# Replicating the *Kuperee* Authentication Server for Increased Security and Reliability

(Extended Abstract)

Thomas Hardjono[1,2] and Jennifer Seberry[1]

[1] Centre for Computer Security Research, University of Wollongong, Wollongong, NSW 2522, Australia
[2] Department of Computing and Information Systems, University of Western Sydney - Macarthur, NSW 2560, Australia

The current work proposes a new scheme for the replication of authentication services in Kuperee based on a public key cryptosystem, in response to the two main shortcomings of the traditional single server solutions, namely those of low availability and high security risks. The work represents further developments in the Kuperee authentication system. The Kuperee server is presented in its simplified design to aid the presentation of the replication scheme. The replication approach is based on the sharing of session pub'ic keys, together with a threshold or secret sharing scheme. However, unlike previous approaches, in the current work the object to be shared-out is instead a session secret key which is not directly available to the (untrusted) Client. The scheme gains advantages deriving from the use of public key cryptology, as well as from the manner in which the secret is shared-out. A comparison with the notable work of Gong (1993) is also presented.

## 1 INTRODUCTION

Developments of networks and nodes in terms of size and number in the world today has made the issue of security is a pressing one. Expansion of these networks, such as the Internet, has brought various possibilities for electronic commerce. In such electronic-based activities a certain level of assurance must be provided, both for the users and the service-providers. This, in turn, has brought into the foreground the security issues related to such networks. These issues include the authenticity of users, the non-repudiation of a user's transactions, integrity and privacy of information traveling within the network, and a host of other issues.

Authentication of users and service-providers within a network is an important foundation stone for other security-related services, independent of whether these are provided by the network or by specific service-providers in the network. The variety of authentication systems and protocols, together with the fact that the encipherment of data must often accompany authentication events, point to the higher layers (eg. in the OSI/ISO stack or the TCP/IP stack) as being the best location to provide such security-related services.

The current work proposes a new scheme for the replication of authentication services based on a public key cryptosystem, in response to the two main

shortcomings of the traditional single server solutions, namely those of low availability and high security risks. The work represents further developments in the *Kuperee* authentication system which was earlier presented in [1].

The choice of a public-key cryptosystem in Kuperee is largely motivated by conviction that as the needs of authentication increases with the expansion of networks, public-key techniques may offer better solutions to the problem of authentication in large-scale networks, compared to the approaches based exclusively on shared-key (symmetric) cryptosystems. This use of a public key, specifically that of [2], distinguishes Kuperee from other systems and provides it with more flexibility in tailoring the protocol steps following the particular demands. The use of the cryptosystems of [2] allows Kuperee to act either as an Authenticating Authority or as a Certifying Authority, or as a simultaneous combination of both. This ability is important in situations in which a special session key must be provided by an authority trusted by two communicating parties, eventhough both parties are already in possession of each other's (publicly available) public-key which have been generated and proclaimed by the parties respectively.

The earlier design of Kuperee suffered from a number of deficiencies which were noticeable soon after the design was published in [1]. The design was also unreasonably complicated. These problems prompted for Kuperee's simplification, reported in [3]. Some known deficiencies of the original design of [1] have again been confirmed recently in [4].

Kuperee, like other authentication systems – such as Kerberos [5, 6, 7] – was originally based on the use of a single authentication server to cater for clients in a given domain [1]. Although this traditional single-server approach has practical advantages, two of its more important shortcomings concerns the availability and performance of the server and the security of the server itself [8]. First, since entities in the distributed system rely on the server for their authentication operations, the server easily becomes a source of contention or bottleneck. The temporary unavailability of the server can lead to a degradation in the overall performance of the distributed system. Secondly, the fact that the server may hold cryptographic (secret) keys belonging to entities in the distributed system makes it the best point of attack for intruders wishing to compromise the distributed system. An attacker that successfully compromises the server has access to the (secret) keys of the entities under the server's jurisdiction. The attacker can then carry-out various active and passive attacks on the unsuspecting entities.

The use of a threshold scheme [9, 10] among a collection of security servers necessitates a client to consult a minimum of $t$ out of $M$ ($t \leq M$) honest servers before the authentication process succeeds or services are granted. This approach, besides increasing the overall security of the distributed system, also reduces the bottleneck present in the previous single-server solutions. An attacker wishing to compromise the cryptographic keys employed within the distributed system must compromise at least $t$ of the $M$ servers, something considerably harder than a single server. The problem of bottlenecks is eased by the existence of

multiple servers, only a subset of which is required by the clients for an instance of authentication.

In the next section the background of Kuperee for the sharing of session keys will be presented. This is followed by the new scheme for the replication of authentication services in Kuperee in Section 3. Section 4 offers some comparisons with an existing approach, while Section 5 closes the paper with some remarks and conclusions. Readers wishing more specific details on the public key cryptosystem employed in Kuperee are directed to the Appendix.

# 2 AUTHENTICATION IN KUPEREE

In the authentication system the entities that interact with the Authentication Server are called *principals* [5]. 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 another server (or another Client) on a remote machine.

There are a number of ways that the Kuperee server can be employed, either as an authentication server or as a certification server, or both [3]. In this current work we first employ Kuperee specifically as an authentication server, mimicking the actions of the Needham-Schroeder protocol [11, 12] within the Kerberos authentication system [5, 7]. That is, Kuperee will be used to deliver a pair of session keys to the principals that require secure and authentic communications. This is then followed by the description of a protocol in which the public-keys within Kuperee are used directly by the principals, thereby making Kuperee rather as a certification authority in the system.

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 authenticates the Client and issues the Client with a *service ticket* which has a shorter lifetime compared to the ticket-granting ticket. On presentation by the Client, the service ticket is used by the service-provider to authenticate the Client, since the service-provider places trust on the TGS (Figure 1). These two stages are also referred to as the credential-initialization and client-server authentication protocols respectively in [13].

## 2.1 Kuperee: Notations

In the public key cryptosystem of [2] a secret key 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 [14] and [15]. 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).

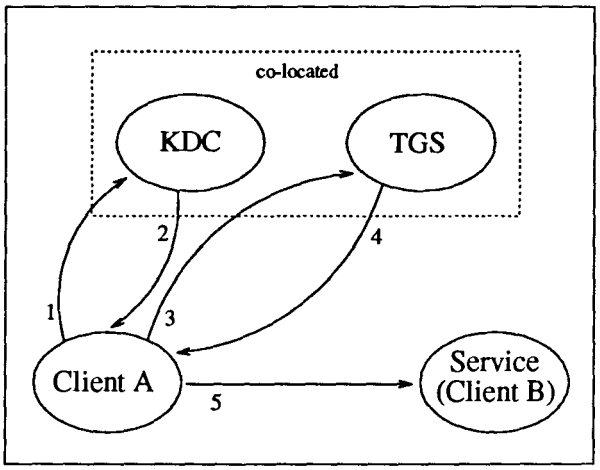


**Fig. 1.** Authentication in Kuperee following Kerberos

The KDC, the Client, the TGS and the Service provider (ie. destination server) have the secret-public key pairs $(X_{kdc}, Y_{kdc} \equiv g^{X_{kdc}})$, $(X_c, Y_c \equiv g^{X_c})$ $(X_{tgs}, Y_{tgs} \equiv g^{X_{tgs}})$, and $(X_s, Y_s \equiv g^{X_s})$ respectively.

The operation "$\{\}_K$" means that the contents within the braces "{}" are enciphered or deciphered using the key $K$. Thus, assuming that $Y_c$ is the public-key of the Client, the operation "$\{\}_{Y_c}$" signifies encryption using the publicly known key $Y_c$ of the Client using the modified encryption algorithm of [2]. The operation provides both confidentiality and integrity. All encrypted messages implicitly includes a nonce.

An important aspect of the protocol is that although a ticket is shown to be encrypted using the public-key of the targeted receiver, the sender's secret-key participates in the encryption. That is, there is *always* sender authentication built into every instance of encryption. In Kuperee, sender authentication and message secrecy are treated as inseparable in a single step. Hence, the receiver knows the identity of the sender and can verify the cryptogram as coming from the (alleged) sender, due to the fact that the receiver must use the sender's public-key in order to decipher the cryptogram.

### 2.2 Kuperee: Sharing of a Session Secret-Public Key Pair

Kuperee mimics Kerberos in that a session secret-public key pair is shared among the KDC-TGS-Client, while another is shared among the TGS-Client-Server. The session key pair is discarded after one instance of authentication.

Here, we assume that the KDC knows the public-key of all principals in the domain. The TGS knows the public key of the KDC and all Clients, while the Client knows the public-key of the TGS and the KDC. The KDC is assumed

to have the highest level of trust equivalent to one holding private keys of the principals. This is necessary as the KDC will also function as a certification authority for the public keys of the entities in the domain. A Client wishing to communicate to another Client may request from the KDC a certified copy of the public key belonging to the second client (and vice versa).

In the following, each session secret-public key pair is denoted by $(k, K \equiv g^k)$, and their subscripts indicates which principals employ the key pair. For clarity, the session keys are shown next to the ticket, rather than within the ticket.

1. Client $\rightarrow$ KDC: $c, tgs, N_c$
2. KDC $\rightarrow$ Client: $\{K_{c,tgs}, tgs, N_c, N_{kdc}\}_{Y_c}$, $\{k_{c,tgs}, N_{kdc}, c, lifetime\}_{Y_{tgs}}$

   The KDC first generates the session key pair $(k_{c,tgs}, K_{c,tgs})$ to be used between the Client and the TGS.
   The session "public" key $K_{c,tgs}$ is delivered to the Client, enciphered under the Client's public-key. The session "secret" key $k_{c,tgs}$ is given to the TGS (via the client) encrypted under the TGS's public-key.

3. Client $\rightarrow$ TGS: $\{A_c, s, N_{kdc}\}_{K_{c,tgs}}$, $\{k_{c,tgs}, N_{kdc}, c, lifetime\}_{Y_{tgs}}$, $N'_c$

   The Client uses the session key $K_{c,tgs}$ obtained from the KDC to encipher the authenticator $A_c = (c, timestamp)$ [5, 7] destined for the TGS.

4. TGS $\rightarrow$ Client: $\{K_{c,s}, s, N'_c, N_{tgs}\}_{k_{c,tgs}}$, $\{k_{c,s}, N_{tgs}, c, lifetime\}_{Y_s}$

   Here the key $k_{c,tgs}$ is not used directly in the manner of public keys. The TGS uses the session key $k_{c,tgs}$ to compute another key $r_{tgs,c}$ as:

   $$r_{tgs,c} \equiv (Y_c)^{X_{tgs}+k_{c,tgs}}$$

   The Client can recover the key as

   $$r_{tgs,c} \equiv (Y_{tgs}\ K_{c,tgs})^{X_c}$$

   since $Y_{tgs}$ is public.
   Note that the Client cannot fabricate $\{k_{c,s}, N_{tgs}, c, lifetime\}_{Y_s}$ without being detected by the recipient Server. This is because in deciphering it, the Server will use keys $X_s$ and $Y_{tgs}$ simultaneously. Only these two keys can succeed. In order to fabricate it, the Client would need to know $X_{tgs}$ which is known only to the TGS.

5. Client $\rightarrow$ Server: $\{A_c, N''_c, N_{tgs}\}_{K_{c,s}}$, $\{k_{c,s}, N_{tgs}, c, lifetime\}_{Y_s}$

   The Client uses the session key $K_{c,s}$ to encipher the authenticator $A_c$. The Server deciphers it using the matching secret session key $k_{c,s}$ obtained from within the ticket.

6. Server $\rightarrow$ Client: $\{c, s, N''_c\}_{k_{c,s}}$

If required, the Server may respond to the Client's request of proving the Server's identity. This can be done by the Server using the session key $k_{c,s}$ to create $r_{s,c}$ as:

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

which is recoverable by the Client as:

$$r_{s,c} \equiv (K_{c,s}\, Y_s)^{X_c}$$

since only the Client knows $X_c$.

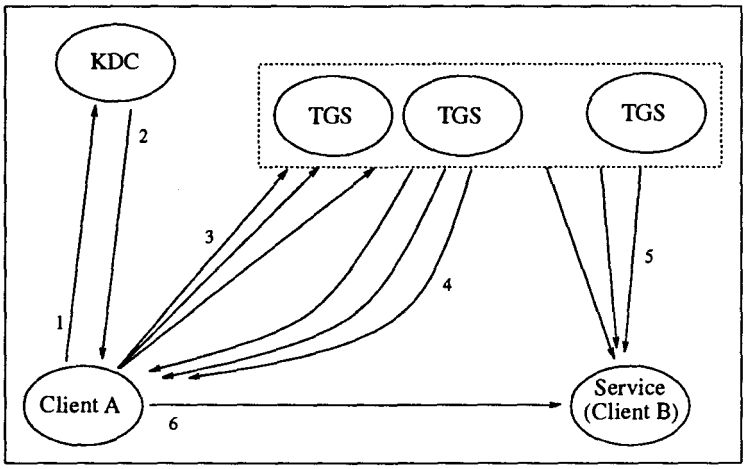


**Fig. 2.** Authentication in Kuperee using several TGSs

## 3 REPLICATING KUPEREE

In this section we extend the proposed protocol of the previous section based on a session secret-public key pair for the purpose of authentication using replicated TGSs. The replication approach is based on the use of secret-sharing schemes (or threshold schemes) [9, 10] to achieve a minimum consensus of honest TGSs. In the current work we do not present any secret-sharing scheme, although the practical scheme of [16] is currently under proposal for Kuperee.

Previous solutions – such as that in [8] – employ a thresholding function to split a parameter (eg. symmetric key) into a number of *shares*, $t$ out of $M$ of which is required at least to recover the original parameter.

In the current work we also employ a thresholding function to achieve the same basic effect. However, in this case the parameter to be shared-out consists

of a session *secret* key which is not directly available to the Client. This conforms with the underlying motivation that there should be a hierarchy of trust assignment in the systems, with the KDC being the most trusted and the Client the least. It is therefore preferable for the Client to hold the public half of the session key-pair, with the Server keeping the secret half.

The current approach is divided into two basic phases. The first is carried-out by the KDC in a continuous manner, where the KDC generates and distributes the keys and the shares to the multiple TGSs. The second phase is invoked whenever a Client requires access to the Server.

### 3.1 Share Generation and Distribution Phase

In this phase the KDC generates a set of session key pairs (secret and public) and associates an identity number with each pair. The secret session key is then broken into a number of shares. These shares, together with the share identity and the session public-key are then distributed to the TGSs in the system.

More specifically, given the session key pair $(k_{c,s}, K_{c,s})$ and given $M$ of TGSs in the system, the KDC generates an identity $ID_{c,s}$ associated with the session key pair and splits the session secret-key into $M$ pieces $P_{c,s_1}, P_{c,s_2}, \ldots, P_{c,s_M}$, where $P_{c,s_i} = thresh_{t.M}(k_{c,s}, i)$ for some threshold function $thresh_{t.M}()$ [8].

The KDC then distributes the parameters

$$\begin{array}{c}(ID_{c,s}, K_{c,s}, P_{c,s_1}) \\ (ID_{c,s}, K_{c,s}, P_{c,s_2}) \\ \vdots \\ (ID_{c,s}, K_{c,s}, P_{c,s_M})\end{array}$$

to the $TGS_1, TGS_2, \ldots TGS_M$ respectively. Each TGS securely holds a database containing these parameters, each entry of which is used for one authentication instance only.

This process is a continuous one, with the KDC ensuring that a ready supply of session key pairs are available for the TGSs.

### 3.2 Authentication Phase

In this phase the authentication over several TGSs occurs, initiated by the Client. The procedure is similar to the traditional single TGS approach, with the addition of one step between the TGSs and the Server (see Figure 2).

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

   Here the Client approaches the KDC in the usual manner with a request to access the TGS.

2. KDC $\rightarrow$ Client:
   $\{ID_{c,s}, K_{c,tgs}, tgs, N_c, N_{kdc}\}_{Y_c}$, $\{ID_{c,s}, k_{c,tgs}, N_{kdc}, c, lifetime\}_{Y_{tgs_i}}$

The KDC provides the Client with a session public key $K_{c,tgs}$ in the ordinary manner together with an identity $ID_{c,s}$ to be presented later by the Client to the TGSs. The KDC also provides a ticket destined for the TGSs which contains the session secret key $k_{c,tgs}$ with its corresponding identity $ID_{c,s}$.

**3.** Client $\rightarrow TGS_i$:
$\{A_c, s, N_{kdc}\}_{K_{c,tgs}}$, $\{ID_{c,s}, k_{c,tgs}, N_{kdc}, c, lifetime\}_{Y_{tgs_i}}$, $N'_{c_i}$

Here the Client obtains the session public key $K_{c,tgs}$ from the previous step and uses it to deliver an authenticator to at least $t$ out of the $M$ TGSs. The Client also forwards the ticket it received from the KDC to each of the TGSs.

**4a.** $TGS_i \rightarrow$ Client: $\{K_{c,s}, s, N'_{c_i}, N_{tgs}\}_{k_{c,tgs}}$

At least one of the TGSs must respond to the Client by delivering the session public key $K_{c,s}$ to the Client. Note that in fact the key used is $r_{tgs_i,c}$ computed as:

$$r_{tgs_i,c} \equiv (Y_c)^{X_{tgs_i}+k_{c,tgs}}$$

If the TGS happens to be dishonest (or has been subverted by an attacker) and delivers a false key, then the authentication process will fail in the ensuing steps.

**4b.** $TGS_i \rightarrow$ Server: $\{P_{c,s_i}, N_{tgs}, c, lifetime\}_{Y_s}$

Each of the $t$ TGSs also deliver to the Server a ticket with the share $P_{c,s_i}$ required by the Server to recover the session secret key $k_{c,s}$.

**5.** Client $\rightarrow$ Server: $\{A_c, N''_c, N_{tgs}\}_{K_{c,s}}$

The Client then uses the session public key $K_{c,s}$ to send the Client's authenticator to the Server.

**6.** Server $\rightarrow$ Client: $\{c, s, N''_c\}_{k_{c,s}}$

As in the single TGS case, the Server uses key $k_{c,s}$ which it now holds (from merging the shares) to compute the actual enciphering key $r_{s,c}$ as:

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

# 4 COMPARISON WITH OTHER APPROACHES

The notion of secret sharing schemes have been in use for sometime in a number of different application [17]. However, in the context of authentication in distributed systems and network security the first notable effort was given by Gong in [8]. It is therefore useful to compare the current proposed scheme for Kuperee with that given in [8].

One of the main underlying differences between the proposed scheme and Gong's scheme in [8] is the use of symmetric (shared-key) cryptosystem in [8]. This has impact on the system on a number of points:

- *Key management.* In [8] the Client and the TGSs (ie. Authentication Servers in [8]) must share a key on a one-to-one basis. This is established by computing the key $K_{a_i}$ – shared between the Client A and the $TGS_i$ – using a hash function $h$ as $K_{a_i} = h(K_a, TGS_i)$ where $K_a$ is the master key known only to the Client A and $TGS_i$ is the unique identity of the $i$-th TGS.
  In our case the overhead in the key management of these shared keys does not exist as the public keys of the Client and the TGSs are readily available in the public domain.

- *Role of the TGSs.* A more pronounced difference lies in the role of the TGSs in Kuperee and in the scheme of [8]. In the replication scheme of Kuperee the TGSs act as a *storage point* for the shares derived from the session secret key. The session secret key is chosen by the most trusted entity, namely the KDC, and it is only ever directly available to one other entity, which is the Server, at the successful completion of the authentication process. It is the KDC that performs the share generation and distribution.
  In the scheme of [8] the TGSs (ie. Authentication Servers) act more as an intermediary in the exchange of two secret parameters $x$ and $y$ between the Client A and the Client B (or the Server B). That is, the TGSs collectively take the role of an *exchange point.* These two parameters $x$ and $y$ are then used by the two parties to compute a common key $K_{AB}$ via some secure one-way function $g$ (ie. Client A sends $x$ to B via the TGSs; B sends $y$ to A via the TGSs; each computes $K_{AB} = g(x, y)$). Thus, the TGSs (ie. Authentication Servers in [8]) do not actually store any shares for longer than the completion of the key exchange period.

- *Selection of participating TGSs.* In the scheme of [8] it is essential that the communicating parties Client A and Client B (or Server B) choose the *same* set of $t$ TGSs. This stems from the fact that the $t$ number of TGSs act as a common exchange point. Thus, if Client B chooses a slightly different set of $t$ TGSs, with some TGS not being in Client A's chosen set, then these non-intersecting set of TGSs will not have the parameter $x$ from Client A. Similarly, the TGSs selected by Client A which are not in the set chosen by Client B will not carry the parameter $y$ from Client B.
  The solution to this dilemma is for both Client A and Client B (or Server B) to select *all* the TGSs when delivering the parameters $x$ and $y$ initially. That is, all the TGSs become the exchange point, receiving copies of parameters $x$ and $y$. After this has occurred, each of the parties can proceed to select any $t$ of the TGSs.
  In the replication scheme of Kuperee the above problem does not exist since all the TGSs are already acting as storage points and are in possession of all the shares respectively. The parties can select any $t$ of the TGSs with no impact on the scheme.

Other differences between the two schemes derive largely from the use of a public key cryptosystem in Kuperee versus a private key (shared key) cryptosys-

tem in [8]. The use of a public key cryptosystem simplifies key management in the domain since any new public-key can be broadcasted by the certification authority. Principal-to-principal secure communications can also be established much more readily without the aid of any trusted third party, albeit with the risk being fully burdened by the two communicating parties (eg. in the case that one of the two turns-out to be a masquerading attacker).

## 5 REMARKS AND CONCLUSION

In this work we have proposed a new scheme for the replication of authentication services based on a public key cryptosystem, in response to the two main shortcomings of the traditional single server solutions. First, since entities in the distributed system rely on the server for their authentication operations, the server easily becomes a source of contention or bottleneck. The temporary unavailability of the server can lead to a degradation in the overall performance of the distributed system. Secondly, the fact that the server may hold cryptographic (secret) keys belonging to entities in the distributed system makes it the best point of attack for intruders wishing to compromise the distributed system.

A comparison with the notable work of Gong [8] has been presented, focusing on the issues of key management, the role of the TGSs and the selection of the TGSs by the communicating parties in the system. Although the two approaches differ in their underlying use of a public key cryptosystem in Kuperee and a shared key (private key) cryptosystem in [8], some of the observations are useful to illustrate the various abilities and usefulness of the two approaches in differing environments.

**Acknowledgements**

We thank Anish Mathuria for indepth comments on Kuperee. This work has been supported in part by the Australian Research Council (ARC) under the reference number A49232172, A49130102 and A49131885, and by the University of Wollongong *Computer Security: Technical and Social Issues* research program.

## References

1. T. Hardjono and J. Seberry, "Authentication via multi-service tickets in the *kuperee* server," in *Computer Security – ESORICS'94: Proceedings of the Third European Symposium on Research in Computer Security* (D. Gollmann, ed.), vol. 875 of *LNCS*, pp. 143–160, Springer-Verlag, 1994.
2. 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.
3. T. Hardjono, "Kuperee simplified," Technical Report Preprint 95-5, Centre for Computer Security Research, Computer Science Department, University of Wollongong, December 1994.

4. Y. Ding and P. Horster, "Why the kuperee authentication system fails," *Operating Systems Review*, vol. 30, no. 2, pp. 42–51, 1996.
5. 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.
6. S. M. Bellovin and M. Merritt, "Limitations of the Kerberos authentication system," *Computer Communications Review*, vol. 20, no. 5, pp. 119–132, 1990.
7. J. T. Kohl, "The evolution of the *kerberos* authentication service," in *Proceedings of the Spring 1991 EurOpen Conference*, (Tromsø, Norway), 1991.
8. L. Gong, "Increasing availability and security of an authentication service," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 5, pp. 657–662, 1993.
9. A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
10. G. R. Blakley, "Safeguarding cryptographic keys," in *Proceedings of the National Computer Conference*, AFIPS Conference Proceedings, Vol.48, pp. 313–317, 1979.
11. 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.
12. R. M. Needham and M. D. Schroeder, "Authentication revisited," *Operating Systems Review*, vol. 21, no. 1, p. 7, 1987.
13. T. Y. C. Woo and S. S. Lam, "Authentication for distributed systems," *IEEE Computer*, vol. 25, pp. 39–52, January 1992.
14. W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, 1976.
15. T. El Gamal, "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.
16. Y. Zheng, T. Hardjono, and J. Seberry, "Reusing shares in secret sharing schemes," *The Computer Journal*, vol. 17, pp. 199–205, March 1994.
17. G. J. Simmons, "An introduction to shared secret and/or shared control schemes and their application," in *Contemporary Cryptology* (G. J. Simmons, ed.), pp. 441–497, IEEE Press, 1992.

## APPENDIX: KUPEREE ALGORITHMS

The approach in Kuperee is based on the public key cryptosystem of [2]. Here 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 [2]. 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.

The following notation is taken directly from [2]. The cryptosystem of [2] 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 $w_{[i\ldots j]}$ $(i \leq j)$ 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 [2]. $G$ 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 $\ell$-bit output strings.

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 generates a session key pair by first calculating $k_{c,tgs} \in_R [1, p-1]$ followed by the calculation $K_{c,tgs} \equiv g^{k_{c,tgs}}$.

The KDC then enciphers the session key $K_{c,tgs}$ intended for the Client who owns the public-key $Y_c$ by invoking *Encipher* (Algorithm 1) with the input parameters $(p, g, r_c, K_{c,tgs})$ where $r_c \equiv (Y_c)^{X_{kdc}+x_c}$ for some random $x_c \in_R [1, p-1]$. The output of *Encipher* (Algorithm 1) that is sent to the Client is in the form $C_c = \{C||y_c\}$ where $y_c \equiv g^{x_c}$.

---

**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**

---

Upon receiving the ciphertext $C_c = \{C||y_c\}$ the Client attempts to decipher the ciphertext by first computing $r' \equiv (Y_{tgs} y_c)^{X_c}$ and using this as input to *Decipher* (Algorithm 2). More specifically, the Client inputs That is, the TGS inputs $(p, g, r', C)$ resulting in the output $K_{c,tgs}$.

**Algorithm 2** *Decipher*$(p, g, r', C)$

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

**end**

In general, the same procedure is followed by each principal who must compute the parameter $r$ (either directly or by selecting a random number $x$) and use it as input to either *Encipher* (Algorithm 1) or *Decipher* (Algorithm 2).

Readers interested in the security of Kuperee are directed to [2] which discusses the security of the public key cryptosystem upon which Kuperee is directly built.