Kerbie: Kerberos-type Authentication using Public-Keys

Thomas Hardjono Yuliang Zheng Jennifer Seberry

Centre for Computer Security Research
Department of Computer Science
University of Wollongong
Wollongong, NSW 2522, Australia
Telephone: +61 42 21 4327
Fax: +61 42 21 4329
Email: thomas/yuliang/jennie@cs.uow.edu.au

Abstract

Kerberos-type authentication protocols have more to offer when they are founded upon public key cryptosystems. In the current paper we argue and illustrate this point by way of presenting a protocol that implements Kerberos using a recent and promising public key cryptosystem, which is secure against the adaptatively chosen ciphertext attacks. The flexibility of the solution is highlighted by extending the protocol to allow the use of one ticket for multiple services. The issue of hierarchical inter-realm authentication is also considered by way of two protocols based on the notion of localized and globalized keys respectively. These protocols represents a step towards a family of solutions based on the combination of the advantages of private key and public key cryptography.
Keywords: Information Security, Authentication, Upper layer protocols, Cryptography, Network Security, Distributed Systems.

1 Introduction

Authentication represents an important security functionality in computer networks and distributed systems. Informally, authentication refers to the ability of the receiver of a message to verify that the message truly came from the alleged sender. Authentication, which is often coupled with secrecy, is usually achieved by way of cryptographic functions. Two philosophies undercurrent in modern cryptology are private (shared or symmetric) key cryptography and public (asymmetric) key cryptography.

In private key cryptosystems the sender and the receiver share the one same key used to encipher and decipher the message text. Decipherability of a piece of ciphertext implies the authenticity of the text. In public key cryptosystems the secret key known only to the sender is related to a public key available to the sender and any other party. Since the secret key and public key are related, any plaintext enciphered using one key can only be deciphered using the other key. Extracting the secret key from the corresponding public key is deemed to be computationally infeasible.

From these two fundamental types of cryptosystems a range of different functionalities can be established by way of building protocols that embody one or the other (or both) of these types of cryptosystems [1]. Due to the complexity and cost of the task of ensuring a correct and tamper-free implementations [2] of such protocols as an integrated part of the architecture (eg. ISO/OSI, Internet) a more common approach has been to place them in the upper layers of such architectures.

One authentication system that has received

considerable attention in the recent past is the *Kerberos* authentication service [3, 4, 5], which is a component within the Project Athena at MIT [6]. Kerberos adopts the private (shared) key approach, and is a trusted third-party authentication service developed from the authentication model of [7]. The Kerberos authentication service provides a trusted intermediary between a *User* or *Client* that requires services from a *Server*. It authenticates the User or Client by way of a shared private key, and it provides a way for the Server to authenticate the User or Client when it request the services of the Server.

Due to the use of private key cryptography, the Kerberos authentication service requires a key to be shared between the User or Client and the trusted authorities in the system, namely the *Key Distribution Center* (KDC) and the *Ticket Granting Server* (TGS) [5]. This, however, requires a great amount of trust to be placed on the implementation of the trusted authorities, since their compromise results in possible masquerade by the attacker of the User or Client whose private key are stored with these trusted authorities.

With this in mind, in this paper we investigate an alternative approach based on the use of a public key cryptosystem, following the footsteps of Kerberos and offering new functionalities such the use of multiple-service tickets. A public key approach gains the advantage of requiring less trust in the implementation of the trusted authorities. This advantage is derived from the nature of public key cryptosystems which only requires the User or Client to share the publicly-known key with the trusted authorities. Hence, the compromise of the trusted authorities does not lead to the possible masquerade of the User or Client.

Another advantage of this approach is the ability of the Clients to use their public keys to perform one-to-one secure communications with other Clients. This ability is not immediately present in Kerberos since any two Clients must first establish a common key via a key-exchange protocol before they can communicate securely.

Finally, contrary to the common public key approaches of using the *de facto* standard RSA cryptosystem [8], we employ the recent and attractive public key cryptosystem of [9] which offers some inherent advantages over RSA. The cryptosystem of [9] has been shown to be secure against the strongest type of attacks, namely the adaptatively chosen ciphertext attacks. In addition, the cryptosystem allows Clients and other participants to share common parameters in a secure fashion, leading to interesting key compositions. Such key compositions are virtually impossible to be achieved using the RSA cryptosystem since the parameters related to the generation of a secret-public key pair must be maintained secret by the owner of the key pair.

In the next section (Section 2) we briefly summarize the Kerberos authentication service (version 5) as described in [5]. We assume the readers are familiar with the basic constructs and mechanisms of Kerberos, and we direct them to [3, 5] for more details on its implementation. In Section 3 we present our approach using the public key cryptosystem of [9]. This is continued in Section 4 by an investigation into the use tickets for multiple services. The algorithms that implement our approach are then given in Section 5. As an afterthought, the flexibility of our approach is further illustrated in Section 6 by inter-realm authentication in a hierarchically organized realms or domains, each managed by a TGS. Section 7 presents a brief discussion and comparison between our approach and Kerberos, whilst Section 8 briefly discusses the security level achieved by our solution. The paper is finally closed in Section 9 by some concluding remarks.

2 Kerberos authentication service

In the Kerberos authentication system [3, 5] the entities that interact with the authentication service are called *principals*. The term can be used for Users, Clients or Servers. Commonly, a user directs a Client (eg. a program) on a machine to request a service provided by a Server on a remote

machine. The Server itself is usually a process on the remote machine, and different services are usually taken to be available on differing remote machines. Kerberos employs two types of credentials to achieve authentications, namely the *tickets* of the form

$$\{s, c, addr, timestamp, lifetime, K_{s,c}\}$$

and the *authenticators* of the form

$$\{c, addr, timestamp\}$$

which is enciphered using a key common to the issuer and the recipient. Here the ticket consists of the Server and Client identities, followed by the Client's network address, a unique timestamp, the lifetime of the ticket and finally by the common key to be shared between the Server and the Client [3]. In this example, the ticket contains the key $K_{s,c}$ shared between the Client c and the Server s. The Server is trusted to generate such shared keys and when the ticket is to be given to a Client c then it must be enciphered using the Client's key K_c.

In brief, the interactions within the authentication service consists of the Client requesting the Key Distribution Center (KDC) for a *ticket-granting ticket* to be submitted by the Client to the Ticket Granting Server (TGS). The TGS then issues the Client with a *service ticket* which has a shorter lifetime compared to the ticket-granting ticket. The service ticket is then used by the Client to request the services of the Server which is mentioned in the ticket. These two stages are also referred to as the credential-initialization and client-server authentication protocols respectively in [10].

The actions of the Kerberos authentication service following the description of [5] is given in the following (Figure 1).

1. Client $\rightarrow$ KDC: c, tgs

2. KDC $\rightarrow$ Client: $\{K_{c,tgs}\}_{K_c}$, $\{T_{c,tgs}\}_{K_{tgs}}$

3. Client $\rightarrow$ TGS: $\{A_c\}_{K_{c,tgs}}$, $\{T_{c,tgs}\}_{K_{tgs}}$, s

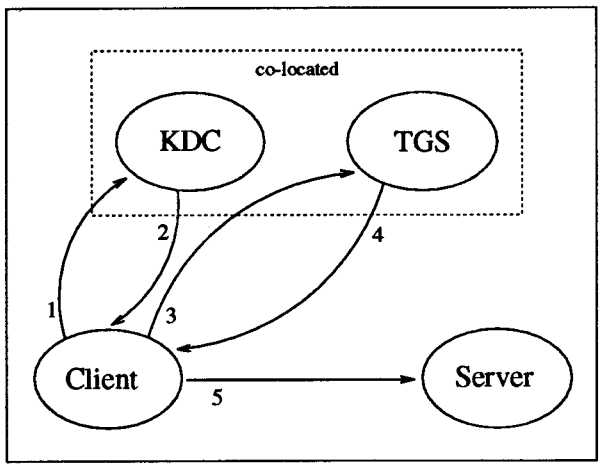

Figure 1: Obtaining a service ticket (after [5])

4. TGS $\rightarrow$ Client: $\{K_{c,s}\}_{K_{c,tgs}}$, $\{T_{c,s}\}_{K_s}$

5. Client $\rightarrow$ Server: $\{A_c\}_{K_{c,s}}$, $\{T_{c,s}\}_{K_s}$

6. Server $\rightarrow$ Client: $\{timestamp + 1\}_{K_{c,s}}$

In step 1 the Client c requests the Key Distribution Center (KDC) for an initial ticket and credentials to be presented to the Ticket Granting Server (TGS).

In step 2 the KDC generates a *session* key $K_{c,tgs}$ that will be shared between the Client and the TGS. A copy of this key is enciphered using the Client's key K_c to guarantee that the Client can obtain it securely. The ticket-granting ticket $T_{c,tgs}$ to be presented to the TGS by the Client already contains a copy of the session key $K_{c,tgs}$. Hence, both the Client and the TGS can later communicate securely using this session key shared between them. Note that the ticket is enciphered using the TGS's private key K_{tgs} known only to the KDC and the TGS.

In step 3 the Client then creates an authenticator A_c to be read only by the TGS (hence enciphered using $K_{c,tgs}$) and presents it to the TGS together with the ticket $\{T_{c,tgs}\}_{K_{tgs}}$ which the Client obtained from the KDC.

On receiving the authenticator and ticket-granting ticket from the Client, the TGS deciphers and authenticates the ticket (step 4). The

TGS then generates another session key $K_{c,s}$ to be shared between the Client c and the Server s. The TGS also creates a service ticket destined for the Server. This ticket $T_{c,s}$ contains a copy of the new session key, and its lifetime is shorter than the lifetime of the initial ticket $T_{c,tgs}$. The TGS knows the private key of every Server s, and ticket is made exclusively for the eyes of the Server by enciphering it using K_s. A copy of the session key $K_{c,s}$ (hidden by enciphering it using $K_{c,tgs}$) accompanies the ticket to the Client.

In step 5, the Client enciphers the authenticator A_c using the session key $K_{c,s}$ which it will share with the Server s. This is then sent to the Server s together with the ticket obtained from the TGS. The Client may request the Server to prove its identity which can be achieved by the Server incrementing the timestamp value by one, and enciphering the result using the session key shared between the Server and the Client (step 6).

3 Public key approach

In this section we propose an approach based on public key cryptography. Our approach is founded on the public key cryptosystem of [9], and some of its constructs that are necessary for the current discussion will be presented in the following. The algorithms that implement the cryptosystem are deferred until Section 5 in order to simplify discussion.

In the public key cryptosystem of [9] a secret key is is chosen randomly and uniformly from the integers in $[1, p-1]$, where the prime p is public and is used to generate the multiplicative group $GF(p)^*$ of the finite field $GF(p)$. The generator of this multiplicative group is denoted as g and it is a publicly known value. This usage of g and p is inspired by notable works of [11] and [12]. The corresponding public key is then produced by using the secret key as the exponent of g modulo p. In the remainder of this paper all exponentiations are assumed to be done over the underlying groups.

For the current usage of the cryptosystem, assume that the secret and public key pairs of the principals are as follows. The KDC has the pair $(X_{kdc}, Y_{kdc} \equiv g^{X_{kdc}})$, the Client has $(x_c, y_c \equiv g^{x_c})$, the TGS has $(X_{tgs}, Y_{tgs} \equiv g^{X_{tgs}})$, while the Server has $(X_s, Y_s \equiv g^{X_s})$. The keys Y_{kdc}, Y_c, Y_{tgs} and Y_s are known to the public as in other systems based on public key cryptography.

Each session secret and public key pair is denoted by $(k, K \equiv g^k)$, and their subscripts indicates which principals employ the key pair. Hence, the pair $(k_{c,tgs}, K_{c,tgs})$ is used for interactions between the Client and the TGS. In our case the tickets to be employed do not contain any keys, hence their form are:

$$\{s, c, addr, timestamp, lifetime\}$$

3.1 Getting an initial ticket

1. Client $\rightarrow$ KDC: c, tgs

2. KDC $\rightarrow$ Client: $K_{c,tgs}$, $C_{c,tgs}$

   The KDC first generates the session key pair $(k_{c,tgs}, K_{c,tgs})$ to be used between the Client and the TGS.

   The cryptogram $C_{c,tgs}$ is the ticket-granting ticket $T_{c,tgs}$ being enciphered as:

   $$C_{c,tgs} \equiv \{T_{c,tgs}\}_{r_{c,tgs}}$$

   where

   $$r_{c,tgs} \equiv (y_c\, Y_{tgs})^{X_{kdc}+k_{c,tgs}} \qquad (1)$$

3. Client $\rightarrow$ TGS: $A_{c,tgs}$, $K_{c,tgs}$, $C_{c,tgs}$, s

   On receiving the enciphered ticket $C_{c,tgs}$ together with its accompanying session public key $K_{c,tgs}$ the Client computes a *Authenticator* $A_{c,tgs}$:

   $$A_{c,tgs} \equiv (Y_{kdc}\, K_{c,tgs})^{x_c} \qquad (2)$$

   The authenticator, the received session public key and enciphered ticket-granting ticket, and

the identity of the destination Server s are then delivered to the TGS.

The TGS employs the session public key $K_{c,tgs}$ to compute its *Decryptor* $D_{tgs,c}$ as:

$$D_{tgs,c} \equiv (Y_{kdc}\, K_{c,tgs})^{X_{tgs}} \qquad (3)$$

This is then used to recreate the key $r_{c,tgs}$ that was used by the KDC to encipher the ticket:

$$r'_{c,tgs} \equiv A_{c,tgs}\, D_{tgs,c}$$

The resulting key $r'_{c,tgs}$ is then used to recover the ticket $T_{c,tgs}$.

3.2 Getting a service ticket

In order to obtain the services of the Server the Client must obtain a service ticket from the TGS to be presented to the Server. We continue the procedure in the following steps.

4. TGS $\rightarrow$ Client: $K_{c,s}$, $C_{c,s}$

In order to encipher the service ticket $T_{c,s}$ the TGS must generate the session key pair $(k_{c,s}, K_{c,s})$ which is used as follows:

$$C_{c,s} \equiv \{T_{c,s} || k_{c,s}\}_{r_{c,s}}$$

where

$$r_{c,s} \equiv (y_c\, Y_s)^{X_{tgs}+k_{c,s}}$$

resulting in the cryptogram $C_{c,s}$.

The cryptogram $C_{c,s}$ and the session public key $K_{c,s}$ are then delivered to the Client.

5. Client $\rightarrow$ Server: $A_{c,s}$, $K_{c,s}$, $C_{c,s}$

As when dealing with the TGS, the Client must compute the authenticator $A_{c,s}$ indicating its desire to use the service provided by the Server:

$$A_{c,s} \equiv (Y_{tgs}\, K_{c,s})^{x_c}$$

This authenticator, the received session public key and enciphered service ticket are then presented to the Server whenever the Client requires the service.

On first being presented with the (enciphered) service ticket, the Server must compute its corresponding decryptor

$$D_{s,c} \equiv (Y_{tgs}\, K_{c,s})^{X_s}$$

The Server is then able to recreate the session key $r_{c,s}$ as:

$$r'_{c,s} \equiv A_{c,s}\, D_{s,c}$$

to be used to obtain $\{T_{c,s} || k_{c,s}\}$.

6. Server $\rightarrow$ Client: $\{timestamp + 1\}_{r_{s,c}}$

If required, the Server may respond to the Client's request of proving the Server's identity. This can be done by the Server reusing the key $k_{c,s}$ that was enciphered together with the service ticket $T_{c,s}$. The key is reused to create $r_{s,c}$ as:

$$r_{s,c} \equiv (y_c)^{X_s+k_{c,s}}$$

to encipher $\{timestamp + 1\}$.

The Client can recreate the key $r_{s,c}$ as:

$$r'_{s,c} \equiv (K_{c,s}\, Y_s)^{x_c}$$

which is then used to recover and check $\{timestamp + 1\}$.

4 Multiple-Service ticket

It would be convenient if a Client could use one service-ticket to access several distinct services offered by various Servers. We can do this as will be shown here.

First, the TGS must be notified by the Client about the q services $s_1, s_2, \ldots, s_q$ that the Client wishes to access. We can repeat the last two steps (ie. steps 4 and 5) as follows:

M3. Client $\rightarrow$ TGS: $A_{c,tgs}$, $K_{c,tgs}$, $C_{c,tgs}$, $m = (s_1, s_2, \ldots, s_q)$

M4. TGS → Client: $K_{c,m}$, $C_{c,m}$ $(R_{c,s_1}, R_{c,s_2}, \ldots, R_{c,s_q})$

The TGS prepares the service ticket $T_{c,m}$, generates the key pair $(k_{c,m}, K_{c,m})$ and enciphers the ticket into $C_{c,m}$:

$$C_{c,m} \equiv \{T_{c,m} || k_{c,m}\}_{r_{c,m}}$$

where

$$r_{c,m} \equiv (y_c\, Y_{s_1}\, Y_{s_2} \cdots Y_{s_q})^{X_{tgs}+k_{c,m}}$$

The TGS must also computer q number of *selectors* $R_{c,s_1}, R_{c,s_2}, \ldots, R_{c,s_q}$ which will be used by the Client to choose among the q specified Servers $s_1, s_2, \ldots, s_q$. These selectors are computed as:

$$\begin{aligned} R_{c,s_1} &= (Y_{s_2}\, Y_{s_3} \cdots Y_{s_q})^{X_{tgs}+k_{c,m}} \\ R_{c,s_2} &= (Y_{s_1}\, Y_{s_3} \cdots Y_{s_q})^{X_{tgs}+k_{c,m}} \\ &\vdots \qquad\qquad \vdots \\ R_{c,s_q} &= (Y_{s_1}\, Y_{s_2} \cdots Y_{s_{q-1}})^{X_{tgs}+k_{c,m}} \end{aligned}$$

The cryptogram $C_{c,m}$, the session public key $K_{c,m}$ and the selectors are then delivered to the client.

M5. Client → Server s_v: $A_{c,m}$, $K_{c,m}$, $C_{c,m}$, R_{c,s_v}

As in the single-service case, the Client must compute its authenticator to be delivered to the Server. Thus,

$$A_{c,m} \equiv (Y_{tgs}\, K_{c,m})^{x_c}$$

However, in this multi-service case, the Client must select the Server from which it requires service. Assuming that the Client requires service from Server s_v $(1 \le v \le q)$, then the Client must employ the selector R_{c,s_v}.

This authenticator $A_{c,m}$, the received session public key $K_{c,m}$, the enciphered service ticket $C_{c,m}$ and the selector R_{c,s_v} must then be presented to the Server s_v.

On first being presented with the (enciphered) service ticket, the Server s_v must compute its corresponding decryptor

$$D_{m,c} \equiv (Y_{tgs}\, K_{c,m})^{X_{s_v}}$$

The Server is then able to recreate the session key $r_{c,s}$ as

$$r'_{c,m} \equiv A_{c,m}\, D_{m,c}\, R_{c,s_v}$$

to be used to decipher the service ticket $C_{c,m}$.

5 Algorithms

As described briefly in Section 3, our approach is based on the public key cryptosystem of [9]. In this section we provide further notations for the cryptosystem and present the algorithm for the encipherment and decipherment of tickets based on a modified version of the original cryptosystem of [9]. The algorithms expresses only the encipherment (decipherment) of the plaintext (ciphertext) tickets, and do not incorporate the steps taken by the KDC, Client, TGS and the Server. Hence, the reader is encouraged to read them in conjunction with the steps provided in Sections 3.2 and 3.1.

5.1 Notation

The following notation is taken directly from [9]. The cryptosystem of [9] employs a n-bit prime p (public) and a generator g (public) of the multiplicative group $GF(p)^*$ of the finite field $GF(p)$. Here n is a security parameter which is greater that 512 bits, while the prime p must be chosen such that $p-1$ has a large prime factor. Concatenation of string are denoted using the "||" symbol and the bit-wise XOR operations of two strings is symbolized using "$\oplus$". The notation $w_{[i\ldots j]}$ $(i \le j)$ is used to indicate the substring obtained by taking the bits of string w from the i-th bit (w_i) to the j-th bit (w_j).

The action of choosing an element x randomly and uniformly from set S is denoted by $x \in_R S$. G is a cryptographically strong pseudo-random string generator based on the difficulty of computing discrete logarithms in finite fields [9]. G stretches an n-bit input string into an output string whose length can be an arbitrary polynomial in n. This generator produces $O(\log n)$ bits output at each

exponentiation. All messages to be encrypted are chosen from the set $\Sigma^{P(n)}$, where $P(n)$ is an arbitrary polynomial with $P(n) \geq n$ and where padding can be used for messages of length less than n bits. The polynomial $\ell = \ell(n)$ specifies the length of tags. The function h is a one-way hash function compressing input strings into ℓ-bit output strings. In the remainder of this paper all exponentiations are assumed to be done over the underlying groups. The reader is directed to [9] for a comprehensive discussion on the constructs of the family of cryptosystems.

5.2 Getting initial and service tickets

In the process of getting an initial ticket the Clients asks the KDC to prepare the ticket to be submitted by the Client to the TGS. The KDC first performs $k_{c,tgs} \in_R [1, p-1]$ followed by the calculation $K_{c,tgs} \equiv g^{k_{c,tgs}}$. The KDC then enciphers the ticket $T_{c,tgs}$ using the key $r_{c,tgs}$ by invoking *Encipher* (Algorithm 1) with the input parameters $(p, g, r_{c,tgs}, T_{c,tgs})$ resulting in the output $C_{c,tgs}$.

Algorithm 1 *Encipher*(p, g, r, T)

1. $z = G(r)_{[1 \cdots (P(n)+\ell(n))]}$.
2. $t = h(T \oplus r)$.
3. $m = (T||t)$.
4. $C = z \oplus m$.
5. output (C).

end

The KDC then sends the resulting ciphertext $C_{c,tgs}$ and the session public key $K_{c,tgs}$ to the Client who proceeds to compute $A_{c,tgs}$. These values are then submitted by the Client to the TGS who tries to decipher $C_{c,tgs}$. This is done by the TGS first computing $D_{tgs,c}$ and using it and the received values as input to *Decipher* (Algorithm 2). That is, the TGS inputs $(p, g, A_{c,tgs}, K_{c,tgs}, C_{c,tgs}, D_{tgs,c})$ and receives the output $T_{c,tgs}$.

Algorithm 2 *Decipher*(p, g, A, K, C, D)

1. $r' = A\ D$.
2. $z' = G(r')_{[1 \cdots (P(n)+\ell(n))]}$.
3. $m = z' \oplus C$.
4. $T' = m_{[1 \cdots P(n)]}$.
5. $t' = m_{[(P(n)+1) \cdots (P(n)+\ell(n))]}$.
6. if $h(T' \oplus r') = t'$ then
   output (T')
   else
   output $(\emptyset)$.

end

The same procedure is followed by the TGS in enciphering the ticket to be submitted by the Client to the Server. The minor difference in this case is that the TGS appends a response key $k_{c,s}$ (ie. the secret half of the session key) to the ticket $T_{c,s}$. This addition does not affect the algorithms and their security in any way. Hence, the TGS invokes *Encipher* (Algorithm 1) with the input parameters $(p, g, r_{c,s}, (T_{c,s}||k_{c,s}))$ resulting in the output $C_{c,s}$. The Server deciphers $C_{c,s}$ into $(T_{c,s}||k_{c,s})$ using *Decipher* (Algorithm 2) with input values $(p, g, A_{c,s}, K_{c,s}, C_{c,s}, D_{s,c})$.

6 Hierarchical inter-realm authentication

The integration of security into distributed systems has introduced the need to manage the information pertaining to the security functions in the distributed system. The most common and important need is that of providing a method to manage cryptographic keys of the components in the distributed system. One approach that may be adopted is that of organizing the components into a hierarchy consisting of a number of domains

or realms, each being managed by an independent trusted authority (eg. TGS). This approach has the advantage of the localized distribution of new keys, hence reducing the replication of keys across the entire distributed system.

Within the context of our discussion a domain or realm can consist of Clients, Servers, a local managing TGS and of other TGSs that manage their own realms. In this manner, the components are organized into a hierarchy based on the TGSs, with each TGS managing a certain number of Clients, Servers and other TGSs. A Client within a realm may request service from a Server in the same domain in the manner previously discussed. However, it is also natural for a Client to request service from a "foreign" Server which is located in a different realm on another part of the hierarchy. In this section we address inter-realm authentication together with some accompanying issues. Our approach is general enough to be applicable to a number of areas, one being the X.500 Directory Services [13].

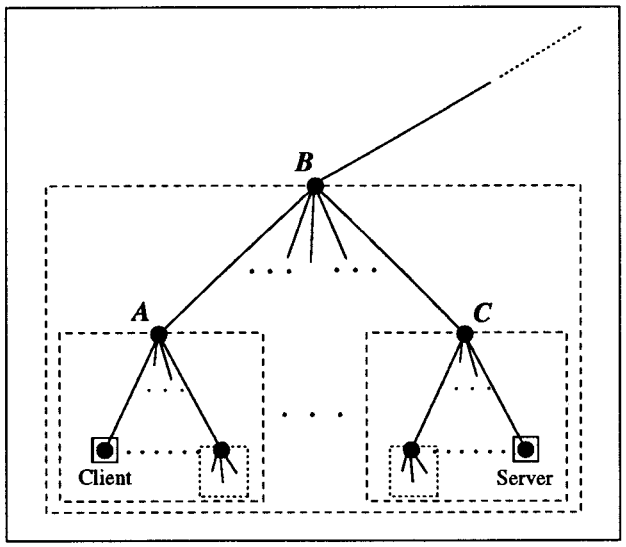

Figure 2: A hierarchy of TGSs

In our usage, a hierarchy is assumed to be a directed acyclic graph and each node in the hierarchy is assumed to have only one parent node. An example of such a hierarchy that will be used in the following discussions is given in Figure 2. In Figure 2 the Client is located in the domain or realm under the jurisdiction of the TGS A. The Client requires the service provided by a Server which is located within the realm of TGS C. In this case the TGS A must enroll the aid of its parent TGS B to forward the Client's request to TGS node C. Two general arrangements of the keys of the nodes in the hierarchy will be considered in the following. Note that the terms "TGS" and "node" will be used interchangeably to simplify discussion. In our example, we assume that the TGSs A, B and C have the key pairs (X_{tgsa}, Y_{tgsa}), (X_{tgsb}, Y_{tgsb}) and (X_{tgsc}, Y_{tgsc}) respectively.

6.1 Localized keys

One possible arrangement of keys in the hierarchy is based on their maintenance on a per realm basis. That is, in this arrangement a TGS node holds the public key of only its parent node and all its children nodes. This arrangement is similar to the arrangement of directories in [14]. Using Figure 2 as an example, TGS node B has the public key of its parent and of TGS nodes A and C. However, B does not have the public keys of the descendants of TGS nodes A and C. In this situation the TGS node A must refer the Client to node A's parent, namely node B. The node B, not knowing the public key of the Server must then refer the Client to B's child node C. Since node C is the trusted authority of the realm in which the Server resides, node C knows the public key of the Server and thus can forward the Client's request to the Server. These steps are shown in the following (Figure 3). Note that in essence, the Client must interact with every TGS node between its own TGS node (A) and the common ancestor node (B), and between the common ancestor node and the destination's TGS node (C). The deeper the Client is located in the hierarchy from the common ancestor, the more interactions it has to perform in order to reach the destination.

1. Client $\rightarrow$ KDC: $c, tgsa$
2. KDC $\rightarrow$ Client: $K_{c,tgsa}$, $C_{c,tgsa}$
3. Client $\rightarrow$ TGS A: $A_{c,tgsa}$, $K_{c,tgsa}$, $C_{c,tgsa}$, s
4. TGS $A \rightarrow$ Client: $K_{c,tgsb}$, $C_{c,tgsb}$

5. Client → TGS B: $A_{c,tgsb}$, $K_{c,tgsb}$, $C_{c,tgsb}$, s

6. TGS B → Client: $K_{c,tgsc}$, $C_{c,tgsc}$

7. Client → TGS C: $A_{c,tgsc}$, $K_{c,tgsc}$, $C_{c,tgsc}$, s

8. TGS C → Client: $K_{c,s}$, $C_{c,s}$

9. Client → Server: $A_{c,s}$, $K_{c,s}$, $C_{c,s}$

10. Server → Client: $\{timestamp + 1\}_{r_{s,c}}$

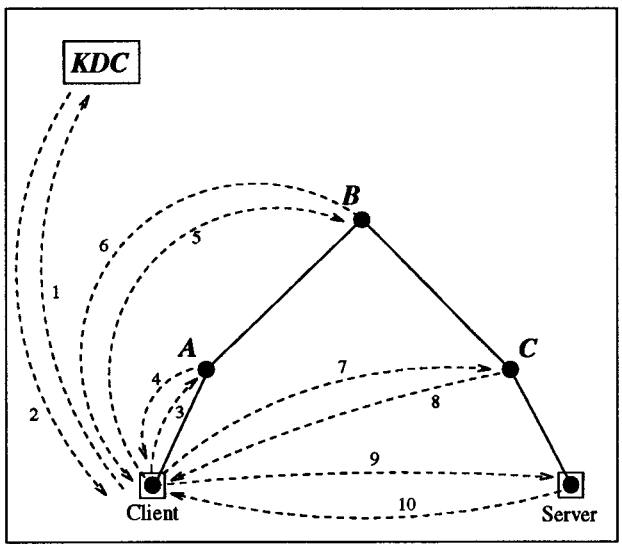

Figure 3: Authentication with localized keys

6.2 Globalized keys

Another possible arrangement of keys in the hierarchy is a more globalized one, in which a node knows not only the public key of all its descendant nodes, but also the public key of all its ancestor nodes (bearing in mind that a node only has one parent). Here a node does not have the public keys of any of its sibling nodes nor that of their descendants. Although such a configuration is costly in terms of the number of messages to be delivered when a node generates a new public key, the gains occur during the interaction with nodes located in other realms. Note that in our approach the public keys are not distributed in a fully globalized manner. That is, since a node does not have the public keys of its siblings, it must request the aid of its parent when dealing with such siblings. However, the steps used in our approach can be modified in a straight-forward manner to suit cases in which a node has a copy of the public key of every other node in the hierarchy.

Returning to our scenario where a Client requires the services offered by a foreign Server, the TGS in the Client's realm has more flexibility in issuing the enciphered ticket. Hence, with a TGS node knowing the public keys of all its ancestors, the node can find a common ancestor between itself and the TGS who manages the realm in which the Server resides. Looking back at Figure 2, the TGS node A (managing the Client's realm) and the TGS node C (managing the Server's realm) have a common ancestor (parent) in the TGS node B.

In this case, the TGS node A must prepare the enciphered ticket to be decipherable by the common ancestor (node B). This common ancestor node B must then locate the desired Server and re-encipher the ticket in such a way that it is decipherable by the Server with the necessary approval of the TGS within the Server's realm. That is, the ticket must be decipherable by the Server with the approval of the TGS node C in the form of node C sending a decryptor for the ticket to the Server. Note that only TGS A and C are involved with the common ancestor B, eventhough both TGSs A and C may have many other ancestors between them and TGS B respectively. Hence, in such a globalized key approach only a maximum of three TGSs (A, B and C) are invoked independent of the depth of the two TGS nodes (A and C) from their common ancestor (B). This scenario is expressed in the following steps (Figure 4).

1. Client → KDC: $c, tgsa$

2. KDC → Client: $K_{c,tgsa}$, $C_{c,tgsa}$

3. Client → TGS A: $A_{c,tgsa}$, $K_{c,tgsa}$, $C_{c,tgsa}$, s

4. TGS A → TGS B: $K_{c,tgsb}$, $C_{c,tgsb}$

   Here

   $$C_{c,tgsb} \equiv \{T_{c,tgsb}\}_{r_{c,tgsb}}$$

   where

   $$r_{c,tgsb} \equiv (y_c\, Y_{tgsb})^{X_{tgsa}+k_{c,tgsb}}$$

5. TGS $A \rightarrow$ Client: $K_{c,tgsb}$

6. Client $\rightarrow$ TGS B: $A_{c,tgsa}$

   where

$$A_{c,tgsa} \equiv (Y_{tgsa}\, K_{c,tgsb})^{x_c}$$

7. TGS $B \rightarrow$ TGS C: $K_{c,tgscs}$, $C_{c,tgscs}$

   The TGS B computes

$$D_{tgsb,tgsa} \equiv (Y_{tgsa}\, K_{c,tgsb})^{X_{tgsb}}$$

   and recreates the key

$$r'_{c,tgsb} \equiv A_{c,tgsa}\, D_{tgsb,tgsa}$$

   in order to decipher $C_{c,tgsb}$ into $T_{c,tgsb}$.

   TGS B then renames the ticket into $T_{c,tgscs}$ and enciphers it as

$$C_{c,tgscs} \equiv \{T_{c,tgscs} || k_{c,tgscs}\}_{r_{c,tgscs}}$$

   where

$$r_{c,tgscs} \equiv (y_c\, Y_{tgsc}\, Y_s)^{X_{tgsb}+k_{c,tgscs}}$$

8. TGS $B \rightarrow$ Client: $K_{c,tgscs}$

9. TGS $C \rightarrow$ Server: $K_{c,tgscs}$, $C_{c,tgscs}$, $D_{tgsc,tgsb}$

   where

$$D_{tgsc,tgsb} \equiv (Y_{tgsb}\, K_{c,tgscs})^{X_{tgsc}}$$

10. Client $\rightarrow$ Server: $A_{c,tgsb}$

    where

$$A_{c,tgsb} \equiv (Y_{tgsb}\, K_{c,tgscs})^{x_c}$$

    The Server then computes

$$D_{s,tgsb} \equiv (Y_{tgsb}\, K_{c,tgscs})^{X_s}$$

    and recreates the key

$$r'_{c,tgscs} \equiv A_{c,tgsb}\, D_{tgsc,tgsb}\, D_{s,tgsb}$$

    in order to decipher $C_{c,tgscs}$ into $\{T_{c,tgscs} || k_{c,tgscs}\}$

11. Server $\rightarrow$ Client: $\{timestamp + 1\}_{r_{s,c}}$

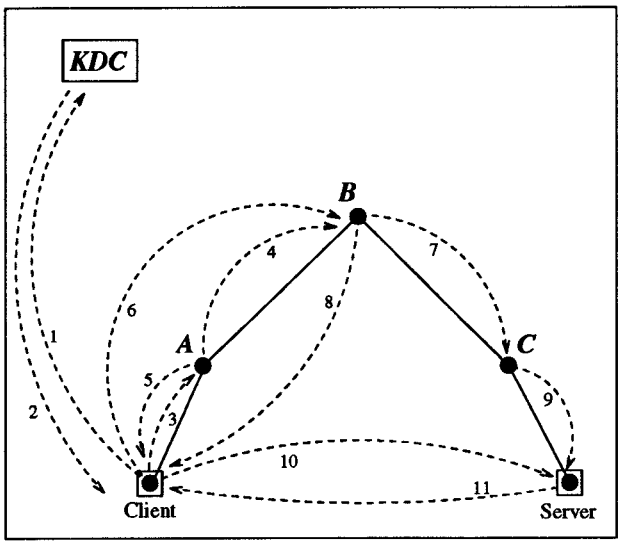

Figure 4: Authentication with globalized keys

7 Discussion

One fundamental difference between our approach and that of Kerberos [3, 5] lies in the underlying usage of the cryptosystem. In Kerberos and other authentication systems based on private (shared) key cryptography, the Clients must share a common key with the trusted authority, such as the KDC or the TGS. This necessitates a high level of security in the implementation of the trusted authority since their compromise immediately leads to the compromise of the Client's key or of the keys belonging to other trusted authorities. This further leads to the possible masquerade of the Client and/or the trusted authorities by the attacker [10].

In contrast, in our approach less trust is required of the authority since the compromise of such an authority (ie. KDC and/or TGS) only reveals to the attacker the public keys of the Clients. Hence, the attacker cannot masquerade as the Client nor as a trusted authority.

With specific reference to Kerberos, our approach also differs in the way the Client's authenticators are used. In [3, 5] the authenticator is used to convince the trusted authority (ie. TGS or Server) that the Client holds the same copy of the session key being shared between them. The authenticator here consists of a piece of text con-

taining the timestamp from the ticket. In our approach the process is less explicit in the sense that the authenticator (decryptor or selector) is a parameter that is required by the trusted authority to decipher the ticket. If an attacker removes, modifies or substitutes the authenticator, the trusted authority (ie. TGS and Server) will not be able to decipher the ticket, leading it to reject the Client's request. Since only the Client has the secret key to create the authenticator, the successful decipherment of the ticket indicates to the trusted authority that the Client has behaved accordingly and no active attacks have occurred. Note that if conditions require more explicit acknowledgment by the Client, the authenticator can be submitted together with the Clients name, address and ticket timestamp through a separate secure channel (eg. a separate instance of the cryptosystem).

An important point worth noting concerns the method of encipherment of the cryptosystem of [9] in contrast to the RSA cryptosystem [8]. In the cryptosystem of [9] all the principals (ie. Clients, KDC, TGSs and Servers) share the same parameters g and p. This leads to the easy formation of keys (Equation (1)), authenticators (Equation (2)), decryptors (Equation (3)) and selectors. From the practical point of view this compares favourably with the RSA cryptosystem in which every principal must accompany its public key with its individual modulus N.

In the case of the RSA cryptosystem, the sharing of the modulus N does not lead to the possible combination of keys belonging to two different principals. More specifically, assume in RSA that P and Q are (secret) primes whose product is N, and that $\phi(N) = lcm(P-1, Q-1)$. The secret and public keys (k, K) in RSA must satisfy $k\,K \equiv 1 \bmod \phi(N)$. The encipherment of a plaintext M is done as $M^k \bmod N$, while the resulting ciphertext can be deciphered by applying the public key K in the same exponentiation manner. Here note that both P and Q are secret to the owner of the keys, and hence $\phi(N)$ must also be secret. This means that the $\phi(N)$ of one node cannot be shared with other nodes, and thus other nodes cannot use it to create combination of keys in the manner of Equation (1).

This apparent difficulty to combine keys in the RSA stems from the fact that exponentiation in RSA is applied to the message. In the cryptosystem of [9] exponentiation is done on the generator g, independent of the message to be enciphered.

8 Security achieved

One of the primary motivating reasons for employing the cryptosystem of [9] is its strength against chosen ciphertext attacks [15]. In such an attack the attacker has access to the deciphering algorithm and can feed the algorithm with any input ciphertext in order to obtain its corresponding original plaintext. From these matching instances the attacker can then obtain information to finally cryptanalyze and break a given ciphertext.

The cryptosystem is promising because it has been show in [9] to be secure, not only against chosen ciphertext attacks, but further against *adaptively* chosen ciphertext attacks. In this type of attacks the attacker is permitted to select the input ciphertext which are *correlated* to the target ciphertext. Hence, the attacker continues to have access to the enciphering algorithm even after the attacker has the target ciphertext. Clearly, the attacker is not permitted to feed the target ciphertext into the deciphering algorithm.

In our mode of usage of the cryptosystem, the weakest point in the scheme is equivalent to solving instances of the discrete logarithm problem [16]. More specifically, in attempting to obtain any secret key that participated in the creation of an authenticator, a descriptor or a selector the attacker is faced with solving a discrete logarithm problem. In attempting to obtain the plaintext ticket from any given ciphertext, the attacker must break the cryptosystem.

9 Conclusion

In this paper we have attempted to argue and illustrate that Kerberos-type authentication protocols have more to offer when they are founded upon public key cryptosystems. A protocol has been presented that implements the Kerberos authentication service [3, 4, 5] using the recent public key cryptosystem of [9], which has been shown to be secure against the adaptatively chosen ciphertext attacks. This protocol has also been extended to allow multiple services to be obtained by using the one same ticket. Another extension relating to hierarchical inter-realm authentication has been illustrated by way of two protocols based on the localized and globalized arrangement of keys. Public key cryptography provides the advantage of one-to-one secure and authentic communications between entities in the system, something which is not immediately available to approaches based on private key cryptography. Our selection of the cryptosystem of [9] is motivated not only by its high level of security, but also by the ease at which session key compositions can be created. The use of public key cryptography also has the advantage in that the trusted authorities can be implemented with less trust since they only maintain the publicly-known keys [10]. The protocols in the current work represents a step towards solutions based on the mixture of private key and public key cryptography (such as in [17]), combining the advantages of both philosophies.

References

[1] V. L. Voydock and S. T. Kent, "Security mechanism in high-level network protocols," *ACM Computing Surveys*, vol. 15, no. 2, pp. 135–171, 1983.

[2] S. T. Walker, "Network security: The parts of the sum," in *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, (Oakland, CA), pp. 2–9, IEEE Computer Society, 1989.

[3] J. G. Steiner, C. Neuman, and J. I. Schiller, "*Kerberos*: an authentication service for open network systems," in *Proceedings of the 1988 USENIX Winter Conference*, (Dallas, TX), pp. 191–202, 1988.

[4] S. M. Bellovin and M. Merritt, "Limitations of the Kerberos authentication system," *Computer Communications Review*, vol. 20, no. 5, pp. 119–132, 1990.

[5] J. T. Kohl, "The evolution of the *kerberos* authentication service," in *Proceedings of the Spring 1991 EurOpen Conference*, (Tromsø, Norway), 1991.

[6] E. Balkovich, S. R. Lerman, and R. P. Parmelee, "Computing in higher education: The Athena experience," *Communications of the ACM*, vol. 28, pp. 1214–1224, November 1985.

[7] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in a large network of computers," *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, 1978.

[8] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–128, 1978.

[9] Y. Zheng and J. Seberry, "Immunizing public key cryptosystems against chosen ciphertext attacks," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 5, pp. 715–724, 1993.

[10] T. Y. C. Woo and S. S. Lam, "Authentication for distributed systems," *IEEE Computer*, vol. 25, pp. 39–52, January 1992.

[11] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, 1976.

[12] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. IT-31, no. 4, pp. 469–472, 1985.

[13] ISO/IEC, "Information Processing Systems - Open Systems Interconnection - The Directory - Information Model," 1989. ISO/IEC 9594-1.

[14] A. D. Birrell, B. W. Lampson, R. M. Needham, and M. D. Schroeder, "A global authentication service without global trust," in *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, (Oakland, CA), pp. 156–172, IEEE Computer Society, 1986.

[15] J. Seberry and J. Pieprzyk, *Cryptography: An Introduction to Computer Security*. Sydney: Prentice Hall, 1989.

[16] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP Completeness*. New York: W. H. Freeman, 1979.

[17] J. J. Tardo and K. Alagappan, "SPX: Global authentication using public-key certificates," in *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, (Oakland, CA), pp. 232–244, IEEE Computer Society, 1991.