# The Development and Testing of the Identity-based Conference Key Distribution System for the RHODOS Distributed System*

Michael Wang[1] and Andrzej Goscinski[2]

[1] (mikew@vast.unsw.edu.au)
School of Computer Science and Engineering
University of New South Wales
Kensington, N.S.W., 2033
Australia

[2] (ang@cm.deakin.oz.au)
Department of Computing and Mathematics
Deakin University
Geelong, Victoria 3217
Australia

**Abstract.** In this paper, we demonstrate that it is possible to develop an authentication service as an integral part of a distributed operating system, subject to some requirements and extensions to the original Koyama–Ohta system. The basic RHODOS requirement is that users cannot be trusted, and therefore they cannot hold any cryptographic parameters, but their own passwords. The Authentication Service supported by the RHODOS distributed operating system provides the following operations: the distribution of the initial cryptographic parameters, user login authentication, one-way and two-way authentication, and conference authentication. The operational properties have been demonstrated by setting up a conference and authenticating conference participants in different circumstance.

**Keywords:** Identity-based Conference Key Distribution, Authentication, Distributed Operating Systems, Distributed Systems

## 1 Introduction

In a distributed computing environment, resources are shared among computing nodes that are located at different geographical locations. Remote access to data is very frequent. As a result, high volume of important and valuable information

* This work was partly supported by the Australian Research Council under Grant A48831034.

Y. Deswarte et al. (eds.), *Computer Security - ESORICS 92*

is transmitted via insecure communication channels. This opens the possibility of attacks on the communication channels by intruders. There are two basic forms of attack: passive and active. In the case of the passive attack, data transmitted on the network can be disclosed to unintended recipients. In the latter case, data transmitted on the network can be modified. Intruders can also deny or delay message delivery. Furthermore, intruders can masquerade as legitimate users to read, modify existing data or submit false data. This implies that users can no longer trust each other. Given this, a distributed system by itself cannot trust the users either. In this paper, we shall only concentrate on the prevention of the active attack and the detection of masquerade.

The requirement imposed by distributed systems is to protect data from tampering and to prevent masquerading. In order to do this, users must be authenticated before logging on to the systems. Detection of data tampering can be achieved via one-way authentication. Protection against masquerading requires the system and user to be able to mutually authenticate each other. This requires two-way (mutual) authentication.

A distributed system offers another unique opportunity: resources can be shared among various nodes. This interaction corresponds to the notion of participants communicating to each other in a conference. Typical applications of a conference scenario are concurrent group work and synchronous remote meetings [GB89, NDV+91]. Authentication in a conference can be achieved by two-way authentication. However, it is believed that there are more efficient methods for conference authentication. Therefore, authentication schemes adopted by a distributed system must also cater for conference authentication in an efficient manner.

The solution to the conference authentication problem can be found in the scheme called Identity-based Conference Key Distribution System (ICKDS) proposed by Koyama and Ohta [KO88b]. It was designed to provide mutual authentication among several parties simultaneously and at the same time to distribute a common key among all parties. Thus, at the end of a conference authentication process all participants can communicate with each other securely using the common conference key. This scheme applies Shamir's identity-based cryptosystem [Sha85] in conjunction with Diffie-Hellman's public-key cryptosystem [DH76]. The idea is to authenticate users by using their identification information. This scheme is general enough to be used for one-way and two-way authentication as well. There is currently no known implementation of ICKDS. Readers are referred to the ICKDS paper [KO88b] for details of the authentication scheme. It will not be elaborated here.

In this paper, we shall describe the design and implementation of an authentication server that is based on the ICKDS scheme but with functionally improved features such that it caters for the needs of distributed systems. The improvements are outlined in Section 3. The development is based on the RHODOS [GGI+91] distributed system. The RHODOS Authentication Server supports conference authentication with simultaneous key distribution. At the same time, the server supports location transparency for conference participants. That is, a participant does not need to know where other participants of the conference are located. The knowledge of their login names will suffice. The implementation is software-based since the Authentication Server is one of the servers of the RHODOS distributed operating system. The decision to place an authentication service in a distributed operating system as its integral part, proposed in [Gos91, GP91], was made for the security and performance reasons. The software-based approach also offers the flexibility for experimentation.

The paper is organised as follows. Section 2 gives an overview of the authentication service provided by RHODOS. Basic assumptions and operational extensions to the Authentication Server are described in Section 3. Section 4 shows how RHODOS overcomes an effective attack on the ICKDS scheme that was published recently. The logical design of the authentication operations is given in Section 5. Current status of our research is reported in Section 6. The conclusion is given in Section 7.

## 2 Overview of the Authentication Server of RHODOS

The authentication service developed for the RHODOS distributed operating system is based on both a symmetric cryptosystem and the Identity-based Conference Key Distribution System [Koy87, KO88b] which is itself based on the public-key cryptosystem. Traces of the basic concept used in zero-knowledge proofs [GMR89] protocols can also be found in the distribution of the initial cryptographic parameters. The RHODOS' authentication service does not restrict authentication to be between users in a conference. It permits user-to-server authentication as well as server-to-server authentication.

The authentication operations of RHODOS can be described as both centralised and distributed. It is centralised in the sense that it relies on a trusted centralised server for *initial identity* authentication. User login authentication is a typical centralised authentication operation. Distributed authentication means the authentication procedure can be carried out by agents of the centralised authentication servers without consulting the centralised server itself. Thus, the centralised server is not involved in every authentication carried out in the system. User initiated authentication operations are typical example of distributed authentication.

Distributed authentication is achieved by having an authentication server in each node. However, there is only one trusted centre as in the ICKDS scheme. This centre is known as the Central Authentication Server (CAS) and all other authentication servers are known as Authentication Server Agents (ASAs). The cryptographic parameters required by ICKDS are only generated by the CAS and distributed securely to the ASAs. They are never exposed to the users — this is the critical assumption and feature of the RHODOS authentication.

From the user point of view the authentication operation in RHODOS is a simple sending of an authentication request to a local ASA. The users do not have any knowledge of the authentication scheme used by their local ASAs. In all cases, only the authentication result is returned to the users. This is done for two reasons. First, the users cannot be trusted to hold any cryptographic information securely. Second, this has the advantage of changing the underlying authentication scheme without affecting users that use the authentication service.

The cryptographic parameters generated by the CAS when it first starts to run are $g$, $P$, $Q$, $R$, $N$, $K_P$, $K_S$ and $S_i$. These cryptographic parameters are defined below. They are identical to those defined by the ICKDS scheme.

- $P$, $Q$, and $R$ are large prime numbers
- $N = P \times Q$
- $g$ is a primitive root over $GF(P), GF(Q)$ and $GF(R)$
- $K_P$ and $K_S$ are the public and private keys of the trusted center, respectively. $K_P$ is a prime such that $(N \cdot R)/2 < K_P < (N \cdot R)$ and $K_S$ has the property such that
$$K_P \cdot K_S \equiv 1 \pmod{L}$$
where $L = \text{lcm}((P-1),(Q-1),(R-1))$.
- Signature $S_i$ for user $i$ whose identification information is $I_i$ is defined as follows:
$$S_i \equiv I_i^{K_S} \bmod (N \cdot R)$$
Note that $I_i \equiv S_i^{K_P} \bmod (N \cdot R)$.

The parameters $(P, Q, K_S)$ are collectively called the *secret system-key*, $(g, N, R, K_P)$ are known as the *public system-key* and $S_i$ is the secret *signature* for user $i$ [KO88b]. The purpose of the secret system-key is to generate the public system-key and secret user signatures. The public system-key and the secret signatures when used together allow clients to be authenticated by the ASAs without consulting the CAS.

# 3 Assumptions of the RHODOS Authentication Environment and Extensions to the ICKDS scheme

Certain assumptions have been made on the operating environment of the RHODOS' authentication service. They are as follows.

- The system environment consists only of workstations.
- Communications with remote entities are via insecure channels.
- System software downloaded to workstations during bootup is authentic.
- The CAS is trusted and is located in a physically secure environment, e.g., behind locked doors.
- Users are not to be trusted.
- Users do not generate or hold any cryptographic information. They only hold their own private passwords.
- Cryptographic parameters are stored in the physical memory of workstations and are protected by the operating system.
- The local RHODOS kernel and local ASA are trusted by their local processes.

- No ASAs trust each other.
- Identities of all communicating entities are unique and well known. In the case of RHODOS, the identities are in the system name form [GGI+91] and is known as a user **SName**.
- System administrators are trusted.

The physical environment for ICKDS [KO88a] is very different to that of RHODOS [GGI+91]. Some changes to ICKDS were needed to incorporate ICKDS into RHODOS.

- ICKDS was designed as a key distribution system and in doing so it realizes only sender authentication. RHODOS, on the other hand, allows client messages to be sent as part of the authentication process. These messages are sent in the clear and are therefore vulnerable to attack. To overcome this problem, checksum of client messages is introduced into the RHODOS authentication protocol to avoid message modification.
- The ICKDS protocol does not prevent message replay. This problem is avoided by use of time-stamps in the RHODOS authentication protocol.
- In ICKDS, the security of the cryptographic parameters are maintained in users' SmartCards. Their security is users' responsibility. SmartCards are not available in RHODOS. Thus, the security responsibility of the cryptographic parameters are given to the Authentication Server Agents (ASAs) themselves.
- The clients of ICKDS are restricted only to users. RHODOS allows server processes as well as user processes to be clients of the ASA.

# 4 RHODOS' Handling of an Effective Attack on ICKDS

At the November 1991 ASIACRYPT conference, an effective attack on the ICKDS scheme was presented in a paper by Shimbo and Kawamura [SK91]. The attack requires the conspiracy of two or more participants of a conference, not including the initiator of the conference. By exchanging cryptographic information generated for conference authentication the conspirators can effectively disclose the conference initiator's secret information (signature).

This attack, however, does not cause any threats in our system. This is because one of our initial assumptions was not to trust the users. Therefore, in our design, no authentication information was ever given to users (or user processes). They are only informed of the success or failure of authentication.

Without direct access to cryptographic information the attackers cannot conspire to disclose secret information in the manner described by Shimbo and Kawamura. It is, however, possible for attackers to obtain cryptographic information by wire-tapping and thus mounting conspiracy attack in the original ICKDS protocol. To overcome this attack, we have added an additional layer of protection by encrypting the cryptographic information used for conference authentication with the public key of the ASAs. The details of this countermeasure is described in Section 5.3.

By not giving the users any secret information and a slight modification of the original ICKDS protocol we have eliminated this conspiracy attack. In fact, with this design, any user level conspiracies have been eliminated.

# 5 Authentication Operations

Authentication operations supported by RHODOS are:

- The distribution of the initial cryptographic parameters;
- User login authentication;
- One-way authentication;
- Two-way authentication; and
- Conference authentication.

The distribution of the initial cryptographic information is based on symmetric cryptosystem. User login authentication is based on a combination of symmetric cryptosystem and asymmetric cryptosystem. The other authentication operations are all based on the ICKDS scheme. The protocols used for the one-way and two-way authentication are basically the same as for the conference authentication but with fewer messages. Their protocols will not be described here. Nevertheless, a brief operational description for them is given in Section 5.4. Details of the authentication protocols can be found in [Wan92].

## 5.1 The Distribution of the Initial Cryptographic Parameters

In any secure environment there is always the problem of how to transmit some common secrets used for authentication from the trusted center to the intended receiver without revealing them to others. In the RHODOS' environment, this involves passing the cryptographic parameters that are generated by the CAS to the ASAs without compromising them. Typically, these cryptographic parameters are transmitted manually to the intended receiver either by the use of special couriers or by the use of special devices such as SmartCards. The latter is the method suggested by [Sha85] and employed by [KO88b]. However, this method is restricted by the use of special hardware devices; RHODOS is not equipped with such devices. Accordingly, the distribution of the cryptographic parameters to ASAs in RHODOS is performed at the time of workstation boot-up. This scheme is based on the use of a symmetric cryptosystem as discussed in [SP89, Gos91]. The scheme is, in fact, similar to the way the initial ticket is obtained in Kerberos [SNS88].

Suppose Workstation A in Fig. 1 has just been turned on by an administrator, *admin*. Existing systems, after self-tests, will send download request to the file server. Workstations in RHODOS will, instead, prompt for a user id and a password, and they will not proceed until a login name and a password are entered. Suppose *admin* has entered his/her login name and password, Workstation A then sends *admin*'s login name to the CAS running on the *Trusted Center* requesting for authentication. The CAS checks its authentication database to confirm that the login name it received is indeed a registered client and it has the authority to boot-up a workstation. The CAS then retrieves *admin*'s private key from the database and uses it to encrypt the necessary cryptographic parameters. These parameters are the public system-key and signatures of *admin* and $ASA_A$. The encrypted data is sent back to the ASA of Workstation A, $ASA_A$. $ASA_A$ then uses *admin*'s private key, converted from *admin*'s password, to decrypt the data sent from the CAS. If the supplied password from *admin* is correct then the converted encryption key will be the same as the one used by the CAS for encryption. $ASA_A$ can therefore decrypt the message and obtain necessary cryptographic parameters. As a consequence, the booting process continues. If *admin*'s password is incorrect then decryption will fail and *admin* is asked to reenter his/her login name and password. The whole process starts all over again.

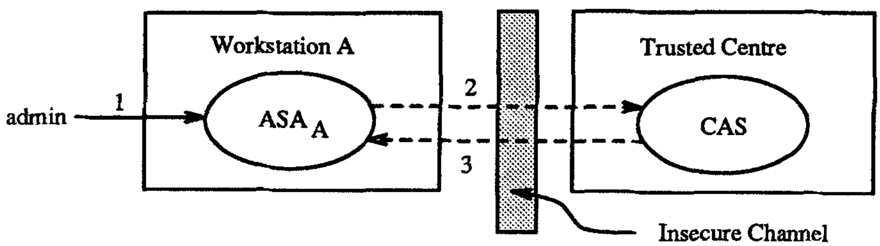


**Fig. 1.** Distribution of the Initial Cryptographic Parameters.

The following protocol describes the information exchanged between various entities involved in this authentication scenario (Fig. 1).

**Message 1.** $admin \rightarrow ASA_A : I_{admin}, passwd$ (Plain-text)
A system administrator, *admin*, powers-on Workstation A and he/she is prompted for his/her login name, $I_{admin}$, and password, *passwd*. He/she enters this information which is passed to the local ASA, $ASA_A$.

**Message 2.** $ASA_A \rightarrow CAS : I_{admin}$ (Plain-text)
$ASA_A$ sends the user's login name, $I_{admin}$, to the CAS requesting authentication of the current user.
CAS checks its authentication database to see whether *admin* is a registered client. If so, the CAS retrieves the symmetric key, $K_{admin}$, which is known only to the CAS and user *admin*, from the authentication database and encrypts the following items using $K_{admin}$:

- *admin*'s login name, $I_{admin}$;
- the public system-key, $(g, R, N, K_P)$;
- *admin*'s signature, $S_{admin}$; and
- $ASA_A$'s signature, $S_{ASA_A}$

**Message 3.**

$CAS \rightarrow ASA_A : \mathcal{E}_{K_{admin}}(I_{admin}, g, R, N, K_P, S_{admin}, S_{ASA_A})$ (Encrypted)
The CAS sends the encrypted message to $ASA_A$. The nature of this message exhibits the basic nature of zero-knowledge proofs in that it is a challenge from CAS to $ASA_A$ to decrypt this message.

$ASA_A$ receives Message 3 and accepts the challenge by converting *admin*'s password to an encryption key, $K'_{admin}$, and tries to decrypt Message 3. If *admin*'s password is correct then $K'_{admin} = K_{admin}$ and thus it can decrypt Message 3, thereby obtaining necessary cryptographic parameters. The booting process then continues. If *admin* is an impostor then he/she does not have the right password which implies $K'_{admin} \neq K_{admin}$. Thus, $ASA_A$ has failed the challenge from CAS.

Irrespective to the results of the decryption both the password and the converted key are destroyed by $ASA_A$ after decryption.

A side effect of this process is that $ASA_A$ is also sure that the information sent to it is genuine since only CAS can encrypt data using the correct user private key.

Note that it is possible for this message to be wire-tapped by intruders. However, since it is encrypted it reveals nothing to the intruders. Furthermore, it is free from insider attack since the system administrator, in this case *admin*, is trusted. Therefore, *admin* cannot possibly wire-tap to obtain this message and then decrypts it with its own key to reveal the cryptographic parameters. This message, however, suffers from password guessing attack as in Kerberos [BM90].

The public system-key obtained from the CAS allows $ASA_A$ to perform authentication with its counterparts that also have the same public system-key without further interaction with the CAS. In fact, the public system-key and $ASA_A$'s signature can be used to authenticate Workstation A to a file server for downloads.

This scenario requires the ASA to be resident in a PROM so it can start to function before system software is downloaded. Since the hardware platform used in RHODOS, at this stage, is the standard Sun 3/50s, hence, this scenario is not implemented. Instead, the cryptographic parameters are passed to $ASA_A$ using the method described for user login authentication.

## 5.2 User Login Authentication

User login authentication is performed with the cooperation of the ASAs and the CAS. The exchanged messages for this scenario are illustrated in Fig. 2. Assume $a$ is the user trying to login, $L$ is taken to be the login process of Workstation A and $ASA_A$ the local ASA of Workstation A. The input from user $a$ to the login process $L$ is omitted from the following protocol.

**Message 1.** $L \rightarrow ASA_A : I_a, passwd$ (Plain-text)

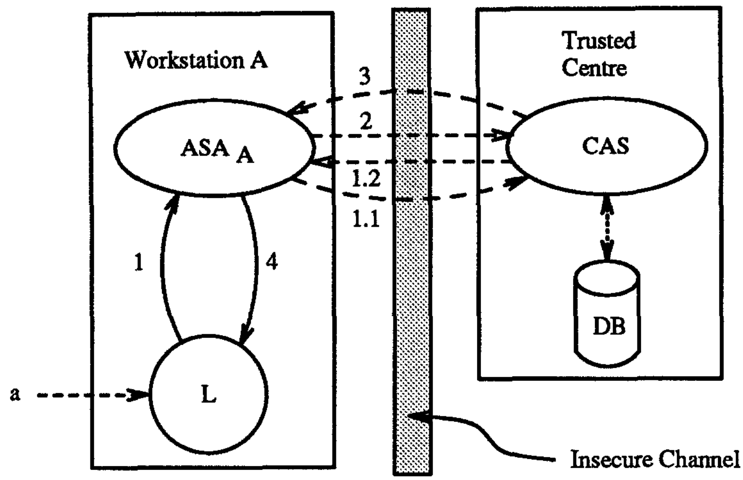


**Fig. 2.** User Login Authentication.

User $a$ tries to login and enters his/her login name, $I_a$, and private password, *passwd*. These are passed to the local ASA, $ASA_A$, by the login process, $L$. $L$ then destroys its copy of $a$'s private password.

**Message 1.1.** $ASA_A \rightarrow CAS$: *enquire_public_key* (Plain-text)

When $ASA_A$ does not possess the public key of CAS it sends a broadcast message requesting for such a key. Note that this message is not necessary if $ASA_A$ already possess the public key of CAS.

**Message 1.2.** $CAS \rightarrow ASA_A : K_{CAS}$ (Plain-text)

The CAS returns its public key, $K_{CAS}$, to $ASA_A$. At the same time, other ASAs may also return $K_{CAS}$ to $ASA_A$. So, if there are discrepancy in the $K_{CAS}$ received by $ASA_A$ then it knows something is wrong. Note that if Message 1.1 was not sent then this message does not appear.

**Message 2.** $ASA_A \rightarrow CAS : (I_a, \mathcal{E}_{K_a}(I_a), K_{ASA})^{K_{CAS}}$ (Encrypted)

Having obtained the public key of CAS, $K_{CAS}$, $ASA_A$ now encrypts the identity of $a$ with $a$'s *symmetric* key, $K_a$, which is converted from $a$'s password. The resulting ciphertext, $\mathcal{E}_{K_a}(I_a)$, is combined with the $ASA_A$'s own public key, $K_{ASA_A}$, and encrypted using the *public* key of the CAS. The result of this is then sent to CAS.

The point in encrypting $a$'s identity with $a$'s private key is to allow CAS to authenticate $a$ directly without taking the risk of compromising the cryptographic parameters to be sent in the next message due to cryptanalysis. By encrypting the complete message it stops attackers from substituting the public key of $ASA_A$ and can therefore decrypt the next message.

The CAS can verify the message came from $ASA_A$ by checking its capabilities. (Capabilities in RHODOS stops uncontrolled passing of access rights and detection of stolen capabilities.) The CAS then checks its authentication database to confirm $a$ as a registered client by decrypting $\mathcal{E}_{K_a}(I_a)$.

**Message 3.**

$CAS \rightarrow ASA_A : (I_a, \mathcal{E}_{K_a}(g, N, R, K_P, S_a, S_{ASA_A}))^{K_{ASA_A}}$ (Encrypted)

When CAS is satisfied that user $a$ is authentic it encrypts (i) the public system-key, $(g, N, R, K_P)$; (ii) $a$'s signature, $S_a$; and (iii) $ASA_A$'s signature, $S_{ASA_A}$ using $K_a$. The resulting ciphertext together with $a$'s identity is then encrypted using the public key of $ASA_A$, $K_{ASA_A}$. This encrypted message is then sent back to $ASA_A$.

This message is encrypted with $K_{ASA_A}$ to stop "insider attack" where $a$ can be wire-tapping in order to obtain the attached cryptographic parameters. The cryptographic parameters are themselves encrypted with $K_a$ to add another layer of protection.

Note that this message is sent only when user $a$ has been authenticated by the CAS.

**Message 4.** $ASA_A \rightarrow L$ : *result* (Plain-text)

If after a time-out period Message 3 was not received by $ASA_A$ then user $a$ is deemed to have failed the login authentication. Otherwise, if Message 3 was received and it decrypts successfully then user $a$ is deemed to be authentic and allowed to logon. The result of the authentication, in the form of *success* or *failure*, is returned to the login process $L$. No cryptographic parameters are given to the users. This includes both the public system-key and user signatures.

Note that, as we indicated earlier, users do not have access to the cryptographic parameters. They are kept by users' local ASAs. When users logoff, their signatures will be destroyed by their local ASAs.

### 5.3 Conference Authentication

**Setting up a Conference.** The conference authentication scenario describes a conference situation among $n$ users (participants) where all of them must be authenticated before the conference can begin. In the conference, there is an initiator of the conference who sends invitations to all potential participants. The invitees then decide whether to participate or not. If they do decide to participate then they send messages back to the initiator to indicate their willingness to participate in the conference.

The messages exchanged for conference authentication are shown in Fig. 3. In the figure, $a$ is assumed to be the initiator of the conference; $b$ and $c$ are the two conference participants; $ASA_A$, $ASA_B$, and $ASA_C$ are the local ASAs of Workstations $A$, $B$, and $C$, respectively.

For this authentication scenario to occur, conference participants $a, b$ and $c$ must have completed the user login authentication. At the same time, $ASA_A$, $ASA_B$ and $ASA_C$ must all possess the public system-key $(g, N, R, K_P)$ and the appropriate signature of their individual clients.

Note that this scenario does not involve the CAS.

**Message 1.** $a \rightarrow ASA_A$ : *conf_name, invitees* (Plain-text)

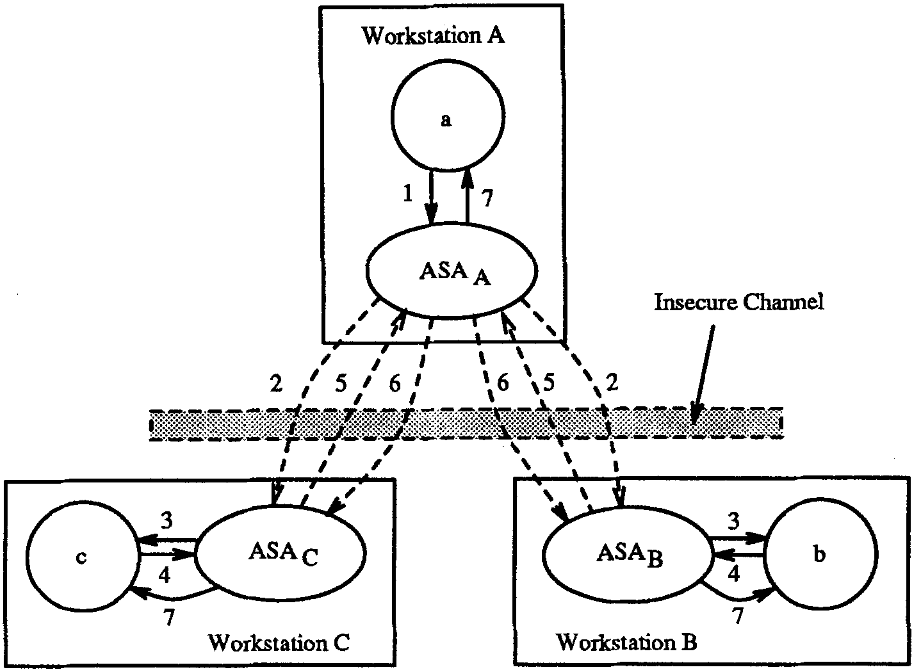


**Fig. 3.** Conference Authentication.

Client $a$ starts an authenticated conference by sending a request to its local ASA, $ASA_A$, with the name of the conference, *conf_name*, and a list of invitees, *invitees*, in this case, $b$ and $c$.
Invitees are designated by their login name. Domain names can also be attached to it, e.g., mikew@vast.unsw.edu.au. The use of a login name provides location transparency for users requesting authentication. However, if necessary, the authentication server will be provided with the exact locations of the invitees.

**Message 2.** $ASA_A \rightarrow ASA_B$: $b$, *conf_invitation*, $K_{ASA_A}$ (Plain-text)
$ASA_A \rightarrow ASA_C$: $c$, *conf_invitation*, $K_{ASA_A}$ (Plain-text)

$ASA_A$ sends the conference invitation to the local ASAs of the invitees, i.e., $ASA_B$ and $ASA_C$. It also informs the other local ASAs of its public key, $K_{ASA_A}$. Note that although it is shown here that the message is delivered one by one to $ASA_B$ and $ASA_C$, however, this message can be sent as a multicast to the appropriate ASAs, thus reducing the number of messages down to one.

**Message 3.** $ASA_B \rightarrow b$ : *conf_invitation* (Plain-text)
$ASA_C \rightarrow c$ : *conf_invitation* (Plain-text)

$ASA_B$ and $ASA_C$ forward the conference invitation to the invitees $b$ and $c$, respectively.

**Message 4.** $b \rightarrow ASA_B$ : $a$, *conf_name* (Plain-text)

$c \rightarrow ASA_C$ : $a$, *conf_name* (Plain-text)

Both $b$ and $c$ must decide whether to participate in the conference or not. If they decide not to then no messages are sent back to their local ASAs. Otherwise, messages are sent back to their local ASAs indicating their intention to join the conference identified as *conf_name*. They obtain the details of the *conf_name* from the invitation, *conf_invitation*.

**Message 5.**

$ASA_B \rightarrow ASA_A$ : $a$, *conf_name*, $K_{ASA_B}$, $(X_b, Y_b)^{K_{ASA_A}}$, *other_info*
(Plain-text and Encrypted)

$ASA_C \rightarrow ASA_A$ : $a$, *conf_name*, $K_{ASA_C}$, $(X_c, Y_c)^{K_{ASA_A}}$, *other_info*
(Plain-text and Encrypted)

Both $ASA_B$ and $ASA_C$ send the join conference message on behalf on their clients to the local ASA of the conference initiator which is $ASA_A$. $ASA_B$ and $ASA_C$ also inform $ASA_A$ of their public key.

The $(X_i, Y_i)$ pair, where $i$ is $b$ or $c$, is defined in a way similar to that of ICKDS but with one added feature. In particular, a hash function value of a time-stamp and a message checksum is introduced into the definition of $Y_i$. $X_i$ and $Y_i$ are defined as follows.

$$X_i = g^{(K_P \cdot p_i)} \bmod (N \cdot R)$$
$$Y_i = S_i \times g^{(C_i \cdot p_i)} \bmod (N \cdot R)$$

where $p_i$ is a secret random number selected by the local ASAs of $i$. $p_i$ has the property that it is co-prime to $(R-1)$ and $p_i \times \tilde{p}_i \equiv 1 \bmod (R-1)$. $C_i$ is a hash value such that:

$$C_i = hash(X_i, \text{TimeStamp}, \text{CheckSum}) .$$

The use of a time-stamp is to avoid playback of messages. The checksum is included to allow detection of modification to the message during transit since it is transmitted in the clear, apart from the $(X_i, Y_i)$ pair. The significance of the use of a hash function is that if the actual message was changed or the entire message was replayed then the receiver will calculate a different hash value.

The $(X_i, Y_i)$ pair is encrypted with the public key of $ASA_A$ to avoid wire-tappers from mounting a conspiracy attack as described in Section 4.

**Message 6.**

$ASA_A \rightarrow ASA_B$ : *conf_name*, *participants*, $(A_{ab}, B_{ab})^{K_{ASA_B}}$, *other_info*
(Plain-text and Encrypted)

$ASA_A \rightarrow ASA_C$ : *conf_name*, *participants*, $(A_{ac}, B_{ac})^{K_{ASA_C}}$, *other_info*
(Plain-text and Encrypted)

$ASA_A$ authenticates Message 5 based on the received $(X_i, Y_i)$ pair. This process is identical to that of ICKDS. If the messages are authentic then $ASA_A$ sends $ASA_B$ and $ASA_C$ a conference established message.

The definition of $(A_{ai}, B_{ai})$, where $i$ is $b$ or $c$, is defined in a way similar to that of ICKDS but again with the added feature. Thus, a hash function value of a time-stamp and a message checksum is introduced into the definition of $B_{ai}$. The reason for adding them is the same as that described for the $(X_i, Y_i)$ pair in Message 5. The $(A_{ai}, B_{ai})$ pair are defined as follows.

$$A_{ai} = X_i^{K_P \cdot r_a} \pmod{N \cdot R}$$
$$B_{ai} = S_a \cdot X_i^{C_{ai} \cdot r_a} \pmod{N \cdot R}$$

where $C_{ai} = hash(A_{ai}, \text{TimeStamp}, \text{CheckSum})$ and $r_a$ is a secret number picked randomly by $ASA_A$.

**Message 7.** $ASA_A \rightarrow a$ : *conf_name, participants* (Plain-text)
$ASA_B \rightarrow b$ : *conf_name, participants* (Plain-text)
$ASA_C \rightarrow c$ : *conf_name, participants* (Plain-text)

$ASA_A$ after sending Message 6 sends the conference established message to the conference initiator $a$ with the list of conference participants.

Both $ASA_B$ and $ASA_C$ authenticate Message 6 in the same way as in ICKDS. If Message 6 is authentic the $ASA_B$ and $ASA_C$ notify their individual clients, $b$ and $c$, that the conference identified by *conf_name* is established and also supply a list of participants of this conference.

At the end of this exchange of messages all the ASAs can derive a common conference key, $K_{conf}$, from the exchanged cryptographic information, i.e., the $(X_i, Y_i)$ and $(A_{ai}, B_{ai})$ values. $ASA_A$ can derive the conference key using the formula:

$$K_{conf} = g^{K_P \cdot K_P \cdot r_a} \bmod R$$

where $r_a$ is a secret random number used by $ASA_A$ to generate the (A,B) values; and $ASA_B$ and $ASA_C$ can derive the conference key using the formula:

$$K_{conf} = A_{a,i}^{\tilde{p}_i} \bmod R$$

where $i$ is $b$ or $c$ and $\tilde{p}_i$ is also a secret number such that $p_i \times \tilde{p}_i \equiv 1 \bmod (R-1)$ with $p_i$ being a secret random number used by $ASA_i$ to generate the $(X_i, Y_i)$ values; $p_i$ is co-prime to $(R-1)$.

**Communications After Authentication.** After the conference authentication process has been successfully performed all participants of the conference, including the initiator, can communicate directly with each other by using the new conference key, $K_{conf}$, described above. That is, after authentication, $a$, $b$ and $c$ can communicate directly without authentication via their local ASAs. Messages transmitted between them are of the form, $\mathcal{E}_{K_{conf}}(M)$, where $M$ is the message and $K_{conf}$ is their conference key. This communication is secure since only genuine parties have the correct encryption key.

### 5.4 One-way and Two-way Authentication

In the case of one-way authentication the sending side generates the (X,Y) pair described under Message 5 of Section 5.3. An one-way authentication message is constructed based on the (X,Y) value and sent to the receiver. The receiver authenticates the message. This completes the one-way authentication.

In the case of two-way authentication the sending side generates the (X,Y) pair described under Message 5 of Section 5.3. A two-way authentication message is constructed based on the (X,Y) value and sent to the receiver. The receiver side authenticates the message and if authentic it generates the (A,B) pair described under Message 6 of Section 5.3. A two-way authentication reply message is then constructed based on the (A,B) value and sent to the sender. This completes the two-way authentication.

## 6 Current Status

A prototype of the Authentication Server had been build on top of Unix. Work will be carried out in the near future to port it to RHODOS. The implementation involved the writing of the Authentication Server itself and a client interface library for the Authentication Server. The Authentication Server can be started as either a CAS or an ASA.

To generate the cryptographic information required for authentication a multi-precision math package was needed. Fortunately, we did not need to write this from scratch. The math package was kindly provided to us by Arjen Lenstra who also made some valuable suggestions in generating large prime numbers. This package was then extended to perform calculations required by the ICKDS scheme.

The current implementation of the cryptographic parameters is such that they are not as strong as that recommended by Koyama and Ohta [KO88b]. They recommended using at least 256 bits for $P$ and $Q$ and 512 bits for $R$. The prototype only uses 60 bits for $P, Q$ and $R$. This is a rather small size but it was done to allow ease of testing since it takes a long time just to generate the public and private system-keys: on a Sun 3/50 it takes 986.9 $\mu sec.$ of CPU time and 16 minutes and 33 seconds of user time. Furthermore, the aim of the prototype was to study the feasibility of incorporating ICKDS into RHODOS; the strength of the cryptographic parameters are of secondary consideration. The prototype, nevertheless, can readily generate cryptographic parameters in the recommended size by changing a couple of constant values in the server.

Details of the implementation and the algorithms are described in [Wan92].

### 6.1 Authentication Server Environment

The testing of the Authentication Server prototype was done on top of Unix rather than RHODOS. The RHODOS Nucleus was replaced by Unix. The role of IPC Manager was simulated by using Unix Sockets [Too90]. A pseudo Name

Server was written to support the functionality required by the Authentication Server.

The emulation of the IPC Manager provided by Unix sockets is such that the latter can be readily replaced by the former without changing any code in the Authentication Server. This emulation provides the same interface as defined by the RHODOS IPC Manager. Thus, from the Authentication Server's perspective it is interacting with the IPC Manager.

To illustrate the work of the Authentication Server we provide some snapshots of the SunView[3] windows used during testing. Figure 4 shows a typical view of the SunViw windows. The top-most two windows represent the logical workstation called *Donald*. The two windows in the next row represent the logical workstation called *Goofy*. The two windows in the third row represents the logical workstation called *Mickey*. For each of the logical workstations, the window on the left is referred to as the ***user window*** and the window on the right is referred to as the ***server window***. The left hand side window at the bottom row of Fig. 4 represents the pseudo Name Server. The window on its right represents the ***time-out process***. This time-out process is used as a timer. When a timer runs out it sends a message to the initiating process. In the following description some particular notation is used to distinguish workstation names from user names. The name of a workstation is represented as *Mickey*. It has the feature of using a upper-case letter and in the italic style. The name of a user is represented as mickey. It has the feature of using a lower-case letter and in the serif font style.

## 6.2 Testing Results

In Fig. 4, the CAS is running on *Mickey*. In the user window of *Mickey*, a user named mickey has successfully logged in and is running the authentication test program. User goofy is logged in on workstation *Goofy* and is also running the authentication test program. The server window of *Goofy* shows its ASA has decrypted a message from the CAS with key "4f5e5e4c73495e4f". Recall in Fig. 4 the CAS is running on *Mickey*. Examining *Mickey*'s server window one sees the top line says "Encrypting with key "4f5e5e4c73495e4f" " which is the same key used by *Goofy*. This means user goofy has entered the correct password and its local ASA has passed the challenge from the CAS based on the supplied password.

Looking at the server window of workstation *Donald* one observes its ASA decrypts a message from CAS with key "075e5d435849495e". Going back to *Mickey*'s server window one sees the second last line says "Encrypting with key "0e085d435849496b" " which is not the same key used by *Donald*'s ASA. This implies user donald did not enter the correct password so *Donald*'s ASA was not using the correct key for decryption. Hence *Donald*'s ASA has failed the challenge from CAS. Therefore, user donald is denied of login. This is shown in *Donald*'s user process window: the password used by donald is "donalddog". The correct password should be "donaldduck".

[3] SunView is a registered trademark of Sun Microsystems, Incorporated.

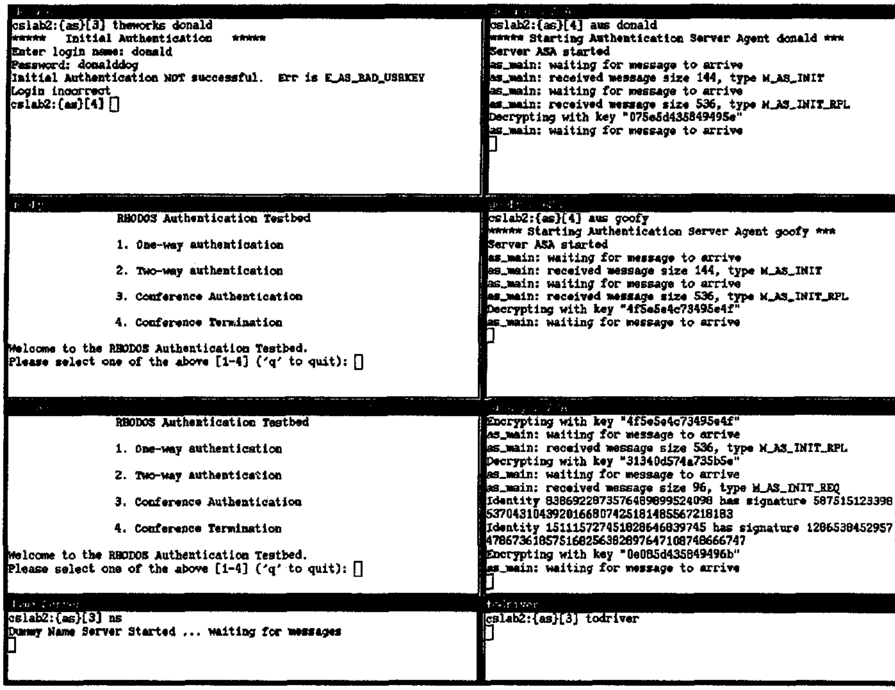


**Fig. 4.** User Login Authentication — with donald failing to log in.

Conference authentication is demonstrated in Fig. 5. In this figure, a conference authentication request is made by user mickey who is known as the conference initiator. The conference name is "Conference 1" and user goofy and donald are the invitees. Looking at the user windows of *Goofy* and *Donald* one can see that both goofy and donald are prompted to join the conference initiated by mickey. They are told of the name of the conference, the name of the initiator and the names of other invitees.

Suppose both goofy and donald decide to join the conference. Figure 6 shows the completion of conference authentication. Examining the user window of all three workstations one can see that all three users are informed of the names of the participants. Looking at their respective ASAs one can see that the conference key was also generated by each ASA and that they have the same key. In fact, if one looks closely at the server window of each of the workstations one can notice the cryptographic information (X,Y) and the (A,B) pair being generated by the Authentication Servers.

A variation of an authenticated conference setup is shown in Fig. 7. In this figure, user goofy is initiating a conference named "Conference 2" and it is inviting both mickey and donald. Looking at the user window of *Mickey* one can see that user mickey was late with its request to join the conference. This is indicated by the error message **E_AS_CONF_NF**. To see the cause of the error message we

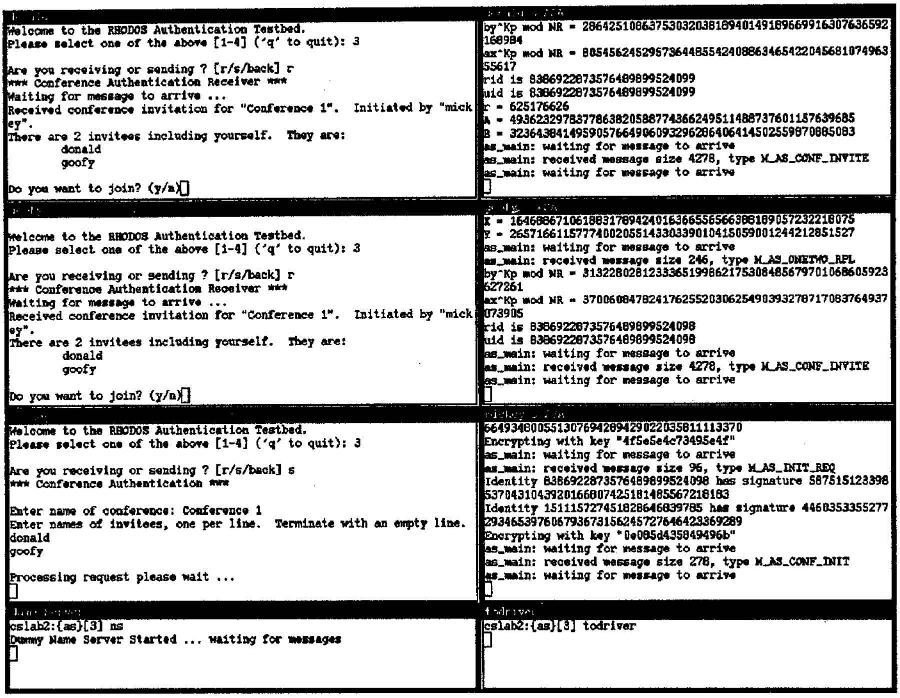


**Fig. 5.** User goofy and donald are invited to a conference.

turn our attention to the server window of *Mickey*. This window shows a timeout message, M_TIMEOUT, was received. The reception of this message indicates to the ASA that the waiting period for user mickey to request to join "Conference 2" has passed. The ASA can only assume mickey does not wish to join "Conference 2" and thus removes all information related to "Conference 2".

Notice also that a timeout message was received by *Goofy*'s ASA. This timeout is caused by the fact that *Goofy*'s ASA was waiting for all the invitees to reply before proceeding with the next stage of the conference authentication operation. In this case, it was waiting for mickey to respond.

## 7 Conclusion

In this work we have designed, implemented and studied an Authentication Server for a distributed system, based on RHODOS. The server has demonstrated its ability to carry out authentication operations that are capable of detecting active attacks and masquerade in a distributed system environment.

One of the most significant results achieved is the ability of the Authentication Server to support conference authentication where all conference participants are authenticated simultaneously and at the same time a conference key is distributed among all participants. This is achieved by use of an improved

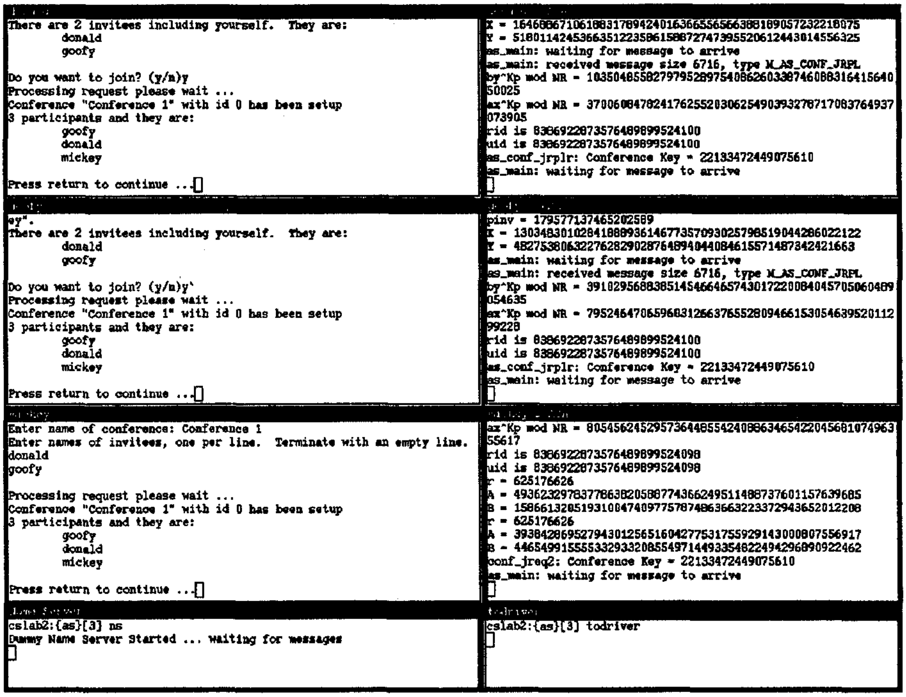


Fig. 6. The completion of a conference authentication.

version of ICKDS scheme. The implementation lies in the ability to detect replay of messages and at the same time allow client messages to be authenticated along with the authentication messages. The Authentication Server also supports client login authentication, one-way and two-way authentication.

Location transparency of authentication clients is another note-worthy feature offered by the Authentication Server. Authentication messages can be delivered to the target user with the initiating user specifying only the login name of the target user and not its current location. This is achieved by the cooperation between the Authentication Server, Name Server and the IPC Manager. This feature is particularly important to distributed systems since authentication clients may not always reside at a fixed location.

The work carried out here provides a testbed for further experiments to test out different strategies, concepts and methodologies. This work has provided the impetus needed for further research in the area of authentication in distributed operating systems.

## Acknowledgment

The authors would like to thank A. Lenstra for providing the math package and his suggestions and T. Hardjono and Y. Zheng for their comments. Moreover,

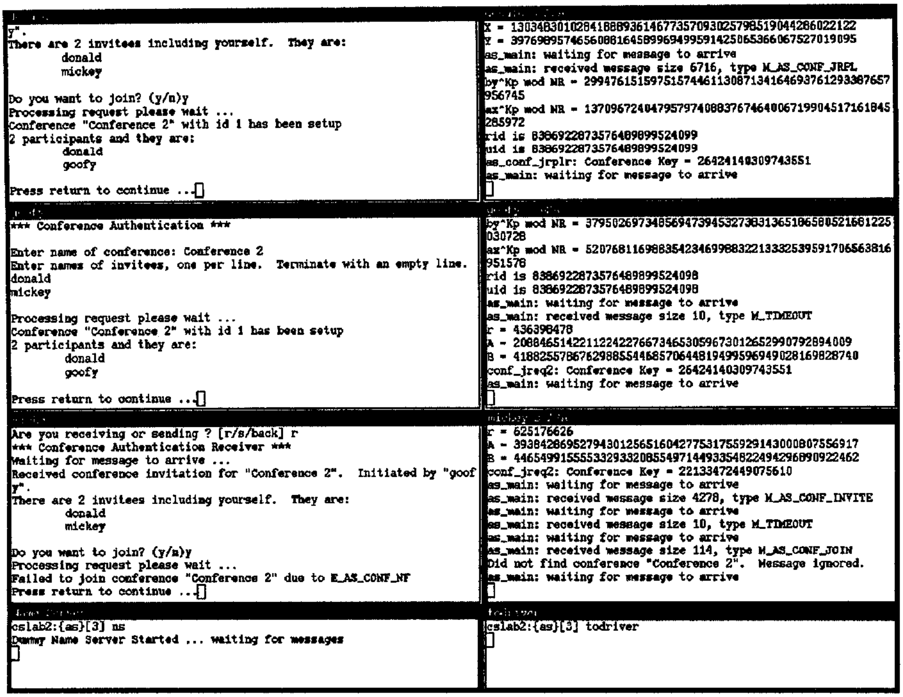


**Fig. 7.** A conference authentication where one of the invitees responded to the invitation too late.

the authors would like to thank the anonymous reviewers for their constructive comments.

# References

[BM90] S. M. Bellovin and M. Merritt. Limitations of the Kerberos authentication system. *Comput. Commun. Rev.*, 20(5):119–132, Oct. 1990.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. on Info. Theory*, IT-22(6):644–654, Nov 1976.

[GB89] A. Goscinski and K. Beaton. A simple distributed computer system for supporting collaboration in distant and synchronous meetings. *Computers in Industry*, 12:95–106, 1989.

[GGI+91] G. W. Gerrity, A. Goscinski, J. Indulska, W. Toomey, and W. Zhu. RHODOS — a testbed for studying design issues in distributed operating systems. In G. S. Poo, editor, *Proceedings of the 2nd Singapore International Conference on Networks (SICON'92)*, pages 268–274, Sept. 3-6 1991.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Computing*, 18(1):186–208, Frbruary 1989. A preliminary version of this paper appeared in the Proceedings of the 27th

Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 174-187.

[Gos91] A. Goscinski. *Distributed Operating Systems: The Logical Design.* Addison-Wesley, 1991.

[GP91] A. Goscinski and J. Pieprzyk. Security in distributed operating systems. *Datenschutz and Datensicherung*, Heft 5:240–252, 1991.

[KO88a] K. Koyama and K. Ohta. Identity-based conference key distribution systems. In C. Pomerance, editor, *Proceedings of CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 175–184. Springer-Verlag, 1988.

[KO88b] K. Koyama and K. Ohta. Security of improved identity-based conference key distribution systems. In C. G. Günther, editor, *Advances in Cryptology — EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 11–19. Springer-Verlag, May 1988.

[Koy87] K. Koyama. Identity-based conference key distribution system. *IEEE Electronic Letters*, 23(10):495–496, May 7 1987.

[NDV+91] J. F. Nunamaker, A. R. Dennis, J. S. Valacich, D. R. Vogel, and J. F. George. Electronic meeting systems to support group work. *Communications of the ACM*, 34(7):40–61, July 1991.

[Sha85] A. Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology — Proceedings of CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer-Verlag, Berlin, 1985.

[SK91] A. Shimbo and S. Kawamura. Cryptanalysis of several conference key distribution schemes (Extended Abstract). In *ASIACRYPT '91 Abstracts*, pages 155–160, Fujiyoshida, Japan, Nov. 11-14, 1991.

[SNS88] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of USENIX Winter Conference*, pages 191–202, Dallas, Texas, February 1988.

[SP89] J. Seberry and J. Pieprzyk. *Cryptography: An Introduction to Computer Security.* Prentice-Hall, 1989.

[Too90] W. Toomey. Emulating a RHODOS process environment under Unix. Technical Report CS90/49, Dept. of Computer Science, University College, University of New South Wales, Canberra, December 1990.

[Wan92] Michael Y-C Wang. The development and study of selected aspects of communication security and authentication schemes for distributed systems. Master's thesis, School of Computer Science and Engineering, University of New South Wales, August 1992.