NCS-1999 使用 IC 卡的身份認證系統之研究

Design of Authentication Systems with IC Cards

薛如珍
元智大學電機暨資訊工程研究所
中壢市遠東路 135 號
giordano@www.mmlab.cse.yzu.edu.tw

黃士殷
元智大學電機暨資訊工程研究所
中壢市遠東路 135 號
shyhin@saturn.yzu.edu.tw

摘要

Kerberos 是改自 Needham-Schroeder protocol 的一 authentication 架構，原是 MIT 中 Athena 專案的一部分發展而來，提供認證（authentication）、完整性（integrity）、機密（confidentiality）、以及授權（authorization）服務。
然而 Kerberos 仍有一些常被提出的弱點如 authentication server 未先確定使用者身份、及容易遭受 dictionary attack 等，許多文獻提出以 Public Key 技術加強之，但往往使用過多 Public Key 而帶來 performance 上的問題（Public Key 與 Symmetric Key 系統相較下，前者使用的時間是後者的許多倍，這是 Public Key 的一大缺點）。
本論文希望配合 Smart Card、Public Key、及 One-time password technique 等技術，對 Kerberos 加以改進，在安全性、方便性、執行效率等考慮點上取得最佳平衡，達到一更完善之 authentication 模型。

關鍵字：身份認證、 Kerberos、 智慧卡、One-time password technique

1. 簡介

Kerberos 是由 MIT 中 Athena 研究計畫的一部份發展而成的 authentication 系統，適用於 Client-Server 模式中，經由值得信任的中央識別伺服器爲其它伺服器提供識別用戶的服務，並爲用戶識別伺服器。Kerberos 目前已發展到 version 5。

然而此系統也有其弱點，其中最常被提到者在於其第一階段沒有經過 mutual authentication 及使用 password 機制（以使用者 password 經過一 one-way hash function 處理而成 symmetric key），雖然爲使用者提供方便及安全儲存方式（password 只記在使用者腦中），卻容易造成 off-line 的 dictionary attack。另一方面，在 key 的管理上也有所不便，使用者必須先到 authentication server 登記 password 的 hash 值以作爲兩者間的 secret key。

本論文希望利用 IC 卡的安全特性作爲安全儲存體，將 Public Key 加解密機制結合入 Kerberos 系統，以期改進上述缺點而做到 mutual authentication、防止 dictionary attack、及幫助 key 的管理。然而，public key 常被提到 performance 方面的問題（public key 加解密使用的時間遠比 symmetric 加解密系統多），故本論文希望能提出方法改進上列缺點，並在 performance 上提昇。

2. Kerberos 簡介

一般來說，網路中的各個伺服器在提供服務之前，會依個別對存取控制的需求訂出各自的識別程序，然而，對於用戶將造成不便，使用者必須對不同的伺服器識別系統記下不同的 key，如何管理保存這些 keys 將對使用者造成困擾。

Kerberos 利用值得信賴的中央識別伺服器對網路使用者及伺服器提供識別服務，使用者只須保存一把與中央識別伺服器溝通的 key，便可與所有建立在此架構下的伺服器溝通。它基本上運用 Symmetric Encryption（DES）技術，參與的使用者、伺服器必須先到 KDC（key distribution center）完成註冊手續，與 KDC 分享一把秘密的 secret key。

2.1 Kerberos V4

Kerberos 系統包含以下角色：

User：一般使用者

Client：User 登入的使用者端系統

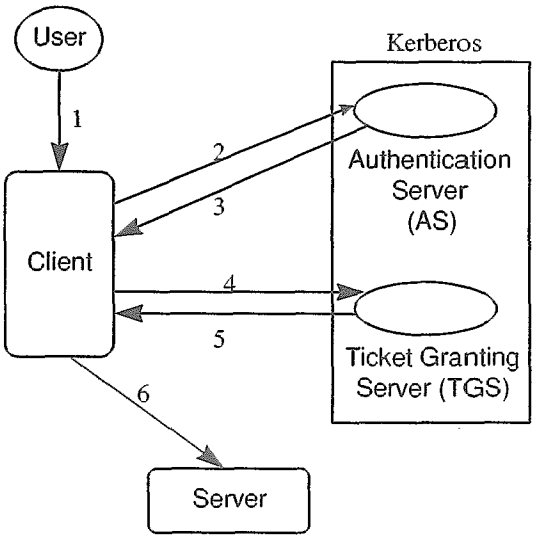
Kerberos：可信任的中央識別系統，其中包含兩種角色

Authentication Server（AS）：爲使用者產生 Ticket-Granting Ticket（TGT），供使用者進入 TGS

Ticket Granting Server（TGS）：爲使用者產生真正用來進入 service server 的 service Ticket

Server：提供 service 的 server

它的運作方式概述如下：



1) user 使用某台機器執行應用程式並輸入自己的 ID

及 password

2) client 端程式以明文向 AS 要求一張"身份證明"，以向某 TGS 證明自己的身份
3) AS 傳回這張以 AS ↔ User 之間的 secret key 加密的"身份證明"，其中包含一張 Ticket（TGT）及一個 temporary session key1

＊TGT 以 AS ↔ TGS 之間的 secret key 加密，並包含同一把 temporary key1，以供往後 User ↔ TGS 之間使用。

＊AS ↔ User 之間的 secret key：User's secret password 經過 hash 產生

4) Client 向 User 要求其 secret password，hash 處理後解開上一階段之"身份證明"，並以取出的 TGT 及 temporary session key1 向 TGT 證明自己的身份（因爲有 secret password 解開"身份證明"）並要求對某 service server 的"身份證明"。
5) TGS 解開 TGT 取出 temporary session key1 驗證 Client 身份。TGS 傳回以 temporary session key1 加密的"身份證明"，其中包含一張 service Ticket 及一個 temporary session key2。

＊service Ticket 以 TGS ↔ Server 之間的 secret key 加密，並包含同一把 temporary key2，以供往後 User ↔ Server 之間使用。

6) Client 以 temporary session key1 解開上一階段之"身份證明"，並以取出的 service Ticket 及 temporary session key2 向 Server 證明自己的身份，開始與 service server 溝通。

Kerberos V4 Message Exchange 格式：

ID_c ：使用 Client 的 User identity

ID_{tgs} ：user 要進入的 TGS 之 id

E_K ：以 key：k 加密{ }內的訊息

K_c ：AS ↔ User 之間的 secret key，由 User's secret password 經過 hash 產生

K_{tgs} ：AS ↔ TGS 之間的 secret key，用以加密 TGT

K_v ：TGS ↔ Server 之間的 secret key，用以加密 service ticket

$K_{c,tgs}$ ：AS 產生之 temporary session key，供往後 User ↔ TGS 之間使用。

$K_{c,v}$ ：TGS 產生之 temporary session key，供往後 User ↔ Server 之間使用。

Ticket ：以 AS ↔ TGS 或 TGS ↔ Server 之間的 secret key 加密以傳送 temporary session key，可重複使用直到 lifetime 結束。

Authenticator：以取出之 temporary session key 加密一些資訊，用以證明使用者身份，不可重複使用，其中之 TS（timestamp）每次更新以防止 reply 攻擊。

Authentication Service Exchange：to obtain TGT

$$(1)C \rightarrow AS: ID_c \| ID_{tgs} \| TS_1$$

$$(2)AS \rightarrow C: E_{K_c}\{K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}\}$$

$$Ticket_{tgs} = E_{K_{tgs}}\{K_{c,tgs} \| ID_c \| AD_c \| ID_{tgs} \| TS_2 \| Lifetime_2\}$$

(B) Ticket-Granting Service Exchange：to obtain service ticket

$$(3)C \rightarrow TGS: ID_v \| Ticket_{tgs} \| Authenticaor_C$$

$$(2)TGS \rightarrow C: E_{K_{c,tgs}}\{K_{c,v} \| ID_v \| TS_4 \| Ticket_v\}$$

$$Ticket_v = E_{K_v}\{K_{c,v} \| ID_c \| AD_c \| ID_v \| TS_4 \| Lifetime_4\}$$

$$Authenticaor_C = E_{K_{c,tgs}}\{ID_c \| AD_c \| TS_3\}$$

Client / Server Authentication Exchange：to obtain service

$$(5)C \rightarrow V: Ticket_v \| Authenticator_c$$

$$(6)V \rightarrow C: E_{K_{c,v}}\{TS_5 + 1\}$$

$$Ticket_v = E_{K_v}\{K_{c,v} \| ID_c \| AD_c \| ID_v \| TS_4 \| Lifetime_4\}$$

$$Authenticator_c = E_{K_{c,v}}\{ID_c \| AD_c \| TS_5\}$$

只有 User 第一次 log in 系統需要打一次 password 取到 ticket-granting ticket（TGT），直到 User log out 或 TGT 過期。此後 User 需要使用其它 Server service 時只需以同一 TGT 再去與 TGS 換其 service ticket 即可。

其中最常被提出來的弱點在於第一階段 User 向 AS 提出要求的訊息是以明文傳送，AS 並未先對使用者進行身份識別，換句話說，任何一個使用者都可以向 AS 宣稱他是某人，而獲得他人的第一階段"身份證明"。

而真正做到身份識別的部分在於只有正確的使用者方有正確的 secret key（hash of password）可解開並使用此"身份證明"。

然而由於 secret key 是由 User 的 password 經 hash 而來，User 爲求記憶方便，很容易出現較 weak 的 secret key，而面臨 dictionary attack 的危機。

一旦攻擊者取得他人的第一階段"身份證明"，再加以 off-line 猜得它人 password，將可爲所欲爲直到使用者發現而更換密碼。

另外，由於 User 的 secret key 都記錄在 AS 的資料庫中，一旦 AS 被攻破，便是一大危機。

2.2 Kerberos V5

Kerberos V5 對 Kerberos V4 的一些缺點作改進並加上新的功能，定義在 RFC1510。以下是一些較爲重要的改變：

Encryption system：在 Kerberos V4 中採用 DES encryption system，而其出口限制造成一些困擾；在 Kerberos V5 中增加欄位，可標明使用之 encryption system，因此可使用任何 encryption technique。

Ticket lifetime：Kerberos V4 以 8-bits 表示 lifetime（每單位代表 5 分鐘，故 lifetime 最長只有 $2^8 \times 5 = 1280$ 分鐘（約 21 小時）對某些應用程式並不適當；Kerberos V5 採用 start time 及 end time 的方式表示 lifetime，較有彈性。

Authentication forward：Kerberos V5 提供"身份證明" forward 的功能，發給某一 Client 的"身份證明"能 forward 給另一 host，使其代表此 Client 的身份使用 service。

Cross-Realm authentication：Kerberos V4 中若有 N 個 realms，必須兩兩交換一把 secret key，共需[N(N-1)]/2 個 secure key exchanges 難以加入太多 realms；Kerberos V5 則加入階層式架構，只須與 parent 及 children 互換 key 即可。

preauthentication：可加入 preauthentication 資訊，讓 AS 事先確定 Client 身份以決定是否發出第一張"身份證明"，使攻擊者要猜測他人 password 更加困難，但無法完全防止（仍可由網路中竊取）。

Option、Flag：在 ticket 中加入 Flag，提供 forwardable、renewable、postdatable ticket。

Nonce：加入 nonce，並在 server 端記錄最近幾次的 nonce 以防止 reply attack。

Kerberos V5 Message Exchange 格式：

Authentication Service Exchange：to obtain TGT

$$(1)C \rightarrow AS : Options \| ID_c \| Realm_c \| ID_{tgs} \| Times \| Nonce_1$$

$$(2)AS \rightarrow C : Realm_c \| ID_c \| Ticket_{tgs} \| E_{K_c}\{K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}\}$$

$$Ticket_{tgs} = E_{K_{tgs}}\{Flags \| K_{c,tgs} \| Realm_c \| ID_c \| AD_c \| Times\}$$

(B) Ticket-Granting Service Exchange：to obtain service ticket

$$(3)C \rightarrow TGS : Options \| ID_v \| Times \| Nonce_2 \| Ticket_{tgs} \| Authentica\,tor_c$$

$$(4)TGS \rightarrow C : Realm_c \| ID_c \| Ticket_v \| E_{K_{c,tgs}}\{K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_v\}$$

$$Ticket_v = E_{K_v}\{Flags \| K_{c,v} \| Realm_c \| ID_c \| AD_c \| Times\}$$

$$Authentica\,tor_c = E_{K_{c,tgs}}\{ID_c \| Realm_c \| Times\}$$

(C) Client / Server Authentication Exchange：to obtain service

$$(5)C \rightarrow V : Options \| Ticket_v \| Authenticator_c$$

$$(6)V \rightarrow C : E_{K_{c,v}}\{Times \| K_{temp} \| Seq\#\}$$

$$Ticket_v = E_{K_v}\{Flags \| K_{c,v} \| Realm_c \| ID_c \| AD_c \| Times\}$$

$$Authenticator_c = E_{K_{c,v}}\{ID_c \| Realm_c \| Times \| K_{temp} \| Seq\#\}$$

3. 身份認證協定的設計

前面提到 Kerberos 最常被提出來的弱點在於 AS 與 User 間的 secret key 是由 User 的 password 經 hash 而來，User 爲求記憶方便，很容易出現較 weak 的 secret key，而面臨遭受 dictionary attack、密碼被猜到的危險。

Kerberos 另一弱點爲第一階段 User 向 AS 提出要求以明文傳送，AS 並未先對使用者進行身份識別，換句話說，任何一個使用者都可以向 AS 宣稱他是某人，而獲得他人的第一階段"身份證明"。（真正做到身份識別的部分在於只有正確的使用者方有正確的 secret key（hash of password）可解開並使用此"身份證明"。）

由以上兩點，攻擊者可以很輕易地取得他人的第一階段"身份證明"，再以 off-line 之 dictionary attack 猜得他人 password，冒充使用者直到使用者發現而更換密碼。即使 Kerberos Version 5 加入 preauthentication 資訊，讓 AS 事先確定 Client 身份以決定是否發出第一張"身份證明"，也只能增加攻擊者要猜測他人 password 的困難度，而無法完全防止（攻擊者仍可由網路中攔截，竊取到此"身份證明"再加以 dictionary attack）。

再則，使用者與 AS 間如何事先交換共通的 long term secret key（hash of password）要如何完成又是另一課題，最安全的方式通常是使用者親自前往向系統管理者登記，然而，每當使用者需要更換密碼時都必須親自跑一趟將造成使用者很大的困擾。

故本章希望以 Smart Card 的安全特性、加上 publick-key 加解密技術來改善 Kerberos 原有弱點，提出一加強架構。

在 Client 端程式及硬體是否可信任的考量下，本章提出兩種架構模型，當不考慮 Client 端的可信任度時（包括 Client 端的軟體程式及記憶體暫留等情況）可使用模型一的架構，否則，使用模型二（此架構使用較多 Smart Card 的運算將花費較多時間）。

爲考慮相容性，本章主要修訂部分在 Client 端及 Authentication Server（Authentication Service Exchange），其他部分（Ticket-Granting Service Exchange、Client / Server Authentication Exchange）通訊仍維持舊架構。

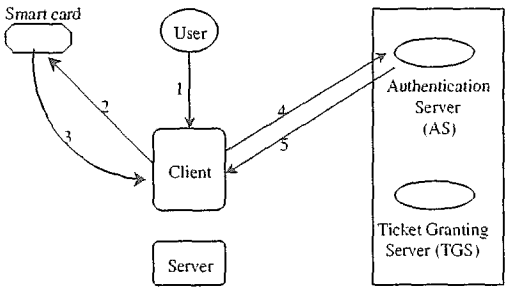
3.1 架構模型一：基本架構

本模型在不考慮 Client 端的可信任性下使用，只更動 Authentication Service Exchange 部分（Client 端及 Authentication Server 端）。

本節重點在於做到 Client 與 Authentication Server（AS）之間的 mutual authentication，並且防止 dictionary attack。

在 mutual authentication 方面，我們在 Client 送給 AS 的訊息中加入 Public Key 技術：首先以 User 的 private key 對部分資料加以簽名，使 AS 能夠事先確定 User（Client）身份，接著再以 AS 之 public key 加密，間接使 Client 確定 AS 的身份。

而在 dictionary attack 的防止上，原來 protocol 考慮 User 使用方便與安全性，採用記憶於 User 腦中的 password 爲 key（加以 Hash 處理），卻帶來了 dictionary attack 的危險（User 不可能記憶一串亂碼）；而在本節提出的架構中，我們以一個 Client 端臨時產生的 K_c 取代原來較弱的 password key 來防止 dictionary attack，並以 Public Key 技術加以簽名、加密，使 K_c 能與原來 password 具有同等地位代表 User，加上配合 IC Card 爲安全儲存體存放 User 的 private key，使用者仍然只需操作一組 password（IC 卡的 PIN）便可完成所有登入工作。本架構的細節說明如下：



1 user : ID_c, PIN
2 C→Sm: PIN, K_c, ID_c, $Realm_p$, ID_{tgs}, $Nonce_c$
3 Sm→C: $E_{K_{privC}}\{K_c, ID_c, Realm_p, ID_{tgs}, Nonce_c\}$
4 C→AS: $E_{K_{pubAS}}\{E_{K_{privC}}\{K_c, ID_c, Realm_p, ID_{tgs}, Nonce_c\}\}$,
$E_{K_c}\{Options, ID_c, Realm_p, ID_{tgs}, Times, Nonce_c\}, K_{pubC}$
5 AS→C: $Realm_p$, ID_c, $Ticket_{tgs}$, $E_{K_c}\{K_{c,tgs}, Flags, Times, Nonce_c, Realm_{tgs}, ID_{tgs}\}$

$Ticket_{tgs} = E_{K_{tgs}}\{Flags, K_{c,tgs}, ID_c, Realm_p, Times\}$

訊息說明：

1) User 打入自己的 identity、smart card 的 PIN(personal Identification Number)
2) Client 產生一把 temporary key K_c 及一 Nonce，並將之與 user 之 ID、PIN、欲溝通之 TGS ID 一起傳送給 SmartCard
3) SmartCard 以本身安全機制（PIN）驗證 User 身份無誤後，以 User 之 private key（Public-Key 技術）對適當資訊簽名，傳送回 Client。
4) Client 以 AS 之 public key 將簽過名的訊息加密，並將其它資訊以 K_c 加密，附上 User 之 public key（或 Certificate 之 issuer name + serial number）送給 AS 要求一張"身份證明"以進入 TGS
5) AS 以本身之 private key 解開 $E_{K_{pubAS}}\{E_{K_{privC}}\{K_c, ID_c, Realm_c, ID_{tgs}, Nonce_1\}$（Client 以此確定 AS 的身份），再以 User 之 public key 驗證 User 之簽名（AS 確定 User 之身份）。確定無誤後，解開加密部分並以 Nonce 確定非 replay，之後 AS 爲 Client 產生一張"身份證明"（包含爲 User ↔ TGS 產生一把 temporary key 及一張 $Ticket_{tgs}$）並以 K_c 加密 temporary key 等資訊傳回給 Client。
6) Client 收到後將加密部分解開，檢查 Nonce 等，若無誤則以 $K_{c,ths}$ 及 $Ticket_{tgs}$ 進行接下來的步驟。

接下來的 Ticket-Granting Service Exchange、Client / Server Authentication Exchange 維持原來的訊息交換架構

3.2 架構模型二：安全架構

本節主要考慮在 Client 端爲不可信任的情況（不可信任的軟體程式及記憶體暫留等情況）下使用，主要修訂部分在 Client 端及 Authentication Server，其它 server 通訊仍維持舊架構。

本節不同於上一節的部分在於整個登入過程中所使用的 key 皆由 IC Card 保存，其中的加解密動作也直接在 IC Card 中完成，Client 程式完全接觸不到這些 key，所以即使是不可信認的 Client 端也無法得到任何足以冒充 User 的資訊。

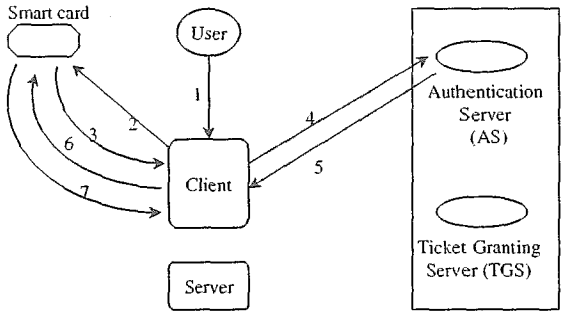
在 Authentication Service Exchange 中，不同於 3.2 節的部分在於：K_c 是由 IC Card 產生並且在 Card 中完成加密動作再交由 Client 送出，所以 Client 在收到 AS 的回應訊息後，必須直接把加密部份送給 IC Card 解密（Client 並沒有 K_c），IC Card 並把取得的 $K_{c,tgs}$ 直接存在 Card 中，所以 Client 只能拿到 Ticket 而無法接觸到 $K_{c,tgs}$。

在 Ticket-Granting Service Exchange 中，由於 Client 並沒有 $K_{c,tgs}$，所以必須將必要的資訊交給 IC Card，由 IC Card 以 $K_{c,tgs}$ 加密製作 Authentication，在取得 IC Card 製作的 Authentication 後再與 Ticket 等資料一併送給 TGS（爲考慮相容性，Client -> TGS 及 TGS -> Client 之訊息資料結構與 Kerberos 原架構相同），相同的，Client 在收到 AS 的回應訊息後，必須直接把加密部份送給 IC Card 解密，並由 IC Card 保存 $K_{c,v}$。

類似上面的步驟，Client/Server Authentication Exchange 中製作 Authentication 的工作仍由 IC Card 完成（IC Card 才有 $K_{c,v}$），再交由 Client 將所有資料一併送出，而 Client 在收到回應訊息後一樣將加密部分交由 IC Card 解密，並由 IC Card 保存之後的 key。

下面我們分成三個小節對整個架構加以詳細介紹，分別是 3.3.1 節的 Authentication Service Exchange、3.3.2 節的 Ticket-Granting Service Exchange、以及 3.3.3 小節的 Client/Server Authentication Exchange。

3.2.1 Authentication Service Exchange



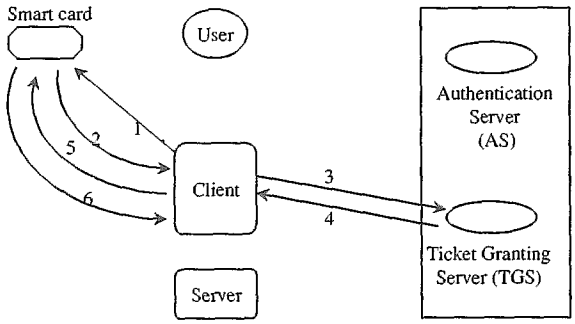
1 user : ID_c, PIN
2 C→Sm: PIN, Options, ID_c, $Realm_p$, ID_{tgs}, Times, $Nonce_c$, K_{pubAS}
3 Sm→C: $E_{K_{privAS}}\{E_{K_{privC}}\{K_c\}\}, E_{K_c}\{Options, ID_c, Realm_p, ID_{tgs}, Times, Nonce_c\}$
4 C→AS: $E_{K_{pubAS}}\{E_{K_{privC}}\{K_c\}\}, E_{K_c}\{Options, ID_c, Realm_p, ID_{tgs}, Times, Nonce_c\}, K_{pubC}$
5 AS→C: $Realm_p$, ID_c, $Ticket_{tgs}$, $E_{K_c}\{K_{c,tgs}, Flags, Times, Nonce_c, Realm_{tgs}, ID_{tgs}\}$
6 C→Sm: $E_{K_c}\{K_{c,tgs}, Flags, Times, Nonce_c, Realm_{tgs}, ID_{tgs}\}, ID_{AS}$
7 Sm→C: response

$Ticket_{tgs} = E_{K_{tgs}}\{Flags, K_{c,tgs}, ID_c, Realm_p, Times\}$

訊息說明：

1) User 打入自己的 identity、smart card 的 PIN(personal Identification Number)
2) Client 產生一 Nonce，並將之與 user 之 id、PIN、欲溝通之 TGS id、Options、Times、及 TGS 之 public key 一起傳送給 SmartCard
3) SmartCard 以本身安全機制（PIN）驗證 User 身份無誤後，產生一 Random Session Key：K_c 存在卡中，接著以 User 之 private key（Public-Key 技術）對 K_c 簽名，再以 AS 之 public key 加密之，並以 K_c（Symmetric Key 技術）加密其他資訊傳送回 Client。
4) Client 將整個 Message 附上 User 之 public key （或 Certificate 之 issuer name + serial number）送給 AS 要求一張"身份證明"以進入 TGS
5) AS 以本身之 private key 解開 $E_{K_{pubAS}}\{E_{K_{privC}}\{K_c\}$（Client 以此確定 AS 的身份），再以 User 之 public key 驗證 User 之簽名（AS 確定 User 之身份）。確定無誤後，解開加密部分並以 Nonce 確定非 replay，之後 AS 爲 Client 產生一張"身份證明"（包含爲 User ↔ TGS 產生一把 temporary key 及一張 Ticket_{tgs}）並以 K_c 加密 temporary key 等資訊傳回給 Client。
6) Client 收到後將加密部分傳送給 SmartCard（SmartCard 方有 K_c）
7) SmartCard 將加密部分解開，檢查 Nonce 等，若無誤則將 temporary key 存在卡中並以適當回應訊息傳回 Client。
8) Client 對回應做適當處理，若無誤則將 Ticket_{tgs} 儲存

3.2.2 Ticket-Granting Service Exchange



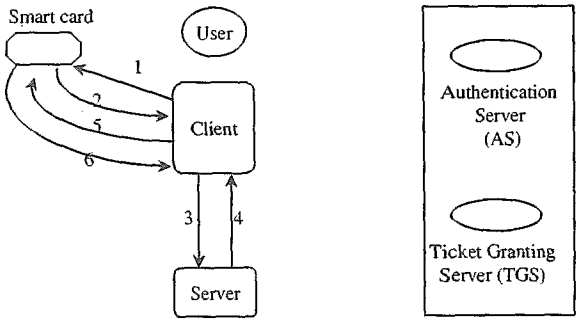
1 C→Sm : $ID_c, Realm_t, Nonce_1, ID_{tgs}, Realm_{tgs}$
2 Sm→C : $Authenticator_c$
3 C→TGS: Options,ID_v, Times,$Nonce_2$, $Ticket_{tgs}$, $Authenticator_c$
4 TGS→C: $Realm_t, ID_c, Ticket_t, E_{K_{c,tgs}}\{K_{c,v}$, Flags,Times,$Nonce_2, Realm_t, ID_v\}$
5 C→Sm : $E_{K_{c,tgs}}\{K_{c,v}$, Flags,Times,$Nonce_2, Realm_t, ID_v\}, ID_{tgs}$
6 Sm→C : response

$Ticket_{tgs} = E_{K_{tgs}}\{Flags, K_{c,tgs}, ID_c, Realm_t, Times\}$
$Ticket_t = E_{K_t}\{Flags, K_{c,v}, ID_c, Realm_t, Times\}$
$Authenticator_c = E_{K_{c,tgs}}\{ID_c, Realm_t, Nonce_2\}$

訊息說明：

1) Client 產生一 Nonce，並將之與 user 之 id、及欲溝通之 TGS id 一起傳送給 SmartCard
2) SmartCard 依 TGS id 取出 $K_{c,tgs}$，以 $K_{c,tgs}$ 加密其他資訊製作 Authenticator 並送回 Client
3) Client 將欲溝通之 Server id、Options、Times、Nonce、Ticket_{tgs} 及 Authenticator 送給 TGS 以要求一張"身份證明"進入 Service Server
4) TGS 檢驗 Ticket_{tgs} 及 Authenticator，若無誤則爲 Client 產生一張"身份證明"（包含爲 User ↔ Server 產生一把 temporary key 及一張 Ticket）並以 $K_{c,tgs}$ 加密 temporary key 等資訊傳回給 Client。
5) Client 收到後將加密部分及 TGS id 傳送給 SmartCard（SmartCard 方有 $K_{c,tgs}$）
6) SmartCard 依 TGS id 取出 $K_{c,tgs}$ 將加密部分解開，檢查 Nonce 等，若無誤則將 temporary key 存在卡中並以適當回應訊息傳回 Client。
7) Client 對回應做適當處理，若無誤則將 Ticket_v 儲存

3.2.3 Client / Service Authentication Exchange



1 C→Sm : $ID_c, Realm_t, Nonce_3, K_{temp}, Seq\#, ID_v, Realm_t$
2 Sm→C: $Authenticator_c$
3 C→V : Options,$Ticket_t$, $Authenticator_c$
4 V→C : $E_{K_{c,v}}\{Times, K_{temp}, Seq\#\}$
5 C→Sm: $E_{K_{c,v}}\{Times, K_{temp}, Seq\#\}$
6 Sm→C: response

$Ticket_v = E_{K_v}\{Flags, K_{c,v}, ID_c, Realm_t, Times\}$
$Authenticator_c = E_{K_{c,tgs}}\{ID_c, Realm_t, Nonce_3, K_{temp}, Seq\#\}$

訊息說明：

1) Client 產生一 Nonce，並將之與 user 之 id、欲溝通之 Server id、K_{temp}、及 Seq#一起傳送給 SmartCard
2) SmartCard 依 Server id 取出 $K_{c,v}$，以 $K_{c,v}$ 加密其他資訊製作 Authenticator 並送回 Client
3) Client 將 Options、Ticket_v 及 Authenticator 送給 Server 以要求開始之後的通訊（要求 Service Server 提供服務）

4) Server 檢驗 $Ticket_v$ 及 Authenticator，若無誤則以 $K_{c,v}$ 加密回覆資訊傳回。
5) Client 收到後將加密部分及 Server id 傳送給 SmartCard（SmartCard 方有 $K_{c,v}$）
6) SmartCard 依 Server id 取出 $K_{c,v}$ 將加密部分解開並以適當回應訊息傳回 Client。
7) Client 做適當的處理，並開始之後的通訊

3.2.4 架構說明

由於配合 Smart Card 的使用，使用者仍然只需記憶一個 password，卻可改以使用較覆雜的 key（以較具強度的 key 取代 password），防止 dictionary attack 的問題。

Smart Card 的安全機制優於其它儲存體，使用 PIN 控制使用權，一旦打入錯誤 PIN 超過預先設定之次數（例如三次），此張卡片將無法再使用，沒有 PIN 的非法使用者無法以其它方式得到 key。本節架構中，加解密動作在卡片中完成，任何使用 key 的動作都必須經由卡片內部的中央處理器單元，key 不會傳到卡片外部（防止 host 上的惡意程式或記憶體殘留），當使用者抽出卡片離開後，其他人即使使用同一台電腦也無法冒用使用者身份（資訊皆在卡片上）。

在 client 的第一個 request（對 AS）加入 Public Key 技術，做到事先 mutual authentication。Client（smart card）以 private key 對 K_c 簽名，再以 AS 的 public key 加密之，當 AS 收到 $E_{pubAS}\{E_{privC}\{K_c\}\}$ 時必須先以自己的 private key 解密取得 K_c，達到向 Client 證明 AS 身份的目的（唯有正確的 AS 方有 private key 可解出 K_c）；同時，AS 可先驗證 client 對 K_c 的簽名，確定 client 身份，AS 只對確認的 Client 發出"身份證明"，解決原先任何人都可獲得"身份證明"的問題。

另外，Public Key 技術以 certificate 管理 key，使用者不必親自前往 AS 登記或更改 secret key，AS 也不再需要儲存 long term user key，防止 attacker 侵入 AS 取得 long term user key 的危險。

本架構經由對 performance 的考量，將使用 Public Key 的動作（Public Key 加解密較慢）簡化到最少。

Public Key 加解密系統的缺點在於其 performance，在 AS 端（接受所有 client 的 request）可能造成很大負擔。

3.3 效率提昇架構

在上一節中，由於 Public key 加解密（簽名、驗證）較花時間（比一般 Symmetric 加解密高出許多），而 AS 必需供應大量 User 的 request，將會對 AS 造成負擔。本節希望加入 keyed hash function、One-time password 等概念來改善上一系統之執行效率。

本節與上一節的不同點在 Authentication Service Exchange 第 4 步與第 5 步 $AS \leftrightarrow C$ 之間的訊息交換、及 K_c 的改變，主要將訊息分成二種型態（以下以< Type 1 >、< Type 2 >稱之）。

使用者首先以 < Type 1 > 訊息登入 AS，此種形態訊息先以 public key 技術確認雙方身份，再依此交換一把在此後代表雙方身份的 key；第二次之後便可使用 < Type 2 > 訊息登入 AS，此種訊息主要利用先前 < Type 1 > 訊息交換的 key，並配合 keyed hash function，每次產生新的 key 來加解密訊息；如此，每次使用 < Type 2 > 訊息來登入 AS 直到 N 次期滿（系統政策規定）或其中一方認爲有必要重新以 Public Key 確認雙方身份時，再以 < Type 1 > 訊息交換新的 key。

本架構仍配合 Public Key 技術使雙方確定對方身份，並使其後的 key list 具有相同性質（mutual authentication），由此，每 N 次 login 到 AS 只有一次使用 Public Key 技術，其它都是使用 symmetric 加解密方式，降低使用 Public Key 的頻率及 AS 的負擔，改善上一節中系統之執行效率。

接著，我們對前面提到的二種型態之訊息加以詳細說明：

< Type1 >

$C \rightarrow AS: E_{K_{pubAS}}\{E_{K_{privC}}\{K, K_c\}\}, E_{K_c}\{ID_c, ID_{tgs}, \sim\}, K_{pubC}$

$AS \rightarrow C: E_{HK_1}\{K_{c,tgs}, ID_{tgs}, Ticket_{c,tgs}, \sim\}$

< Type 2 >

[first time]

$C \rightarrow AS: E_{HK_2}\{ID_c, ID_{tgs}, \sim\}, ID_c$

$AS \rightarrow C: E_{HK_3}\{K_{c,tgs}, ID_{tgs}, Ticket_{c,tgs}, \sim\}$

[second time]

$C \rightarrow AS: E_{HK_4}\{ID_c, ID_{tgs}, \sim\}, ID_c$

$AS \rightarrow C: E_{HK_5}\{K_{c,tgs}, ID_{tgs}, Ticket_{c,tgs}, \sim\}$

⋮

[nth time]

$C \rightarrow AS: E_{HK_{2n}}\{ID_c, ID_{tgs}, \sim\}, ID_c$

$AS \rightarrow C: E_{HK_{2n+1}}\{K_{c,tgs}, ID_{tgs}, Ticket_{c,tgs}, \sim\}$

$HK_1 = H_K\{K_C\}$

$HK_2 = H_K\{H_K\{K_C\}\} = H_K\{HK_1\}$

⋮

$HK_n = H_K\{HK_{n-1}\}$

訊息說明：

< Type 1> 第一次登入 AS 或希望重新交換一把 secret key 時：

1) Client（Smart Card）端先產生 2 個 random symmetric key K 及 K_c，同樣予以簽名、加密送給 AS
2) AS 收到後，以 K_c 解開其它訊息，並以 K 及 K_c 計算出 HK_1 記下來且用以加密 reply。
3) Client（Smart Card）收到 reply 後同樣求出 HK_1，以 HK_1 解開 reply 並將 HK_1 記下，此後同上一節。

< Type 2 > 平常 log in AS 時：

1) Client 取出記錄的 HK_n 加以 hash 處理成爲此次加密 request 的 key HK_{n+1}，並在記錄中以 HK_{n+1} 取代 HK_n

2) AS 同樣取出 HK_n 且 hash 處理成爲 HK_{n+1} 解開 request，再 hash 處理 HK_{n+1} 成爲 HK_{n+2} 用以加密 reply，最後在記錄中以 HK_{n+2} 取代 HK_n

3) Client 在收到 reply 後再以 HK_{n+1} 求出 HK_{n+2} 來解密 reply，並在記錄中以 HK_{n+2} 取代 HK_{n+1}

說明：

由於每次使用不同的 key （One-time password 概念），attacker 即使以暴力攻擊法得到此次的 HK_n，但 HK_n 用過便丟棄不再使用，且 attacker 無法取得 K，故 HK_n 對 attacker 毫無用處。

如此一來，每 N 次 login 到 AS 只有一次使用 Public Key 技術，其它都是使用 symmetric 加解密方式，降低使用 Public Key 的頻率及 AS 的負擔，改善上一節中系統之執行效率。

由上，Authentication Service Exchange 將修改如下：

< Type 1 >

1 user: ID_c, PIN

2 C→Sm: PIN, Options, ID_c, Realm$_c$, ID_{tgs}, TimesNonce$_c$, K_{pubAS}

3 Sm→C: $E_{K_{pubAS}}\{E_{K_{priC}}\{K, K_c\}\}, E_{K_c}\{$Options, ID_c, Realm$_c$, ID_{tgs}, TimesNonce$_c\}$

4 C→AS: $E_{K_{pubAS}}\{E_{K_{priC}}\{K, K_c\}\}, E_{K_c}\{$Options, ID_c, Realm$_c$, ID_{tgs}, TimesNonce$_c\}, K_{pubC}$

5 AS→C: Realm$_c$, ID_c, $Ticket_{tgs}$, $E_{HK_1}\{K_{c,tgs}$, Flags, TimesNonce$_c$, Realm$_{tgs}$, $ID_{tgs}\}$

6 C→Sm: $E_{HK_1}\{K_{c,tgs}$, Flags, TimesNonce$_c$, Realm$_{tgs}$, $ID_{tgs}\}$, ID_{AS}

7 Sm→C: response

$Ticket_{tgs} = E_{K_{tgs}}\{$Flags, $K_{c,tgs}$, ID_c, Realm$_c$, Time$\}$

< Type 2 >

1 user: ID_c, PIN

2 C→Sm: PIN, Options, ID_c, Realm$_c$, ID_{tgs}, TimesNonce$_c$

3 Sm→C: $E_{HK_n}\{$Options, ID_c, Realm$_c$, ID_{tgs}, TimesNonce$_c\}$

4 C→AS: $E_{HK_n}\{$Options, ID_c, Realm$_c$, ID_{tgs}, TimesNonce$_c\}$, ID_c

5 AS→C: Realm$_c$, ID_c, $Ticket_{tgs}$, $E_{HK_{n+1}}\{K_{c,tgs}$, Flags, TimesNonce$_c$, Realm$_{tgs}$, $ID_{tgs}\}$

6 C→Sm: $E_{HK_{n+1}}\{K_{c,tgs}$, Flags, TimesNonce$_c$, Realm$_{tgs}$, $ID_{tgs}\}$, ID_{AS}

7 Sm→C: response

$Ticket_{tgs} = E_{K_{tgs}}\{$Flags, $K_{c,tgs}$, ID_c, Realm$_c$, Time$\}$

3.4 安全性再加強架構

本節主要的著眼點在於 Authentication Service Exchange 中，AS 回應於 Client 的訊息裡包含的 $K_{c,tgs}$ 是一把重要性很高的 key，因爲惡意的使用者一旦取得此 key，將可使用它冒用使用者身份進入任何 Server 一段時間（直到此 key 使用期限終止，通常爲八個小時）。

由上一節 AS 回應給 Client 的訊息中可看出，$K_{c,tgs}$ 是以某一個 HK_{n+1} 加密起來，爲防止 attacker 真的以暴力攻擊法取得 $K_{c,tgs}$（一再 try HK_{n+1} 的值，雖然以此方法成功的機會非常微小），本節希望以 HK 序列中的一個元素取代原來由 AS 產生的 $K_{c,tgs}$，因爲此元素爲 Client（IC Card）與 AS 已事先知道的，所以便不必在包含在給 Client 看的訊息中，AS 只要直接將它放到 Ticket 中，再把 Ticket 包入給 Client 看的訊息裡，如此便有兩層的加密保護，使 $K_{c,tgs}$ 更安全。

詳細訊息說明如下：

原來的訊息：

$C \rightarrow AS: \{ID_c, ID_{tgs}, \sim\}HK_n, ID_c$

$AS \rightarrow C: \{K_{c,tgs}, ID_{tgs}, Ticket_{c,tgs}, \sim\}HK_{n+1}$

attacker 可能以暴力攻擊法一再 try HK_{n+1} 的值而得到 $K_{c,tgs}$，b 如此將可扮演 Client 一段時間。

本節之加強架構：

$C \rightarrow AS: \{ID_c, ID_{tgs}, \sim\}HK_n, ID_c$

$AS \rightarrow C: \{ID_{tgs}, Ticket_{tgs}, \sim\}HK_{n+1}$

$Ticket_{tgs} = \{HK_{n+2}, \sim\}K_{tgs}$

因爲 HK_{n+2} 是 Client 早已知道的，便不需在 reply 中出現了，AS 只負責將 HK_{n+2} 包入 ticket 中，並將 ticket 放入加密訊息裡得到雙重保護（K_{tgs} 、K_{n+1}）。

但此方法將把 ticket 再度放回加密部分增加加密資料（Kerberos version 5 已將 ticket 搬出加密部分）。

4. 結論

本論文主要以 Kerberos 身份認證模型爲基礎，對其存在的缺點加以改進，提出新的身份認證架構。

首先以 Public Key technique 修改 Kerberos 架構，並配合 IC Card 作爲安全儲存體，在 Client 是否可信認的考量下提出兩個新架構，分別爲在 Client 可信認狀況下使用的基本架構及在 Client 不可信認狀況下使用的安全架構，此兩架構都做到了 Client 與 AS 間的 mutual authentication 及防止 dictionary attack。

在 performance 的考量下，本論文接著提出一個效率提昇架構，其中使用到了 keyed hash function technique 及 One-time password 概念，以期減少 AS 端 Public Key 技術的使用量，而以 Symmetric Key 取代（Public Key 的計算時間是 Symmetric Key 的許多倍）。

最後，本論文再提出一個安全性再加強架構，使整個系統的安全性更加提昇。

參考文獻

[1] M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication," Research Report 39, Digital Equipment Corp. Systems Research Center, Feb 1989.

[2] B.C. Neuman and T. Ts'o, "Kerberos : An Authentication Service for Computer Networks," IEEE Coommunications Magazine, v.32, n.9, Sep 1994, pp.33-38

[3] J.T. Kohl and B.C. Neuman, "The Kerveros Network Authentication Service（v5）," RFC 1510,Sep 1993.

[4] Bill Bryant, "Designing an Authentication System：a Dialogue in Four Scenes,"February 1988.

[5] Cary Gaskell and Mark Looi , "Integrating Smart

Cards Into Authentication Systems," Cryptography : Policy and Algorithms, July 1995, pp. 270-281

[6] Marvin A. Sirbu and John Chung-I Chuang, "Distributed Authentication in Kerberos Using Public Key Cryptography," IEEE,1997.

[7] D. Davis, "Kerberos Plus RSA for Word Wide Web Security," Proceedings of the USENIX Workshop on Electronic Commerce, July 1995

[8] D.E. Denning and G.M. Sacco, "Ti*estamps in Key Distribution Protocols," Communication of the ACM, August 1981

[9] A.O. Freier , P. Karlton, and P.C. Kocher, "Secure Socket Layer 3.0," Internet Draft, March 1996

[10] International Telegraph and Telephone Consultative Committee (CCITT), "Recommendation X.509 : The Directory Authentication Framework," 1998

[11] RSA Laboratories, "PKCS #7 : Cryptographic Message Syntax Standard." November 1993

[12] M. Sirbu and J.C. Chuang, "Public key Based Ticket Granting Service in Kerberos," Internet Draft, May 1996

[13] Ravi Ganesan, "Yaksha : Augmenting Kerberos with Public Key Cryptography," IEEE 1995

[14] Boyd, C., "Digital Multicignatures," Cryptography and Coding, Clarendon Press, Oxford 1989, H.J. Beker and F.C. Piper, Ed

[15] Ganesan R. and Y. Yacobi, "A Secure Joint Signature and Key Exchange System," Bellcore Technical Memorandum, TM-ARH-, 1994

[16] R. Ganesan, "A New Key Escrow System," In publication process

[17] J. T. Kohl, "The Evolution of the Kerberos Authentication Service," EurOpen Conference Proceedings, May 1991

[18] R. Rivest, A. Shamir and L. Adelman, "On Digital Signatures and Public-Key Cryptography", Communications of the ACM, v.27, n.7, July 1978

[19] R. Molva, G. Tsudik, E. van Herreweghen, and S.Zatti, "KryptoKnight Authebtication and Key Distribution System," Proceedings of European Symposium on Research in Computer Security, Toulouse, France, Nov 1992.

[20] Wenbo Mao and Colin Boyd, "Methodical Use of Cryptographic Transformations in Authentication Protocols," IEE Proceedings, Comput. Digit. Tech., Vol. 142, No. 4, July 1995, pp. 272-278

[21] J.G. Steiner, B.C. Neuman, and J.I. Schiller, "Kerberos : An Authentication Service for Open Network System," USENIX Conference Proceedings, Feb 1988, pp. 191-202

[22] B.C. Neuman and T. Ts'o, "Kerberos : An Authentication Service for Computer Networks," IEEE Communications, September 1994

[23] B.C. Neuman, B. Tung, and J. Trostle, "Public Key Cryptography for Initial Authentication in Kerberos," Internet Draft, October 1996