Kuperee Simplified

Thomas Hardjono

Centre for Computer Security Research
University of Wollongong, NSW 2522, Australia
email: thomas@cs.uow.edu.au

19 December 1994

Abstract
In this current work we simplify the authentication protocol of Kuperee by removing all its features and presenting it in the minimum form required to achieve authentication in the Kerberos manner. This minimal form of Kuperee facilitates the further modification of the various parameters of Kuperee to achieve different effects, taking advantage of the fact that underlying Kuperee is a public-key cryptosystem which allows for various key compositions. Two approaches for authentication in Kuperee are discussed. The first relies on the sharing of a session secret-public key pair much in manner of the Needham-Schroeder protocol. The second approach is based on the use of a group-shared longer term public-key, which exploits the nature of the public key cryptosystem employed.

1. Introduction

The *Kuperee* authentication/certification server was first presented in [1] as public-key based alternative to the Kerberos authentication system [2, 3]. In [1] a number of advantages of Kuperee was discussed. These include the binding together of two entities by a third entity, leading to the need of equal co-operation by the two entities in order to complete the authentication procedure. In addition, Kuperee features *multi-service* tickets, one of which can be used by a Client to access multiple service providers with a single ticket.

In this current work we simplify the authentication protocol of *Kuperee* by removing all its features and presenting it in the minimum form required to achieve authentication in the Kerberos manner. This minimal form lends itself to several modifications which can make use of the public-key nature of Kuperee. These may include the use of certified public keys of the principals rather than private session-keys such as that in Kerberos.

In the following we will discuss the simplified Kuperee, focusing on the procedure used to achieve an authentication instance.

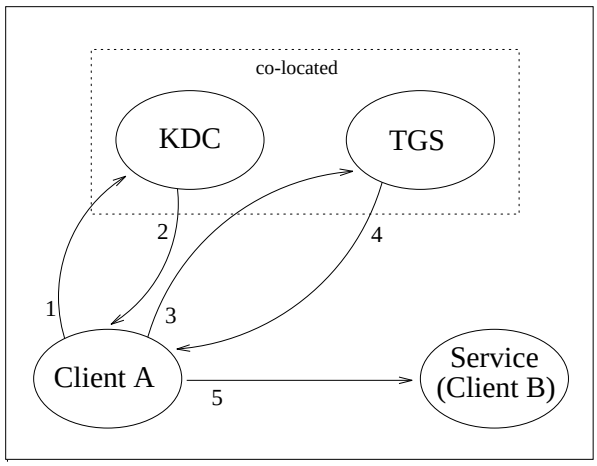

Figure 1. Authentication in Kuperee following Kerberos

2. Authentication in Kuperee

The approach of Kuperee [1] mimics that of the Kerberos authentication system [2, 3] which is based on the Needham-Schroeder protocol [4, 5] and which uses a symmetric (private-key) cryptosystem. However, unlike Kerberos, Kuperee employs an asymmetric (public-key) cryptosystem [6] which provides it with a number of different features to Kerberos (see [1]). In the following we will briefly review the Kuperee authentication system as a point of reference, using the terminology of Kerberos for ease of understanding.

In the authentication system the entities that interact with the Authentication Server 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 another server (or another Client) on a remote machine.

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 [7].

2.1. Kuperee: Notations

In the public key cryptosystem of [6] 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 multiplica-

tive 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 [8] and [9]. 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.

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 [6].

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. 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

In this approach 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 of 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 and 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. The tickets are assumed to contain the usual parameters, such as the sender and receiver identity, timestamp, lifetime and other necessary parameters.

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

2. KDC $\rightarrow$ Client: $\{K_{c,tgs}\}_{Y_c}$, $\{T_{c,tgs} || k_{c,tgs}\}_{Y_{tgs}}$

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

The "public" session-key $K_{c,tgs}$ is delivered to the Client, enciphered under the Client's public-key. The "secret" session-key $k_{c,tgs}$ is placed inside or together with the ticket $T_{c,tgs}$ destined for the TGS. These are enciphered under the TGS's public-key.

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

The Client uses the session-key $K_{c,tgs}$ to encipher the authenticator A_c [2, 3], the result of which is decipherable only by the TGS since TGS holds the matching session-key $k_{c,tgs}$ obtained from the ticket.

4. TGS $\rightarrow$ Client: $\{K_{c,s}\}_{r_{tgs,c}}$, $\{T_{c,s}||k_{c,s}\}_{Y_s}$

The TGS reuses the session-key $k_{c,tgs}$ by calculating the key $r_{tgs,c}$ as:

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

The Client can decipher $\{K_{c,s}\}_{r_{tgs,c}}$ by re-covering the key as

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

since Y_{tgs} is public.

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

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

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 session-key $k_{c,s}$. The key is reused to create $r_{s,c}$ as:

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

which is recoverable by the Client as:

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

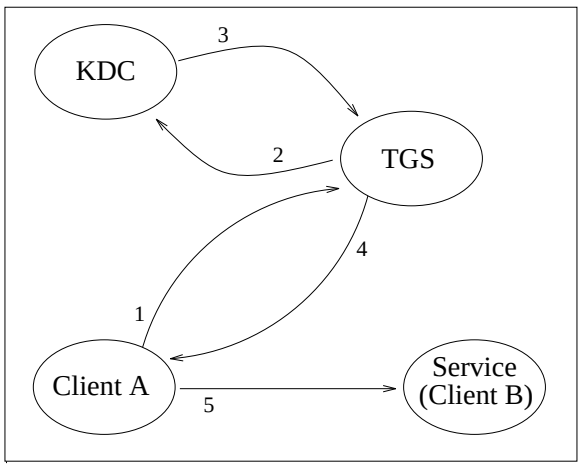

Figure 2. Authentication in Kuperee based on Group Sharing of Public Keys

2.3. Kuperee: Group Sharing of Long Term Public Keys

Another approach to authentication using Kuperee is to designate the KDC and the TGS more as a certification authority for the public-keys being used in the system. More specifically, in this approach the Client employs two pairs of secret-public key pairs, one designated with a longer life-time compared to the other. The long term key pair is shared only among the KDC-TGS-Client, with the KDC being the certification authority for the TGS. The short term key pair is used by the the TGS-Client-Server, with the TGS being the certification authority for the Server.

For this approach, assume that the Client is associated with the key pairs $(X_{c_1}, Y_{c_1} \equiv g^{X_{c_1}})$ and $(X_{c_2}, Y_{c_2} \equiv g^{X_{c_2}})$. Here, we assume that the KDC securely generates the pair (X_{c_1}, Y_{c_1}) and delivers them securely to the Client, after which the KDC erases X_{c_1} (Alternatively, the Client can generate the pair, but an authentic delivery of Y_{c_1} must be performed in such a way that the KDC is convinced of the identity of the Client associated with Y_{c_1}). Similarly, the TGS generates the key pair (X_{c_2}, Y_{c_2}) and delivers it securely to the Client. The TGS then erases X_{c_2} and acts as a certification authority for Y_{c_2}. The authentication procedure is given in the following (Figure 2).

1. Client $\rightarrow$ TGS: c, s

2. TGS $\rightarrow$ KDC: c

3. KDC $\rightarrow$ TGS: $\{T_{c,tgs} || Y_{c_1}\}_{Y_{tgs}}$

4. TGS $\rightarrow$ Client: $\{Y_s\}_{Y_{c_1}}$, $\{T_{c,s} || Y_{c_2}\}_{Y_s}$

5. Client → Server: $\{A_c\}_{Y_s}$, $\{T_{c,s}||Y_{c_2}\}_{Y_s}$

6. Server → Client: $\{timestamp+1\}_{Y_{c_2}}$

Note that Steps 2 and 3 are unnecessary once the TGS already obtains a certified copy of the public-key Y_{c_1}. In contrast, due to the shorter life-time of public key Y_{c_2} we assume that the TGS must always deliver the most recent copy of Y_{c_2} to the Server (or the Server may ask a copy of it from the TGS).

3. Kuperee: Algorithms

The approach in Kuperee is based on the public key cryptosystem of [6]. 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 [6]. 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 [6]. The cryptosystem of [6] 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 $w_{[i\cdots j]}$ $(i \leq j)$ 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 [6]. 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 [6] for a comprehensive discussion on the constructs of the family of cryptosystems.

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).

4. Remarks and Conclusion

In this brief work we have simplified the Kuperee authentication system by stripping it of the various features contained within the original version of [1]. This has resulted in the protocol being fairly similar in appearance to the Needham-Schroeder protocol employed in Kerberos, with a crucial difference being the public-key cryptosystem employed in Kuperee. Using this minimal form a number of possibilities for other features of Kuperee present themselves, taking advantage of the public-key cryptosystem where principals in the system have knowledge of each other's public-key. These will be the subject of further study, focusing also on the possible further simplifications based on the public nature of the public-keys of the principals.

Acknowledgements

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 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.

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

4 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.

5 R. M. Needham and M. D. Schroeder, "Authentication revisited," *Operating Systems Review*, vol. 21, no. 1, p. 7, 1987.

6 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.

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

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

9 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.