Hierarchical Tree Approach to Group Key Management using the Group Diffie-Hellman Protocol

by

Peter King Pong Au

B.Sc., University of British Columbia, 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

April 2007

Abstract

As a result of the increasing popularity of group-oriented peer-to-peer applications, there is an increasing demand for security services in this kind of environment. Key management plays an important foundation role in security services. Quite a few key management solutions have been proposed for peer groups. However, they usually have limited scalability. This paper considers the scalability of a group-oriented environment and proposes a group hierarchy solution to resolve the problem.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank to my supervisor, Dr. Son Vuong, for his inspiration, guidance and encouragement for the past 5 years. It would have been impossible for me to get this far without his support and understanding. I'm also grateful to have Dr. Alan Wagner for being my second reader and comments he provided.

I would also like to thank my family members for their endless love, especially my wife, Jany, for her understanding, support and patient.

Finally I want to dedicate this work to my son, Joseph, who was born during the time of this research. He has brought me quite a few of troubles, but at the same time, given me endless hours of happiness.

Peter Au
The University of British Columbia

Chapter 1

Introduction

1.1 Motivation

The increasing popularity of distributed, group-oriented collaborative applications has prompted the need for security services to provide communication privacy and integrity in these contexts. Such services should be specifically geared towards group-oriented collaborative environments and optimized for such environments. However, security mechanisms for collaborative communication are usually complicated and expensive compared with the mechanisms used for the non-collaborative multicasting communication commonly found on in today's Internet.

Security requirements for group-oriented collaborative environments present some interesting research challenges. Key management, which is the foundation of all security requirements, plays an important role in this research area. In the past several years, quite a number of key management solutions have been proposed for peer groups. Key management models for peer-group operations have been addressed by several other researchers. However, they have usually targeted on internal security issues within a single independent group with a simple security model, such as group formation and internal membership operations. On the other hand, the issues of external membership changes, where security requirements can be inherited or shared between groups, remain relatively unexplored.

A fully functional scalable collaborative environment requires not just reliable peer communication within a single group, but also a security mechanism for multiple groups interacting with each other. In a scalable collaborative environment, group is an important concept. Each group is a collection of peers who can share the same functionality, such as level of security. There may also be existing security relationship between groups through inheriting or sharing mechanisms. Therefore, the security requirements in such an environment can be much more complicated than simple peer-to-peer security within a single group. A more sophisticated key management scheme is thus required to address security concerns in these environments.

1.2 Overview of Diffie-Hellman Protocol

The Diffie-Hellman key agreement protocol was developed by W. Diffie and M. Hellman in 1976 and was published in a paper called "New Directions in Cryptography" [1]. It was subsequently proposed as an Internet standard by the Internet Engineering Task Force (IETF) as RFC 2631 [2]. It is also known as the exponential key agreement protocol. The protocol allows two parties to exchange a secret key through an insecure medium without sharing any prior secrets.

The Diffie-Hellman Key Agreement Protocol requires both parties to have key pairs. By combining the one party's private key with the other party's public key, both parties can compute the same shared secret number. This number is usually used as a key-encryption key to encrypt the content-encryption key, which in turn is used to encrypt the content data.

We will now look at the Diffie-Hellman key agreement protocol in more detail. The protocol requires two system parameters: p and g. Both parameters are considered as public values and may be used by all users in the system. Parameter p is a prime number (usually very large) and parameter g (also called a generator) is an integer smaller than p. These two parameters are picked with the following property:

For every number n between 1 and p-1 inclusive, there is a power k of g such that $n = g^k \bmod p$.

Suppose we have two parties X and Y, who want to agree on a shared key using the Diffie-Hellman key agreement protocol. First, each party generates its own private value (a.k.a. private key (PK)), say, a and b, respectively. Then they derive their public value (a.k.a. public key or blind key (BK)) using the parameters p and q, as follows:

$$\text{BK} = g^{(\text{PK})} \bmod p$$

We simplify this procedure by using $\alpha = g \bmod p$, as,

$$\text{BK} = \alpha^{(\text{PK})}$$

Now, party X's public key is $g^a \bmod p = \alpha^a$, and party Y's public key is $g^b \bmod p = \alpha^b$. The parties then exchange their public keys with each other. Finally, they each combine their own private key with the other's public key, and calculate the shared secret key as follows:

$$\text{shared secret key } k = (\text{Other's BK})^{(\text{own PK})}$$

Party X computes $(g^b)^a \bmod p = \alpha^{ab}$, and party Y computes $(g^a)^b \bmod p = \alpha^{ba}$. Because $\alpha^{ab} = \alpha^{ba} = k$, both parties now have k as the shared secret key.

Later researchers extended the two-party Diffie-Hellman protocol to n-parties. They called it the "natural" extension [8] to two-party Diffie-Hellman protocol by generalizing the calculation to n-parties. The shared secret key, usually referred to as the group key (GK) in a group setting, is calculated as follows:

$$\text{GK} = \alpha^{\text{PK}_1 \text{PK}_2 \text{PK}_3 \ldots \text{PK}_n}$$

More information regarding the Group Diffie-Hellman Protocol can be found in the later section "Previous Related Works".

1.3 Exponentiations of DH-family Protocols

The Diffie-Hellman key agreement protocol is also known as the exponential key agreement protocol because of its exponentiation requirement.

As described in the last section, given two system parameters, p and g, the blink key (BK) is an exponentiation, BK = $\alpha^{(PK)}$, where α is called the exponentiation base, which is equal to g mod p. After the two parties have exchanged the blink keys with each other, they calculate the shared secret key k using another exponentiation, k = (Other's BK)$^{(own\ PK)}$. Therefore, to successfully complete a 2-party Diffie-Hellman key agreement protocol, each party needs to compute at least two exponentiations.

In n-parties Diffie-Hellman agreement protocol, the shared group key (GK), as described in the last section, is calculated as GK = $\alpha^{PK_1 PK_2 PK_3 \ldots PK_n}$. To calculate this group key, intermediate values are passed between group members to gather the private keys, and multiple exponentiations are necessary to complete the protocol. Because exponentiation is costly, the number of exponentiations in the system becomes one of the important measures in the protocol complexity.

1.4 Synopsis

In this work, we address the key management security requirements of a dynamic, scalable, collaborative environment in a hierarchical security model, where the security level of different groups can be inherited or shared. We use one of the well-known solutions to peer-group key management, the Diffie-Hellman Key Exchange Protocol, to address the internal key exchange between peers within each group. We then apply the same protocol in a special way to address inter-group key management issues, by arranging groups in a hierarchical tree structure according to inheritance of different levels of security requirements in the collaborative environment.

This thesis is divided into six sections. In Chapter 2, we briefly review the Diffie-Hellman Key Exchange Protocol and the various extensions devised by other researchers, and describe how it is applied to our work. In Chapter 3, we describe our proposed new design and algorithm regarding the organization of peer groups in a hierarchical tree structure and how the Diffie-Hellman Key Exchange can be extended for use in this tree structure. In Chapter 4, we do some analysis on the proposed design. Chapter 5 explains the implementation of our design in a media-sharing application. In the final section, we describe further possible research works and provide conclusions based on our results.

Chapter 2

Background and Related Works

2.1 Background

2.1.1 Peer-to-peer communication

The Peer-to-peer communication model has a history as old as the Internet. The original Internet, ARPANET, the purpose of which was to share resources around the U.S., was built based on a peer-to-peer system. The first few hosts on the ARPANET – UCLA, SRI, UCSB, and the University of Utah – were independent hosts with equal status. The ARPANET connected them together not in a master/slave or server/client relationship, but rather as equal peers [25]. Two long-established networks that are still in use today, DNS and Usenet, have peer-to-peer components. However, the rise of the commercial Internet with its millions of home users during 1990s changed the networking model significantly. Server/client-based networking protocols such as SLIP and PPP became common when using slow-speed modem connection. Also server/client-based application protocols, such as Telnet and FTP, became popular. The networking patterns switched more to downloading data rather than sharing resources. Eventually, asymmetric high-speed network links, such as ADSL and cable networks, became common during the late 1990s.

However, server/client-based communication requires a centralized server to handle intensive tasks, which can easily become a bottleneck and a point of failure. In recent years, peer to peer communication has come back into the picture as the "future" communication model. During the late 1990s and early 2000s, peer-to-peer technology was once again becoming popular through some popular file-sharing applications such as

Napster[26] and KaZaA[27], and later in the popular file sharing protocol, BitTorrent[28]. Peer-to-peer telephony applications such as Skype[30], and peer-group-oriented applications such as video conferencing and internet broadcasting sport events [31][32], are also becoming known to Internet users.

A true peer-to-peer network should have all nodes or peers joined together dynamically to participate in some or all intensive tasks that would otherwise be handled by the centralized server. A decentralized peer-to-peer network has several advantages over a centralized server/client network. Theoretically, a decentralized peer-to-peer network is able to scale indefinitely without a significant increase in search time or network bandwidth. It distributes the workload over individual nodes or peers, where their number should always increase in direct proportion to the network size.

Security became a major concern during the late 1990s as the Internet became popular among the general public, and people began to rely on the computers for important communications and tasks. As server/client-based networks and applications have been dominant since the Internet started to become commercialized, most mature security technologies are based on the server/client communication model. As peer-to-peer-based networks and applications have become more popular, further research into security issues in the peer-to-peer environment has been stimulated.

W. Diffie and M. Hellman first published the two-party Diffie-Hellman Key Exchange Protocol [1] in 1976, whereas the first public key cryptosystem was published two years later by Rivest et al. [15]. Since then, various solutions have been proposed to apply the Diffie-Hellman key generation and management concept in a multi-party setting, that is, in a peer-to-peer environment. The main difficulty is to efficiently apply the concept without causing an exponential growth in complexity. During the past 30 years, quite a number of researchers have been working on and proposing new solutions.

2.1.2 Group Key Management

In peer-to-peer environment, communication between peers is called group communication. One of the ways to protect group communication is encryption. An encryption algorithm takes an input data, such as a group message, and performs some conversions on it using a key to generate a ciphered data. The key is a carefully selected number chosen from a large range of values, so that there is no easy way to recover the original data from the ciphered data other than by knowing the key. The messages are protected by encryption using the chosen key, which in the context of group communication is called group key. Only those who know the group key are able to recover the original data.

A secured group communication is relying on a properly selected group key. Therefore, group key management plays an important role in the secured group communication. One of the most important aspects in key management is how the keys are created and then transported to the authenticated peers. Basically, there are two different models used for key creation and transportation: **key distribution** and **key contribution**. In **key distribution**, a centralized key server is responsible for creating and distributing keys to authenticated users. This server is also usually responsible for authenticating users at the time of distributing keys. Since this process matches the server/client communication model well, most server/client-based networks and applications employ **key distribution** as their primary key management model. In **key contribution**, users contribute their own secrets to generate the group key by communicating with each other. There is more communication overhead in **key contribution**, as users need to contact each other at least once in the initialization stage. Authentication is also usually more complicated. However, key computations are distributed among users, avoiding computation bottleneck and a single point of failure.

In peer-to-peer settings, either model has its pros and cons. Some people use the traditional server-client **key distribution** model, as it is relatively more efficient; on the other hand, some people whose work is based on a more pure peer-to-peer approach use

the key **contribution** model. We will look at several different approaches to key management in the peer-to-peer environment in the next section.

2.2 Previous Related Works

A number of approaches to peer-group key management have been proposed. In general, these approaches can be divided into four main categories:

1. Simple Key Distribution Centre (SKDC)
2. Group Diffie-Hellman (GDH)
3. Logical Key Hierarchy (LKH)
4. Distributed Sub-group (DS)

2.2.1 Simple Key Distribution Centre (SKDC)

SKDC is the earliest and simplest solution to the peer-group key management problem. The approach is based on the **key distribution** model. It uses a centralized key assignment method, where a Group Controller (GC), sometimes called Key Distribution Centre (KDC), is responsible for creating, assigning, and managing all keys. Some of the well-known examples are Kerberos developed by MIT [3] [4], and the Group Key Management Protocol (GKMP) by Harney and Muckenhirn [5] [6]. Also, there are some more recent researchers who have proposed some centralized key management schemes in a multicast or group environment such as [32] [33] [34] [35] [36]. SKDC basically applies the key management of the traditional server-client communication model to a peer-group environment. However, this server-client model also brings with its known server-client communication problems, such as a central bottleneck and a single point of failure. Therefore, some researchers such as Poovendran et al. [37] raised the problem of fault-tolerance of GKMP. Some other researchers would not consider this as a proper solution to the peer group environment at all, as it defeats the advantages of peer-to-peer communication [29].

2.2.2 Group Diffie-Hellman (GDH)

It has been 30 years since the 2-party Diffie-Hellman key agreement protocol was first proposed [1]. In the meantime, quite a few researchers have extended the protocol to the group setting. GDH is an approach applying the renowned two-party key agreement protocol in a generalized, n-party environment, where all peers talk to each other to agree on a common group key. Some of the earlier notable examples are Ingemarson et al [13] and Burmester and Desmedt [14]. We can also find some recent examples using GDH such as Cliques by Steiner et al. [7] [8] [9], and also the Octopus protocol [10]. GDH-family protocols are based on a pure **key contribution** concept, honouring the peer-to-peer communication model and taking advantage of it. Bresson et al. [16] [17] [18] [19] [20] showed that solving the Group Computational Diffie-Hellman problem is at least as hard as solving the Two-party Computational Diffie-Hellman problem. So, the security of GDH is ensured. However, some of the research results are only of theoretical interest, while others have demonstrated a scalability issue [10].

2.2.3 Logical Key Hierarchy (LKH)

Scalability has long been an issue in key management in a peer-group setting. LKH is an approach that takes advantage of a data tree structure to minimize size, storage and computational cost. In general, the peers would be arranged at the leaf level of a hierarchy tree. Intermediate keys are computed up the tree, from the leaves to the root, until the final group key is agreed upon at the root of the tree. In some cases, intermediate keys are useful in membership operations of the group. There have been quite a few of variants and theoretical studies in this area, especially in key trees [22] [24], key graphs [23], and Hierarchy Binary Tree proposed by Wallner et al. [38] and Caronni et al. [39] [40]. Some researchers put the idea forward in more practical approaches. Examples include the One-way Function Tree by D. McGrew and A. Sherman [11] [41], and also the Tree-based Group Diffie-Hellman (TGDH) by Y. Kim et al. [12] [21]. For these contributions, peers are organized in one single group. The group keys are either managed by a single group controller, or the keys are both contributed and distributed by

the members themselves. The single group controller becomes a crucial point of failure. The entire group will be affected if there is a problem with the controller such as mechanical malfunction. If the group keys are distributed by the members themselves, some researchers criticize that, as the group grows larger, too much key-refreshing overhead makes the approach impractical [43].

2.2.4 Distributed Sub-group (DS)

Some researchers proposed to split the large group into smaller sub-groups. Examples of these contributions include Iolus [44] and Intra-domain Group Key Management [45]. Each sub-group has a different sub-group controller, minimizing the problem of concentrated work. This distributed control collaborates to avoid the central failure, so that local sub-group failure might not affect the whole system.

Iolus, proposed by Suvo Mittra [44], is a framework with a hierarchy of agents that split peers into different sub-groups. Each sub-group is managed by a Group Security Agent (GSA), and all GSAs are grouped together to form a top-level group which is also controlled by a top-level GSA. Iolus uses independent keys for each group. The lack of a global group key makes key refreshing in a sub-group to be treated locally. The drawback is that if a peer needs to transmit a piece of data from one group to the other, the encrypted data is needed to be translated as it is traveling across group boundary, as there are no common group keys that are known to the peers of different sub-groups [42].

On the other hand, Intra-domain Group Key Management, proposed by Hardjono et al. [45], is an intra-region group key management scheme. There are two region levels, one *trunk region*, and one or more *leaf region*. The *leaf region* is further divided into administratively scoped areas. There are one Domain Key Distributor (DKD) and many Area Key Distributors (AKD) for each administratively area. The DKD and AKDs are placed in an extra level group called All-KD-group. The All-KD-group is used for DKD to transmit key-refreshing messages to the AKDs. All areas in the domain use the same group key. For this reason, data do not need to be translated when passing from one area to another. Also, DKD does not need to keep track of all group members, but only keep

track of the AKDs. Nonetheless, since Intra-domain Group Key Management employs a DKD as a single controller, it presents the problem of a single point of failure if the DKD is interrupted [42].

Chapter 3

Design and Algorithms

3.1 Overview

The **key distribution** model is efficient as there is a central key distribution node or server to generate and manage the keys. However, it is limited by the traditional server-client communication disadvantages, defeating the advantages of peer-to-peer communication. On the other hand, **key contribution** model retains all the advantages of peer-to-peer communication, but is known to be inefficient and not very scalable. We propose an approach that is based on both **key distribution** and **key contribution** and also takes advantage of LKH to yield a simple but highly efficient and scalable key agreement method that is effective in a group-oriented collaborative environment. Inspired by the model of DS, we also make use of the logical or intermediate keys in LKH to create a multi-level security environment.

3.2 Simple Group

To form a group with a limited number of group members, key agreement using the **key contribution** concept would yield the best advantages. Therefore, a small *local group* is formed using one of the known key contribution methods to form a group and generate its corresponding group key. The key can be generated by any known key agreement method that makes use of the key contribution concept, for example, one of the proved Group Diffie-Hellman Protocols [7] [8] [9] [10]. However, because a key agreement

protocol using the **key contribution** concept has the problem of scalability, the number of members who can come together to share the same group key would be very limited.

As an example, say members M_1, M_2 and M_3 come together to form group A. By exchanging the necessary information between each member, they generate a group key K_A (See Figure 3.1).

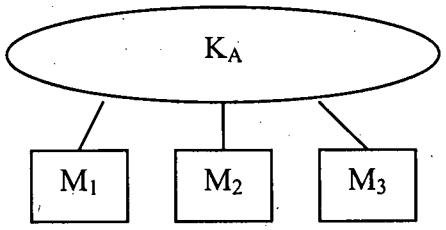

Figure 3.1 Simple Group

Using the Group Diffie-Hellman Protocol, with K_1, K_2 and K_3 as the corresponding secrets (private keys) of M_1, M_2 and M_3, the members exchange the corresponding blind key (public key) $BK_1 = \alpha^{K1}$, $BK_2 = \alpha^{K2}$ and $BK_3 = \alpha^{K3}$ between each others. The group key K_A is then calculated by each member as follows:

$$\mathbf{K_A = \alpha^{K_1 K_2 K_3}}$$

Now M_1, M_2 and M_3 can communicate securely with each other by using the group key K_A.

3.3 Hierarchy of Groups

In a fully functional scalable collaborative environment, the number of members should not be limited, and security controlled by a single group key is not a favourable feature

when different types of members require different levels of security control. Say members of two different groups want to communicate with each other in a traditional key agreement method. There is no choice but to discard the old group key, and generate a new group key that includes contributions from all members of these two groups. This leads to the formation of a bigger group containing all members from both groups. This method is inflexible and may not be feasible when the number of members reaches to a level where scalability and efficiency become issues. Also, it is not possible for each original group to maintain their existing privacy as the old group keys have been discarded or deemed unusable. In a fully functional scalable collaborative environment, this is not a favourable scenario.

Therefore, we introduce a concept of group hierarchy that can provide security in different levels (or areas), but still allow different members from different levels (or areas) to communicate securely with each other. This technique also gives the whole environment more room to scale, because the number of members in each group is limited but the total number of groups is unlimited.

Referring to the example from the previous section, say there are other members who have formed another group B, with group key K_B, using a similar group agreement protocol to that used in group A. If a member of group B wants to communication with members of group A, a super-group C will be formed (see Figure 3.2), with group A and group B as members. A group key K_C is generated using the same key agreement method as in group A and B, with group key K_A and K_B as secret. The necessary message exchanges required by the key agreement protocol could be done by a group leader of each group. This group leader can be elected at the time the group is formed, or by any other election process after the group is formed. After the necessary message exchanges between the two group leaders, a new super-group key K_C is generated and known to both group leaders. The group leaders then forward this new super group-key K_C to their group members. To hide this new group key K_C from external entities, the new key K_C will be encrypted by group key K_A (for members of group A) or K_B (for members of group B).

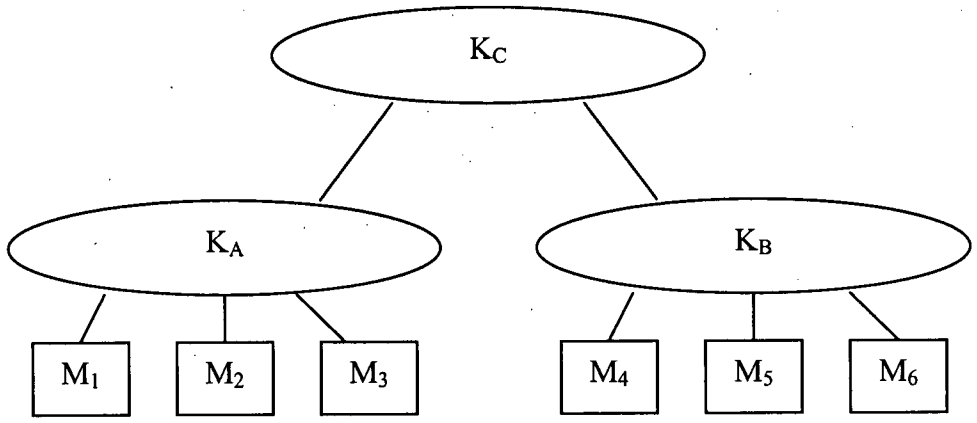

Figure 3.2 Hierarchy of Groups

Using the Diffie-Hellman protocol, the leader of group A calculates the blind key $BK_A = \alpha^{KA}$ and the leader of group B calculates the blind key $BK_B = \alpha^{KB}$. They exchange these blind keys and can each calculate group key K_C, as follows:

$$\mathbf{K_C = \alpha^{K_A K_B}}$$

The leader of group A will then encrypt this new group key K_C by K_A and send it to all members of group A. The leader of Group B will do the same, using K_B to encrypt instead.

Now each member will have 2 group keys, the local group key (K_A or K_B) and the super-group key (K_C). Members can then communicate securely, using either the local group key (for information shared within the same local group) or the super-group key (for information shared with members of the other group).

Note that the role of group leader can be taken by any one of the members, which is an important characteristic to eliminate the possibility of a single point of failure and an efficiency bottleneck.

The same idea can be applied to formation of other local groups or super-groups into a hierarchy of groups with different levels (or areas) of security, creating a flexible and scalable collaborative environment.

Note that each member in the environment will know a set of group keys along a path from their local group to the root group. For example, in Figure 3.3, M_1 would know its local group key K_A and also all of the group keys along the path to the root group, that is, K_C, K_D and K_F. This gives M_1 the flexibility to encrypt data or messages at different levels of security, depending on its target audience. We call the path that leads from the local group to the root group, the **key path**.

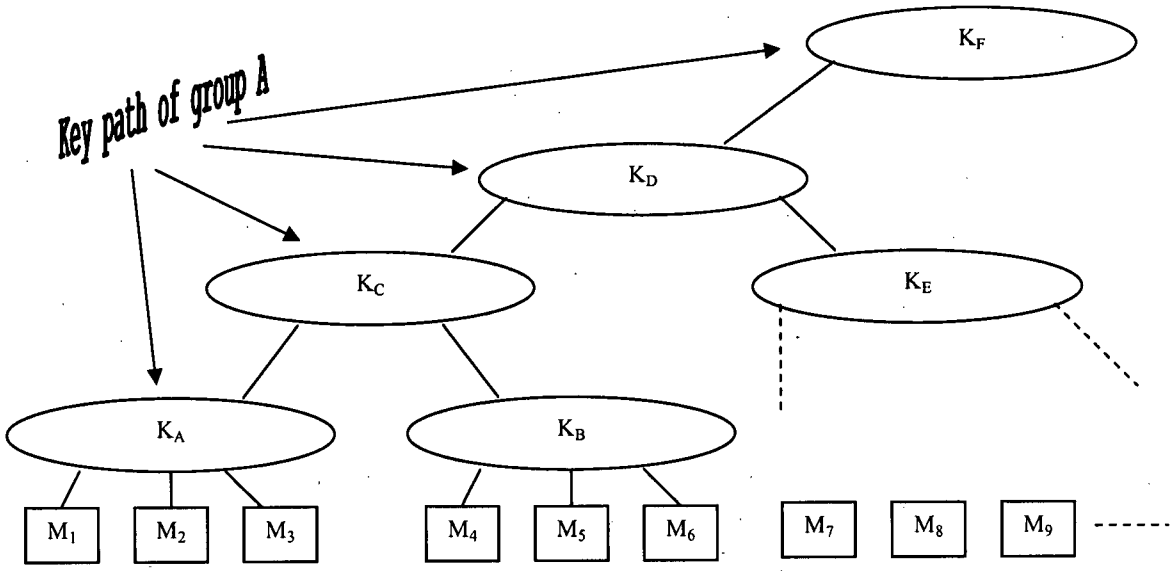

Figure 3.3 Key path

3.4 Membership Changing

For a scalable collaborative environment, membership operations like membership addition, deletion, merging of groups and group division are quite common. But in a scalable collaborative environment with many groups and memberships, we do not want each membership operation affecting every member in the whole environment. One of the goals of our protocol is to restrict the impact of membership operations only to those members who are immediately related to the group where the membership operation is occurring. This does not mean other members will be unaware of the membership changes, but they should at least not get involved in the key re-calculation process resulting from the change in membership. What they will get is just a notice of membership change and the new key(s) that have been created because of the membership change (and of course the information regarding the old key(s) that should be discarded).

Referring again to our previous example, say member M_3 is going to be removed from group A as depicted in Figure 3.4. Group key K_A undoubtedly needs to be refreshed to maintain the privacy of group A. The remaining members M_1 and M_2 would need to get involved to contribute the new group key. Say M_1 and M_2 have got together and create a new group key K_A' by exchanging new blind keys $BK_1' = \alpha^{K1'}$, $BK_2 = \alpha^{K2'}$ where,

$$K_A' = \alpha^{K_1'K_2'}$$

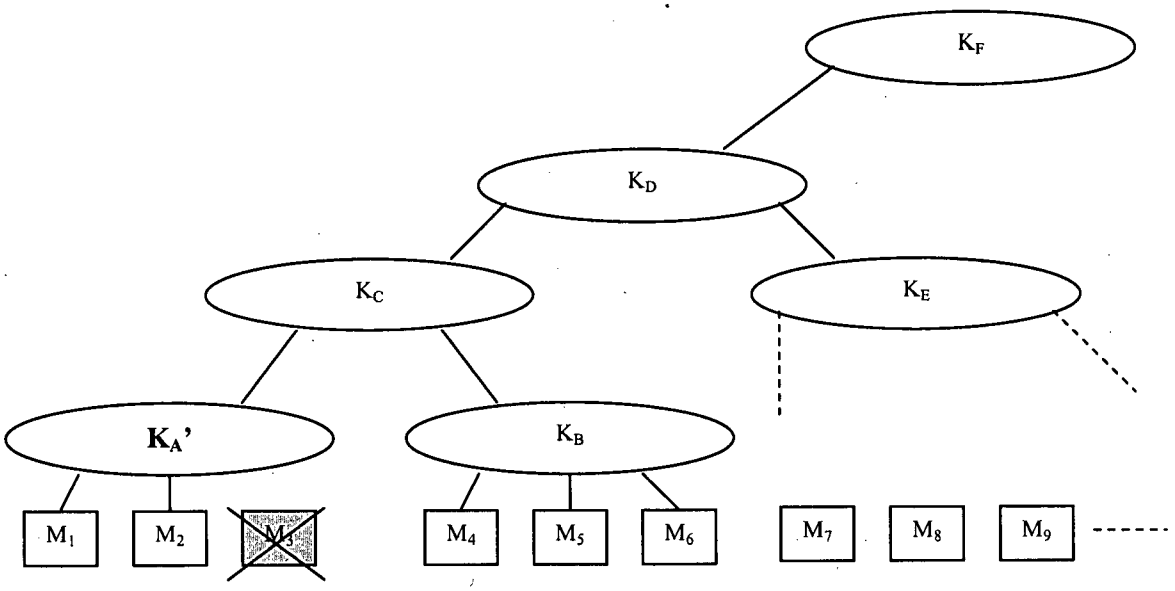

Figure 3.4 Member leaving

Group key K_C of group C, which is the parent group of group A, will now need to be refreshed because it is one of the known group keys to M_3, the deleted member, or, in our terminology, it is one of the keys along the *key path* of M_3. Members or member groups of group C now need to get involved to refresh the group key of group C. In our example, we have two member groups, A and B, belonging to group C, therefore the group leaders of groups A and B will take the responsibility of refreshing the group key of group C. The group key of group C should be calculated based on the group key from group A and group B. We already have a new group key K_A'. On the other hand, K_B has never been made known to M_3. So, if is safe to calculate the new group key K_C' by using the new group key K_A' with the old key K_B, as follows:

$$K_C' = \alpha^{K_A' K_B}$$

Note that we don't want to refresh K_B, and that it is not necessary to, because refreshing K_B would require the involvement of all members from group B, which violates our idea of not impacting members that do not immediately belong to the group with the membership changes.

Now we have a new group key K_C' (see Figure 3.5), and the leader of group A and group B will encrypt this new group key K_C' with the corresponding local group key (K_A' or K_B) and send it to their members.

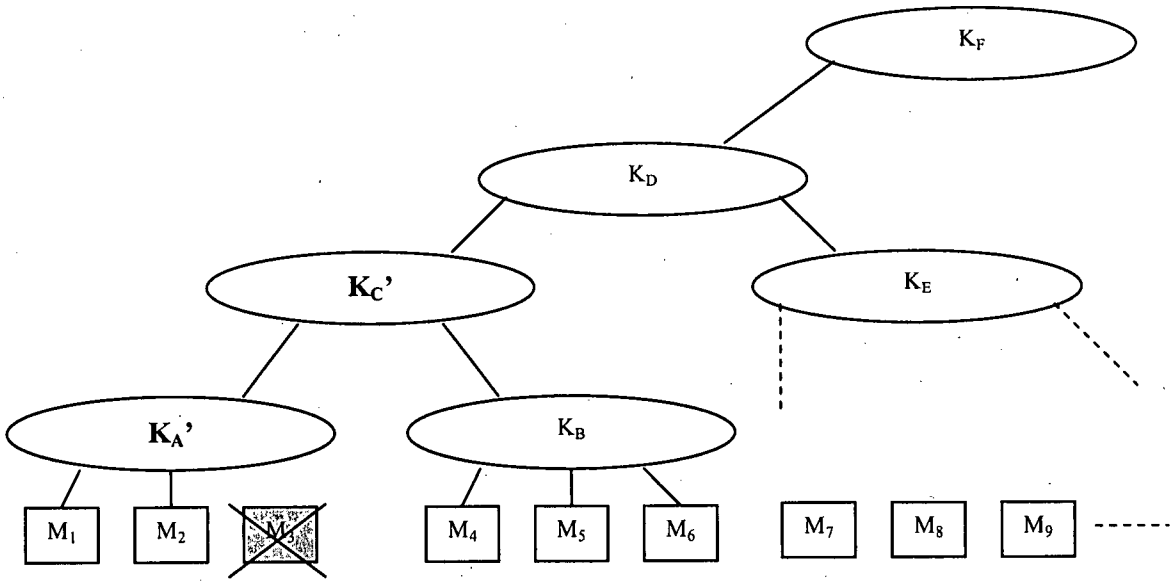

Figure 3.5 Key refresh of parent group

Similar key refreshing of all keys along the *key path* of group A will happen subsequently, as shown in Figure 3.6. This protocol requires only the group keys along the *key path* of the group where the membership operation happens to refresh. Only members of the group where the membership operation happens, plus the group leaders of sibling groups along the *key path* would need to get involved in the key refreshing process. This

procedure significantly reduces the number of members affected because of the membership changes, and also reduces the number of exponential calculations required by the key-refreshing process. The analysis of the advantages of this procedure can be found in the later section "Efficiency of the Protocol" of next chapter.

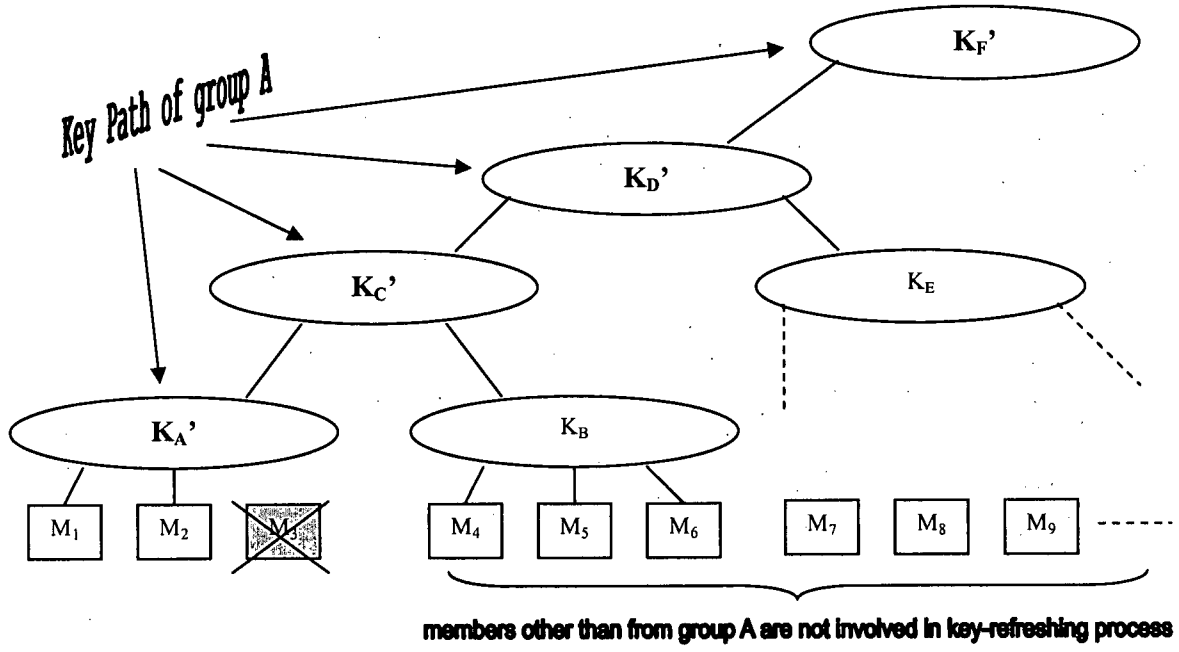

Figure 3.6 Key refresh along key path

Chapter 4

Analysis

4.1 Efficient Diffie-Hellman-Based Protocols

K. Becker et al. in a research report [10] defined and proved lower bounds of some of the characteristics of the Diffie-Hellman-based protocols. In other words, they found out what an efficient Diffie-Hellman-based protocol should consist of, by dividing Diffie-Hellman-Based protocols into *broadcasting* and *non-broadcasting* types. Because in our approach we need to broadcast some of the messages, we are more interested in the *broadcasting* type.

Becker et al. [10] defined the lower bounds of any efficient Diffie-Hellman-based protocol as follows:

Total number of messages >= N
Total number of rounds >= $\log N$

where N is the total number of members involved in the protocol.

In other words, a Diffie-Hellman-based protocol with the following complexities would be considered to be an efficient protocol:

Total number of messages = $O(N)$
Total number of rounds = $O(\log N)$

We will use these lower bounds and complexities as the basis for analysis of the efficiency of our protocol.

4.2 Efficiency of the Protocol

For all key exchange protocols, there are usually 2 types of key-refreshing processes. First is the Initial Key Agreement (IKA) process, where all members come together to agree upon the first group key(s). This process happens in the very first stage where all peers just come together and nobody knows each other. Second is the Subsequent Key Refreshing (SKR) process, which happens after IKA produces the first group key. Several situations require SKR, for example, various different membership changes and key expiration.

As we employ the **Key Contribution** concept, the IKA process is usually more resource-consuming, as it has no choices but all members are required to contribute towards the initial group key(s). Therefore, there is not much streamlining we can do, since every member needs to be involved at least once during the process. Based on the principle of a scalable collaborative environment as described above, a group hierarchy is formed voluntarily when necessary. Therefore, an arbitrary initial formation requiring IKA is not meaningful. However, we still describe the IKA, assuming there is some arbitrary number of members coming together to form a tree structure according to our approach. But this is just for comparison purposes and does not reflect the principles we mentioned earlier.

On the other hand, for all efficient GDH protocols, the number of members participating in the SKR process is a crucial factor in the protocol's efficiency. Our approach to improving the efficiency of the GDH protocol is to reduce the number of members taking part in the key-refreshing process. Only members of the group where the membership operation occurs, plus the group leaders of sibling groups along the *key path*, would need

to get involved in the key-refreshing process. This significantly reduces the number messages being passed around and the number of exponential calculations, which improves efficiency.

The following notations are used throughout this section:

N:	Total number of members
n:	Number of members in the group where key refreshing is necessary
b:	Number of groups at the leaf level of the hierarchy tree
d:	Level of the hierarchy tree
T:	Total number of groups

Different possible settings and scenarios are discussed in this section. The settings are arbitrary but can easily be created and enforced through application or administration policies. For each scenario, we look at the three common measures of communication protocol complexity:

1. Total messages
2. Total number of rounds
3. Number of broadcast messages

The two key refreshing-processes, IKA and SKR, will be analyzed in each measure.

4.2.1 Scenario 1: Balanced Binary Tree with Members in Leaf Level Only

We first look at a Balanced Binary Tree (BBT) which is the simplest and easiest case to understand (See Figure 4.1). We organize the hierarchy of groups into a BBT, and members are only allowed to join groups at the leaf level, which makes the case even simpler but still realistic and practical.

In this scenario, all non-leaf groups will use an efficient 2-party DH algorithm to generate the group key. Because of the nature of 2-party DH algorithm, where n is always 2, the complexity is constant for all non-leaf groups, which is O(1).

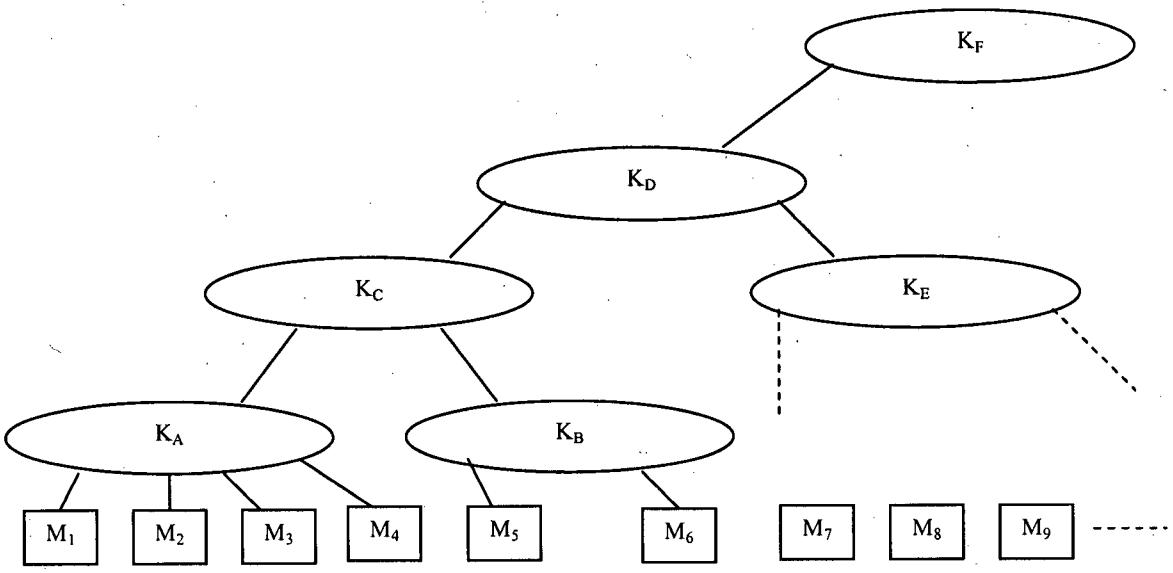

Figure 4.1 BBT with Members in Leaf Level Only

Analysis of *Total Messages* in the case of IKA

Based on the process described in the previous section, an optimized GDH process within the affected group would require $O(n)$ messages to refresh the group key [10]. Let leaf groups G_1, G_2, ... and G_b have n_1, n_2, n_3, ... and n_b members respectively. Because each leaf group needs to do a local IKA itself to find a local group key, the total number of messages would be $O(n_1 + n_2 + n_3 + \ldots + n_b) = O(N)$. We then need to create all sub-group keys above the leaf level. For a BBT, there are $T/2$ sub-groups above the leaf level. Each sub-group key is generated by an efficient 2-party DH algorithm that requires a fixed number of messages, which is O(1). Therefore, we need $T/2 \cdot O(1) = O(T/2)$ messages to create all sub-group keys above the leaf level. After each creation of parent keys, we need to broadcast that parent key back to the corresponding members. In total, we need another $O(T/2)$ messages, because there are $T/2$ parent keys ($T/2$ sub-groups above leaf level) that need to be broadcast. We add up all of these numbers to obtain the total number of messages, as follows:

Total messages: $O(N) + O(T/2 + T/2)$

$O(N + T)$

or $O(N + (2 \cdot b - 1))$ for $T = 2 \cdot b - 1$

$O(N + b)$

Analysis of *Total Messages* in the case of SKR

Based on the process described in the previous section, an optimized GDH process within the affected group would require $O(n + 1)$ messages to refresh the group key (one additional message is for the new member when that member is added). All parent keys along the *key path* are required to refresh. For each parent key along the key path to refresh, an efficient 2-party DH algorithm (the original DH) would require a fixed number of messages, which is O(1). If there are d levels of the tree (the length of the *key path*), we would need $O(d)$ messages to refresh all of the parent keys. For each group, we need to broadcast the parent key back to the corresponding group members, which need another $O(2d)$ messages (one for each side of the child tree, using different keys to encrypt). We add up all of these numbers to obtain the total number of messages, as follows:

Total messages: $O(n + 1) + 3O(d)$

$O(n + 3d)$

or $O(n + 3 \cdot \lg b)$ for $d = \lg b$

$O(n + \lg b)$

Analysis of *Total Rounds* in the case of IKA

Similar arguments to those in the above analysis in "Total messages in the case of IKA" can be applied in "Total Rounds". Here, however, all of the IKA of the local leaf groups are done in parallel. Therefore, the groups with most rounds will dominate this stage. Assuming we are using an optimized GDH protocol as described in Becker[10], which requires O(lg n) rounds, we need $O(\lg(\max(n_1, n_2, n_3, \ldots, n_b)))$ rounds for all of the leaf-level groups to finish the IKA process. Now, after the last leaf-level group finishes

calculating its group keys, the groups in the $(d - 1)$ level start simultaneously and calculate their group keys in parallel, so do all higher levels. Because all non-leaf level groups require constant time to complete their 2-party DH algorithm, and we have d levels, $d \cdot O(1) = O(d)$ rounds are required for all non-leaf groups to calculate their group keys. At the end, the final broadcast messages of the root key need $O(d)$ rounds to complete. Therefore, the total number of rounds can be calculated as follows:

Total rounds: $O(\lg(\max(n_1, n_2, n_3, \ldots, n_b)))+ O(d + d)$

$O(\lg(\max(n_1, n_2, n_3, \ldots, n_b)) + 2d)$

or $O(\lg(\max(n_1, n_2, n_3, \ldots, n_b)) + 2\lg b)$ for $d = \lg b$

$O(\lg(\max(n_1, n_2, n_3, \ldots, n_b)) + \lg b)$

Analysis of *Total Rounds* in the case of SKR

Similar arguments to those in the above analysis in "Total Messages in the case of SKR" can be applied here. We have $O(\lg(n + 1))$ to refresh the group key (one additional message is for the new member when a new member is added). Then, we would need $O(d)$ messages to refresh all of the parent keys along the *key path*. At the end, the final broadcast messages needs only one round to complete. Therefore, the total number of rounds can be calculated as follows:

Total rounds: $O(\lg(n + 1)) + O(d + d)$

$O(\lg n + 2d)$

or $O(\lg n + 2\lg b)$ for $d = \lg b$

$O(\lg n + \lg b)$

Analysis of *Broadcast Messages* in the case of IKA

All sub-group keys, plus the group key, need to be broadcast back to the corresponding members, yielding the following:

Broadcast messages: $O(T)$

or $O(2 \cdot b - 1)$

$O(b)$

Analysis of *Broadcast Messages* in the case of SKR

All sub-group keys along the *key path*, plus the group key, needed to be broadcast back to the corresponding members, yielding the following:

Broadcast messages: $O(d)$

or $O(\lg b)$

Summaries

Our analysis for Scenario 1: Balanced Binary Tree with Members in Leaf Level Only, is summarized in Table 4.1 and Table 4.2.

Messages:	$O(N + b)$
Rounds:	$O(\lg (\max(n_1, n_2, n_3, \ldots, n_b)) + \lg b)$
Broadcasts:	$O(b)$

Table 4.1 Summary for IKA of Scenario 1

Messages:	$O(n + \lg b)$
Rounds:	$O(\lg n + g\ b)$
Broadcasts:	$O(\lg b)$

Table 4.2 Summary for SKR of Scenario 1

General Analysis for Scenario 1

IKA is obviously more costly than SKA. Actually, IKA in this scenario is not efficient as described by Becker [10], where an efficient Diffie-Hellman-based protocol should have O(N) complexity in *Total Messages*. However, this higher IFA complexity is offset by a

lower complexity in SKR, where the complexity depends on the number of members in the local affected group, rather than on the whole member population.

In fact, the efficiencies of SKR in this scenario are generally based on two characteristics: the number of levels (or the number of leaf groups, where d=lg b) and the number of members in the group where key refreshing is necessary. The number of level is a fixed number based on the arbitrary application policy setting. So, the only other significant factor will be the number of members in the affected group.

For *total messages*, we need $O(n + d)$ total messages to complete the process. The efficiency depends on the number of members in the affected group. The worst case scenario would be $n = N$, i.e., all members happens to be in the affected group. Then, we have $O(N + d)$. But this case does not make sense. It would be meaningless to create parent keys and broadcast to other branches that have no members in them. We can easily avoid this situation if we have a good application level policy to prevent overcrowding a single group, and, for optimized performance, to evenly distributed members among the leaf groups.

In the next scenario, we assume that members are evenly distributed among all groups in the leaf level.

4.2.2 Scenario 2: Balanced Binary Tree (BBT) with Members Equally Distributed among Groups in the Leaf Level

It is not completely unrealistic to assume all members are evenly distributed among all groups in the leaf level, as shown in Figure 4.2; this could be done easily through an application level policy of regarding member joining. In this scenario, we will always have $n = N/b$ members, as all leaf groups have the same number of members. Since all other assumptions and constrains are the same as in the last scenario, we can just substitute $n = N/b$ into the formulae in the last scenario.

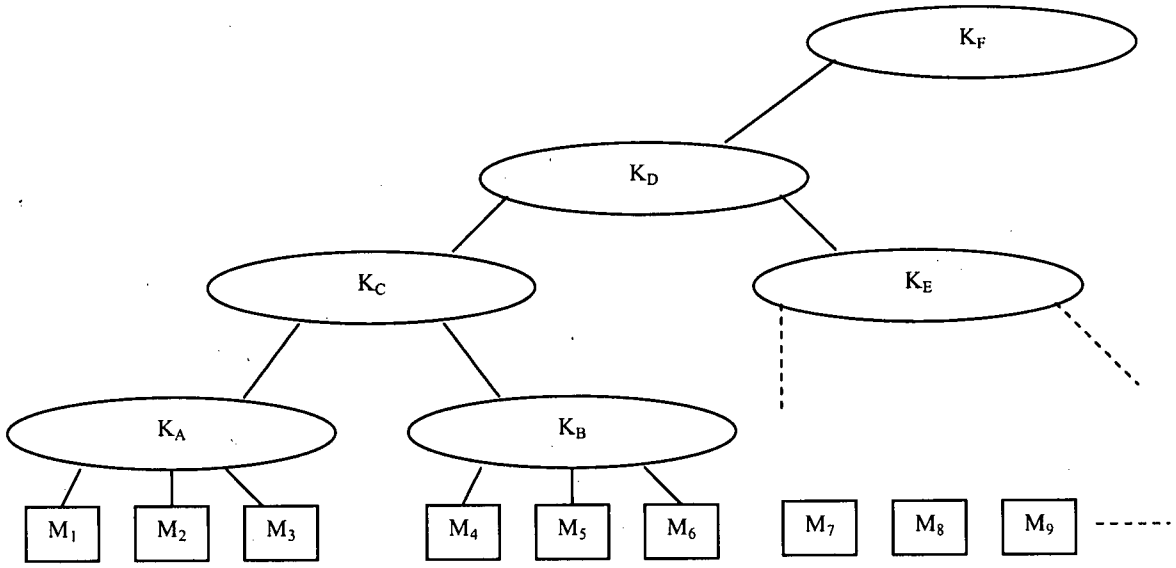

Figure 4.2 BBT with Members Equally Distributed in the Leaf Level

Summaries

Our analysis for Scenario 2: Balanced Binary Tree (BBT) with Members Equally Distributed among Groups in the leaf level, is summarized in Table 4.3 and Table 4.4.

Messages:	$O(N + b)$
Rounds:	$O(\lg N/b + \lg b)$
Broadcasts:	$O(b)$

Table 4.3 Summary for IKA of Scenario 2

Messages:	$O(N/b + \lg b)$
Rounds:	$O(\lg N/b + \lg b)$
Broadcasts:	$O(\lg b)$

Table 4.4 Summary for SKR of Scenario 2

General Analysis for Scenario 2

As we have assumed all members are evenly distributed among all groups in leaf level, we can do a more thorough analysis by looking at the growth of several key characteristics as the number of members increases.

For a 3-level BBT ($b = 8$), we plot N against ($N/b + \lg b$), i.e., *total members* vs. *total messages*, in Figure 4.3.

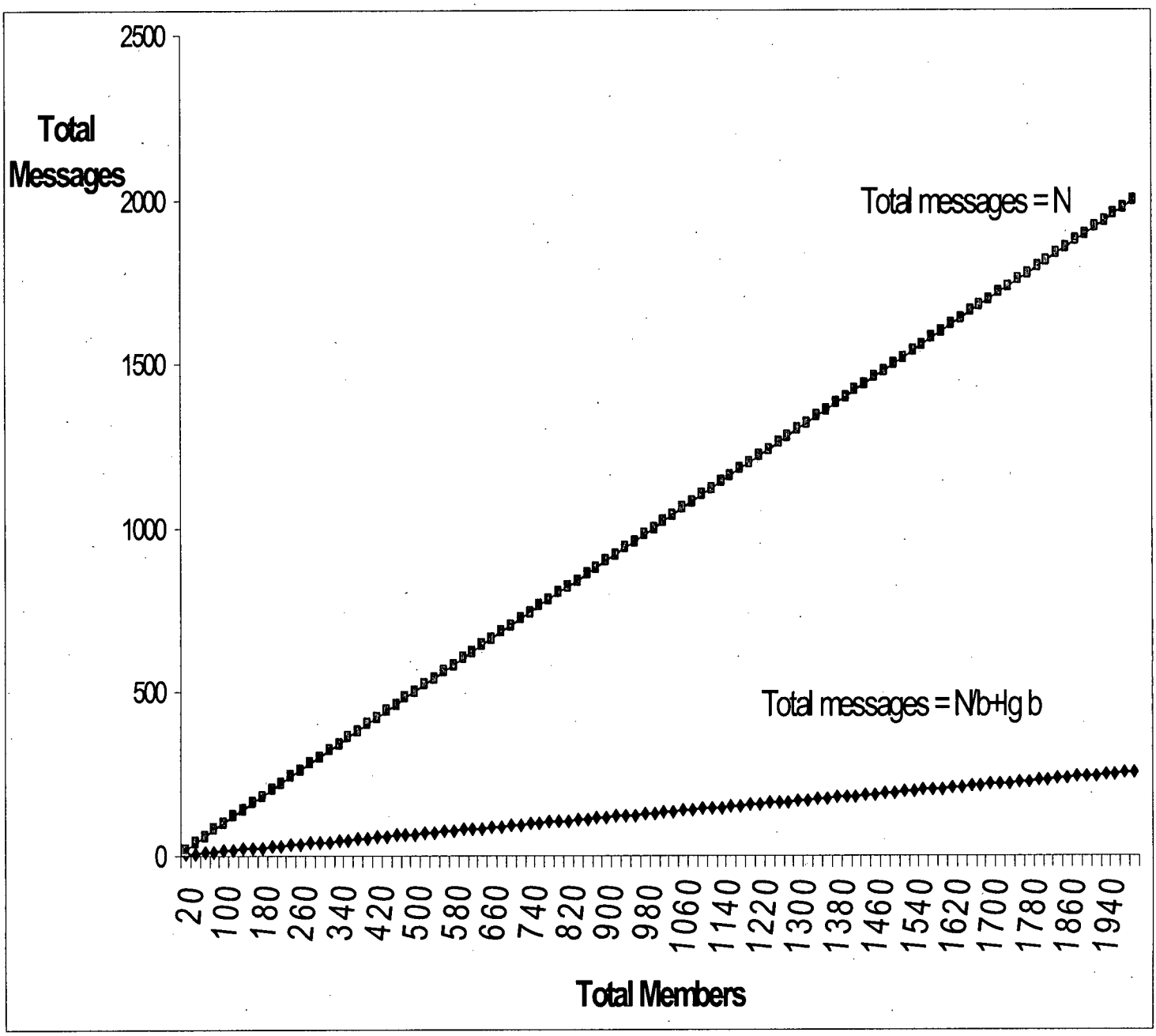

Figure 4.3 Total Members vs. Total Messages

We can clearly see a linear relationship in Figure 4.3, but it has a slope significantly less than 1, which indicates that $O(N/b + \lg b)$ grows considerably slower than O(n). Actually, for $b > 1$, $O(N/b + \lg b) < O(N)$.

Let us take a look at the number of rounds. In Figure 4.4, we have a graph showing N plotted against (lg (N/b) + lg b), i.e., *total members* vs. *total rounds*. In this graph, we can see that the total number of rounds is optimized as N gets larger.

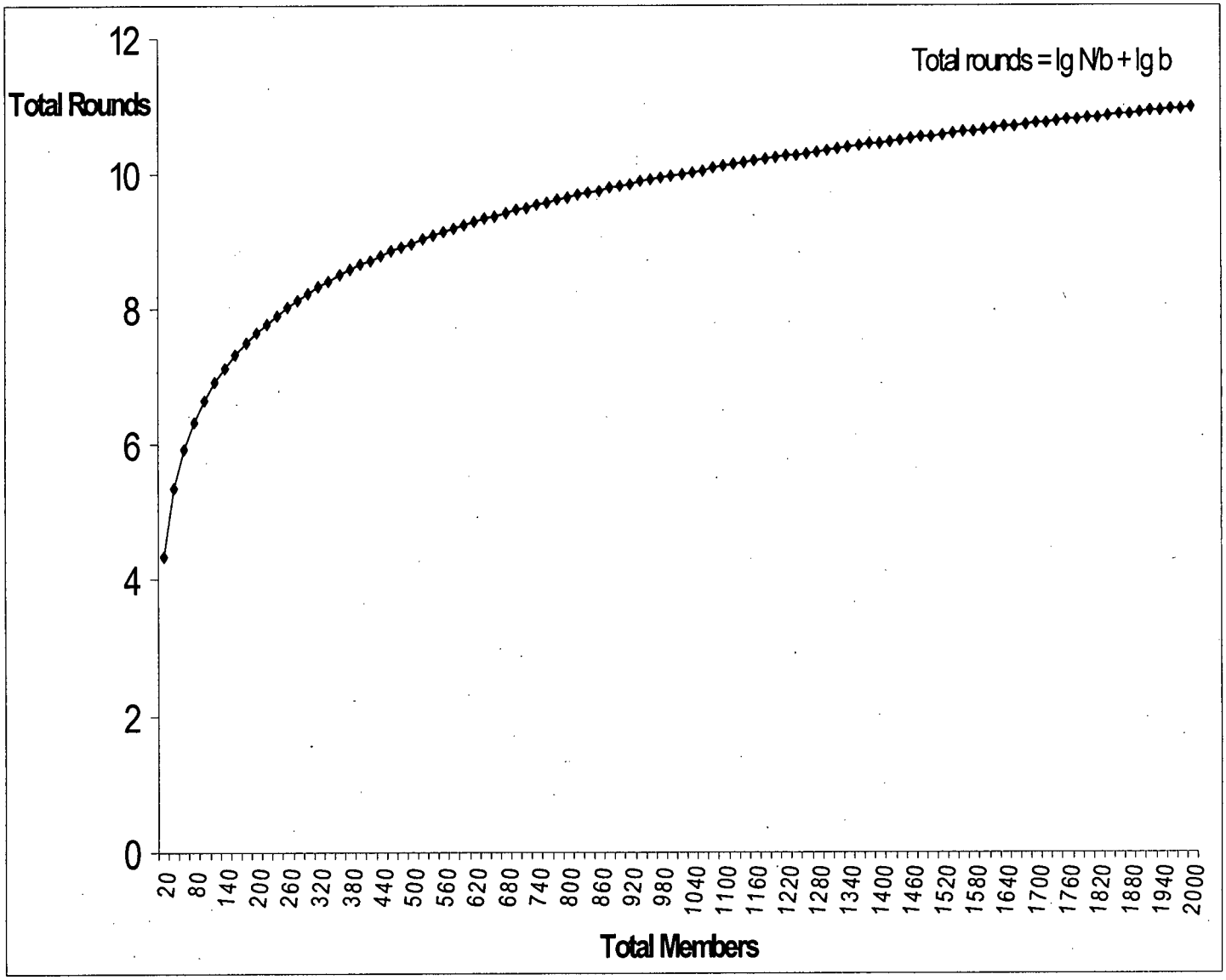

Figure 4.4 Total Members vs. Total Rounds

Let us look at the effect of the number of levels. If we fix the total number of members and vary the number of levels of the BBT, we can analyze how the number of levels affects the total messages. Say for a fixed total number of members $N = 20000$, we plot $(N/b + \lg b)$ against b, i.e., *total messages* vs. *number of levels*, as in Figure 4.5.

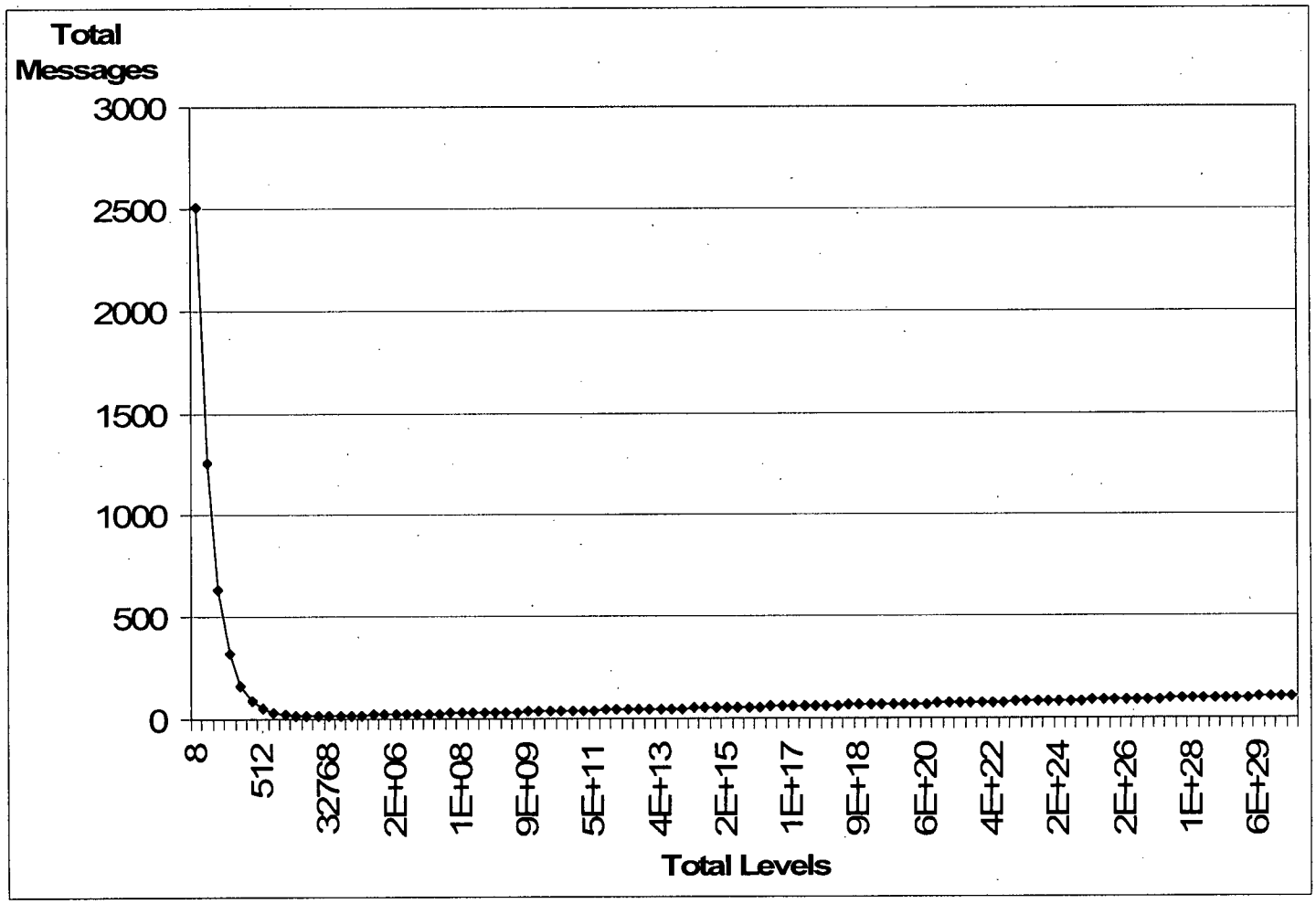

Figure 4.5 Total Messages vs. Number of Levels

With the total number of members fixed and the number of levels increasing, we can see that there is a drastic decrease in the total number of messages. However, this number will be get optimized pretty quickly, and further increases in the number of levels will not help, but will actually increase the number of messages. In fact, it is unrealistic to have the number of levels approaching or exceeding the total number of members.

Chapter 5

Implementation

5.1 Overview

To test our research in a more realistic environment, we have implemented our approach, using Java, in a media-sharing application where users can securely share their media data, such as pictures and homemade video, with each other. As in our research approach, users can create different levels of hierarchy and media can be shared with different groups of users using different levels of security.

This application offer basic features for users to select, organize and display media content. Besides these basic functionalities, it integrates with a peer-to-peer (P2P) communication module that is able to communicate and share media content with other users in the model described in this research.

5.2 User Interface

Figure 5.1 shows the main menu of the application interface. The application interface is organized similar to other media organization software, where users can add existing media content into the program and do some basic organization. The application is intended to run in full screen like other common media organization software. The main menu provides a list of several tools and major operations, such as Photo Album, File

Browser, Settings, Exiting, etc. Users can use the mouse cursor to highlight and initiate the selected tools or operations.

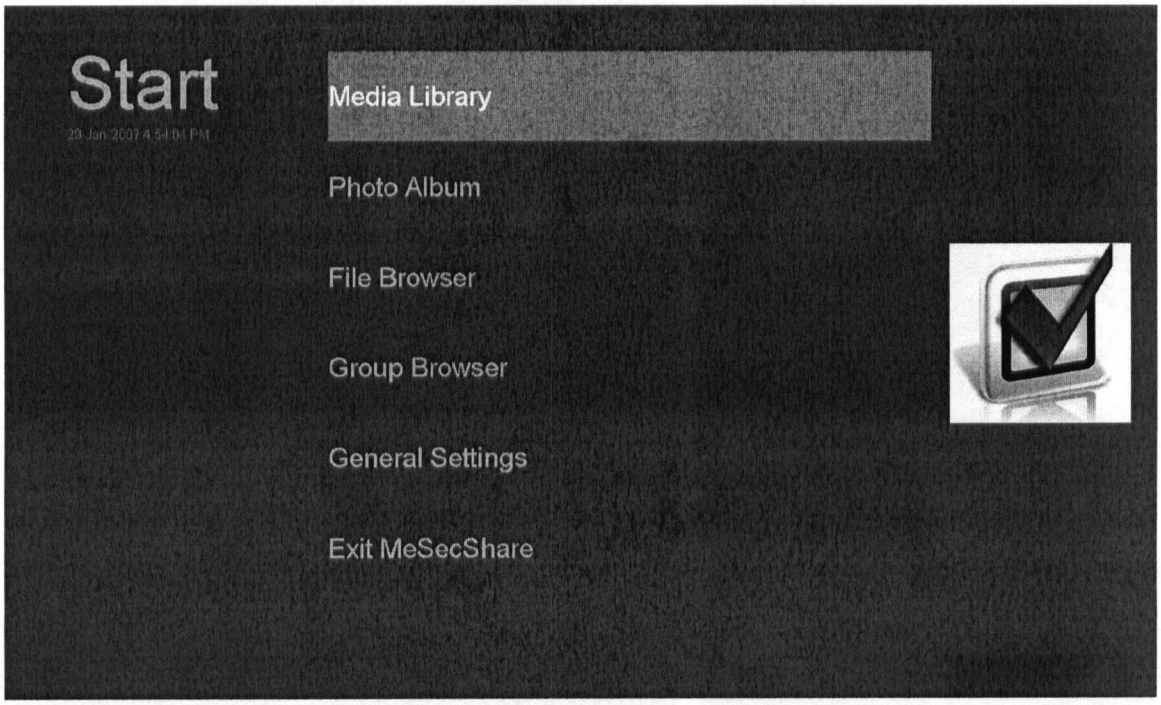

Figure 5.1 Application Main Menu

After a particular tool or operation is initiated, depending on the tool or operation, the interface usually rearranges itself into 2 or 3 horizontally aligned panels. The left panel features a list of available operations; the centre panel is the main display panel for the tools; and the optional right panel contains additional information.

5.2.1 File Browser

File Browser tool can be selected from the main menu, and is provided for users to search and select media data. When they find the specific media data, they can add them to media organization tools such as Media Library or Photo Album.

The first time the File Browser tool is started, it displays the file listing of the user's file system root. From there, users may navigate to other parts of the file system and select

their favour media data files. Figure 5.2 shows the File Browser tool as it first starts, with a typical file system root in Microsoft Windows.

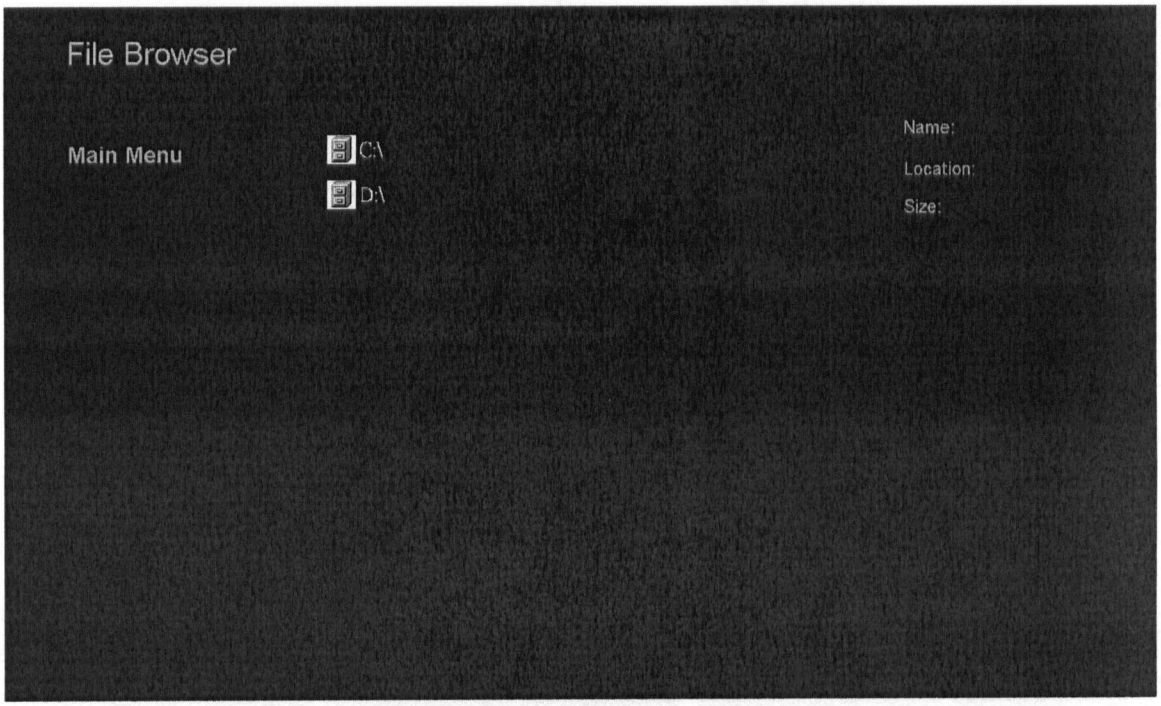

Figure 5.2 File Browser Showing the C and D Drives of a Windows File System

When the users navigate through the file system and find a favourite media data file or folder, they can highlight the particular file or folder and select the "Add to..." operation from the left panel; a popup window will ask where the highlighted content should be added to. A list of media organization tools is provided for user to choose from and they can select the appropriate tool to handle the media type they have selected. For example, in Figure 5.3, the user finds the "My Pictures" folder containing some favourite picture files, and wishes to add them to the Photo Album tool. They then highlight the "My Pictures" folder and select the "Add to..." operation from the left panel. A popup window appears and the user selects "Photo Album" as the destination. "My Pictures" will then be added to "Photo Album".

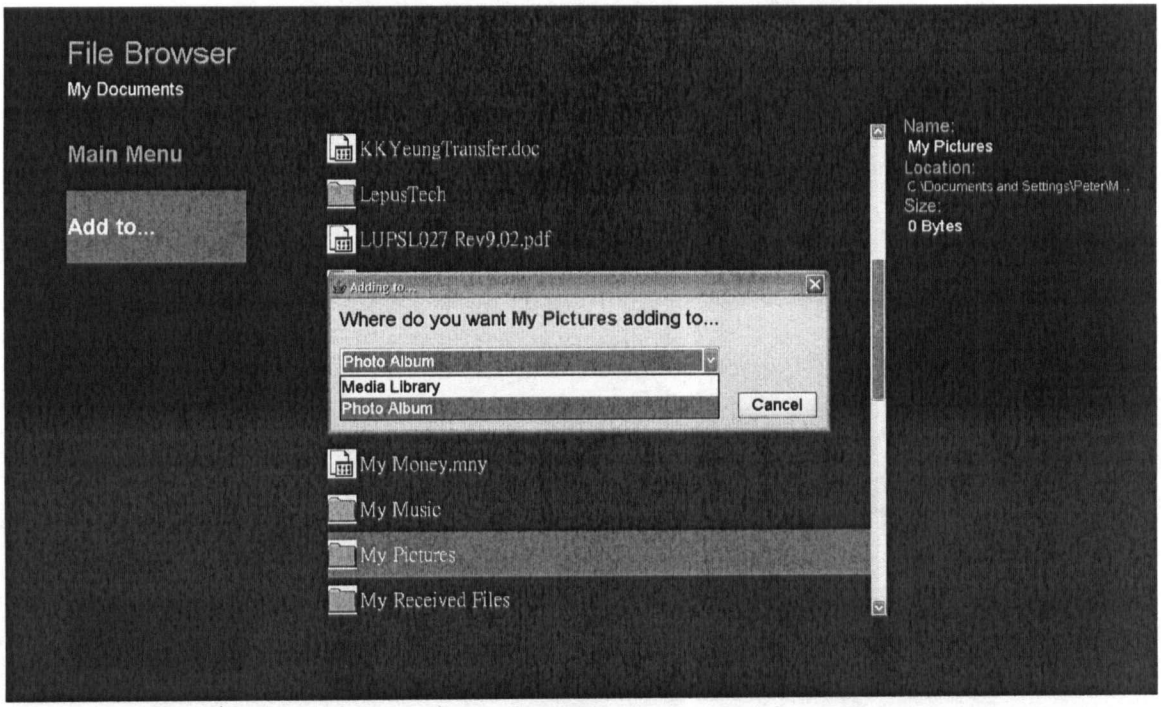

Figure 5.3 Adding Picture Folder to Photo Album in File Browser

5.2.2 Media Organization Tools

The application can accommodate several different media organization tools to properly handle and organize different types of media data. We have implemented a Photo Album tool to handle and organize the various picture types of media files.

In the last section, we mentioned that users are able to select their favourite media files or folders and add them to the appropriate media organization tool. In the Photo Album tool, each folder added by users through the File Browser tool will become an album. All of the added albums are listed in the initial page of the Photo Album tool. Users can select and open the specific album and view the media content of that album. Figure 5.4 shows the pictures files in the "My Pictures" album.

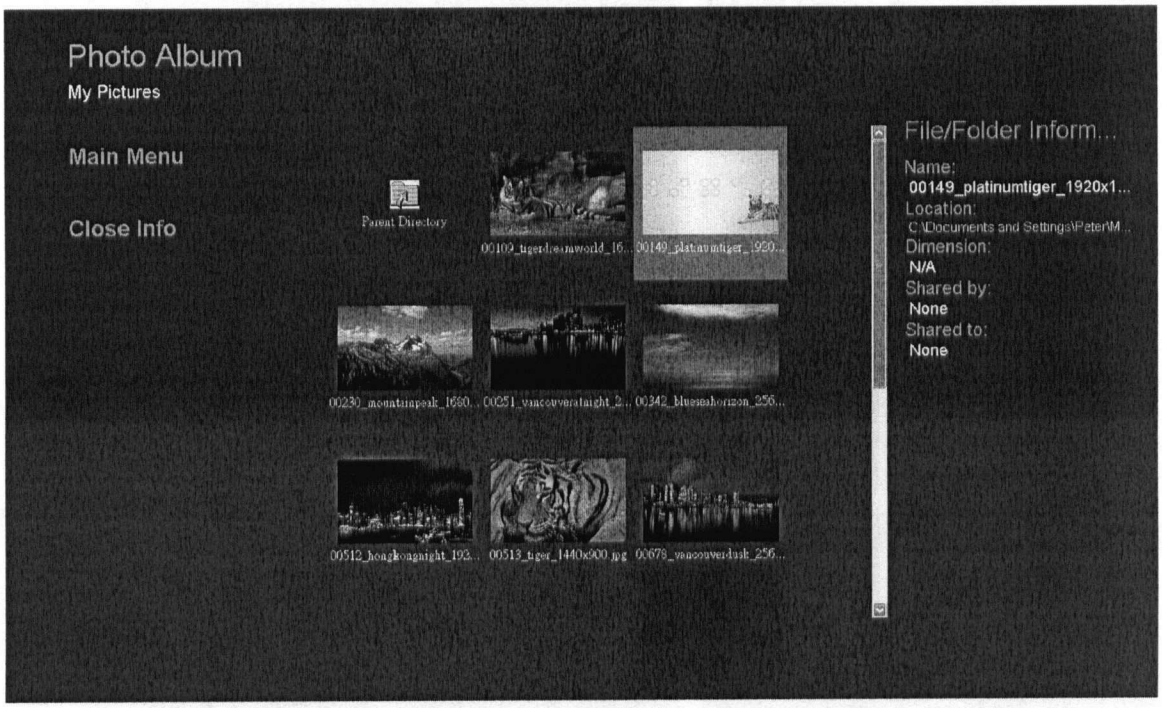

Figure 5.4 Photo Album Tool

5.2.3 Group Browser

The Group Browser tool is the portal to the secure P2P communication and media-sharing features of our application. Its main purpose is to give users an interface to browse through the groups, peers and their shared media resources. It also provides functionalities to manipulate hierarchy, group and peer relationships, which is an implementation of our research approach.

If the user has not created a group and has not been asked to join a group, the Group Browser offers them a choice to create a new group, as shown in Figure 5.5. After the user has created a new group, they will automatically become the manager of the new group. The group manager has the capability of inviting other peers to join its group, so that a group with multiple members can be formed. Figure 5.6 shows a group manager inviting a new peer to join his or her group. Neighbours, which are peers that have

established secure relationships, can then start to securely share media content with each other.

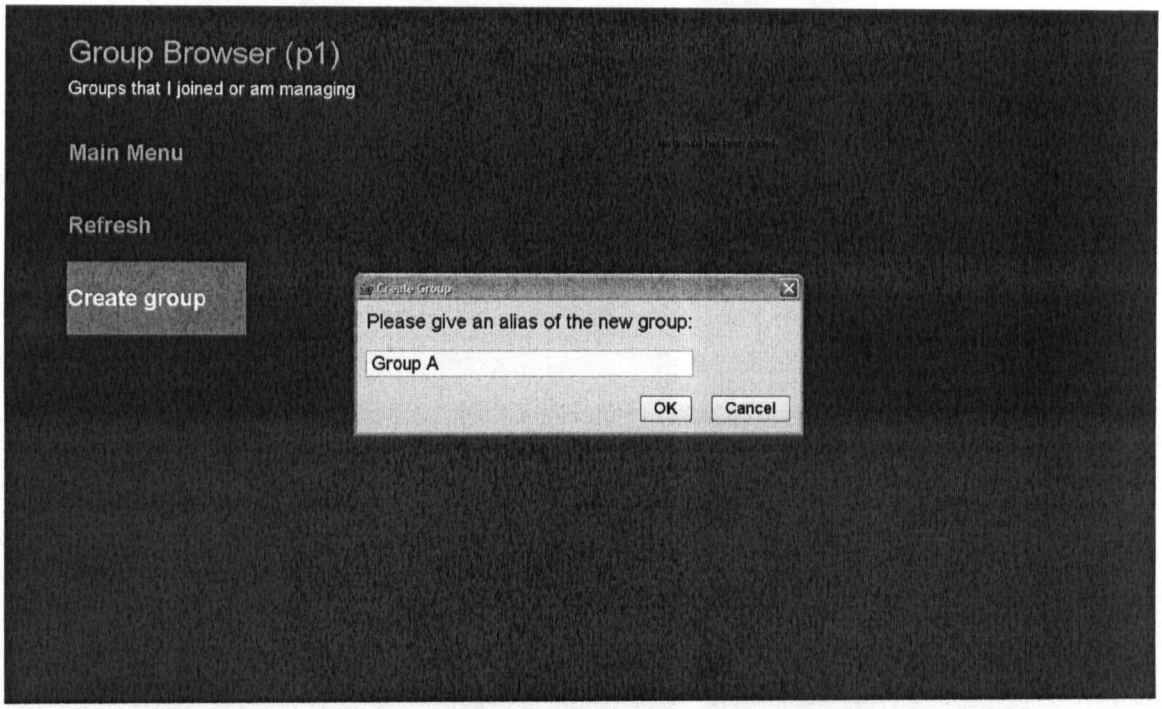

Figure 5.5 Group Creation in Group Browser

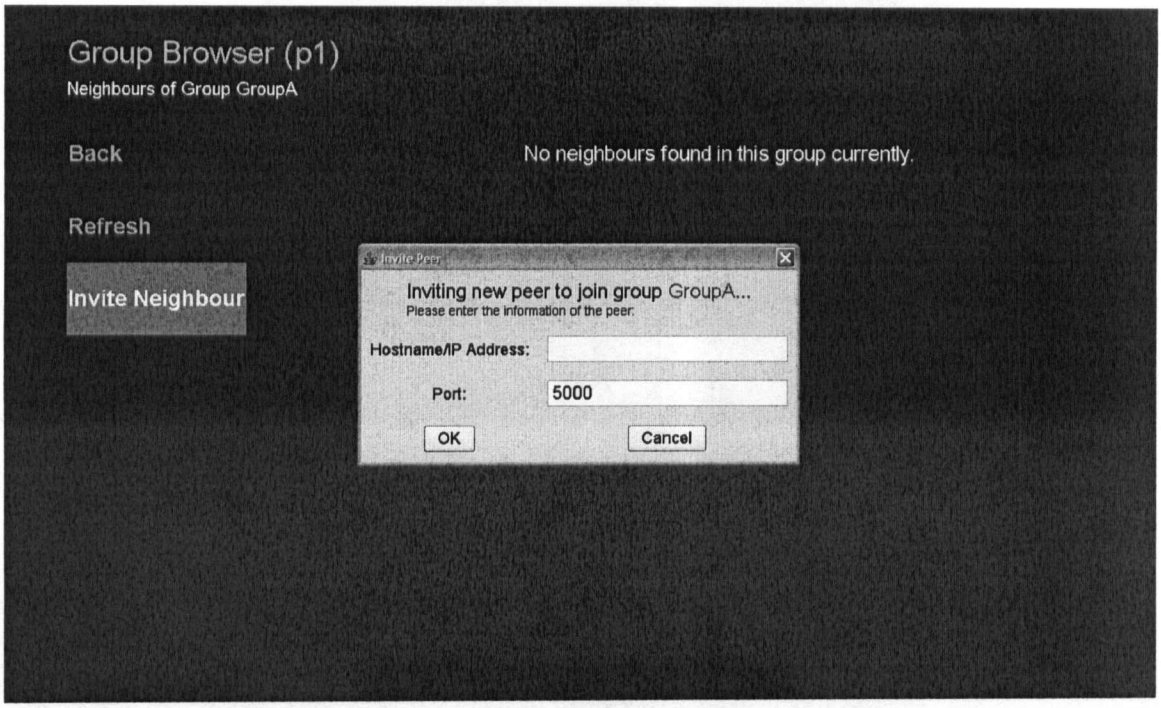

Figure 5.6 Inviting Neighbour in Group Browser

Besides inviting peers, the group manager is also able to invite other groups to join as its group's child group. This process establishes the hierarchy of group relationships. The child group's members then have the right to communicate and share media content with the parent group's member at the secure level of the parent group. We discuss the group relationships and hierarchy in the next section.

The Group Browser has 3 basic views: group, neighbour and resource. In group view, users can browse the groups they manage or join (called principal groups), and other groups that have relationships (called related groups). By clicking on a specific group, users can access the neighbour view in which they can see all of the neighbours within that specific group. By clicking on a specific neighbour, users go into the resource view, which shows all of the resources that this particular neighbour is sharing. From this view, users can add other neighbours' shared resources into their own media organization tools. We discuss this procedure in more detail in Section 5.4.

5.3 Secure P2P Communication

This media-sharing application has an implementation of our research approach to create a secure P2P communication environment so that neighbours can communicate and share media content securely. We now look at this implementation in more detail and explain how the secure P2P Communication works in this application.

5.3.1 Group Diffie-Hellman Implementation

All groups will perform a group Diffie-Hellman (GDH) key agreement to create a group key, which is used in the secure communication between neighbours. After a certain period of time, which is an application setting, the system will do a key refreshing to create a new key. The key agreement is implemented using the Java Security package with the Java Cryptography Extension (JCE). We have evaluated several optimizations of GDH [7] [8] [9] [10] and their key-refreshing processes. We did not implement all of the optimizations, however, as the performance of GDH within a single group is not a major measurement of our research approach. Rather, we concentrate on how well the hierarchy groups work, and their practicality.

5.3.2 Group and Group Manager

A group is formed when a peer starts a new group. The peer who creates the new group will automatically become the group manager. At the application level, the group manager is responsible for the general administration of the group. Therefore, the group manager has the following privileges:

1. Naming the group
2. Inviting another peer to join the group
3. Inviting another group to become the group's child group

Other privileges that the group manager should have but that have not been implemented as of the latest version, include the following:

1. Renaming the group
2. Removing neighbour from the group
3. Removing child group
4. Destroying the group

At the secure communication level, the group manager has the same responsibility as the group leader as discussed in Section 3 regarding the design of our research approach and the algorithms involved. In general, the group manager has the following responsibilities:

1. Initiating the Initial Key Agreement (IKA) process
2. Expiring the old key and initiating the Secondary Key Refreshing (SKR) process
3. Taking part in the key-refreshing process of other groups
4. Encrypting any new parent group keys and distributing them to the group members

5.3.3 Neighbours

A neighbour is a peer with whom a secure connection has been established. It could be another member of the group the user belongs to, or of the group that has a relationship with the group the user belongs to. In other words, a neighbour should have one or more group keys associated with it that the user can use to securely communicate with it.

A group key is first generated by performing the IKA within the specific group. The group manager has the privilege of inviting a new peer to join the group. When the group manager invites the first member, an IKA occurs between the group manager and the new peer, and a new key is generated for the group. Communication between the group manager and the new peer, now a neighbour, is secured. The group manager can subsequently invite other peers to join. SKR occurs every time a new peer joins, and an updated group key is generated.

5.3.4 Group Relationships and Group Hierarchy

A group relationship is established when a group manager invites another group to join as a child group. One group manager can send an invitation to another group manager to join. The invitation would contain the group name of the invitee group. The other group manager can accept the invitation only if the invitee group is managed by it.

When the group relationship is being established, the parent group manager will initiate the SKR of the parent group to refresh the parent group key. The child group manager will get involved in the SKR on behalf of the child group. When the new parent group key is generated, the child group manager will encrypt it by the current child group key and distribute it to the group members. The group members are then able to communicate with the parent group's members using the new parent group key.

Each group manager is free to establish parent-child relationships with other groups. Hence, a group hierarchy is formed. The SKR process is carried out as discussed in Section 3.

5.4 Media Content Sharing

Sharing media content is simple, once neighbours and group relationships are properly set up, as described in the last section. Let us look at a simple example of five peers forming two groups where these 2 groups have a parent-child relationship. These relationships are shown in Figure 5.7.

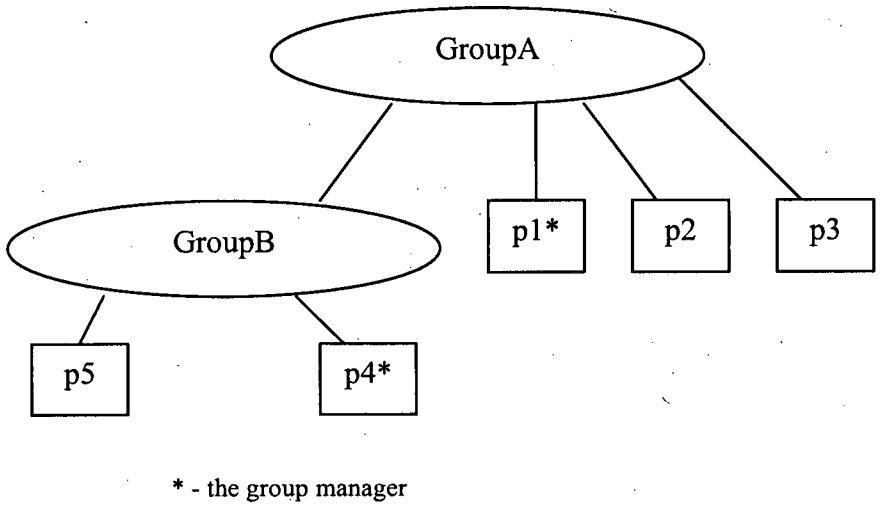

Figure 5.7 Group Relationships Example

Figure 5.7 shows the neighbours and group relationships in our five-peer example. Neighbours *p1* and *p4* are the group managers of *groupA* and *groupB* respectively. Neighbours *p2* and *p3* are members of *groupA*, and neighbour *p5* is member of *groupB*. *GroupA* is the parent group of *groupB*; to see it another way, *groupB* is the child group of *groupB*.

In our example, Figure 5.8 shows the group browser of neighbour *p1* using group view. The view presents *p1* a manager of *groupA*, and *groupA* having a child group called *groupB*. All group names are clickable in this view. By clicking on a group name, we go into the neighbour view of that particular group. Figure 5.9 shows the neighbour view of *groupA*. We can see that neighbours *p2* and *p3* are shown as the neighbours of *p1* in *groupA*. Note that the peer would not see itself listed in its own neighbour view.

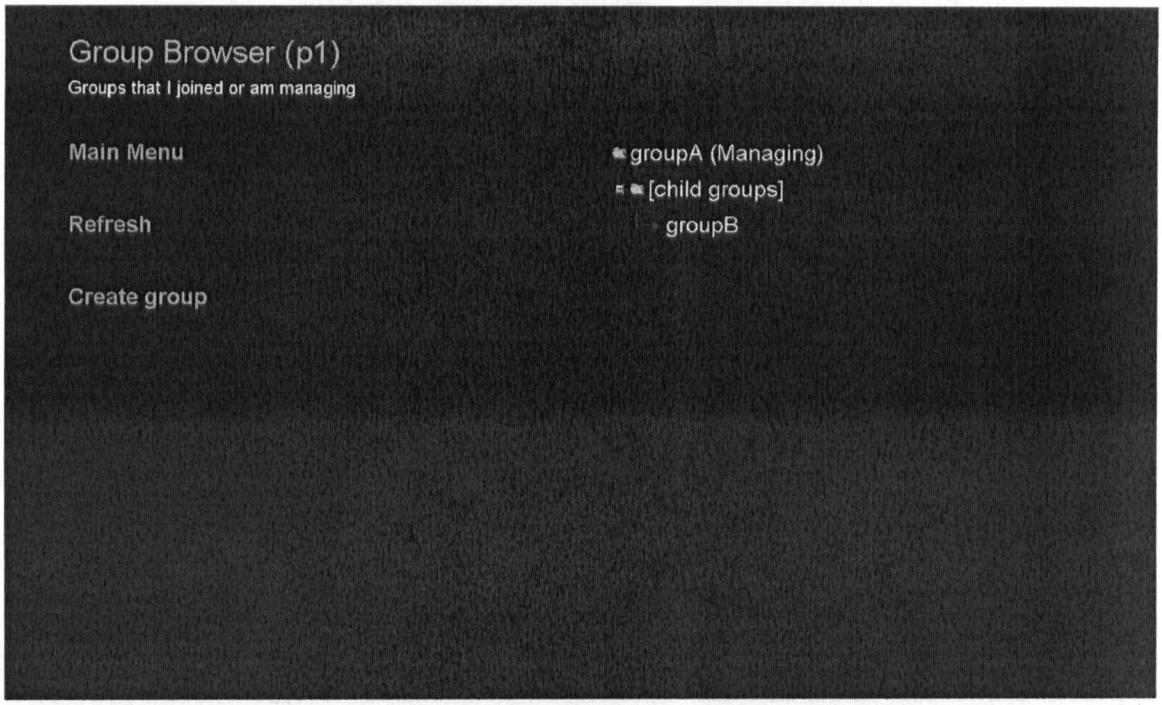

Figure 5.8 Group View of Group Browser of Neighbour *p1*

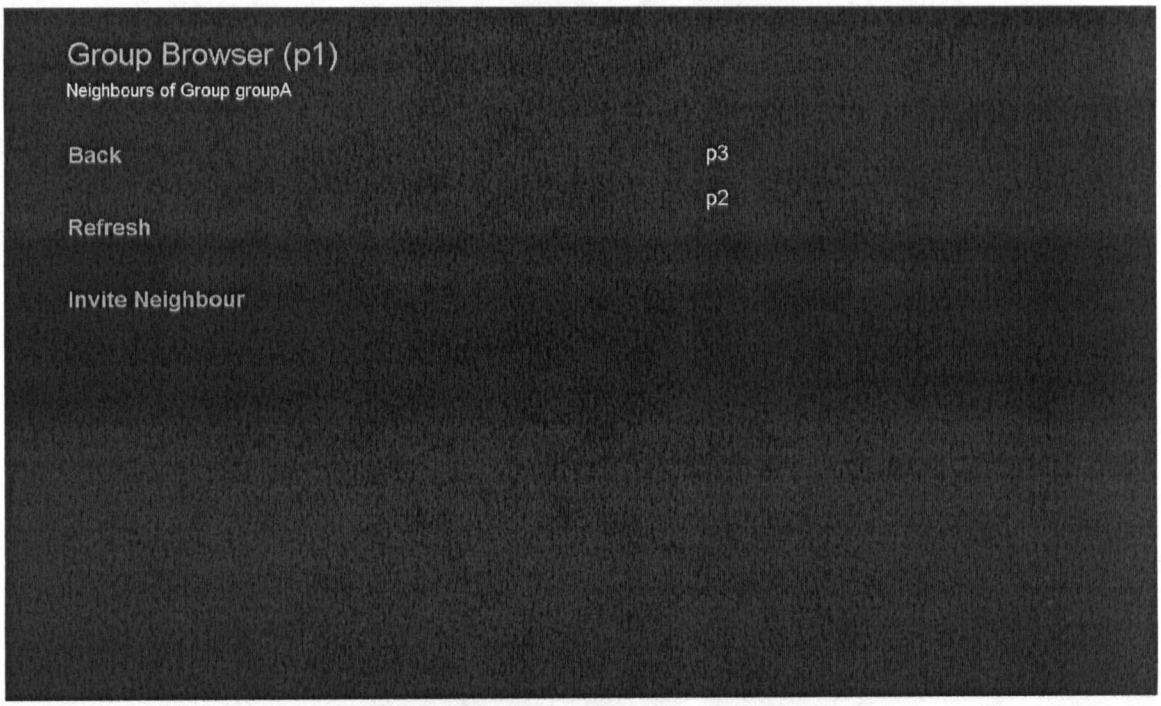

Figure 5.9 Neighbour View of *groupA* from *p1*

Each neighbour is able to share its added media resource with other neighbours. Adding a media resource means that the media content has been added to one of the media organization tools. When sharing the media content, a neighbour can pick either the group that it creates or joins (principal group), or any other group that has a relationship with his principal group (related group).

Let us return to our example and go to neighbour *p2*. Figure 5.10 shows the Photo Album tool of neighbour p2. Neighbour *p2* has an album called "testAlbum" added to his or her Photo Album tool and wishes to share this album with other neighbours. A popup window is displayed to let the user to pick one of the principal or related groups of *p2* where the media resource is to be shared on that group security level. Let us select *groupA*, the principal group of *p2*, in the popup window and press the "OK" button. *The* Photo album "testAlbum" is marked to be shared, as shown in Figure 5.11. The album information panel on the left shows the photo album is shared to *groupA*.

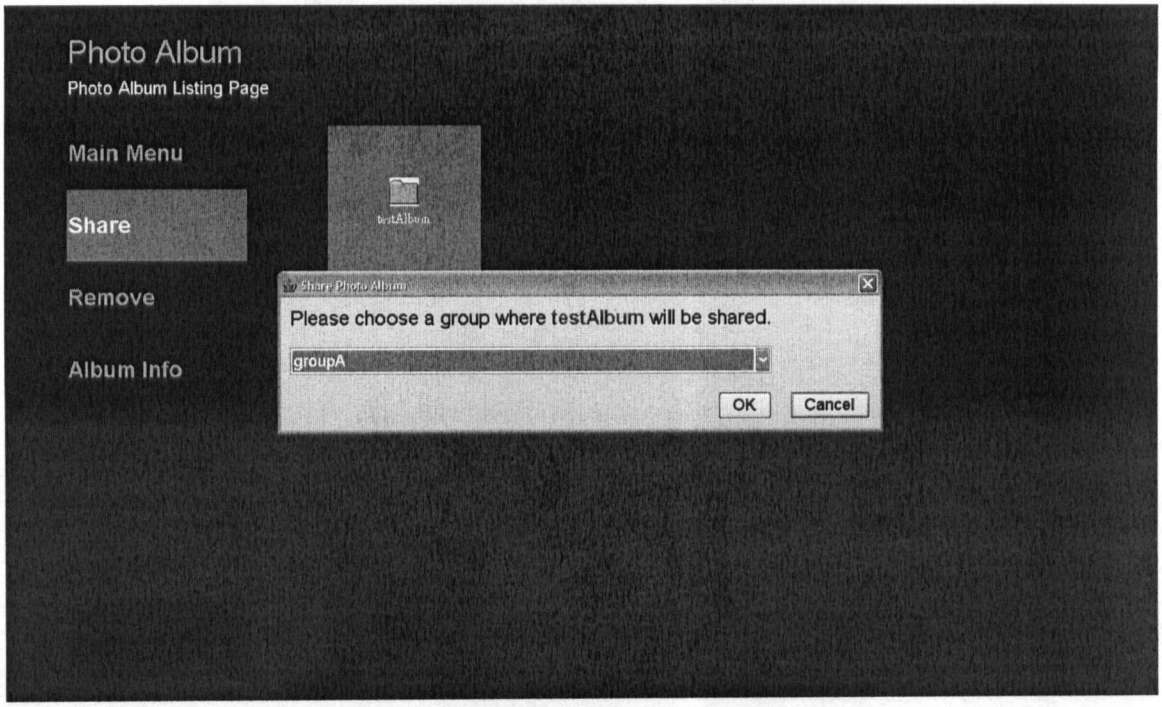

Figure 5.10 Neighbour *p2* sharing a photo album with *groupA*

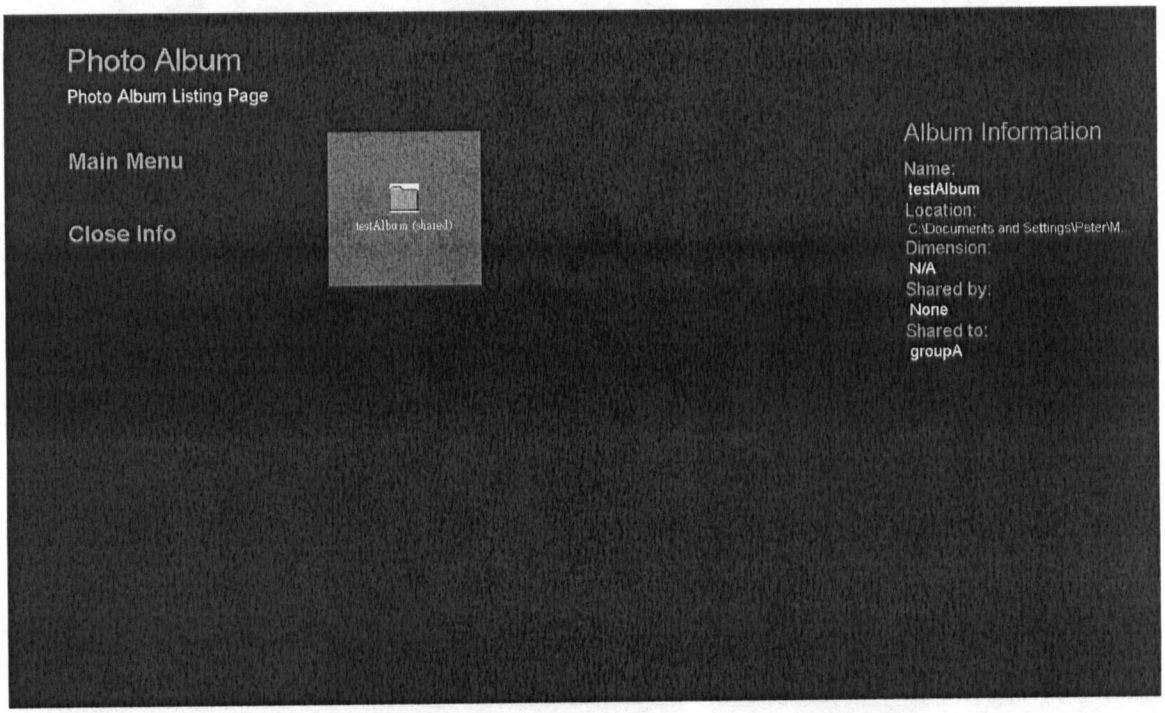

Figure 5.11 Photo Album "testAlbum" is marked as "shared"

After neighbour *p2* shares its photo album, the application will automatically notify other (related) neighbours of its new shared resource. We now go back to neighbour *p1* in Figure 5.12, which shows the resource view of *p1* viewing its neighbour *p2*. We can see that *p2* is sharing a photo album resource item called "testAlbum".

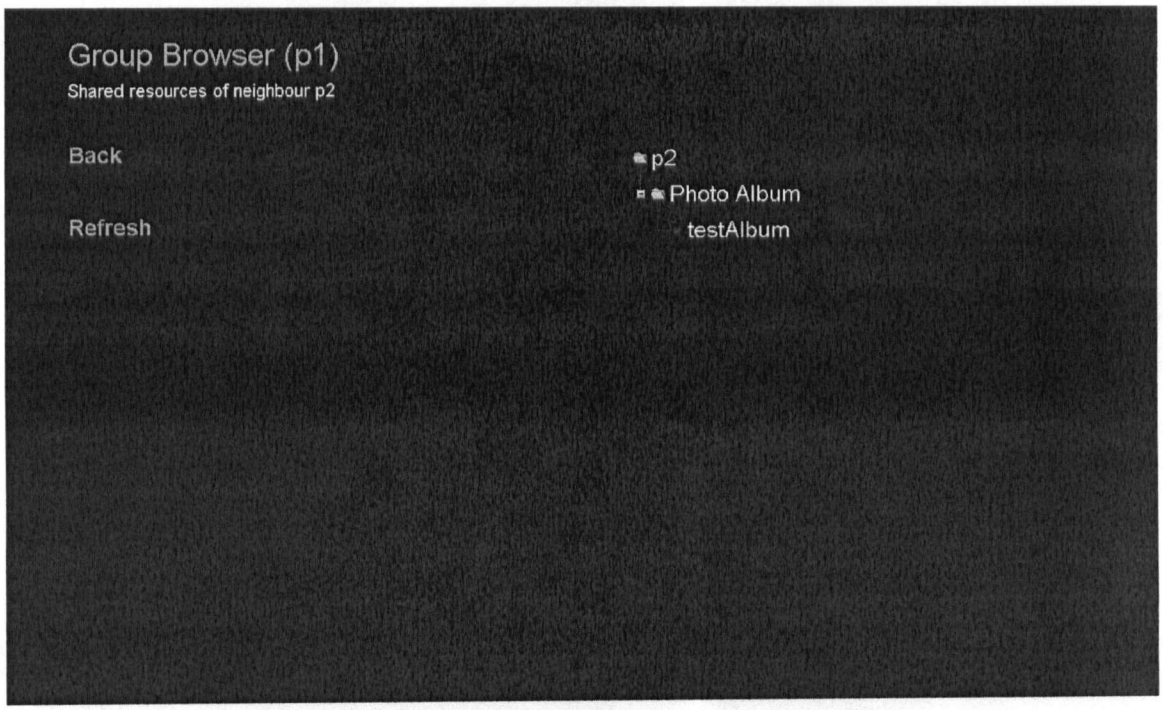

Figure 5.12 Resource View of *p1* viewing *p2*

Neighbour *p1* can now add *p2*'s "testAlbum" to its own photo album. By highlighting the shared resources "testAlbum", users will get a "Take Resource" operation on the left panel. They can then use this operation to add the "testAlbum" to their own Photo Album tool. Figure 5.13 shows this operation.

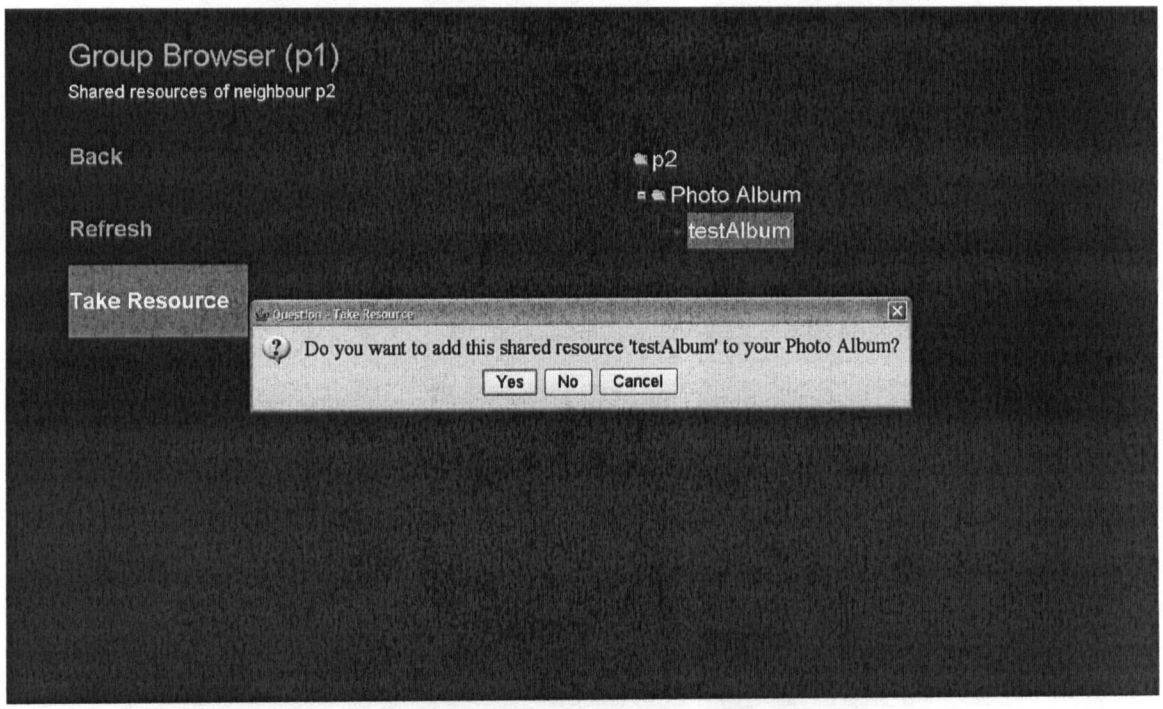

Figure 5.13 "Take Resource" operation in Group Browser

In the Photo Album tool of *p1* shown in Figure 5.14, we can see that the photo album "testAlbum" of neighbour *p1* has now been added to the Photo Album tool of neighbour *p2*. The right information panel indicates that the album is shared by *p2*. The content of the shared photo album will begin to transfer to *p1* as soon as the album is first shared using the security level of *groupA*.

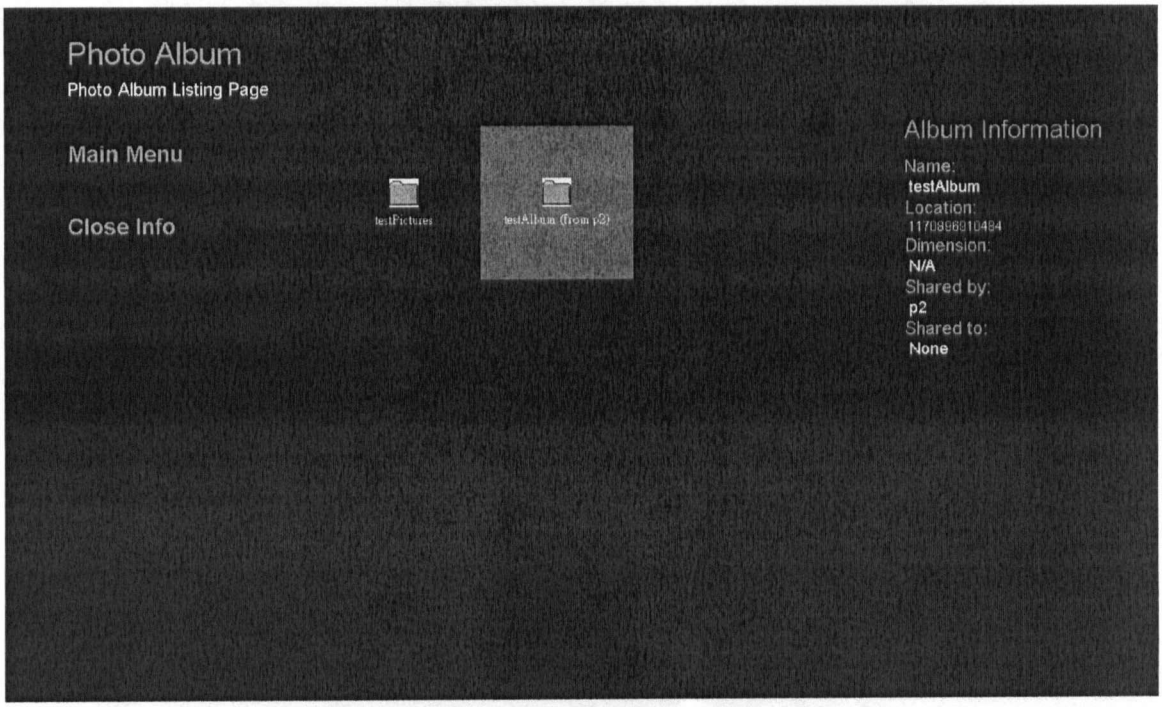

Figure 5.14 The Photo Album tool of *p1* after adding *p2*'s shared album

Chapter 6

Conclusions and Future Works

6.1 Conclusions

There is a trend toward increased use of peer to peer applications. One can easily foresee that peer-to-peer communication will become as mainstream a communication model as the server-client model because of its advantages. However, the security of peer-to-peer communication has not matured sufficiently to warrant the trust of the majority of Internet users.

Security requirements for the peer-to-peer environment present quite a number of interesting research challenges. Key management, which is the heart of the security process, plays an important role in this research area. We propose a functional key management model for a collaborative peer-to-peer environment, focusing on reducing the number of affected peers in the secondary key refreshing process, in order to make the environment more scalable. The key management model also introduces the idea of a hierarchy of security levels, which makes the environment more practical and realistic.

Our research approach is inspired by both the Logical Key Hierarchy (LKH) and the Distributed Sub-group (DS) concepts. We take advantages of both concepts, and merge them into a model that minimizes the drawbacks of the two concepts. Our approach takes the advantage of data tree structure of LKH that minimize size, storage and computational cost, and organizes the peers into sub-groups, which distributes the key management works to multiple sub-group managers and avoids the single point of failure

problem. The localization of the key refreshing also minimizes the overhead of the key-refreshing process.

We have also implemented a media-sharing application using our research approach, where users can securely share their media content with each other in a group hierarchy environment using different security levels. The application is a partial proof-of-concept implementation, which shows that our approach is practical, and demonstrates the usefulness of grouping and levels of security.

6.2 Future Works

Chapter 3 describes our proposed new design and its algorithms, using examples and descriptive language. The algorithms can be formalized, and some protocol verification can be done.

The security of the Group Diffie-Hellman problem has been verified and proved to be a computational hard problem [16]. Our design is based on the Group Diffie-Hellman-based protocol for creating a group key for each individual group, so the security of our protocol has the same level of security. However, there are some variants that might affect the integrity of the security, for example, re-using the old group key to contribute to the new group key in the higher level, broadcasting the new group key, etc. Each of these variants can be more thoroughly analyzed in the future.

We have made a few assumptions when analyzing the algorithms in Chapter 5. As these assumptions create some ideal situations, the efficiency of the proposed protocol might in practice be slightly different from the data. While the real environment might never be able to be described and theoretically analyzed, there are a few other practical scenarios that more closely resemble the real environment that may be interesting to analyze further.

The media-sharing application implemented in this research can be potentially improved in quite a few areas. For example, implementing other media tools besides Photo Album, a better implementation of the GDH-based protocol, and a relational database to make the application more scalable, etc., might be explored.

Statistics could be collected from practical usage of the application. These statistics would help in analyzing the protocol from a practical point of view, to complement the theoretical analysis done in this research.

Bibliography

[1] W. Diffie and M. E. Hellman. New Directions in Cryptography, IEEE Transactions on Information Theory, IT-22(6):644-654, November 1976.

[2] E. Rescorla. Diffie-Hellman Key Agreement Method, RFC 2631, IETF, June 1999.

[3] B. Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks, IEEE Communications, 32(9):33-38. September 1994.

[4] John T. Kohl and B. Clifford Neuman. The Kerberos Network Authentication Service (V5), RFC 1510, IETF, September 1993.

[5] H. Harney and C. Muckenhirn. Group Key management Protocol (GKMP) Architecture, RFC 2094, IETF, July 1997.

[6] H. Harney and C. Muckenhirn. Group Key management Protocol (GKMP) Specification, RFC 2093, IETF, July 1997.

[7] M. Steiner, G. Taudik, and M. Waidner. Cliques: A New Approach to Group Key Agreement, Technical Report RZ 2984, IBM Research, December 1997.

[8] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman Key Distribution Extended to Group Communication, SIGSAC: 3rd ACM Conference on Computer and Communications Security, pages 31-37. ACM SIGSAC, 1996.

[9] M. Steiner, G. Tsudik, and M. Waidner. Key Agreement in Dynamic Peer Groups, IEEE Transactions on Parallel and Distributed Systems, March 2000.

[10] K. Becker and U. Wille. Communication Complexity of Group Key Distribution, 5th ACM Conference on Computer and Communications Security, San Francisco, California, November 1998, pp. 1-6, ACM Press.

[11] D. A. McGrew and A. T. Sherman. Key Establishment in Large Dynamic Groups Using One-Way Function Trees, Technical Report No. 0755, TIS Labs at Network Associates, Inc., Glenwood, MD, May 1998.

[12] Y. Kim, A. Perrig, and G. Tsudik. Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups, 7th ACM Conference in Computer and Communication Security 2000, pages 235-241, November 2000.

[13] I. Ingemarson, D. Tang, and C. Wong. A Conference Key Distribution System, IEEE Transactons on Information Theory, 28(5), 1982, 714-720.

[14] M. Burmeester, Y. Desmedt. A Secure and Efficient Conference Key Distribution System, Advances in Cryptology – EUROCRYPT'94, LNCS 950, Berlin: Springer 1994, pages 275-286.

[15] R. L. Rivest, A. Shamir, and L. M. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems, Communications of the ACM, 21(2):120-126, February 1978.

[16]] E. Bresson, O. Chevassut and D. Pointcheval. The Group Diffie-Hellman Problems, 9th Annual International Workshop on Selected Areas in Cryptography of the ACM, pages 325-338, 2002.

[17] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably Authenticated Group Diffie-Hellman Key Exchange, Proceedings of ACM CCS '01, pages 255-264. ACM Press, November 2001.

[18] E. Bresson, O. Chevassut, and D. Pointcheval. Provably Authenticated Group Diffie-Hellman Key Exchange – the Dynamic Case, Proceedings of Asiacrypt '01, volume 2248 of LNCS, pages 290-309. Springer-Verlag, December 2001.

[19] E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic group Diffie-Hellman Key Exchange under Standard Assumptions, Proceedings of Eurocrypt '02, volume 2332 of LNCS, pages 321-336. Springer-Verlag, May 2002.

[20] E. Bresson, O. Chevassut, and D. Pointcheval. Group Diffie-hellman Key Exchange Secure Against Dictionary Attacks, Proceedings of Asiacrypt '2002. Springer, December 2002.

[21] Y. Kim, A. Perrig, and G. Tsudik. Tree-based Group Key Agreement, ACM Transactions on Information and System Security, Vol. 7, Issue 1, pages 60-96, February 2004.

[22] X. Li, R. Yang, M. Gouda, and S. Lam. Batch Updates for Key Trees. Technical Report, University of Texas, Austin, September 2000.

[23] C. K. Wong, M. Gouda, and S. Lam. Secure Group Communications Using Key Graphs, Proceedings of SIGCOMM, September 1998.

[24] X. Li, Y. Yang, M.Gouda, and S. Lam. Batch Rekeying for Secure Group Communications, Proceedings of Tenth International World Wide Web Conference (WWW10), (Hong Kong, China), May 2001.

[25] Nelson Minar, Marc Hedlund, Clay Shirky, Tim O'Reilly, et. al. Peer-to-Peer: Harnessing the Power of Disruptive Technologies, edited by Andy Oram, O'Reilly Press, First Edition February 2001.

[26] Napster. Mar 2, 2007. Napster, March 2, 2007 <http://www.napster.com>

[27] KaZaA. Mar 2, 2007. KaZaA, March 2, 2007 <http://www.kazaa.com>

[28] BitTorrent. Mar 2, 2007. BitTorrent March 2, 2007 <http://www.bittorrent.com>

[29] M. Amnefelt and J. Svenningsson. Keso - A Scalable, Reliable and Secure Read/write Peer-to-peer File System, 2004.

[30] Skype. Mar 2, 2007, Skype March 2, 2007 <http://www.skype.com>

[31] S. Paul. Multicast on the Internet and its applications, Kluwer Academic Publishers, 1998.

[32] W. Trappe, J. Song, R. Poovendran, and K.J.R. Liu. Key Distribution for Secure Multimedia Multicasts via Data Embedding, Proc. IEEE ICASSP'01, pp. 1449-1452, May 2001.

[33] D.M. Wallner, E.J. Harder, and R.C. Agee. Key Management for Multicast: Issues and Architectures, Internet Draft Report, Sept. 1998.

[34] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner. The VersaKey Framework: Versatile Group Key Management, IEEE Journal on selected areas in communications, vol. 17, no. 9, pp. 1614–1631, Sep. 1999.

[35] A. Perrig, D. Song, and D. Tygar. ELK, A New Protocol for Efficient Large-group Key Distribution, Proc. IEEE Symposium on Security and Privacy, pp. 247 –262, 2001.

[36] S. Banerjee and B. Bhattacharjee. Scalable Secure Group Communication over IP Multicast, JSAC Special Issue on Network Support for Group Communication, vol. 20, no. 8, pp. 1511 –1527, Oct. 2002.

[37] R. Poovendran, S. Ahmed, S. Corson, and J. Baras. A Scalable Extension of Group Key Management Protocol, Technical Report TR 98-14, Institute for Systems Research, 1998.

[38] D.Wallner, E. Harder, and R. Agee. Key Management for Multicast: Issues and Architectures, RFC 2627, June 1999.

[39] M.Waldvogel, G. Caronni, D. Sun, N.Weiler, and B. Plattner. The VersaKey Framework: Versatile Group Key Management. IEEE Journal on Selected Areas in Communications (Special Issue on Middleware), 17(8):1614–1631, August 1999.

[40] G. Caronni, M. Waldvogel, D. Sunand, and B. Plattner. Efficient Security for Large and Dynamic Multicast Groups. Workshop on Enabling Technologies, (WETICE 98), IEEE Comp Society Press, 1998.

[41] D. Balenson, D. McGrew, and A. Sherman. Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization, IETF Internet draft, August 2000.

[42] M. Burmester and Y. Desmedt. A Secure and Scalable Group Key Exchange System, Information Processing Letters, 94(3), pp. 137-143, 2005.

[43] S. Rafaeli. A Decentralized Architecture for Group Key Management. Computing Department, Lancaster University, September 2000.

[44] S. Mittra. Iolus: A Framework for Scalable Secure Multicasting, ACM SIGCOMM, volume 27,4 of Computer Communication Review, pages 277–288, ACM Press, New York, September 1997.

[45] T. Hardjono, B. Cain, and I. Monga. Intra-Domain Group Key Management Protocol, IETF Internet draft, September 2000.