理員重新開啟帳戶之後，使用者才可以再登入。當然，系統管理員可詢問使用者是否有從事不正常登入的事實，從中可以判斷出是否有被入侵的現象。

【（B）離線密碼猜測】

無論如何，使用者的密碼還是必須儲存於系統之內，雖然儲存的密碼都會經過特殊處理（加密或雜湊演算），但這些處理技術大多是公開的；攻擊者只要取得密碼檔案，再用已知的處理方法，要尋找出原來密碼明文也並非不可能。再說，一般主機的系統檔案也無法完全隱藏，攻擊者欲找出密碼檔案的儲存位置並不難。也就是說，離線密碼猜測是攻擊者由密碼密文，以及處理密碼的演算法中，猜測出密碼明文的方法。雖然密碼是不定長度並且可能是一些無意義的字串，但如果配合社交攻擊法的猜測密碼明文，也是一種值得嘗試的攻擊法。

雖然克服離線密碼猜測最根本的關鍵在於密碼的處理過程，但隨著應用系統的特性，對於密碼的處理方式也有不同的做法，以下介紹幾種密碼處理方法。

13-2-4 密碼處理技巧

將使用者所輸入的密碼經過特殊處理之後，再儲存於系統檔案裡；至於密碼處理並不侷限於隱密性功能而已，也可以作為系統與使用者之間的認證工具，通常我們將處理過的密碼稱為『共享密鑰』或『主密鑰』。又隨著主密鑰可能扮演的角色，也會牽涉到密碼的處理技巧；譬如，若主密鑰扮演加密鑰匙的功能時，則必須依照所採用的加密演算法，去決定主密鑰的長度，以下介紹幾種密碼的處理技巧。

【（A）單向加密演算法】

所謂『單向加密演算法』（One-way Encryption）係利用加密技巧來達到雜湊演算法的功能；做法是將密碼當作加密鑰匙，以某一固定的數值（大多取 0）作為明文輸入，如此連續加密多次所輸出的密文既是主密鑰。如 Unix 就是採用 DES 加密演算法處理密碼的隱密工作，它將使用者所輸入的密碼作為加密鑰匙（8 個 7 位元的 ASCII 字元，即 56 個位元），並以 64 位元的 “0” 作為明文，連續重複加密 25 次後所得到的密文，作為密碼的儲存（或主密鑰，如圖 13-6 (a)）。經過 25 次加密處理之後，要

破解密文的確非常困難，但是經過 25 次運算相對耗費許多系統時間；另一種改進方法是加入『鹽』（Salt）的做法（如圖 13-6 (b) 所示），將『鹽』當作明文的輸入，再以密碼為加密鑰匙計算出儲存密碼；如果針對每一個使用者給予不同數值的『鹽』，想破解它可就不容易，如此，可以減少重複計算的次數，以提高系統效益。

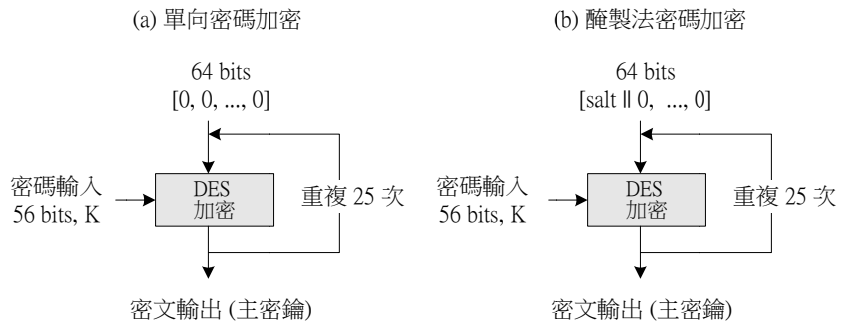


圖 13-6 單向加密法的密碼處理

【（B）單向雜湊函數】

除了利用加密方法來達到隱密性功能外，採用『單向雜湊函數』（One-way Hash Function）也是很普遍的方法；最常見的演算法是 MD4、MD5 與 SHA-1 等方法。簡單的做法是將密碼直接經過演算法計算後，所得到的雜湊碼作為密文儲存(或主密鑰)；也可以將密碼加入『鹽』之後，再經過雜湊演算法處理，如圖 13-7 (a) 與 (b) 所示。

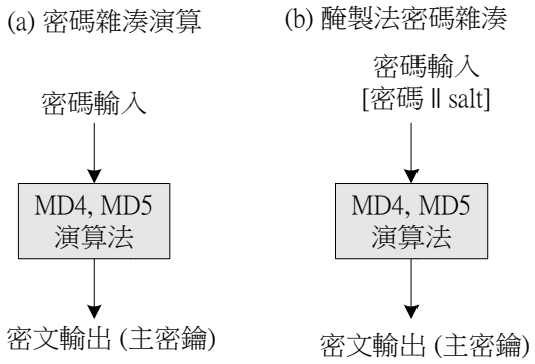


圖 13-7 單向雜湊演算法的密碼處理

【(C）一次密碼】

許多 Internet 網路應用(如 Telnet、FTP)是延伸 Unix/Linux 系統而來。早期 Unix 系統是被設計成多人使用環境，使用者可於系統的主控台或終端機上登入。隨著網路的普及，使用者也可以透過網路登入系統（遠端登入）。在這種情況之下，使用者所登入

的電腦（終端機或工作站）並不處理密碼，而祇將密碼以明文方式傳送給系統主機，如此說來，處理密碼完全是主機的工作，不需要與工作站協議如何處理密碼，所以工作站與主機之間就沒有完整的認證協定可依循。

如此說來，某些網路應用上還是需要以明文方式傳遞密碼，盜取者想要竊取密碼是一件容易的事。更何況目前無線網路風行，攻擊者並不需要網路連線，只要一台訊號接收器便可輕易收集到所希望的密碼。為了克服密碼明文傳送的方法，唯有將密碼使用的次數減到最低，所謂的『一次密碼』（One-time Password），顧名思義，每一個密碼祇使用一次。

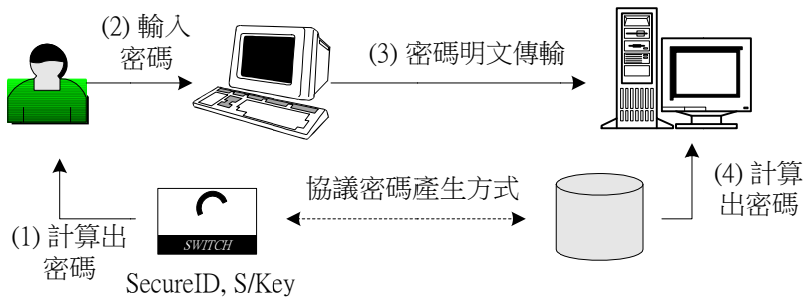


圖 13-8 一次密碼的運作程序

圖 13-8 為一次密碼產生的方法；使用者擁有一張小卡片或計算機，其中內建了一個預先程式化的認證函數、以及持有者的序號。當使用者欲登入系統時必須使用這張卡片，當輸入卡片的密碼之後，才會輸出一個當次使用的密碼，此密碼的產生是依據時間與一個秘密函數或序號進行某一種運算。使用者由卡片密碼與所計算出來的數字組合成為一個通行的密碼，然而這些計算方式必須與主機的計算方式相同，如此一來，主機與使用者之間每次認證時便可採用不同的密碼（運作方式如圖 13-8 的訊號次序）。這種密碼產生器的內部演算法是不會公開的，目前使用較普遍的是 Security Dynamics 公司所發行的 SecureID 卡。

另一種做法是類似密碼簿的方法。使用者持有一個 IC 卡，卡內建有一序列的密碼順序，係由主機系統下載而來的，所以主機與 IC 卡儲存相同的密碼序列。當使用者登入系統時，便依照密碼系列的順序來選擇密碼（電腦會自動選取），選用過的密碼便將其刪除；使用者也是利用卡片的密碼與選用的密碼組合成一個祇使用一次的通行密碼，目前使用較普遍的密碼簿是 Bellcore 公司所發行的 S/Key 卡片。

13-3 單向認證協定

『單向認證』（One-way Authentication）表示只能夠確認某一方的身份而已，一般是指回應者（如伺服器）確認發起者（如使用者）身份的認證協定。單向確認大多使用於系統登入方面的應用，至於電子商務方面，因為大多需要雙方的認證，所以不適合使用單向認證；儘管如此還是有很多應用系統僅採用單向認證來辨別要求服務者的身份。以下分別來討論兩種單向認證協定。

13-3-1 單向共享密鑰認證

在一般主從式（Client/Server）應用系統之下，使用者大多需要在伺服器上建立一個帳戶，作為登入伺服器時使用，系統會利用使用者密碼建立了一個與使用者共享的『共享密鑰』（Shared Secret），每一個使用者分別與伺服器享有一個獨一無二的共享密鑰，這些觀念在前一小節已詳細介紹過（如圖 13-5 所示）。接下來，我們探討雙方如何利用此共享密鑰來認證身份。

就前一節的介紹，我們不希望將密碼(無論明文或密文)直接在網路上傳輸，然而，要如何驗證雙方所持有的共享密鑰是相同的。一般是由回應者（伺服端）擔任確認的工作，以確定發起者(客戶端)是自己的用戶沒錯，至於確認對方共享密鑰的方法，大多採用『盤問/回應』（Challenge /Response）認證協定，其又有下列幾種做法：

【（A）明文盤問】

圖 13-9 為明文盤問機制，雙方所持有的共享密鑰為 $K_{Alice\text{-}Bob}$。首先，發起者(Alice)向回應者（Bob）聲明身份（訊號 (1)）；Bob 收到對方聲明身份之後，便發送一個『盤問』（Challenge，由亂數 R 所構成）資料給 Alice（訊號 (2)）；Alice 收到盤問資料之後，將共享密鑰與 R 經過某一個演算法計算（$f(K_{Alice\text{-}Bob}, R)$）之後，回傳給 Bob（訊號 (3)）；最後 Bob 以自己所持有的共享密鑰與 R 代入相同的演算法計算，所得的值與 Alice 傳送過來的值比較是否相同，即可確定發起者是否為 Alice。

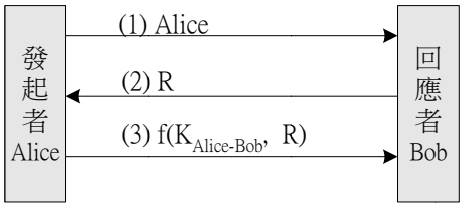


圖 13-9 明文盤問的共享密鑰認證

其中計算 $f(K_{Alice-Bob}, R)$ 可以是雜湊演算法（如 MD4、MD5、HMAC）或加密演算法（如 DES），主要能將共享密鑰植入亂數 R 中即可，並非真的要隱藏亂數 R，因此，只要雙方協議好採用何種演算法即可。明文盤問協定的特性如下：

- ◈ 僅回應者可以利用共享密鑰來確認發起者的身份，而發起者祇能以對方網路位址去確定所欲通訊者的身份。
- ◈ 攻擊者可以利用 IP 位址偽裝成回應者（如 Bob），以詐騙發起者（如 Alice）。
- ◈ 攻擊者可以由網路上竊聽到盤問明文與回應密文，並以所得到的訊息破解出共享密鑰（因為演算法是公開的）。
- ◈ 攻擊者可以入侵到回應者的資料庫中，尋找出發起者（如 Alice）的共享密鑰；只要能取得共享密鑰，完全不需要去破解出密碼，攻擊者照樣可以偽裝成發起者或回應者（Alice 或 Bob）。
- ◈ 如果能取得共享密鑰，也有可能由共享密鑰破解出密碼；雖然很困難，但也不能說不可能。

值得注意的是，各種證認協定都有其優劣點，鮮有十全十美的認證協定；至於應該採用何種協定來實現，完全視應用系統安全性的考量而定。也就是說，一般安全性較高的認證協定，可能需要的運作程序較複雜，至於一些簡單的系統，能省則省。

【（B）密文盤問】

圖 13-10 為另一種變形，回應者利用加密後的密文盤問發起者，可以稍微改善明文盤問的缺點。回應者（如 Bob）先利用共享密鑰將盤問訊息（亂數，R）加密，之後再傳送給發起者（如 Alice）；如果發起者能將其解密回原來的明文，便能確定它所持

有的密鑰與回應者相同，同時可以確定發起者的身份（真的是 Alice）。密文盤問的特性如下：

- ◈ 回應者必須採用可逆的加密演算法（$K_{Alice-Bob}[R]$），而不可以使用單向加密或單向雜湊演算法；亦是，加密後的密文是可以反解密成明文。
- ◈ 攻擊者可由網路竊取亂數的明文與密文、以及公開的加密演算法，從中可以破解出雙方共享密鑰（$K_{Alice-Bob}$），這要比由單向雜湊碼中破解共享密鑰容易得多了。
- ◈ 攻擊者無法偽裝成回應者的身份（利用網路位址），但可偽裝成發起者以騙取更多的盤問密文。

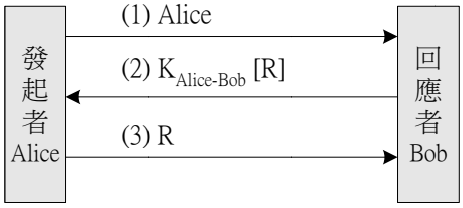


圖 13-10 密文盤問的共享密鑰認證

【（C）時間戳記盤問】

前面兩種認證協定最大的缺點，是需要一個盤問的明文（R）在網路上傳輸；如果能利用雙方都認定的亂數，並且可以隨時改變其數值，以取代這個亂數便能解決一部份問題。有一種可能，倘若雙方時序能達到同步的話（接近於同步，至少不要相差太遠），便能利用隨時變動的『時間戳記』（Timestamp）來取代亂數 R，其運作情形如圖 13-11 所示。圖 13-11 (a) 係採用加密演算法，發起者（如 Alice）將共享密鑰作為加密鑰匙，向當時的時間戳記加密，再傳送給回應者（如 Bob），回應者同樣利用共享密鑰向該訊息解密，再比較加密後的時間戳記與自己當時的時間戳記之間的差異，倘若兩者相差在容許的範圍之內，表示回應者可以確認發起者的身份（Alice）。

圖 13-11 (b) 係採用雜湊演算法將共享密鑰植入時間戳記裡。發起者將時間戳記與共享密鑰一起加入雜湊演算法中計算，之後將所得到的雜湊值與時間戳記傳送給回應者。回應者首先比較所收到的時間戳記是否在可容許範圍，如果可以的話，再以同樣的演算

法計算出雜湊值，其中會用到發起者的時間戳記與自己擁有的共享密鑰；如果收到的與自己計算的雜湊值相同的話，便可以確定發起者的身份。

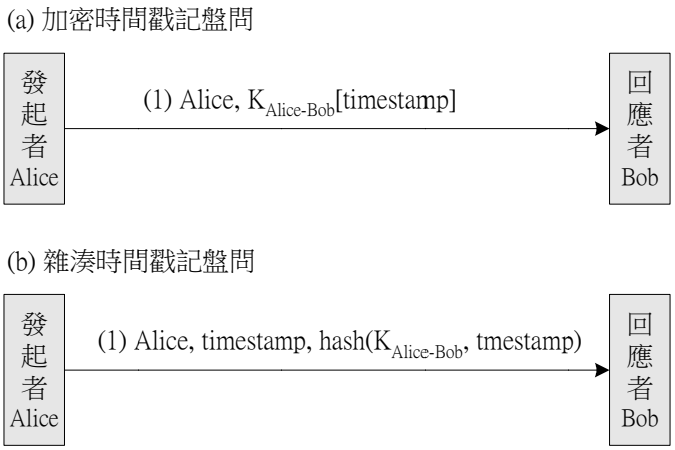


圖 13-11 時間戳記的共享密鑰認證

當然囉！攻擊者也可以將它的時序調整到回應者可以接受的範圍，再來攫取時間戳記的密文，這種破解技巧大多與前面兩種協定相同。但簡化的協定比較難觀察出正從事於身份認證的訊息（只有一筆訊息），這對整個運作程序而言，比較有隱密性的功能。另一方面，加入時間戳記亦可防禦重播攻擊，接下來會介紹到。

13-3-2 單向公開鑰匙認證

許多系統利用公開鑰匙認證對方的身份，圖 13-12 就是利用公開鑰匙達成單向認證的運作協定；執行該協定之前，發起者（如 Alice）必須擁有公開鑰匙配對 $\{K_{Ra}, K_{Ua}\}$，其中前者為私有鑰匙，後者為公開鑰匙，回應者（如 Bob）必須取得 Alice 的公開鑰匙（K_{Ua}）。公開鑰匙的單向認證有下列兩種實現方法。

【（A）私有鑰匙回應】

圖 13-12 (a) 為私有鑰匙回應的運作程序。Alice 發送帳戶名稱之後（訊號 (1)），Bob 利用一個亂數 R 來盤問 Alice（訊號 (2)），接下來，Alice 利用自己的私有鑰匙向該盤問訊息簽署（E_{KRa}[R]，假設使用加密方法），再回傳給 Bob，最後 Bob 再利用對方的公開鑰匙（K_{Ua}）驗證該簽署碼是否正確，如果正確的話，則表示對方確實是 Alice 無誤。

基本上，盤問訊息都是 32 位元的亂數，大多無需採用加密演算法，僅採用數位簽章技巧（如 DSS）即可，所以發起者將 32 位元的亂數簽署後得到一個簽署碼，再回傳給回應者，而回應者則利用另外一把鑰匙認證該簽署碼是否正確。

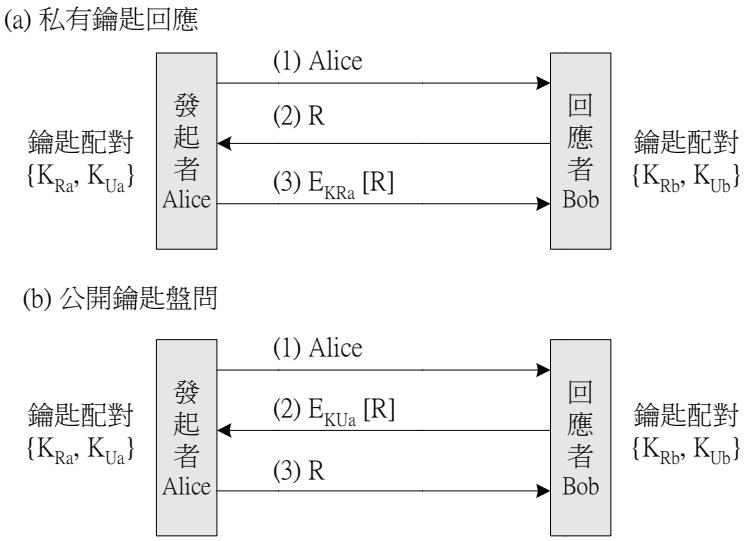


圖 13-12 公開鑰匙的單向認證協定

【（B）公開鑰匙盤問】

圖 13-12 (b) 是公開鑰匙認證的另一種變形。回應者（如 Bob）利用發起者的公開鑰匙（K_{Ua}），向亂數 R 加密之後（$E_{KUa}[R]$），再以此密文來盤問發起者（如 Alice）；如果發起者能用自己的私有鑰匙向該密文解密回明文的話，則回應者便能確定它的身份無誤（真的是 Alice）。這種認證協定必須採用可逆的加密演算法，不可以採用單向的數位簽章技巧，否則發起者無法將盤問密文解密回原來的明文。

看起來，公開鑰匙認證協定好像比較安全，問題是如果分配與儲存公開鑰匙。如果回應者（伺服器）系統內儲存許多使用者的公開鑰匙，攻擊者只要入侵到系統內修改這些公開鑰匙，便可以達到欺騙回應者（伺服器）的手段，何況這些公開鑰匙也不是什麼機密。因此，利用公開鑰匙達到單向認證，通常需要其他機制的輔助才行，譬如，數位憑證或 PKI 系統等等（第七、九章介紹）。

13-4 相互認證協定

『相互認證』（Mutual Authentication）表示通訊雙方可以互相確定對方身份的協定。相互認證在電子商務上非常的重要，入侵者可能冒充客戶身份來詐騙伺服器，也可能冒充伺服器來詐騙客戶。如何達成相互認證的措施，同樣可區分為秘密鑰匙系統與公開鑰匙系統兩種機制，以下分別說明之。

13-4-1 相互密鑰認證 – 運作

不同於單向認證機制，僅回應者確認發起者所持有鑰匙與自己的相同，而相信對方身分。相互認證則必須雙方能確定彼此持有的共享鑰匙是否相同，再相互確認身分。基本上，還是利用『盤問/回應』（Challenge/Response）機制便可達到雙方認證的功能。如圖 13-13 是相互共享密鑰認證的運作程序，在未通訊之前雙方必須持有共享密鑰（K_{AB}），而此密鑰大多是發起者的密碼所計算而來。在圖 13-13 的訊號 (2) 與 (3) 是由 Bob （大多是伺服端）盤問 Alice 的共享密鑰，而訊號 (4) 與 (5) 是 Alice（大多是客戶端）盤問 Bob。其中 $f(K_{AB}, R)$是雙方將共享密鑰植入盤問亂數（R_1、R_2）演算法，它可以是雜湊演算法（如 MD5 或 HMAC）或加密演算法（如 DES）。

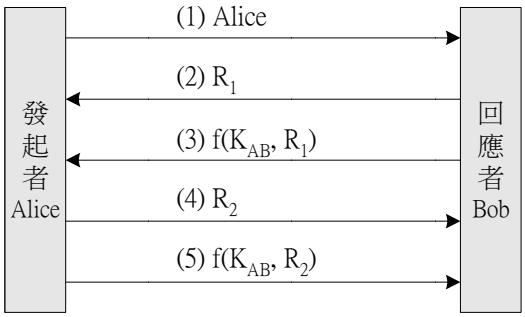


圖 13-13 相互共享密鑰協定的運作程序

【（A）簡化運作程序】

我們可以簡化圖 13-13 的運作程序，將訊號 (1) 與 (4) 結合成一個訊號，亦將訊號 (2) 與 (3) 整合在一起，如圖 13-14 所示。當 Alice 聲明自己身份時，並攜帶這盤問訊號 R_2（圖 13-14 訊號 (1)），同時 Bob 回應 Alice 時，也攜帶另一個盤問訊號（R_1）來測試對方（訊號 (2)）。

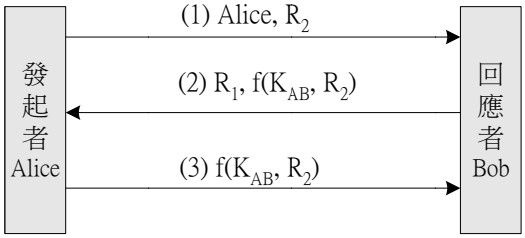


圖 13-14 簡化的共享密鑰運作程序

【（B）反射攻擊法】

使用簡化的共享密鑰協定容易遭受『反射攻擊法』（Reflection Attack）擊破。如圖 13-15 所示，例如 Trudy 由其它管道知 Alice 是 Bob 的客戶端名稱，它就可以假冒 Alice 的名字向 Bob 發出盤問要求。首先 Trudy 利用 Alice 的名字發出盤問訊號 R_2（訊號 (1)）。Bob 即時送出回應訊號 $f(K_{AB}, R_2)$，並送出自己的盤問訊號 R_1 給 Trudy（訊號 (2)，誤認為是 Alice）。Trudy 再利用 Bob 的盤問訊號 R_1 再要求盤問 Bob（訊號 (3)）。Bob 會誤認為 Alice 想要建立另一個連線而要求盤問。而且盤問訊號是一個亂數，很難去偵測自己是否曾經送出同樣的亂數；因此 Bob 再回應一次，利用雙方的秘密鑰匙對 R_1 加密，同時再送出自己的盤問訊號（訊號 (4)）。這時候，Trudy 直接取 Bob 回應之 $f(K_{AB}, R_1)$當作訊號 (2) 的回應，再傳送給 Bob（訊號 (5)）。接下來，Bob 解密之後得到的亂數 R_1，與訊號 (2) 的 R_1 相同，便相信發起者是 Alice，其實是 Trudy 冒充的。

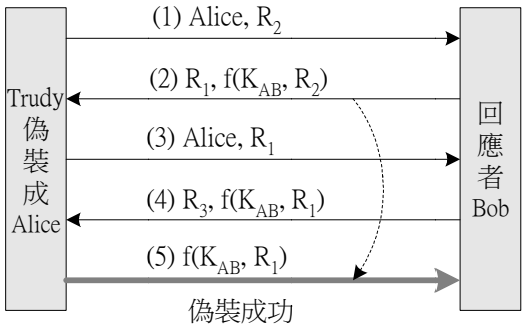


圖 13-15 反射攻擊法的運作程序

當然 Bob 可限制每一個人一次只能要求一條連線，如此就可以避免入侵者的反射攻擊。但如果發生第一次的回應訊號遺失，而對方要求再盤問重新連線可能會被拒絕，

除非第一次要求認證時能確定身份，否則這個協定很容易被擊破。我們將會介紹其它協定以彌補這方面的缺憾。

13-4-2 相互密鑰認證 – 改良

簡化的相互共享密鑰認證也容易遭受『密碼猜測』（Password Guessing）攻擊。攻擊者只要連續發出聲明身份及盤問亂數（如圖 13-14 訊號 (1)，Alice, R_1），便可以由回應者得到該處理盤問亂數的密文（$f(K_{AB}, R_1)$），且得到亂數的明文與密文配對（$\{R_1, f(K_{AB}, R_1)\}$。如果訊息取得足夠的話，攻擊者就可找出共享密鑰（K_{AB}），無論產生密文的演算法是何種機制（雜湊演算法或加密演算法）。我們稍加修改圖 13-14 的運作程序，就可以避免密碼猜測攻擊，如圖 13-16 所示，Alice 聲明身份時，並不攜帶盤問亂數（訊號 (1)），再由 Bob 盤問 Alice（訊號 (2)，R1）；Bob 可以確定對方身份無誤之後（利用訊號 (3)，$f(K_{AB}, R_1)$），再回應 Alice 的盤問（訊號 (4)，$f(K_{AB}, R_2)$）。如果，Bob（一般都是伺服器）無法確定 Alice 身份時，並不會回應它的盤問，如此一來，攻擊者就無法連續得到亂數與密文的配對了。

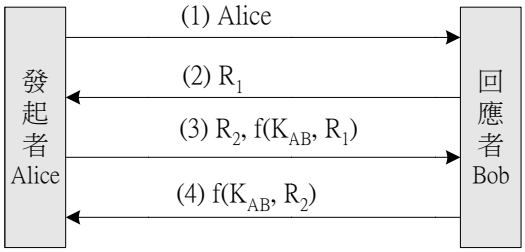


圖 13-16 改良型的共享密鑰認證

13-4-3 相互公開鑰匙認證

利用公開鑰匙達到雙方的認證程序可就簡單多了，但先決條件是雙方必須有公開鑰匙與私有鑰匙配對，並且公開鑰匙要能安全的分配給通訊對方。分配公開鑰匙並沒有想像中那麼簡單，本書已在第四章討論過，不再重複，一般都需要數位憑證或 PKI 系統來輔助（將在第九章介紹），這裡僅討論它的運作程序。

圖 13-17 為相互公開鑰匙認證的運作程序，假設發起者（如 Alice、客戶端）的鑰匙配對為 $\{K_{Ua}, K_{Ra}\}$，回應者（如 Bob、伺服端）為 $\{K_{Ub}, K_{Rb}\}$。首先，Alice 聲明自己身份，並利用 Bob 的公開鑰匙向亂數 R_1 加密，再傳送給 Bob（訊號 (1)、

$E_{KUb}[R_1]$）。Bob 利用自己的私有鑰匙解密盤問亂數（R_2），並利用 Alice 的公開鑰匙向亂數 R_2（Bob 隨機取出）加密，再一起回傳給發起者（訊號 (2)、$E_{KUa}[R_2]$）。Alice 驗證所盤問的亂數(R_1)是否相同，如相同的話，則可以確認對方身份(是 Bob 無誤)；接下來，利用自己的私有鑰匙向盤問亂數解密（R_2），並回傳給 Bob。Bob 收到 Alice 的回應亂數（R_2），比較是否與自己產生的相同，如果相同的話，便可以確定發起者身份（是 Alice 無誤）。

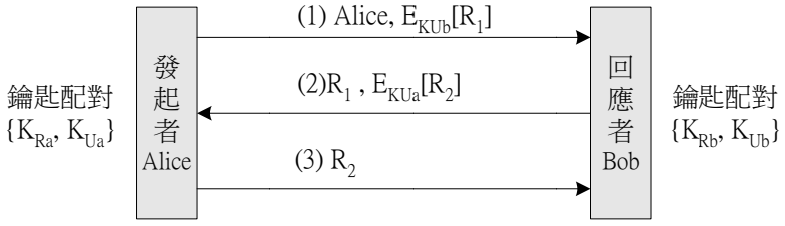


圖 13-17 相互公開鑰匙認證的運作程序

13-5 集中式用戶認證

13-5-1 集中式用戶認證簡介

『鑰匙分配中心』(Key Distribution Center, KDC)[74] 大多是使用於封閉性網路（或組織單位的網路）的應用系統之中，負責分配使用者的秘密鑰匙(Secret Key)，所以使用者持有此秘密鑰匙才可以存取系統內的網路資源。為何需要 KDC 來分配鑰匙，這是值得深思的問題，我們分三個部分來介紹 KDC 的概念。

13-5-2 主機使用者的確認

『帳號/密碼』是一般伺服系統中認證使用者身份最基本的方法，如先前所介紹的，使用者利用密碼建立與伺服器之間的溝通，雙方身份確認的方法，是以共享密鑰為基礎。然而，就多個使用者而言，主機與每一使用者都必須建立一把獨立的共享密鑰，如此說來，主機本身必須維護多把的共享密鑰，而且每個使用者的資源存取權限不盡相同，主機通常需要建立一個『存取控制表』(Access Control List, ACL)(如圖 13-18 所示)；其記錄方式又分兩種，一種是資源可讓那些使用者使用，另一種是使用者可存取那些資源，目前大多採用前者。有了 ACL 規劃使用者權限之後，就可以進一步記錄所有使用者存取資源的情形，如此便可達到『稽核』(Audit)的功能了。

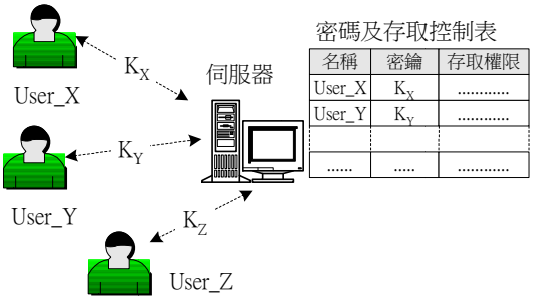


圖 13-18 使用者與伺服器之間的共享密鑰

13-5-3 工作群組使用者的確認

除非使用者的數目非常多，否則主機對使用者的管理多半不會衍生特殊問題，但如果應用環境存在眾多伺服器（或主機）時，問題就不是那麼簡單了。基本上，使用者在每一個不同的伺服器上都必須建立帳號，才可以使用該伺服器的資源，相同的道理，每一個伺服器也必須建立一個 ACL 管理使用者，如此一來，多部伺服器與多個使用者便衍生許多帳戶管理上的困難，如圖 13-19 所示。

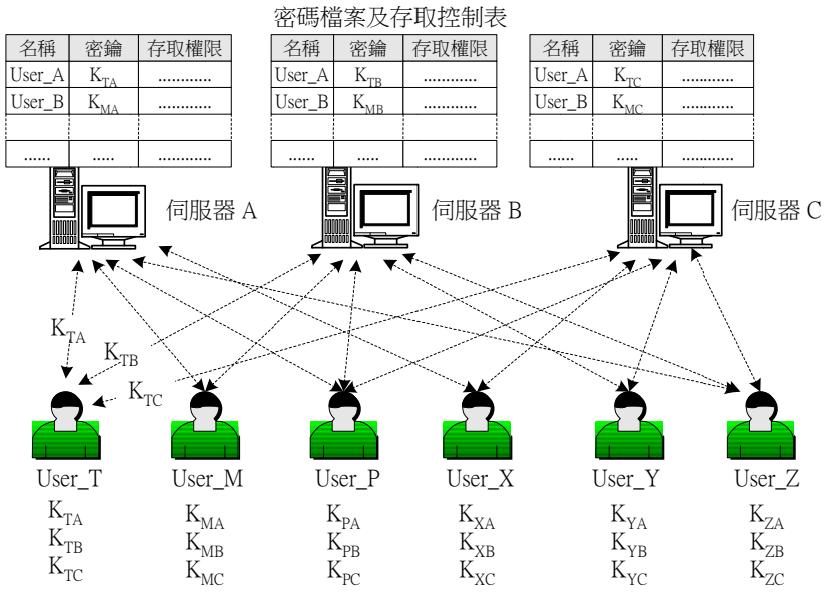


圖 13-19 多個伺服器與使用者的管理

基本上，我們會將工作性質相同的伺服器（或主機）組合成一個『工作群組』（Workgroup），方便使用者找尋所需要的資源所在之處，當然使用者必須在伺服器上建立帳戶始可存取。當系統環境還不是很複雜的情況下，管理者分別於各個伺服器上建立使用者帳戶尚可以承擔，當然還必須在每一伺服器上建立存取控制表。一旦系統環境變得很大的時候，系統管理就變得瑣碎且複雜，恐有賴其他機制來輔助始可行。

13-5-4 網域使用者的確認

我們可以發現，將工作性質相同的伺服器（或工作站）整合成一個工作群組，亦不能滿足大型應用系統的需求。如果，我們可以將工作性質相同的使用者與伺服器整合成一個群組，或許就能解決大部分的困難，這就是『網域』（Domain，Kerberos 稱為 Realm）的概念。在一個網域之下，它將工作性質相同的使用者與伺服器（與工作站）組合在一起，並給予一個網域名稱（Domain Name），每一網域內至少必須存在一部『網域控制台』（Domain Controller, DC），DC 管理該網域下所有使用者與電腦（伺服器或工作站）。每一使用者也必須在 DC 上建立一個『網域使用者』帳戶，才可以存取網域上的資源。基本上，DC 負責管理使用者帳戶，至於存取控制表的維護還是保存在各個伺服器上，所以 DC 負責登入的工作，伺服器自己則管理存取控制，其運作程序如圖 13-20 中訊號號碼的次序。

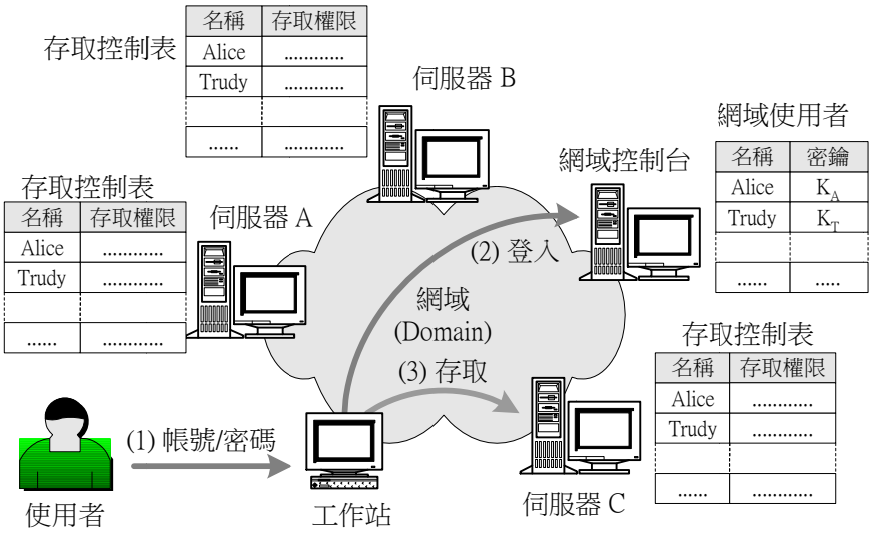


圖 13-20 網域的概念

這樣一來，使用者的共享密鑰僅建立在與 DC 之間，而非在所有的伺服器上。當使用者與 DC 之間以某一種機制（單向或相互認證）確定身份之後，它便可以取得通往伺服器的『鑰匙』，再利用這把鑰匙與伺服器通訊，這就是『鑰匙分配中心』（Key Distribution Center, KDC）的概念。簡單的做法是，DC 確定使用者身份與所欲通訊對象之後，分別給予使用者與伺服器一把『會議鑰匙』（Session Key），他們之間再利用這把鑰匙來通訊。

13-5-5 網域控制台 – Win NT

圖 13-21 為 Windows NT 網域的認證程序（Windows 2000 已採用 Kerberos 系統）。使用者於工作站上輸入帳號與密碼要求登入網域（訊號 (1)、(2)），工作站與 DC 之間利用『盤問與回應』協定確定身份之後，DC 會給予一個『符記』（Token）給工作站（訊號 (3)），工作站取得符記後，便可通往所欲存取的伺服器（訊號 (4)）。如此說來，DC 僅扮演著鑰匙分配中心（KDC）的功能，至於使用者如何登入 KDC？KDC 如何分配鑰匙？又使用者與伺服器如何共同確認 KDC 所分配的鑰匙？這些問題待下一節再說明。

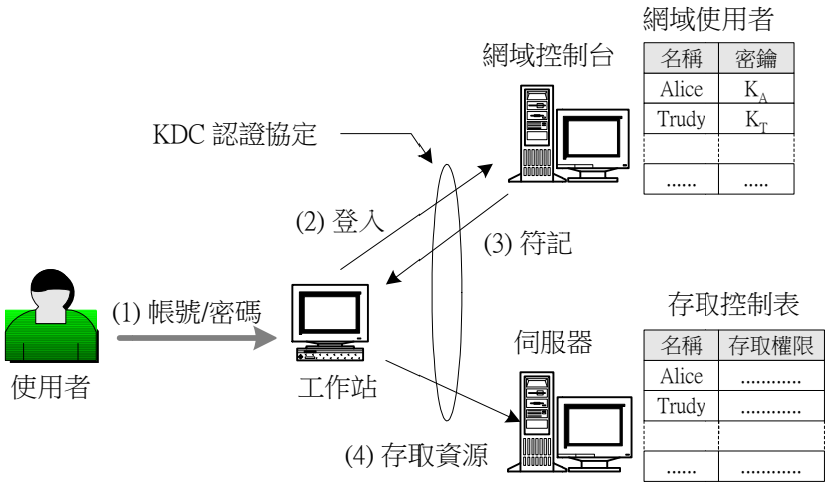


圖 13-21 Windows NT 網域認證