Scalable and fault-tolerant key agreement protocol for dynamic groups

A. Abdel-Hafez¹, A. Miri¹ and L. Orozco-Barbosa², *,†

¹School of Information Technology & Engineering, University of Ottawa, Ottawa, Ontario, Canada.
²Instituto de Investigación en Informática, Universidad de Castilla-La Mancha, Albacete, Spain.

SUMMARY

With the widespread use of the Internet, the popularity of group communication-based applications has grown considerably. Since most communications over the Internet involve the traversal of insecure networks, basic security services are necessary for these collaborative applications. These security services can be facilitated if the authorized group members share a common secret. In such distributed applications, key agreement protocols are preferred to key distribution protocols. In the past two decades, there have been many proposals for key agreement protocols. Most of these protocols are not efficient and limit the size of the underlying group. In this paper, we consider the scalability problem in group key agreement protocols. We propose a novel framework based on extension of the Diffie–Hellman key exchange protocol. The efficiency of our protocol comes from the clustering of the group members, where the common session key is established collaboratively by all participants. We present the auxiliary protocols needed when the membership changes. We show that our protocol is superior in complexity in both communication and computation overheads required to generate the session key. Copyright © 2006 John Wiley & Sons, Ltd.

1. INTRODUCTION

With the widespread use of the Internet, the popularity of group communication-based applications has grown considerably. Group communication is a means of providing multi-point to multi-point communication by organizing processes in groups. Current group-oriented applications include live multi-party conferences, online video games, collaborative workspaces, remote consultation and diagnosis systems for medical applications, contract negotiation and distributed interactive simulation and much more. Many of these applications disseminate and exchange sensitive and/or classified information. In practice most of these applications use the Internet as the underlying communications network, so basic security services—such as traffic integrity, entity authentication, and confidentiality—are necessary for these collaborative applications.

These security services can be facilitated if the authorized group members share a common secret (known in the literature as a group or session key. Thus one of the main design challenges in secure and reliable group communication systems is the group key management. A key management protocol is a process whereby a shared secret key becomes available to two or more authorized parties, for subsequent cryptographic use. Key management protocols can be subdivided broadly into:

*Correspondence to: L. Orozco-Barbosa, Instituto de Investigación en Informática, Universidad de Castilla–La Mancha, Campus Universitario s/n, 02071 Albacete, Spain.
†E-mail: luis.orozco@uclm.es

Contract/grant sponsor: NSERC.
Contract/grant sponsor: Council of Science and Technology of Castilla–La Mancha; contract/grant number: PBC-03-001.
1. Key transport protocol (centralized key management): this is a key establishment technique in which one party creates or otherwise obtains a secret value, and securely transfers it to others.

2. Key agreement protocol (contributory key management): this is a key establishment technique in which two (or more) parties contribute information, which jointly establishes the shared secret key, such that no party can predetermine the resulting value (see Rueppel and van Oorschot [1] for an overview of modern key agreement protocols).

Whereas secure communication between two parties (unicast) is well understood, little concrete consideration has been given to the security and privacy issues involved in multiparty information exchanges. Most efforts in multiparty settings have been focused on extending the well-known two-party Diffie–Hellman (DH) key exchange protocol [2] to operate as a multi-party key agreement protocol. However, contrary to a common initial impression, secure group communication is not a simple extension of secure two-party communication [3]. Beyond the fulfillment of security requirements, most of the proposed group key agreement protocols suffer from lack of efficiency. Generally, group communication security protocols are multi-round protocols. Consequently, protocol efficiency is of great concern due to the direct relation of the number of participants to computation and communication complexity.

In this paper, we specifically focus on key agreement protocols, the cornerstone of the security services, within a group of members with a highly dynamic membership nature. Mainly, we consider the efficiency problem in key agreement protocols, where we propose a novel, efficient, reliable framework for $n$-party key agreement protocol. Although our framework is completely distributed, as an access controller we still need a central server (network administrator, NA*) to control the group membership services such as adding or deleting members and controlling the indexing scheme of the group members.

The remainder of this paper is organized as follows. The next section presents the challenges of group communication security. In the third section, we review some previous related work. The fourth section describes the architecture of the proposed system and explains the main Initial Key Agreement Protocol. The fifth section describes the way our protocol handles the membership changes to keep the security of the generated session key. The sixth section presents performance evaluations of the protocols and comparisons with other protocols. Our conclusions and future work plans are given in the final section.

2. SECURITY CHALLENGES IN MULTICAST

In multicast, just as in unicast, an adversary (who can be a legitimate group member) may eavesdrop on confidential communications, disrupt a session’s data exchange, inject unauthorized or bogus traffic, block a session’s progress, masquerade as someone else to join the session, or initiate and operate a bogus session. Therefore, the privacy, integrity, availability, and authenticity of a multicast service must be protected. However, the nature of multicast makes it inherently more vulnerable to different types of attacks which have no counterpart in unicast and cannot be effectively dealt with by simply using trivial extensions to secure unicast. The following describes the characteristics specific to group communication and the possible attacks, which are clearly different from that available for unicast communication:

- Multicast involves a group of principals and this can potentially make it easier for an attacker to pose as another legitimate principal or to try attacks in parallel. In fact, in contrast to unicast communications where the communicating parties know the identity of each other, knowledge of the session membership is typically not guaranteed in multicast, and session membership is hard to be controlled.
- Multicast data is transmitted over multiple channels of the network at the same time, which presents many more opportunities for intercepting the traffic.
- There is no mechanism that can prevent either a legitimate group member, or non-group member, from overwhelming the group members with spurious data, which can result in a denial-of-service attack.

*In practice, the NA can be distributed or replicated with each NA being responsible for its nearest cluster(s). In this case all the NAs should coordinate among themselves to ensure consistency and uniqueness of the group (multicast) addresses.
When an attack does take place, multicast ensures that a large number of group members are affected.

From the previous discussion, it is clear that group membership control and secure key establishment mechanisms are two major components of a secure multicast protocol.† If the group membership is dynamic, the secret key needs to be updated so that members do not get access to data sent before they join the group or the data sent after they leave the group.

Apart from satisfying the above-mentioned security requirements, the desirable design features of the security system architecture should also be

• compatible with the underlying network protocols;
• scalable to the scope of the global Internet; and
• transparent to higher-level applications and services.

Transparency keeps high-level applications free from details such as authentication and checking of credentials and policies. In addition, the security mechanisms residing at the session and/or presentation level should place no restriction on the underlying networking mechanisms. As such, they must coexist with networking standards and must not depend on changes to the transport and switching elements in the (inter)network. Moreover, the architecture design should be localizable in the sense that an area of a network may install the trusted multicast facility and achieve firewall-style self-protection even if other portions of the (inter-)network do not yet support trusted multicast. This feature is important to facilitate the gradual introduction of the new technology to the existing millions of host machines. Finally, the architecture has to be flexible enough to support a variety of session control policies as required by higher-level applications. These features together will make trusted multicast easier to integrate into an existing environment such as the Internet.

3. RELATED WORK

Since the publication of two-party Diffie–Hellman key exchange (DH) in 1976, various solutions have been proposed to extend the DH key exchange mechanism to n-party Key Agreement Protocol. The earliest attempt to provide contributory group key agreement is due to Ingemarsson et al. [4] This protocol requires synchronous start-up and executes in n − 1 rounds. Members must be arranged in a logical ring. In a given round, every participant raises the previously received intermediate key value to the power of its exponent and forwards the result to the next member. After (n − 1) rounds, every member has computed the same key \( S_n \). This work did not address the case of rekeying after membership changes.

Another interesting scheme was presented by Steer et al. [5] The protocol (referred to as STR) takes n rounds to complete. In some ways STR is similar to GDH.2 [6], which will be discussed below (‘Group Diffie–Hellman GDH.2’). Both take the same number of rounds to complete the protocol and compute the session key. Also, both accumulate keying material by traversing group members one per round.

However, the generated session key in STR has a very different structure, given by \( S_n = \alpha^{N_{n-1} \cdots N_a N_{n-2}} \). Interestingly, STR is well suited for adding new members; it takes only two rounds to add a new member. Member deletion, on the other hand, is difficult in STR since there is no natural group controller.

Another notable result is due to Burmester and Desmedt (BD) [7], whose protocol executes in only three rounds:

• Each member, \( M_i \), generates its random exponent \( N_i \) and broadcasts \( Z_i = \alpha^{N_i} \).
• Each member, \( M_i \), computes and broadcasts:

\[
X_i = \left( \frac{Z_{i+1}}{Z_{i-1}} \right)^{N_i}
\]

†Membership control should be supported by the underlying group communication protocol, but it should be integrated with the key establishment protocol.
• Each member, $M_i$, can now compute the key:

$$S_n = Z_{i-1}^{nN} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \ldots X_{i-2} \mod p$$

The key defined by BD is different from all protocols discussed thus far, namely:

$$S_n = \alpha^{N_1 N_2 + N_3 + \ldots + N_n N_1}$$

The protocol has been proven secure given that the Computational Diffie–Hellman (CDH) problem is intractable. The CDH problem can be simply stated as follows: given $\alpha^x \mod p$ and $\alpha^y \mod p$ it is hard to find $\alpha^{xy} \mod p$. While the BD protocol is efficient and secure, it is not well suited to dynamic groups.

The TGDH protocol [8] is the only protocol that addresses the scalability issue in its key agreement protocol. TGDH is an adaptation of key trees in the context of fully distributed, contributory group key agreement. The tree is organized in the following manner: each node (each with ($i$, $v$) exponents) and one cardinal value containing $a$ gains.

As shown in Algorithm 1, Steiner et al. have addressed the dynamic membership issues in their group key agreement proposal [6] and have proposed a family of Group Diffie–Hellman (GDH) protocols (namely, GDH.1, GDH.2, and GDH.3). Their protocols can be considered as a generic extension of the two-party Diffie–Hellman [9]. GDH provides contributory authenticated key agreement, key independence, key integrity, resistance to known key attacks, and perfect forward secrecy. Their protocol suite is fairly efficient in leave and partition operations, but the merge protocol requires as many rounds as the number of new members to complete the key agreement procedure. One of their basic group key agreement protocols (without authentication) is briefly described in the following section. Later in the ‘Complexity Analysis’ section we will compare our protocols with GDH.2 and TGDH, by highlighting the advantages and performance gains.

3.1 Group Diffie–Hellman GDH.2

In this section we briefly describe the Group Diffie–Hellman protocol GDH.2 (see Algorithm 1) proposed in Atenies et al. [6] Let $p$ be a prime integer and $q$ a prime divisor of $p - 1$. Let $G$ be the unique cyclic subgroup of $\mathbb{Z}_p^*$ of order $q$, and let $\alpha$ be a generator of $G$. We assume that $M_i$ shares (or is able to share) with each $M_i$, a distinct secret $K_{iv}$. The protocol consists of two stages: upflow and downflow. The purpose of the upflow stage is to collect contributions from all group members. As shown in Algorithm 1, $M_i$ receives the contribution from $M_{i-1}$ and composes $i$ intermediate values (each with $(i - 1)$ exponents) and one cardinal value containing $i$ exponents. For example, if $M_4$ receives the set

$$\{\alpha^{N_1 N_2 N_3}, \alpha^{N_3 N_2}, \alpha^{N_1 N_3}, \alpha^{N_2 N_3}\}$$

then it outputs the set

$$\{\alpha^{N_1 N_2 N_3 N_4}, \alpha^{N_3 N_2 N_3}, \alpha^{N_1 N_2 N_4}, \alpha^{N_3 N_2 N_4}, \alpha^{N_2 N_2 N_3}, \alpha^{N_1 N_3 N_4}\}$$

The cardinal value in this example is $\alpha^{N_1 N_2 N_3 N_4}$. By the time the upflow reaches $M_n$, the cardinal value becomes $\alpha^{N_{n-2} \cdot N_{n-1}}$. Thus $M_n$ is the first group member to compute the key $K_n$. At the final step of the upflow stage, $M_n$ computes the last batch of intermediate values. In the second stage $M_n$ broadcasts the intermediate values to all group members. By receiving these values each group member can calculate the session key $S_n$. 
Algorithm 1 (Group Diffie–Hellman Protocol GDH.2)

**Step 1 (Upflow): round i; i ∈ [1, n − 1]**

1. $M_i$ selects $N_i \in \mathbb{Z}_p^*$

2. $M_i \leftarrow \alpha^{N_{i_1} \cdots N_{i_{l_i}}}_{N_i} \rightarrow M_{i+1}$

**Step 2 (Broadcast): round n**

3. $M_n$ selects $N_n \in \mathbb{Z}_p^*$

4. $M_i \leftarrow \alpha^{N_{i_1} \cdots N_{i_{l_i}}}_{N_i} M_n$

5. Upon receipt of the above, $M_i$ computes $\alpha^{N_{i_1} \cdots N_n}_{N_{i_1}} = \alpha^{N_1 \cdots N_n} = S_n$

In the case of a member addition, under the GDH.2 protocol, the last member $M_n$ saves the contents of the upflow messages. $M_n$ generates a new exponent $\tilde{N} \in \mathbb{Z}_p^*$ and computes a new upflow message and sends it to the new member, $M_{n+1}$. Then $M_{n+1}$ generates its own exponent and computes the new key. Finally, under the normal operation of the protocol (where $M_{n+1}$ takes $M_n$’s rule), $M_{n+1}$ computes $n$ intermediate values and broadcasts to all the other group members.

Roughly speaking, the communication and computation costs of performing GDH.2 increase linearly with the size of the group members. This will affect the efficiency of the protocol, especially when the group size increases. In the following section we introduce our approach to deal with this scalability problem.

4. CLUSTER GROUP DIFFIE–HELLMAN (CGDH)

In this section, we present a novel framework for an efficient and scalable key agreement protocol (CGDH), which is based on multilevel clustering of the universal group into small, size-bounded clusters. The idea of clustering was first proposed in Banerjee and Khuller [10] to create hierarchies for wireless networks and was used in Banerjee and Khuller [11] as a scalable and secure distribution of the group (session) key. The efficiency of our proposed scheme comes from the fact that all the clusters at a given level can perform the required steps concurrently. This feature, from the complexity point of view, reduces the time required to generate the session key. The proposed framework requires clustering the group members into a hierarchical structure, where the number of clusters reduces systematically from the base level level; to the top level. This level consists of only one cluster $C_{1,1}$. Although our extension is based on the distribution of group members into (approximately) equal-size clusters, all the members contribute equally in generating the group session keys, which is a major requirement in securing dynamic peer groups. The main advantage of this protocol over (HGĐH) [12] is that the group session key and the intermediate clusters’ session keys are never transmitted through the network. Furthermore, the members of a cluster can communicate securely within the cluster by either using another cluster session key (wherein the cluster representative adds a different secret value to the contributions of the preceding members and broadcasts the intermediate values to the cluster members only) or by applying a strong hash function to its cluster session key.

4.1 Notation and Definitions

- $n$ = number of group participants
- $d$ = maximum number of cluster members
- $l$ = total number of levels
\( i, v, j \) = indices of levels, clusters and members

\( i = 1, 2, \ldots, l \)

\( v = 1, 2, \ldots, d(d - 1)^{l-2} \)

\( j = 1, \ldots, d - 1 \)

\( G \) = cyclic algebraic group

\( \alpha \) = exponentiation base; generator of \( G \)

\( C_{i,v} \) = the \( v \)th cluster in the \( i \)th level

\( \{M_{i,v}\}_s \) = the descendants of member \( M_{i,v} \)

\( N_C \) = maximum number of members per cluster

\( E_S \) = number of serial exponentiations

\( M_S \) = maximum message size

\[ \{M_{i,v}\}_s = \{M_{i+1,(d-1)(v-1)+1}, \ldots, \} = \{M_{x+i+1,(d-1)(v-1)+1} \}_{x=1, \ldots, d-1} \]

4.2 Initial Members Distribution

At the initial stage, the group members are organized into a hierarchical structure, as shown in Figure 1. For clarity purposes, unless stated otherwise, the hierarchical structure used throughout our discussion is well balanced; that is, all the clusters have an equal number of members. The hierarchical structure consists of \( l \) levels, each level comprising one or more clusters. Levels are numbered sequentially starting from \( \text{level}_1 \) which consists of one cluster, and ending at \( \text{level}_l \) which consists of \( d(d - 1)^{l-2} \) clusters. The hierarchical structure is built starting from the lowest level, \( \text{level}_l \), by regrouping every \( d \) members into a cluster, each one denoted by \( C_{i,v} \), where \( v = 1, \ldots, d(d - 1)^{l-2} \). Each cluster in \( \text{level}_l \) chooses a cluster representative. Thereafter, \( d - 1 \) of these representatives are regrouped with another new member (which will be the representative of this cluster) and will constitute an upper level cluster, \( \text{level}_{l-1} \). This process will continue until the last \( d \) representatives constitute the root cluster \( C_{1,1} \).

The total number of group members \((n)\) in a fully balanced hierarchical structure can be represented as a function of the maximum number of members per cluster \((d)\) and the total number of levels \((l)\) as follows:

\[ n = d \sum_{i=1}^{l} (d - 1)^{i-1} = d \frac{(d - 1)^{l-1} - 1}{d - 2} \]

Similarly, the maximum number of clusters \((N_C)\) in the hierarchy can be calculated as a function of \((d,1)\) as follows:

\[ N_C = 1 + d \sum_{i=2}^{l} (d - 1)^{i-2} = 1 + d \frac{(d - 1)^{l-1} - 1}{d - 2} \]

Figure 1. Distribution of group members

From (1) and (2) we can state the relation between \( n \) and \( N_C \) as follows:

\[
N_C = n - d(d - 1)^{l-1} + 1
\]

Note that the construction process can start from the top; i.e., \( d \) members can construct the root cluster \( C_{1,1} \); then, each member in \( C_{1,1} \) can be a representative to its lower-level cluster and chooses another \( d - 1 \) members within a certain distance to constitute another cluster, and the process continues until all the members are accounted for. If the hierarchy is well balanced (as shown in Figure 2), the number of clusters in the lower level would be \( d(d - 1)^{l-2} \).

In this hierarchical structure, the indexing scheme of members (clusters) can be used to determine the position of any member (cluster) in the hierarchy based on the maximum number of members \( (d) \) in each cluster. Moreover, we can identify its neighbours in the same cluster. Also from this indexing scheme we can determine the cluster representative chosen by \( N_A \), its ancestors up to the root cluster and also the children's clusters down to the \( \text{level}_l \). After organizing the members into a multilevel hierarchical structure, the actual protocol starts as described in the following section.

### 4.3 Initial Session Key Generation

The initial key generation protocol (Algorithm 2) takes place once the hierarchical structure has been created. The main protocol consists of \( l \) steps, and each step comprises in turns of \( v \) simultaneous \( d \) rounds (where \( v \) is equal to the number of clusters in this level), so the total number of rounds is \( R = ld \). After successfully performing each step, every cluster will be represented by its representative in conducting the next step as shown in Algorithm 2, which uses the generated cluster session key as its secret share in conducting the next step. As in the two-party case, all the group members agree a priori on a cyclic group \( G \). Let \( \alpha \) be a generator of \( G \).

The protocol starts from the lowest level, \( \text{level}_l \). For each cluster in the \( \text{level}_l \), \( C_{i,v} \; v = 1, 2, \ldots, d(d - 1)^{l-2} \), each member \( M_{i,v} \; j = 1, \ldots, d - 1 \), receives the contributions from \( M_{i,v} \) and composes \( j \) intermediate values, each with \( j - 1 \) exponents and one cardinal value containing \( j \) exponents (\( M_{i,v} \) can be thought of as a cluster initiator, where it uses \( \alpha \) as its cardinal value). For example, as shown in Figure 2, \( M_{3,2} \) receives \( \alpha^{d_2} \) and produces \( \{\alpha^{d_2}, \alpha^{d_2}, \alpha^{d_2} \} \). The cardinal value in this example is \( \alpha^{d_2} \). By the time the upflow reaches \( M_{l-1,v} \) (the cluster \( C_{i,v} \)'s representative to the upper level, \( \text{level}_{l-1} \)), the cardinal value becomes...
\[ \alpha \prod_{i}^{d} \] Here, \( y \) is equal to the remainder of \( \left( \frac{v}{d-1} \right) \) if \( v \mid (d-1) \), otherwise equal to \( -\frac{v}{d-1} \) if \( v \nmid (d-1) \) and \( v_{l} = \left[ \frac{v}{d-1} \right] \). Upon receiving this message, \( M_{r_{-1},r_{1}}^{y} \) adds its secret exponent \( r_{i_{-1},r_{1}}^{y} \) to the message. By adding its secret exponent to the cardinal value, \( M_{i_{-1},r_{1}}^{y} \) can compute the cluster session key and broadcasts the intermediate values \( \alpha \prod_{i}^{d} \) to its \( l \) cluster’s members \( M_{l_{0}}^{y} \). All cluster members are then able to generate the cluster session key \( \alpha \prod_{i}^{d} R_{l_{0},r_{1}}^{y} \), which will be considered (in the second step) as this cluster’s secret share \( (R_{l_{0},r_{1}}^{y}) \). In the second step, which will be conducted by \( l_{0},r_{1} \) members, the same procedure within each cluster will be repeated; namely, each member in each cluster sends its public share to the next member in the cluster (which is the upflow rounds) but using the lower-level cluster session key \( (R_{l_{0},r_{1}}^{y}) \) as its secret share instead of picking another random value. For example, the secret share for \( M_{1,1}^{2} = R_{1,1}^{1} = \alpha^{d_{1}^{1}d_{2}^{1}d_{3}^{1}} \) and its intermediate (public) value is \( R_{1,1}^{2} = \alpha^{d_{1}^{1}d_{2}^{1}d_{3}^{1}} \). In the downflow messages the cluster’s representative \( M_{l_{0},r_{2}}^{y} \), where \( v_{2} = \left[ \frac{v_{1}}{d-1} \right] = \left[ \frac{v}{(d-1)^{2}} \right] \), broadcasts the intermediate values to all its descendants \( \{ M_{l_{0},r_{2}}^{y} \} \). For example (refer to Figure 2), \( M_{1,1}^{1} \) broadcasts \( \alpha^{d_{1}^{1}d_{2}^{1}d_{3}^{1}} \) to \( \{ M_{1,1}^{1}, M_{1,1}^{2}, M_{1,1}^{3} \} \) and broadcasts \( \alpha^{d_{1}^{1}d_{2}^{1}d_{3}^{1}} \) to \( \{ M_{1,1}^{1}, M_{1,1}^{2}, M_{1,1}^{3} \} \). This procedure run from the \( l_{0},r_{1} \) to \( l_{0},r_{2} \). At \( l_{0},r_{1} \) each member uses its \( l_{0},r_{2} \) cluster session key as its secret share. For instance, in our example, the secret value for \( M_{1,1}^{1} \), \( R_{1}^{1} = \alpha^{d_{1}^{1}d_{2}^{1}d_{3}^{1}} \). Similarly, the secret keys of \( M_{1,1}^{2} \) and \( M_{1,1}^{3} \) are \( R_{1}^{1} = \alpha^{d_{1}^{1}d_{2}^{1}d_{3}^{1}}, \) and \( R_{1}^{1} = \alpha^{d_{1}^{1}d_{2}^{1}d_{3}^{1}}, \) respectively. As in the previous steps, each member, \( M_{l_{0},r_{1}}^{y} \), in cluster \( C_{l_{0}} \), raises the received intermediate values to the power of its private input and forwards the result to the next member in the cluster. The downflow stage takes place when the last member, \( M_{1,1}^{d} \), receives the last upflow message and computes the group session key \( S_{n} \). \( M_{1,1}^{d} \) then raises the intermediate values it has received to the power of its private key \( R_{1}^{1} \) (\( R_{1}^{1} \) in our example) and broadcasts the intermediate values to the other members, \( M_{l_{0},r_{j}}^{y} ; j \in [1, d-1] \), and their descendants, \( \{ M_{l_{0},r_{j}}^{y} \} ; j \in [1, d-1] \). At the same time \( M_{1,1}^{d} \) broadcasts its cardinal value (of course, before adding its secret exponent) to its descendants, \( \{ M_{l_{0},r_{1}}^{y} \} \). For example, \( M_{1,1}^{1} \) broadcasts \( \alpha^{R_{1}^{1}R_{1}^{2}} \) to its descendants \( \{ M_{1,1}^{1}, M_{1,1}^{2}, M_{1,1}^{3}, M_{1,1}^{4}, M_{1,1}^{5}, M_{1,1}^{6} \} \) who have the value \( R_{1}^{1} \), broadcasts \( \alpha^{R_{1}^{2}R_{1}^{3}} \) to \( M_{1,1}^{1} \) and its descendants \( \{ M_{1,1}^{1}, M_{1,1}^{2}, M_{1,1}^{3}, M_{1,1}^{4}, M_{1,1}^{5}, M_{1,1}^{6} \} \), who have the value \( R_{1}^{2} \) and broadcasts \( \alpha^{R_{1}^{2}R_{1}^{3}} \) to \( M_{1,1}^{1} \) and its descendants \( \{ M_{1,1}^{1}, M_{1,1}^{2}, M_{1,1}^{3}, M_{1,1}^{4}, M_{1,1}^{5}, M_{1,1}^{6} \} \), who have the value \( R_{1}^{1} \). At this time all the group members are able to generate the group session key, which is \( S_{n} = \alpha^{R_{1}^{1}R_{1}^{2}R_{1}^{3}} \).

Algorithm 2 (Initial key generation protocol)

1. For \( i = l \) down to 2

1.1 \( v \leftarrow d(d-1)^{y-2} \)

1.2 For \( x = 1 \) to \( v \)

1.2.1 For \( y = 1 \) to \( d-2 \)

1.2.1.1 \( M_{ix}^{y} \Rightarrow M_{ix}^{y+1} : \left\{ \alpha \prod_{i}^{d} \right\}_{v_{x}^{y} \rightarrow \alpha^{v_{x}^{y} \cdots v_{x}^{y}}} \)
The previous discussion shows that every group member participates in creating the group session key, which enables each member of the group to assure the freshness of the generated session key. Also, the protocol is very efficient since most of the computation and communication performs in parallel over the clusters. Due to this parallelism, we consider only the serial cost of computation.

In summary, CGDH Initial key agreement protocol has the following characteristics:

<table>
<thead>
<tr>
<th>Rounds</th>
<th>$R = ld$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages</td>
<td>$M = dN_c + 1$</td>
</tr>
<tr>
<td>Exponentiations</td>
<td>$Ex_S = \left\lfloor \frac{d}{2} + 1 \right\rfloor$</td>
</tr>
<tr>
<td>Max. message size</td>
<td>$M_S = d$</td>
</tr>
</tbody>
</table>

We should mention that in spite of the number of messages in our protocol being larger than that of the flat settings (GDH-2, for example), the maximum message size in our protocol is much smaller. Besides, if a failure happens in the flat setting, the whole protocol halts: the whole process has to be
restarted, implying that all the messages have to be retransmitted. In the case of our protocol, the fault
can be easily isolated. For example, if a message from any member at any level is belated due to any
failure, after a certain time (a specified time out), the whole cluster will be discarded from the group, and
the protocol completes without this cluster members, who can join afterwards just as one member joins
(the cluster representative). Another advantage is that in our setting the transmission delay is much
smaller than that of the flat setting, since a reasonable choice will be to distribute the members accord-
ing to a geographical base; i.e., in our protocol the clustering is based on a maximum specified distance
between all cluster members, the only long distance messages will be between the cluster representa-
tives, and therefore the link failure will affect only inter-representative communication. This is not the
case in the flat setting, where there may be many transcontinental messages, which may take a lot of time
and may traverse unreliable channels. Also we should mention that in spite of the fact that the total expo-
nentiation in our protocol is larger, the average exponentiation (per member) is much smaller than that
in the flat setting. Also most of the exponentiations are performed concurrently, which fairly distribute
the load between the group members. Lastly, the special rule of the last member in GDH-2 is distributed
in our protocol between the clusters’ representatives, which easily distributes the working load and helps
in fault isolation. Failure of one representative affects only its cluster (and its descendants), while other
group members will not be affected.

As already mentioned, the nature of group communication is not static. The membership of the group
mutates during the course of the communication. After any membership changes, the session keys should
be changed accordingly. Our protocol provides the underlying efficient auxiliary protocols to support the
regeneration of session keys after any membership changes. In the following section, we introduce the
way our protocol supports these changes.

5. GROUP MEMBERSHIP CHANGES

In the previous section, we have assumed that the group membership is determined in advance and that
all the members have been authorized to participate in the group communication prior to execution of
the key generation protocol. However, owing to the dynamic nature of group-oriented applications, our
key agreement protocol must handle adjustments to group secrets after any membership change. The
notification of any membership changes is supported by the underlying group communication system.
The numbering scheme of the remaining members must be updated accordingly.

The membership changes include a new member(s) joining in and an old member(s) leaving (or getting
evicted from) the group. Moreover, due to environmental factors such as network failure, groups can be
partitioned. Similarly, when network partitions heal, multiple groups need to merge into one. In addi-
tion to the aforementioned membership operations, periodic refreshes of group secrets are advisable to
limit the amount of ciphertext generated with the same key and to recover from potential compromises
of members’ contributions of a prior session key.

In the following subsections, we will explain how our protocol supports the aforementioned
operations, that is to say the join, leave, merge, partition and key refresh protocols. In these subsec-
tions, we assume that every member can determine (unambiguously) an insertion point in the hierar-
chical structure. In other words, we assume that after checking the eligibility of the new member to
join the group, NA will provide the joining member with its position in the hierarchical structure
accompanied by the public parameters used in the protocol. Also, NA is responsible for informing the
other group members of the identification of the new joining member. Similarly, when any mem-
ber leaves, NA is the entity responsible for updating the membership view for all the remaining
members.

5.1 Join Protocol

The main security requirement of a member joining in is the secrecy of the previous group keys with
respect to both outsiders and the new group member. In our protocol, this can be achieved as follows
Algorithm 3: A user willing to join the group initiates the protocol by sending a join request message to the network administrator NA who checks the eligibility of the new member to join this group. If this condition is verified, NA replies with the needed cryptographic public parameters \( C_p (\alpha \text{ and } p) \) and also the specific insertion point (a specific cluster \( C_{l,v} \) and the member’s number within the cluster). Usually, a new member joins the protocol at level \( l \), since the vacant places are always in the lowest level.\(^†\) The insertion point should keep the balance of the hierarchical structure; i.e., the new member joins the cluster with the lowest number of members so as not to increase the levels of the hierarchy. On the other hand, if the group members were distributed in a geographical base (as we have assumed), the new member joins the nearest cluster. If the number of members in one cluster becomes too large, it can be split into two or more clusters.

After the join request has been granted, the join protocol begins. Suppose that the new member joins cluster \( C_{l,v} \). The cluster’s representative, \( M^y_{l-1,v} \), who is supposed to keep the upflow message received during the previous change, picks another secret exponent \( r^y_{l,v} \) computes another upflow message and sends it to the new member, who is slated to take the role of the cluster representative. Upon receiving the intermediate values, the new member adds its secret share \( r^y_{l,v} \) and broadcasts the intermediate values to the other clusters’ members. All the cluster members and their descendants are now able to generate their new cluster session key. After regenerating the cluster session key, the (old) cluster representative, \( M^y_{l-1,v} \), requests that its ancestor cluster, \( C_{l-1,v} \), refreshes its session key. \( M^y_{l-1,v} \) uses the new lower cluster’s session key as a new secret share. According to its position in the cluster \( y = 1, 2, \ldots, d - 1 \), \( M^y_{l-1,v} \), adds its public share to the message received from its preceding member in the last conduction of the key generation/regeneration protocol and sends this message to its next member. By the time the new intermediate values have been received by \( C_{l,v} \)’s representative, \( M^y_{l,v} \), it puts its share (which may have changed) and broadcasts these intermediate values to its descendants. This process continues until the root cluster regenerates a new group session key.

Algorithm 3. (Join Protocol for New Member \( M^y_{l,v} \))

1. \( M^y_{l-1,v} \) \( \Rightarrow \) \( M^y_{l,v} \) : \( \alpha \prod_{x \in [1,d-1]} r^y_{l,v} \)

2. \( M^y_{l,v} \) \( \Rightarrow \) \( \{M^y_{l-1,v}\}_{x} \) : \( \alpha \prod_{x \in [1,d-1]} r^y_{l,v} \)

3. \( r^y_{l,v} = \alpha \prod_{x \in [1,d-1]} r^y_{l,v} \)

4. For \( i = l - 1 \) to 2

4.1 \( x_1 = \left[ \frac{v}{d - 1} \right] \)

4.2 Complete the steps as the Initial Key Generation Protocol

5. Redo the Broadcasting Round

\(^†\)The reason for this choice will be explained in the following subsection, which explains the Leave protocol.
The cost of the join protocol depends on the position of the cluster in the overall hierarchical structure where the new member joins (i.e., the level where this cluster is located and also the positions of the representative of each ancestor cluster up to the root cluster). The following table shows the cost of the join protocol where we consider that the new member joins at the lowest level \( l \) and all the representatives are located in the first position in the cluster numbering, which is the worst case in our setting, i.e., the balanced structure.

<table>
<thead>
<tr>
<th>Rounds</th>
<th>( R = 2 + (l - 1)d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages</td>
<td>( M = 2 + (l - 1)d )</td>
</tr>
<tr>
<td>Exponentiations</td>
<td>( E_{x_5} = d(l - 1)\frac{d+1}{2} + l + 1 )</td>
</tr>
<tr>
<td>Max. message size</td>
<td>( M_5 = d )</td>
</tr>
</tbody>
</table>

5.2 Leave Protocol

Leave protocol is initiated in two cases: (1) when a member wishes (voluntarily) to leave the session group; or (2) the NA wants to expel a member from the group (maybe due to a certain policy or security compromise). The main security requirement, when a member leaves (especially if forced to leave) is the secrecy of the subsequent group keys with respect to both outsiders and former group members. After granting the leave request to a member, NA sends a notification message to the other members of her/his cluster to update the numbering scheme of their cluster. This change in the indexing scheme will result in replacing the removed member by the last member of its lower level cluster. For example, if member \( M_{2,2} \) from cluster \( C_{2,2} \) leaves the group (Figure 2), the last member of the cluster \( C_{3,4} \) \( M_{3,4} \) will take his/her place. This updating process of member distribution continues until \( level \), since the vacant places will always be in \( level \). After the member numbering has been updated, the key regeneration protocol starts. The protocol starts from the lowest level cluster (cluster \( C_{3,4} \) in our example). The new cluster representative of this cluster picks another secret share and adds it to the last message it has received from the preceding member in the cluster during the last rekeying protocol (or from the initial key generation, if it is the first membership change in the session course), and broadcasts these new intermediate values to the remaining cluster members. In the next stage, the protocol runs as the initial key generation protocol. The only difference is that the protocol starts from this new representative, who adds the new share to the contributions from its preceding member and forwards the results to the next member. This procedure continues until the root layer as described in the Initial Key Generation Protocol (‘Initial Session Key Generation’, above). The main difference between this protocol and Initial Key generation one is that only a subset of the group members will be involved in the key regeneration protocol (namely, in our example the members of \( C_{3,4}, C_{2,2}, C_{1,1} \) will be involved), while the other member will only participate in the last round (Broadcasting round).

Since only a subset of group members participate in regenerating the new group session key, the cost of performing this protocol is based on the position of the departing member. The worst case happens if the departed member was in the lowest level and the position of the representatives up to the root level are the first members in their clusters. The best case occurs where the member who left was in the root level. The following metrics define the cost of the leave protocol, where now we consider again the worst case.

<table>
<thead>
<tr>
<th>Rounds</th>
<th>( R = 1 + (l - 1)d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages</td>
<td>( M = 1 + (l - 1)d )</td>
</tr>
<tr>
<td>Exponentiations</td>
<td>( E_{x_5} = 2 + d(l - 1)\frac{d+1}{2} )</td>
</tr>
<tr>
<td>Max. message size</td>
<td>( M_5 = d )</td>
</tr>
</tbody>
</table>
5.3 Merge Protocol

The merge operation is used to add \( m > 1 \) members to the current group of \( n \) members. As mentioned earlier ('Join Protocol'), the members always join the group by the lowest level. If \( m < d \), and if there are empty spaces to accommodate these members, then it can be treated as the join protocol used \( m \) times from the point of view of the cluster where the \( m \) members will join. But with respect to the other clusters it will be considered as if only one member was joining. Otherwise if there are no empty spaces to accommodate these new members, they can construct a new cluster and their representative joins the group as if only one member was joining. If the new members join at different clusters at the same time, it will also be considered as the join protocol has been conducted \( m \) times. The only difference in this case is that the number of members involved in the protocol will be increased according to the number of clusters where the new members join. We should note that the cost of the merge protocol is approximately the same as the join protocol, since the joined clusters at the same level perform the required protocol steps concurrently. If \( m \geq d \), the \( m \) members can construct a new cluster(s) with their cluster session key(s) and one (or more) member (which will be considered as the cluster representative(s)) can join an old cluster as one (or more) member(s). From the point of view of the old group members, the number of joining members in this case equal to \( \left\lceil \frac{m}{d} \right\rceil \). So with respect to the old \( n \) members, it will be considered as one (or more) member(s) requests to join the current group. The indexing of the \( n + m \) members needs to be updated by the NA.

5.4 Partition Protocol

The partition operation removes \( m \) members from the current group of \( n \) members. If the partition occurs due to a network failure, two cases have to be considered: (1) The members to be removed belong to the same cluster. In this case, the NA will deal with this situation as if only one member (the representative) is leaving. If the representative remains in the group he/she will regenerate a new key as if this representative had joined the group or had initiated a key refresh protocol. (2) The \( m \) members belong to different clusters; then the NA can determine whether the remaining \( n - m \) members need to construct a new hierarchy or keep the same hierarchy. In both cases, the remaining members need to regenerate a new cluster session key taking into account the required modification to the members' indexing scheme. If the partition is due to multiple leave requests, it will be considered as if the leave protocol is performing \( m \) times.

As shown in the previous discussion, the clustering approach is more efficient in the case of group partition, since with high probability the departing members belong to the same cluster, which makes the cost of the partition protocol for \( m \) members the same as just one member leaving.

5.5 Key Refresh Protocol

The key refresh protocol updates the group session key. This protocol is initiated by the NA and it depends on the application policy. Generally, if the group membership is static for a certain period, a key refresh protocol should be initiated. A key refresh protocol is proposed for these reasons:

- to limit the amount of available ciphertext encrypted with the same key;
- to limit exposure due to loss of session key;
- to avoid long-term storage of secret keys.

Key refresh protocol can be considered as a special case of the leave protocol with no departing members. It starts, as Leave protocol, from the lowest level. The cluster representative of a certain cluster, \( C_{v} \) (chosen by NA), picks a new secret share and reruns the broadcasting round to its cluster members. After generating a new cluster session key, the protocol runs exactly as the Leave protocol.

If a certain cluster needs to refresh its cluster session key, the protocol can be executed for this cluster independent of the rest of the group members. In this case, the cluster will regenerate its new cluster session key.
session key without changing the group session key. The cost of refreshing the group session key is similar to that of the Leave protocol.

6. COMPLEXITY ANALYSIS

Beyond the security of the system, the complexity of the protocol has always been an important issue when designing group key management systems. From the conceptual perspective, we are interested in two major cost aspects: the cost of communications (number of rounds; number and type of messages; and the maximum bandwidth) and the cost of computations. Here, for the computational cost, we consider the cost of modular exponentiations, since it is the most costly computational process. There is always a trade-off between both costs, but this is typically based on the underlying communication facilities and on the applications. For example, the cost of communication in a modern high-speed LAN setting can appear negligible in comparison to the cost of modular exponentiations. On the other hand, the communication cost is very important in high-delay networks (e.g., WANs, Internet).

In this section we analyze the communication and computation costs of the aforementioned protocols, namely, initial key generation, join, leave, merge, and partition protocols. In our analysis, we consider the following costs:

**Number of Rounds** \((R)\). The total number of rounds is the number of actions among members that need to be done serially (Mullender [10], pp. 133–134). During a round, each member takes the following actions in the specified order:

1. He/she sends a message to their neighbour and/or broadcasts a message to their lower-level cluster members.
2. He/she receives the message sent to them by their parent or neighbour.
3. He/she changes the values of the messages received by adding their exponent.

**Number of Messages** \((M)\). The total number of messages plus the total number of broadcasts sent according to the protocol. A message is a single packet sent from one member to another, while a broadcast is a message sent by a member and received by a subset (or all) of the group members. So the number of messages measures the number of packets sent (where a broadcast message is considered as one message).

**Number of Exponentiations** \(\text{Ex}_S\). Since modular exponentiation is the most expensive operation (it requires \(O(\log^3 p)\) bit operations in \(\mathbb{Z}_p^*\), the major concern in reducing the computation overhead is to reduce the total number of exponentiations needed to be executed by the members during the protocol.

**Maximum Message size** \(M_s\). The maximum message size is considered as the size of the largest message sent during protocol processing. From the largest message size, we can calculate the commutative message size. From this we can easily derive the concrete bandwidth requirement once the concrete parameters, such as the group \(G\) and its encoding, are known.

We will compare our protocol with the contributory group key agreement scheme GDH.2 described elsewhere [6,14]. To the best of our knowledge, GDH.2 of the Cliques [14] is the best protocol providing contributory authenticated group key agreement, and is capable of dynamic membership events, and of handling both partitions and merges. Also, we will compare our protocol with TGDH [8]. TGDH is a new group key agreement protocol based on Diffie–Hellman key trees, and it is the first proposal to consider the scalability problem. The computational complexity of the TGDH protocol is reduced from \(O(n)\) to \(O(\log n)\), but the communication cost is slightly larger than that of GDH.2.

The overhead of our protocol will depend on the structure of the hierarchy, i.e. the maximum number of members in any cluster \(d\) and the maximum number of levels \(l\) in the hierarchy. As we have seen from the characteristics of each protocol, there is a trade-off between \(d\) and \(l\). The number of rounds in the initial protocol requires \(O(ld)\) for generation of the session key. In the case of membership change...
protocols, on the other hand, it requires a lower number of rounds compared with the initial key generation protocol (as expected) to regenerate the session key. The number of messages and the computation overhead of the initial protocols depend on the maximum number of clusters, $N_C$ (which is approximately $O(d^{l+1})$), so it will be affected more by increasing the number of levels. In the case of membership change protocols, the number of messages and the computation overheads require $O(ld)$ to regenerate the session key. To conclude, the number of levels has to be minimized to achieve the best efficiency, but as the same time the maximum number of members in any cluster has to be bounded above so as not to lose the benefits of our protocol.

Compared with GDH.2, it is clear that the computation overhead of our protocol is superior, as on average it requires $O(ld)$ to generate/regenerate the session key compared to $O(n)$ required by GDH.2, which is approximately $O(d^l)$ (recall that using our protocol the group size $n$ can be represented as follows: $n = d\frac{(d-1)^l-1}{d-2} = d^l$. With respect to the communication overhead, the number of rounds in our initial key agreement protocol is $ld$, which is lower than that of GDH.2, which requires $O(n)$. The number of messages, on the other hand, is greater than that of GDH.2, but the maximum message size in our protocol is $d$, compared with $n$ in GDH.2, since our protocol allows parallel handling of the messages through the network. As a result, the total load on the network from our protocol is less than that of GDH.2.

The communication overhead of TGDH, $2nh$, where $h$ is the height of the tree, is slightly larger than that of GDH.2, which is $n$; it is clear that our protocol is superior with respect to the communication overhead. The computational overhead is approximately the same in both protocols, especially in the case of membership change protocols, since both protocols are multi-round protocols. But our approach is more efficient for large-size groups. Since the tree in TGDH is binary, the depth of the tree $h$ ($h = \log n$ if the tree is fully balanced) is larger than the number of levels, $l$, of our protocol, which allows our protocol to have a lower number of rounds and exponentiations. Also the number of users involved in the key regeneration is lower in our protocol. A summary of the comparisons is given in Tables 1, 2 and 3.

<table>
<thead>
<tr>
<th>Operation Protocol</th>
<th>Initial key generation</th>
<th>GDH.2</th>
<th>TGDH</th>
<th>CGDH</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>$n$</td>
<td>$2h$</td>
<td>$ld$</td>
<td>$d(N_C) + 1$</td>
</tr>
<tr>
<td>$M$</td>
<td>$n$</td>
<td>$2nh$</td>
<td>$d(N_C) + 1$</td>
<td></td>
</tr>
<tr>
<td>$ExS$</td>
<td>$\frac{n(n+3)}{2}$</td>
<td>$2nh$</td>
<td>$\left[d\frac{d+1}{2} + 1\right]$</td>
<td></td>
</tr>
<tr>
<td>$MS$</td>
<td>$n(n - 1)$</td>
<td>$2n$</td>
<td>$d$</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Initial key generation cost

<table>
<thead>
<tr>
<th>Operation Protocol</th>
<th>Join protocol</th>
<th>GDH.2</th>
<th>TGDH</th>
<th>CGDH</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>2</td>
<td>6</td>
<td>$2 + d(l - 1)$</td>
<td></td>
</tr>
<tr>
<td>$M$</td>
<td>2</td>
<td>2</td>
<td>$2 + d(l - 1)$</td>
<td></td>
</tr>
<tr>
<td>$ExS$</td>
<td>$3n + 2$</td>
<td>$\frac{3h}{2}$</td>
<td>$(l - 1)\left(d\frac{d+1}{2}\right) + l + 1$</td>
<td></td>
</tr>
<tr>
<td>$MS$</td>
<td>$n$</td>
<td>$2n$</td>
<td>$d$</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Join protocol cost
Scalability of key distribution protocols have been considered using different approaches (see, for example, references 8, 11, 15, 16). In this paper we have presented a novel framework for a scalable and fault-tolerant key agreement protocol. The most important feature of our protocol is the adoption of a clustering structure in key agreement protocols. Using clustering, we were able to reduce the cost of session key generation/regeneration and easily isolate any member or link failure away from the other group members. This allowed us to develop a scalable key agreement protocol. Our approach is an efficient and scalable modified extension of the well-known DH key exchange protocol to a multiparty key agreement protocol. Although we used a modified GDH.2 (which is used only to explain the idea behind our protocol) as a building block to generate a cluster session key, the group session key is different from that of GDH.2. The efficiency of our protocol is also independent of the building block protocol used to generate the cluster session key. We should point out that although we have focused on key agreement protocols, known to be less efficient than key distribution protocols, our protocol is comparable to the best-known key distribution protocol. Moreover, regarding the storage requirement for each of the members, our protocol is superior. Actually, the group members in our proposal do not store intermediate keys to generate/regenerate the session key.

Some considerations deserve further study. First and foremost, although we have (heuristically) shown that our protocol is efficient with large numbers of group members, there are many practical issues that still need to be discussed. For example, we have claimed in our protocol that all the clusters at the same level generate their session keys at the same time, which needs synchronization between subgroups. An efficient implementation of our protocol is required to consider these and other practical issues. Also, our complexity analysis was built on a ‘balanced’ hierarchical structure, where the maximum number of members in each cluster is equal for all clusters and the number of levels is the same for each branch. This assumption considers a uniform distribution of the members. Another analysis should be performed with this constraint removed while building the structure, that is, when the cluster sizes and the branch lengths are variable.

Second, we plan to investigate use of faster algorithms for key agreement protocols. In particular, we will focus on approaches leading to reduced key length and/or reduced computation overhead, such as those using elliptic curves.

Third, as already mentioned, the proposed protocol is the first step in building a complete authenticated key agreement protocol for group communication. We proposed that all the communications run over authenticated channels, where our protocol should be secure against passive attacks. We still need to discuss possible active attacks and modify our protocol to add some security attributes like entity and message authentication, key confirmation, key freshness, etc., which enable our protocol to withstand these attacks.

ACKNOWLEDGEMENTS

This work was supported by an NSERC (Canada) grant and by the Council of Science and Technology of Castilla–La Mancha under project PBC-03-001. This work was performed while L. Orozco
Barbosa was at the School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada.

REFERENCES


AUTHOR’S BIOGRAPHIES

**A. Abdel-Hafez** received his PhD in Electrical and Computer Engineering from the University of Ottawa, Canada, in 2003. From 2000 to 2003, he held a teaching assistant position at the School of Information Technology and Engineering, University of Ottawa. His current research interests are wireless communications, computer security and networking.

**A. Miri** received his PhD in Electrical and Computer Engineering from the University of Waterloo, Canada, in 1997. From 1999 to 2001, he was at the University of Waterloo as a Postdoctoral Fellow. Since 2001, he has been with the School of Information Technology and Engineering (SITE) at the University of Ottawa, where he is currently an Associate Professor. His research interests include digital communication, networking and applied security.

**Luis Orozco-Barbosa** received his PhD from the University Pierre and Marie Curie, France, in 1987. From 1991 to 2002, he was a Faculty member of the School of Information Technology and Engineering (SITE), University of Ottawa. He is now Director of the Research Institute of Informatics at Albacete, Spain. His current research interests include wireless communications, security protocols and performance evaluation.