

A Key-chain Based Keying Scheme For Many-to-Many Secure Group Communication

Dijiang Huang, Deep Medhi
University of Missouri–Kansas City

We propose a novel secure group keying scheme using *hash chain* for *many-to-many* secure group communication. This scheme requires a *key predistribution center* to generate multiple hash chains and allocates exactly one hash value from each chain to a group member. A group member can use its allocated hash values (secrets) to generate group and subgroup keys. Key distribution can be either offline or online via the key distribution protocol. Once keys are distributed, this scheme enables a group member to communicate with any possible subgroups *without* help of the key distribution center, and without having to leave the overall group, thus avoiding any set-up delay. Our scheme is suitable for applications where the population of a system is stable, group size is moderate, subgroup formation is frequent, and the application is delay sensitive. Through analysis, we present effectiveness of our approach.

Categories and Subject Descriptors: C.2.0 [COMPUTER-COMMUNICATION NETWORKS]: General—*Security and protection*; C.2.4 [COMPUTER-COMMUNICATION NETWORKS]: Distributed Systems —*Distributed applications*; G.2.1 [DISCRETE MATHEMATICS]: Combinatorics—*Permutations and combinations*; G.2.2 [DISCRETE MATHEMATICS]: Graph Theory—*Path and circuit problems*

General Terms: Security, theory

Additional Key Words and Phrases: Hash chain, key chain, secure group communication, many-to-many secure group communication

1. INTRODUCTION

Group communication applications can be broadly classified into two types: *one-to-many* and *many-to-many*. For *one-to-many* communication, shown in Fig. 1, a sender generates/communicates traffic to $n - 1$ receivers. Examples of applica-

© ACM, 2004. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in PUBLICATION, VOL 7, NO 4, November 2004 http://portal.acm.org/browse_dl.cfm?idx=J789

This work was partially supported by DARPA under agreement No. F30602-97-1-0257 and by the University of Missouri Research Board.

Authors address: Dijiang Huang and Deep Medhi, Computer Science and Electrical Engineering Department, University of Missouri, 550F Flarsheim Hall, 5100 Rockhill Road, Kansas City, MO. 64110; email: dhuang@umkc.edu, dmedhi@umkc.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2004ACM 1094-9224/04/1100-0001 \$5.00

tions are pay-TV, software distribution, secure distribution of copyright-protected material (e.g. music), audio streaming, and so on. On the other hand in *many-to-many* communications, each member can potentially be both a sender and a receiver. Generally speaking, *many-to-many* communications require formation of both sender-group and receiver-group which are ad hoc and the applications usually are delay sensitive. For example, potential applications are dynamic private conferencing, peer-to-peer interactive applications, online interactive games, and so on.

In this paper, our focus is on *secure* group communication, and more importantly, *many-to-many* secure communications. It may be noted that the majority of current work on secure group communication is for *one-to-many*. In this case, centralized group management schemes are usually suitable; for example, the sender is the group management controller. In Fig. 1 (*one-to-many*), if we consider the population for group communication to be n , sender 1 may want to distribute private information to any possible subset of the set: $\{2, 3, \dots, n\}$. Thus, the research on *one-to-many* secure group communication have been focussed on group members' revocation – to construct the desired subgroup with minimal user storage and group management overhead. Thus, the intuitive approach for *many-to-many* communication is to use a centralized key distribution server that is responsible for subgroup formation. However, this approach has several deficiencies. First, the introduction of the trusted third party for group management will cause additional group setup delay; this is not suitable for interactive realtime applications. Second, the centralized framework is vulnerable to single point failure. Third, the centralized key server must be always available online; this deficiency restricts the usage for certain applications. For example, in a wireless ad hoc network, the key server must maintain connections with all group members, which severely constrains the mobility of mobile users.

Another approach for *many-to-many* secure group communication utilizes the distributed group management schemes, in which the group members collaborate to build the group key, such as the Group Diffie-Hellman (GDH) method [Ateniese et al. 1998; Burmester and Desmedt 1996; Steiner et al. 1996; 1998; Kim et al. 2000]. This approach requires a linear number of expensive public-key operations. In addition to the computation overhead imposed by public-key operations, each user needs to negotiate with its communication peers to maintain the communication group. This may not be desirable for delay sensitive and real-time interactive applications.

In this paper, we develop a hybrid scheme that capture the merits of both centralized and distributed approaches for *many-to-many* secure group/sub-group communication that is suitable for delay sensitive applications which can not tolerant communication set-up delay. Our approach has two phases: a *key predistribution phase* and a *group communication phase*. Similar to the centralized key server approach, in the *key predistribution phase*, a Key Distribute Center (*KDC*) distributes the keys and index $\langle \text{key } ID, \text{user } ID \rangle$ to each group member via offline methods or online secure channels. During the *group communication phase*, which is different from the centralized key server approach, a group member can derive subgroup communication keys from its predistributed keys *without* relying on

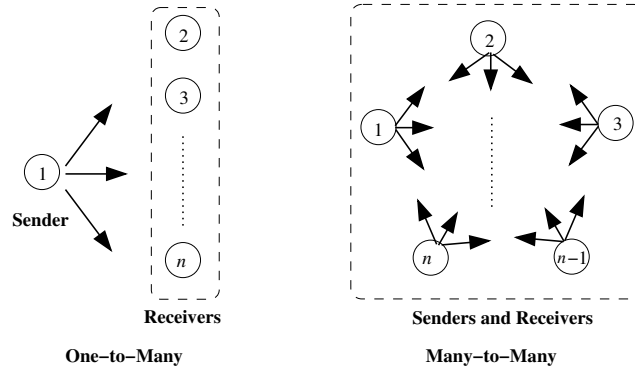


Fig. 1. *one-to-many* and *many-to-many* communications

KDC. In other words, the subgroup management is decentralized. Compared to the GDH approach, there are no negotiation procedures and no expensive public key operations involved for setting up such subgroup communication. We refer to our proposed group management scheme during the *group communication phase* as “self-management”. To reduce the storage overhead, we build a key-forest structure (described later in Section 5) via multiple key chains. Each “tree” (a key chain) in the key-forest structure represents a particular key derivative relations among all group members. We propose a key distribution scheme that builds the relations among multiple key chains which enable a group member to derive desired subgroup keys *without* needing any help from the *KDC*.

Since the group member itself can validate the subgroup communications through its predistributed keys and index $\langle \text{key ID}, \text{user ID} \rangle$, our proposed scheme is useful for applications that cannot tolerate delay (due to overhead of subgroup set-up) and have frequent occurrence of subgroups. To summarize, our proposed secure group keying scheme is applicable with the following assumptions: (a) the overall group population is stable, (b) the group size is moderate and is restricted by the storage requirement of a group member, (c) the subgroup change is frequent, and (d) the application is delay sensitive.

The rest of this paper is organized as follows. In Section 2, we present a brief literature survey related to our work. In Section 3, we list the notations that are used through this paper. The requirements to design *many-to-many* secure group communication is presented in Section 4. In Section 5, we present some fundamental theorem, lemmas, corollaries and algorithms for the linear key-chain structure and the multiple key-chain structure. Based on this foundation, we discuss corresponding keying protocols in Section 6. We then follow up with performance assessments of the secure group keying scheme in Section 7. Finally, we present our summary in section 8.

2. RELATED WORK

Secure group communication keying schemes can be broadly classified into decentralized schemes and centralized schemes. The most well-known decentralized scheme is the Group Diffie-Hellman (GDH) method [Ateniese et al. 1998; Burmester

and Desmedt 1996; Steiner et al. 1996; 1998; Kim et al. 2000]. This approach requires a linear number of expensive public-key operations. The works in [Alves-Foss 2000; Burmester and Desmedt 1995; Kim et al. 2001; 2000] reduce the number of public-key operations. Besides the computational overhead imposed by public-key operations, each user needs to negotiate with its communication peers to maintain the communication group. In general, this is not suitable for delay sensitive and real-time interactive applications.

For centralized secure group communication keying schemes (which are all non-public key solutions), we categorize them into two groups: one from the information theory community, and the other from the Internet community.

2.1 Secure group communication schemes from the information theory community

Secure group communication schemes proposed by the information theory community are broadly classified into i) *key pre-distribution scheme (KPS)* and ii) *broadcast encryption scheme (BES)*.

KPS is inspired by Blom [1985] and extensively studied by Blundo et al [1993; 1995; 1996; 1998], [Gong and Wheeler 1990; Korjik et al. 1995; Kurosawa et al. 1995; Leighton and Micali 1994; Stinson and van Trung 1998]. *KPS* requires a *trust authority* to distribute secret information in such a way that only privileged subsets (pre-specified) of participants are able to compute certain keys. For *zero-broadcasting* ($t, \leq w$)-*KPS*, providing t members subgroup communication and preventing collusion among at most w members, a user need to carry $\binom{t+w-1}{t-1}$ keys. In the worst case with $t + w = n$ and $t = w$, the maximum number of keys a user would need to possess is $\binom{n-1}{\lfloor n/2 \rfloor}$.

BES consists of a key pre-distribution phase, followed later by a broadcast message which is to be decrypted only by a privileged subset (pre-specified) of participants [Berkovits 1992; Fiat and Naor 1994; Blundo et al. 1993; Blundo et al. 1994; 1996; Just et al. 1994]. Stinson's survey [1997] summarizes that this approach requires a user to possess $\sum_{j=0}^w \binom{n-1}{j}$ keys for *BES* to prevent the collusion among w users. Naor et al. [2001] developed a subset-difference method which requires $\frac{1}{2} \log^2 n$ keys being stored at members with $2w$ communication overhead.

2.2 Secure group communication schemes from the Internet community

Within the Internet community, we can classify secure group communication keying schemes as *key oriented schemes (KOS)* and *framework oriented schemes (FOS)*. Three excellent surveys are [Dondeti et al. 1999], [Moyer et al. 1999], and [Rafaeli and Hutchison 2003]. Generally speaking, *KOS* uses key derivative relations to build up the keying scheme. Group members use their secrets to generate the desired group key (See [Briscoe 1999; Waldvogel et al. 1999; Wallner et al. 1999; Wong et al. 2000]). Snoeyink et al [2001] studied the multicast key distribution based on *KOS*. They gave the lower bound for adding or deleting a user as $\Omega(\log n)$. For a sequence of m additions or evictions, the total distribution cost is $\Omega(m \log n)$. Sherman and McGrew [2003] proposed the One-way Function Tree (OFT) scheme in which the key distribution cost for bulk additions (m members) remains in the order of $\Omega(m \log n)$ and the key distribution cost for bulk eviction is improved to the order of $\Omega(m \log(n/m))$.

FOS uses hierarchical group relations to set up group keys. *FOS* can be either flat that has one group management center, or multi-level groups that the group management overhead is distributed to multiple group management centers (See [Gong and Shacham 1994; Harney and Muckenhirn 1997; Ballardie 1996; Mittra 1997]).

2.3 Summary

Most of the work mentioned above are designed for one-to-many group communication. While some of these approaches can be used in *many-to-many* group communications by considering the fact that a *many-to-many* communication is made up of “many” *one-to-many* communications. When one considers issues such as communication overhead, storage constraints, and delay-sensitive requirements, very few are well suited for *many-to-many* secure group communication.

3. NOTATIONS

The following notations will be used throughout the rest of the paper:

\mathcal{N}	All positive integer
\mathcal{U}	Group of users $\{u_c c = 1, \dots, n\}$, $ \mathcal{U} = n$
u_c	Group member $u_c \in \mathcal{U}$
\mathcal{K}_n	Graph \mathcal{K} composed of n vertices
$\mathcal{K}(\mathcal{E}, \mathcal{V})$	Graph \mathcal{K} composed of set of edges $\mathcal{E} = \{e_{ij} i, j = 1, 2, \dots, n\}$ and set of vertices $\mathcal{V} = \{v_i i = 1, 2, \dots, n\}$
s	Number of <i>Hamiltonian</i> cycles that used to create a fully connected graph \mathcal{K}_n , where $n = 2s + 1$.
t	Identify a <i>Hamiltonian</i> cycle, where $t = 1, \dots, s$
$d(u_i, u_j)$	The distance between two members u_i and u_j in a <i>Hamiltonian</i> cycle, where $d(u_i, u_j) = \min\{\vec{d}(u_i, u_j), \overleftarrow{d}(u_i, u_j)\}$
$\vec{d}(u_i, u_j)$	The distance between u_i and u_j in the increased order, where $\vec{d}(u_i, u_j) = (j - i)$ and $j > i$
$\overleftarrow{d}(u_i, u_j)$	The distance between u_i and u_j in the decreased order, where $\overleftarrow{d}(u_i, u_j) = (n + i - j)$ and $j > i$
$K_{\hat{i}}$	A key chain $\{k_{i_0}, \dots, k_{i_r}\}$, indices $\hat{i} = \{i_0, \dots, i_r\}$, $r \leq n - 1$ and $ K_{\hat{i}} = r + 1$ is the length of the key chain
$K_{\hat{i}}^{(t)}$	A key chain allocated following <i>Hamiltonian (permutation) cycle</i> t
k_{i_j}	A key in key chain $K_{\hat{i}}$, $j = \{0, \dots, r\}$. The number of derived subgroups of k_{i_j} is given by $ k_{i_j} = K_{\hat{i}} - j = r - j + 1$
$k_{i_j}^{(t)}$	A key in key chain $K_{\hat{i}}^{(t)}$
k'	A temporary session key
\mathcal{S}	Subgroup of users, $\mathcal{S} \subseteq \mathcal{U}$
$\bar{\mathcal{S}}$	Complementary subgroup of \mathcal{S} , $\mathcal{S} \cap \bar{\mathcal{S}} = \phi$ and $\mathcal{S} \cup \bar{\mathcal{S}} = \mathcal{U}$
\mathcal{S}_{i_j}	Subgroup of users that use key k_{i_j}
L	The subgroup size $L = \mathcal{S} $
$p^{(t)}$	t^{th} <i>permutation</i> cycle, where $t = \{1, 2, \dots, s\}$ and $s = (n - 1)/2$
$p_k^{(t)}$	k^{th} <i>permutation</i> generated by <i>permutation</i> cycle $p^{(t)}$
$p_I^{(t)}$	Index of a <i>permutation</i> cycle $p^{(t)}$, $p_I^{(t)} = p_0^{(t)}$

$S_k^{(t)}$	The sequence of a <i>permutation</i> $p_k^{(t)}$
$h^j(\cdot)$	Hash function, which hash input j times
$f(\cdot)$	Key generating function
$E(\cdot)$	Encrypting function
$D(\cdot)$	Decrypting function
$\mathcal{I}(\cdot)$	Index function
\Rightarrow	Derivative relation
\parallel	Concatenate operation

4. MANY-TO-MANY SECURE GROUP COMMUNICATION REQUIREMENTS

In this section, we discuss design requirements for secure group communication that support *many-to-many* communication considering the following aspects: communication phases, storage constraint, self-management, and key agreement protocol. All our analysis is based on a group of n users and each of them is identified by u_c , where $c = 1, \dots, n$.

4.1 Secure group communication phases

Our proposed secure group communication scheme includes two phases: the *key predistribution phase* and the *group communication phase*. In the *key predistribution phase*, a *KDC* distributes the keys and index $\langle \text{key ID}, \text{user ID} \rangle$ to each group member via offline methods or online secure channels. Then, the group members enter the *group communication phase*. During the *group communication phase*, a group member can derive subgroup communication keys from its predistributed keys *without* relying on *KDC*. We refer to our proposed group management procedure during the *group communication phase* as “self-management”.

4.2 Storage constraint

We assume that the key server is responsible for keys predistribution. A group member needs sufficient space to store the secrets pre-installed by the key server in the *key predistribution phase* and uses the predistributed secrets to setup group keys during the *group communication phase*. Two factors restrict the group member’s storage space: the applications and the hardware restriction. A typical *many-to-many* application is private conferencing which involves intensive interactive private communications among group members. It can be used in either a wired network environment or an ad hoc network environment. If the size of a group is n , then the possible number of subgroups is $2^{n-1} - 1$. Current low-end mobile devices such as personal digital assistants (PDAs) have memory size ranged from 8MB to 128MB or more. Thus, we assume that the end user has the key space limitation of 1MB to support $2^{n-1} - 1$ number of subgroup communications¹.

4.3 Self-management

Our proposed scheme is suitable for real-time interactive applications and the subgroup memberships may change frequently. This type of applications demands the minimal delay for sub-group setup for communication. We require that a group member can self-generate the desired subgroup keys *without* the help of key server

¹For high-end devices, such as laptops, such storage limitation is not necessarily an issue.

after the *key predistribution phase*. This requirement is also helpful for ad hoc network where the key server may not be online all the time.

4.4 Key agreement protocol

The sender can use a session key k' to encrypt data and use subgroup keys as *key-encrypting key (KEK)* to encrypt the session key that is attached to the encrypted data. Multiple subgroup keys can be used as *KEKs* to fulfil desired subgroup composition. Using the encrypting function $E(\cdot)$ and decrypting function $D(\cdot)$, the sender can encrypt the data as follows:

$$\langle [i_{1j_1}, \dots, i_{rj_r}, E_{k_{i_1j_1}^{(t_1)}}(k'), \dots, E_{k_{i_rj_r}^{(t_r)}}(k')], E_{k'}(Data) \rangle$$

$$(i_1 \dots i_r \in \{1, 2, \dots, n-1\}; j_1 \dots j_r \in \{1, 2, \dots, n-1\}; t_1 \dots t_r \in \{1, 2, \dots, s\})$$

The superscript t_r of k stands for the key chain being derived from the appropriate *Hamiltonian* cycle. Algorithm 5.1 gives the complete description on how to generate the *Hamiltonian* cycles and Section 6 presents how to use the generated *Hamiltonian* cycles to build our keying scheme. $\{i_1, \dots, i_r\}$ is the key chain list. The subscription j_r tells which key element is used in the corresponding key chain. If a receiver possesses a key element j_k in the key chain i_r , if and only if $j_r \geq j_k$, then she can hash the key $k_{i_rj_k}^{t_r}$ by $j_r - j_k$ times to derive key $k_{i_rj_r}^{t_r}$. The description of the key message is presented in Section 6.1.2. Once the encrypted message is received, the group members can determine the following information: (1) source of the senders, (2) subgroup formation. In order to facilitate the key agreement protocol to achieve these two goals, the proposed secure group keying scheme considers that (a) the key *id* is served as the group *id*, and (b) the key *id* tells the group formation.

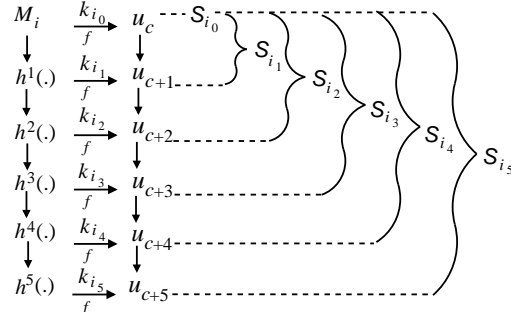
Once a receiver receives this message, she can search the key chain list $\{i_1, \dots, i_r\}$ to find out if she belongs to the receiver-group. If so, she can use the decryption function $D(\cdot)$ to decrypt the session key k' , and then use the session key to decrypt the data.

5. USING HASH CHAIN TO BUILD KEY CHAIN

In this section, we present the preliminaries which are building blocks for our secure group keying protocols. We first show how we can use a hash function [Rivest 1992] to construct a hash chain, and subsequently, to form a key chain. We present three lemmas in regarding to forming *non-colluding* (defined in definition 5.1) subgroups which are based on the linear structure of a key chain. Then, we discuss two lemmas: Lemma 5.6 is used for building the unbalanced keying protocol that is suitable for centralized multicast applications; Lemma 5.7 is used for building the balanced keying protocol that is suitable for decentralized multicast applications. More discussion about balanced and unbalanced keying protocols will be given later in Section 6.

We start with function $h(\cdot)$ which is a hash function that maps an arbitrary length message M to a fixed length message digest MD . It satisfies the following properties:

- (1) $h(\cdot)$ is publicly known.
- (2) Given M , it is relatively easy to compute $h(M)$.

Fig. 2. An example of keys allocation ($n = 6, r = 5$)

- (3) Given MD , it is computationally infeasible to find a message M such that $h(M) = MD$.
- (4) Given M and $h(M)$, it is computationally infeasible to find a message $M' (\neq M)$ such that $h(M') = h(M)$.

A hash chain is a sequenced hash values with the linear derivative relations among them. For example, we use $h^j(\cdot)$ to denote using hash function $h(\cdot)$ on a message j times. The outputs of the multiple hash operations construct a hash chain. A hash chain maintains a linear derivative relations among multiple hash values, i.e., $h^j(\cdot)$ can derive $h^r(\cdot)$ when $j < r$.

Consider message M_i to be the i^{th} initial key element that is used by a user to generate the cryptographic key denoted by k_{i_0} , where $k_{i_0} = f(M_i)$, and where $f(\cdot)$ is a key generating function; this key generating function is publicly known and generates the exact length of cryptographic key. Thus we have the following relation:

$$k_{i_j} = f(h^j(M_i)) \quad (j > 0).$$

Since $h(\cdot)$ is publicly known, we know that the derivative relations of a hash chain are:

$$M_i \Rightarrow h^1(M_i) \Rightarrow \dots \Rightarrow h^j(M_i) \Rightarrow \dots \Rightarrow h^r(M_i). \\ (1 < j < r)$$

Then, through function $f(\cdot)$, corresponding key chain derivative relations are:

$$k_{i_0} \Rightarrow k_{i_1} \dots \Rightarrow k_{i_j} \Rightarrow \dots \Rightarrow k_{i_r}. \\ (1 < j < r)$$

The derivative relations among multiple keys form a linear hierarchical structure from top (k_{i_0}) to the bottom level (k_{i_r}). One straight forward key allocation example is shown in Fig. 2, which is composed of 6 keys ($r = 5$). If we allocate a key to each group members from u_c to u_{c+5} , each group member can derive its lower-level keys. Then 6 subgroups can be formed via the key chain K_i , denoted by S_{i_0} to S_{i_5} . Note that these subgroups are *non-colluding* subgroups, in which s set of members cannot collude to derive the keys used by a given subgroup unless at least one of members belongs to that subgroup.

Definition 5.1. The *non-colluding* feature of the group key management is described as follows: $\forall \mathcal{S} \in \mathcal{U}$, we have $\forall u_i \in \mathcal{S}, u_i \Rightarrow K_{\mathcal{S}}$ and $\forall u_j \in \mathcal{U} \setminus \mathcal{S}, u_j \not\Rightarrow K_{\mathcal{S}}$.

Based on the above discussion we has the following lemmas:

LEMMA 5.2. *For a key chain with length $|K_{\mathcal{Z}}| = r + 1$, it is possible to form $r + 1$ non-colluding subgroups.*

PROOF. A key chain $K_{\mathcal{Z}}$ contains $r + 1$ keys, which can be assigned to $r + 1$ group members. These group members can form $r + 1$ subgroups from size 1 to $r + 1$. \square

LEMMA 5.3. *For a key chain with length $|K_{\mathcal{Z}}| = r + 1$ and a user group $|\mathcal{U}| = n$, $n = r + 1$ is the minimum length of key chain to form n non-colluding subgroups when one key is uniquely assigned to each group member.*

PROOF. If we assign two keys k_{i_m} and k_{i_l} to a user u_c , and there exist a key k_{i_x} where $l < x < m$, we cannot assign k_{i_x} to any other users except u_c . This is because key k_{i_m} has been assigned to u_c , u_c should not derive any keys which is higher than its level. But key k_{i_l} will allow u_c to derive k_{i_x} . Then, this would not be a *non-colluding* subgroup. Also, if more than one keys are assigned to a user, it will increase the length of a key chain. Thus, from Lemma 5.2 we know that a key chain is at least of length n and exactly one key from the key chain is assigned to a user, which can guarantee every user is a member of a subgroup and there is no collusion problem. \square

LEMMA 5.4. *Key k_{i_j} can serve as a subgroup key with the group size of $j + 1$.*

PROOF. Key k_{i_j} is hashed j times, which can be derived by j keys (or group members). \square

5.1 Multiple key chains for unbalanced keys allocation

We extend the linear structure of a key chain to a non-linear structure by combining multiple key chains. All these properties provide critical ingredients to our keying science due to efficiency and flexibility. In this section we present the methods for unbalanced keys allocation. It is used for building a unbalanced keying protocol that is suitable for centralized multicast applications, in which a multicast center initiates the majority of multicast sessions.

Assigning more than one key from multiple key chains to a user is very similar to arranging the group member's positions in different key chains. We assume \mathcal{U} is fixed with size n ; then, assigning keys to a group member is the same as to follow permutations in different key chains. Our goals are two folds:

- (1) Maximizing the number of subgroups that can be covered.
- (2) Minimizing the number of keys possessed by a group member.

Note that these two goals contradict each other. We show how we can achieve the minimized number of keys possessed by a group member under certain subgroup coverage requirements. In order to evaluate the efficiency of the keying scheme we define the term of $\Delta k / \Delta g$ ratio.

Definition 5.5. $\Delta k / \Delta g$ ratio is ratio of the incremental number of keys to the incremental number of covered subgroups:

$$\Delta k / \Delta g = \frac{\mathcal{I}(K_i) - \mathcal{I}(K_l)}{\sum_{j=l+1}^i |\mathcal{S}_j|} \quad (i > l \geq 1) \quad (1)$$

Equation 1 specifies the increased number of subgroup keys needed when one subgroup is added in. $\mathcal{I}(\cdot)$ is the index function where $\mathcal{I}(K_i) = i$ and $\mathcal{I}(K_l) = l$. Further, $\mathcal{I}(K_i) - \mathcal{I}(K_l) = |i - l|$ stands for the increased number of keys of a group member when adding key chains from $K_{(l+1)}$ to K_i . Here, we use Lemma 5.3, and each key of a key chain is uniquely assigned to a group member. $l \geq 1$ means that there already exists at least one key chain. The summation $\sum_{j=l+1}^i |\mathcal{S}_j|$ stands for the summation of distinct subgroups supported by key chains from $K_{(l+1)}$ to K_i . An efficient keying scheme always come with the least value of the $\Delta k / \Delta g$ ratio. For example, assigning a unique key to each subgroup without derivative relation, $\Delta k / \Delta g$ ratio is 1. If we have n members, the number of keys a group member possessed is 2^{n-1} . When n is very big, this keying scheme is impractical. Thus, our task is to find the minimal $\Delta k / \Delta g$ ratio while the maximum number of subgroups are supported.

In order to study how we can allocate keys from multiple key chains, we use some preliminary definitions from the *permutation* theory (see *Appendix*).

First, we consider the maximum number of subgroup keys possessed by a group member using $n - 1$ key chains.

LEMMA 5.6. *Consider user $u_c \in \mathcal{U}$, where $|\mathcal{U}| = n$, $n \geq 3$ and $c = \{1, \dots, n\}$. If we allocate $n - 1$ keys k_{i_j} ($0 \leq j \leq n - 1$, $i = \{1, \dots, n - 1\}$) to user u_c , each of these keys from a unique key chain, then the minimum $\Delta k / \Delta g$ ratio is $1/(n - 2)$ and the maximum number of subgroups the user u_c can communicate with is $n^2 - 3n + 4$.*

PROOF. We number the users in the group \mathcal{U} from 1 to n . From Lemma 5.2 and Lemma 5.4, we know that each key chain can maximally form n subgroups. When a user possesses all k_{i_0} ($i = \{1, \dots, n - 1\}$), he can communicate with the maximum number of subgroups. When we have $n - 1$ key chains, we can assign all the head of the $n - 1$ key chains (k_{i_0}) to a user, which can maximize the number of subgroup keys the user can derive. We need to assign keys in $n - 1$ key chains to n users systematically, which can form the maximum number of distinct subgroups. This is the same as arranging the *permutation* of n users. If we assign number 1 to the user u_1 and $2, \dots, n$ to the rest $n - 1$ users u_c ($c \in \{2, \dots, n\}$), we use *permutation* cycles (see Definition A.3 in *Appendix*) to assign keys to avoid subgroups overlapping. The *permutation* can be represented by two cycles $p = \{\{1\}, \{n, 2, 3, \dots, n - 1\}\}$. If we multiply p to identity *permutation* $I = \{1, 2, \dots, n\}$ by $n - 1$ times, the result of multiplication is the same as identity *permutation* I :

$$I = I \cdot p^{n-1}$$

For each multiplication, the *permutation* cycle $\{n, 2, 3, \dots, n - 1\}$ causes the element from position 2 transposing to position n and other elements (excluding at the positions 1 and 2) transposing to the immediate left position. Thus, we can have $n - 1$ distinct permutations $I, I \cdot p, \dots, I \cdot p^{n-2}$. Note that, in each permutation, the first element is 1 and the second elements will be distinct elements from 2 to n . As shown in Fig. 3, the permutation sequences are shown from right to left.

row \ col	1	2	3	...	i	...	$n-2$	$n-1$
1	1	1	1	...	1	...	1	1
2	2	n	$n-1$...	$n-i+2$...	4	3
...
j	j	$j-1$	$n-i+j$...	$j+2$	$j+1$
...	$j-i+1$
$n-1$	$n-1$	$n-2$	$n-3$...	$n-i$...	2	n
n	n	$n-1$	$n-2$...	$n-i+1$...	3	2

Fig. 3. $n - 1$ key chains.

Each column represents a key chain which is indexed by i and each row element represents a key $k_{i,j}$ which is indexed by j . Beginning the second row, the key elements above the dash line is indexed by $n - i + j$ and the key elements on and below the dash line is indexed by $j - i + 1$.

Because of the top-bottom hierarchical key-chain structure, we need to do a top-down search to find if there exists overlapped subgroups. We randomly pickup column i ; the second element in this column is $n - i + 2$ which is above the dash line. In order to find the same subgroup members composition, other key chains should contain the element $n - i + 2$. In the immediate right key chain, the element $n - i + 2$ is one level below (row 3) and $n - i + 1$ is on the same level (row 2). But on column i , the element $n - i + 1$ is on the bottom level (row n). This means that we need to search the group members all the way down to the bottom, then we can find two totally matched subgroups. We also observe that, due to the cycle, the right/left key chains always maintain the structure that element $n - i + 1$ is above element $n - i + 2$, which means there is no subgroups which can match except that the subgroup contains all group members. Thus, in permutations of $I, I \cdot p, \dots, I \cdot p^{n-2}$, there is no same formation of subgroup except the subgroup that contains all group members. The total number of subgroups is $n(n - 1) - 2(n - 1) + 2 = n^2 - 3n + 4$. Thus, $\Delta k / \Delta g$ ratio is equal to the increment of 1 key to the incremental of supported subgroups, which is $1/(n - 2)$.

If we add one more key chain and still put 1 in the first place, there will be overlapping of subgroups with previous subgroups. Thus, increasing one key, the incremented number of subgroups will be less than $n - 2$, and then the new ratio is $\Delta k / \Delta g < 1/(n - 2)$. \square

The *Proof* of Lemma 5.6 uses one particular *permutation* cycle $\{n, 2, 3, \dots, n - 1\}$ from the second position to the last position. The number of permutations of a given number n having exactly 1 *permutation* cycles is defined by the *Stirling numbers of the first kind* $S_1(n, 1) = (n - 1)!$ [Comtet 1974]. In fact, for any one of $(n - 1)!$ *permutation* cycles, Lemma 5.6 is valid and provable.

col	1	2	3	...	i	...	$n-1$	n	
row	1	1	n	$n-1$...	$n-i+2$...	3	2
2	2	1	n	...	$n-i+3$...	4	3	
...	$n-i+j+1$	
j	j	$j-1$	$j+2$	$j+1$	
...	$j-i+1$	
$n-1$	$n-1$	$n-2$	$n-3$...	$n-i$...	1	n	
n	n	$n-1$	$n-2$...	$n-i+1$...	2	1	

Fig. 4. n key chains.

5.2 Multiple key chains for balanced keys allocation

Lemma 5.6 addresses key allocation issues for a particular user (u_1 in our proof of Lemma 5.6). For more general case, we assume every group member is treated equally. In this section we present methods for balanced keys allocation. They are used for building a balanced keying protocol that is suitable for decentralized multicast applications, in which every member has the same opportunity to initiate a multicast session.

LEMMA 5.7. *Consider user $u_c \in \mathcal{U}$, where $|\mathcal{U}| = n$, $n \geq 1$ and $c = \{1, \dots, n\}$. There exists n distinct key chains K_i and $|K_i| = n$. If we allocate n keys k_{i_j} ($1 \leq i \leq n, 0 \leq j \leq n-1$) to user u_c , each from a unique key chain, the number of subgroups supported by n key chains is $n^2 - n + 1$. On average, the number of subgroups that a user can communicate with is $n(n-1)/2 + 1$.*

PROOF. Using *permutation* cycle $p = \{n, 1, 2, \dots, n-1\}$, we can build the *permutation* matrix, which is shown in Fig. 4. Then, we have n distinct permutations $I, I \cdot p, \dots, I \cdot p^{n-1}$. Each column represents a key chain. Following the *permutation* sequence, we can build non-overlapping subgroups to assign keys to each group member. The proof is the same as in the proof for Lemma 5.6. The only difference is that for Lemma 5.6, the length of *permutation* cycle is $n-1$, and for Lemma 5.7, the length of *permutation* cycle is n . Thus the total number of subgroups is the total number of subgroups in the *permutation* matrix minus $n-1$ overlapped groups (all members group), that is $n \cdot n - (n-1) = n^2 - n + 1$. The number of subgroup keys that user u_c can generate directly is given as follows:

$$\begin{aligned}
 \sum_{i=1}^n |k_{i_j}| - (n-1) &= \sum_{j=0}^{n-1} (|K_i| - j) - (n-1) \\
 &= \sum_{j=0}^{n-1} (n-j) - (n-1) \\
 &= \frac{n(n-1)}{2} + 1.
 \end{aligned}$$

□

To give a more visualized view of our scheme, we consider the scenario where every group member is a vertex (each group member in \mathcal{U} is mapped to a vertex in \mathcal{V} : $v_c \in \mathcal{V}$, $u_c = v_c$ and $c = \{1, 2, \dots, n\}$) of a graph $\mathcal{K}(\mathcal{E}, \mathcal{V})$. To find a full *permutation* cycle (covers every number in a *permutation* cycle) is the same as to find a *spanning* cycle² of the graph. The directed edge e_{ij} ($e_{ij} \in \mathcal{E}$ and $i, j = \{1, 2, \dots, n\}$) between two nodes i and j means a transposition of position i to position j in the *permutation*. Assuming that a group has n members, in which we associate them to n vertices in a graph, where n is odd ($n = 2s + 1$ and $s \in \mathcal{N}$). These n vertices can form a fully connected graph which we refer to as \mathcal{K}_{2s+1} . We now state the following important theorem [Harary 1969] which is applicable here.

THEOREM 5.8. *The fully-connected graph \mathcal{K}_{2s+1} is the sum of s spanning cycles, where $s \in \mathcal{N}$. (For Proof, refer to [Harary 1969], p. 89)*

This theorem states that any fully connected graph \mathcal{K}_{2s+1} can be constructed by s independent *spanning* cycles. These s *spanning* cycles have no overlapped edges, and each cycle has exactly n edges. Since our keying scheme is based on a sequence of hash values, we use the term “*Hamiltonian* cycles”³ instead of “*spanning* cycles”.

There are many ways to create s independent *Hamiltonian* cycles. We present an algorithm that creates s *Hamiltonian* cycles for graph \mathcal{K}_{2s+1} , where $2s + 1$ is a prime.

ALGORITHM 5.1 HAMILTONIAN CYCLES CREATION ALGORITHM.

<i>Initial</i>	<i>Given n entities: v_1, v_2, \dots, v_n, where $n = 2s + 1$, $s \in \mathcal{N}$</i>
<i>Procedures</i>	<i>construct a set of s paths denoted by \mathcal{P}_t, on the points $v_1, v_2, \dots, v_{2s+1}$ as follows:</i>
	<i>(1) create s paths: $\mathcal{P}_t = v_1 v_{t+1} v_{2t+1} v_{3t+1} \dots v_{2st+1}$, where $t = \{1, \dots, s\}$. All the subscripts are taken as integers (mod $2s + 1$)</i>
	<i>(2) the Hamiltonian cycle \mathcal{Z}_t is then constructed by joining v_1 to the endpoints of \mathcal{P}_t</i>
<i>End</i>	

For any group of users \mathcal{U} ($|\mathcal{U}| = n$ and $n = 2s + 1$, $s \in \mathcal{N}$), we can always find s *Hamiltonian* cycles (or *spanning* cycles or *permutation* cycles) which have no overlapped edges. Using Algorithm 5.1, we have the index of *permutation*⁴ that is constructed by *Hamiltonian* cycle \mathcal{Z}_t :

$$p_I^{(t)} = \{\{1, t + 1, 2t + 1, \dots, 2st + 1\} \pmod{(2s + 1)}\} \quad (2)$$

$p_I^{(t)}$ is the first *permutation* ($p_I^{(t)} = p_0^{(t)}$). The *permutation* cycle is given as:

$$p^{(t)} = I \cdot (\{2, 3, \dots, 2s + 1, 1\})^t \quad (3)$$

²A *spanning* cycle is the one that covers every nodes in a graph exactly once.

³A *Hamiltonian* cycle visits each vertex exactly once in sequence.

⁴Note here that the superscript (t) does not stand for the power of *permutation* or sequence. It means the *permutation* or sequence derived from the t^{th} *Hamiltonian* cycle.

Thus, there exists $2s$ permutations, which are given by:

$$p_k^{(t)} = I \cdot (p^{(t)})^k, \quad (k = \{1, 2, \dots, n-1\}) \quad (4)$$

The sequence of a *permutation* indexed by $p_I^{(t)}$ is represented by $S_k^{(t)}$, where $k = \{1, 2, \dots, n-1\}$. We also have $S_0^{(t)} = p_0^{(t)} = p_I^{(t)}$.

6. SECURE GROUP KEYING PROTOCOL

Based on discussion so far, if we associate each subgroup member to a unique vertex in a fully meshed graph by following the tour of a continuous path, finding a subgroup key is equivalent to find a continuous path among these subgroup members in the graph. Based on this important observation, we now discuss our secure group keying protocol in detail.

6.1 Balanced keying protocol

In the balanced keying protocol, we treat each group member equally. The balanced keying protocol is based on Lemma 5.7, Theorem 5.8 and Algorithm 5.1.

Given $\mathcal{U} = \{u_c | c = 1, 2, \dots, n\}$ and n key chains, where $n = 2s + 1$ and $t \in \{1, 2, \dots, s\}$. We have the following algorithm:

ALGORITHM 6.1 KEY ASSIGNMENT ALGORITHM.

Initial Select a unique $t \in \{1, 2, \dots, s\}$; initialize variable $k = 0$

Step 1 (1) for key chain K_i , using Algorithm 5.1, a Hamiltonian cycle \mathcal{Z}_t is a permutation cycle $p^{(t)}$ given by Equation (3).

(2) permutations $p_k^{(t)}$ are given by Equation (4) and the corresponding sequence is $S_k^{(t)}$.

(3) denote $\mathcal{I}_j(S_k^{(t)})$ for the $(j+1)^{th}$ element in sequence $S_k^{(t)}$.

(4) assign k_{i_j} to user $u_{\mathcal{I}_j(S_k^{(t)})}$, where $j = \{0, 1, \dots, n-1\}$

Step 2 $k = k + 1$, if $k \geq n$, Stop. Otherwise goto step 1

Algorithm 6.1 presents how to distribute keys to group members from a given key chain. The following algorithm shows how to distribute keys to group members by following the tour of s Hamiltonian cycles.

ALGORITHM 6.2 KEY-CHAIN ASSIGNMENT ALGORITHM.

Step 1 Given a graph \mathcal{K}_n , where $n = 2s + 1$, $s \in \mathcal{N}$, build s Hamiltonian cycles using Algorithm 5.1

Step 2 By taking the tour of s Hamiltonian cycles using Algorithm 6.1, build s sets of keys allocation

6.1.1 *An example of balanced keying protocol.* We illustrate the keying scheme via an example with the group size 7.

EXAMPLE 6.1. We consider 7 group members as 7 vertices in a graph, shown in Fig. 5. Following Algorithm 5.1, we build 3 separate Hamiltonian cycles. Node 1 is the starting point of each of these three Hamiltonian cycles. These three cycles form a fully meshed graph \mathcal{K}_7 . Following Algorithm 6.1 and Algorithm 6.2, we form the permutation table in Fig. 6. The keys allocation is shown in Table I.

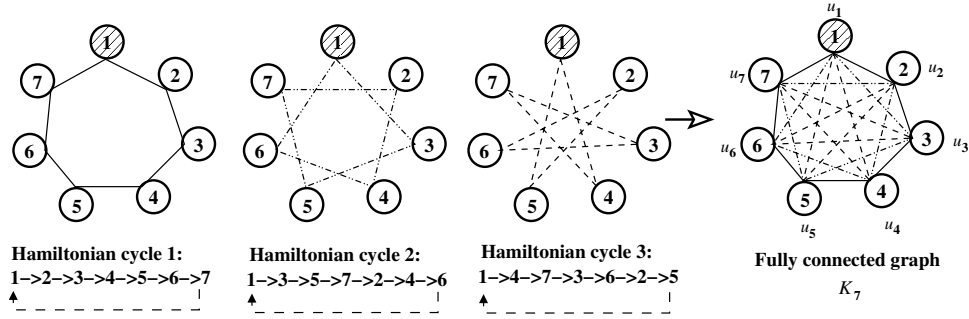


Fig. 5. An example of *Hamiltonian* cycles for 7 nodes ($n = 7, s = 3$)

	Cycle 1 (t=1)	Cycle 2 (t=2)	Cycle 3 (t=3)
Index ($p_0^{(i)}$)	1 2 3 4 5 6 7	1 3 5 7 2 4 6	1 4 7 3 6 2 5
Identity (I)	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
Permutation cycle $p^{(i)}$	7 1 2 3 4 5 6	6 7 1 2 3 4 5	5 6 7 1 2 3 4
Permutation $p_1^{(i)}$	2 3 4 5 6 7 1	3 4 5 6 7 1 2	4 5 6 7 1 2 3
Sequence $S_1^{(i)}$	6 2 5 3 4 7 1	3 5 7 2 4 6 1	4 7 3 6 5 2 1
Permutation $p_2^{(i)}$	3 4 5 6 7 1 2	5 6 7 1 2 3 4	7 1 2 3 4 5 6
Sequence $S_2^{(i)}$	2 5 3 4 7 1 6	5 7 2 4 6 1 2	7 3 6 2 5 1 4
Permutation $p_3^{(i)}$	4 5 6 7 1 2 3	7 1 2 3 4 5 6	3 4 5 6 7 1 2
Sequence $S_3^{(i)}$	5 3 4 7 1 6 2	7 2 4 6 1 3 5	3 6 2 5 1 4 7
Permutation $p_4^{(i)}$	5 6 7 1 2 3 4	2 3 4 5 6 7 1	6 7 1 2 3 4 5
Sequence $S_4^{(i)}$	3 4 7 1 6 2 5	2 4 6 1 3 5 7	6 2 5 1 4 7 3
Permutation $p_5^{(i)}$	6 7 1 2 3 4 5	4 5 6 7 1 2 3	2 3 4 5 6 7 1
Sequence $S_5^{(i)}$	4 7 1 6 2 5 3	4 6 1 3 5 7 2	2 5 1 4 7 3 6
Permutation $p_6^{(i)}$	7 1 2 3 4 5 6	6 7 1 2 3 4 5	5 6 7 1 2 3 4
Sequence $S_6^{(i)}$	7 1 6 2 5 3 4	6 1 3 5 7 2 4	5 1 4 7 3 6 2

Fig. 6. An example of *permutations* for 7 group members ($n = 7, s = 3$)

Table I. $n=7$ group (only subscript i_j of k_{i_j} is shown in this table)

\mathcal{K}_7 nodes	1	2	3	4
Member	u_1	u_2	u_3	u_4
cycle 1	1 ₀ 2 ₆ 3 ₅ 4 ₄ 5 ₃ 6 ₂ 7 ₁	1 ₁ 2 ₀ 3 ₆ 4 ₅ 5 ₄ 6 ₃ 7 ₂	1 ₂ 2 ₁ 3 ₀ 4 ₆ 5 ₅ 6 ₄ 7 ₃	1 ₃ 2 ₂ 3 ₁ 4 ₀ 5 ₆ 6 ₅ 7 ₄
cycle 2	1 ₀ 2 ₆ 3 ₅ 4 ₄ 5 ₃ 6 ₂ 7 ₁	1 ₄ 2 ₃ 3 ₂ 4 ₁ 5 ₀ 6 ₆ 7 ₅	1 ₁ 2 ₀ 3 ₆ 4 ₅ 5 ₄ 6 ₃ 7 ₂	1 ₅ 2 ₄ 3 ₃ 4 ₂ 5 ₁ 6 ₀ 7 ₆
cycle 3	1 ₀ 2 ₆ 3 ₅ 4 ₄ 5 ₃ 6 ₂ 7 ₁	1 ₄ 2 ₄ 3 ₃ 4 ₂ 5 ₁ 6 ₀ 7 ₆	1 ₃ 2 ₂ 3 ₁ 4 ₀ 5 ₆ 6 ₅ 7 ₄	1 ₁ 2 ₀ 3 ₆ 4 ₅ 5 ₄ 6 ₃ 7 ₂
\mathcal{K}_7 nodes	5	6	7	
Member	u_5	u_6	u_7	
cycle 1	1 ₄ 2 ₃ 3 ₂ 4 ₁ 5 ₀ 6 ₆ 7 ₅	1 ₅ 2 ₄ 3 ₃ 4 ₂ 5 ₁ 6 ₀ 7 ₆	1 ₆ 2 ₅ 3 ₄ 4 ₃ 5 ₂ 6 ₁ 7 ₀	
cycle 2	1 ₂ 2 ₁ 3 ₀ 4 ₆ 5 ₅ 6 ₄ 7 ₃	1 ₆ 2 ₅ 3 ₄ 4 ₃ 5 ₂ 6 ₁ 7 ₀	1 ₃ 2 ₂ 3 ₁ 4 ₀ 5 ₆ 6 ₅ 7 ₄	
cycle 3	1 ₆ 2 ₅ 3 ₄ 4 ₃ 5 ₂ 6 ₁ 7 ₀	1 ₄ 2 ₃ 3 ₂ 4 ₁ 5 ₀ 6 ₆ 7 ₅	1 ₂ 2 ₁ 3 ₀ 4 ₆ 5 ₅ 6 ₄ 7 ₃	

In Table I, each group member has 21 keys (in a column) which belong to three different cycles (due to space limitation, we only present the subscripts (i_j) of

the k_{i_j} in each row). $K_i^{(t)}$ is a key chain that is constructed by following the t^{th} *Hamiltonian* cycle ($t = \{1, 2, \dots, s\}$) to allocate keys to group members. And similarly, we can add notation $^{(t)}$ to key $k_{i_j}^{(t)}$. Fig. 5 shows that any two-group members are connected by a direct-link and they both can derive a two-member subgroup key to form a secure communication. For example, if we randomly select u_1 and u_4 then, in the *Hamiltonian* cycle 3, u_1 and u_4 are adjacent, and in the third key chain, u_4 has the key $k_{1_1}^{(3)}$ and u_1 can derive it from its $k_{1_0}^{(3)}$. These relations can be applied to any combination of two group members. Note that these *Hamiltonian* cycles cannot cover every combination of selected group members. In a *Hamiltonian* cycle with n entities, the number of l different group members in a continuous path is n , where $l < n$. Thus, in a group with n members, using Algorithm 5.1 there will be s cycles formed. The number of l adjacent members in s cycles is $s * n$. In Example 6.1, when $s = 3$, the number of 3-member subgroups is $\binom{7}{3} = 35$ and $s * n = 21$. Hence, we can infer that three *Hamiltonian* cycles cannot totally cover every possible combination of subgroup members. However, we can build relations among the cycles to solve this problem.

6.1.2 Key messages. We can use a session key k' to encrypt data and use subgroup keys as *key-encrypting key (KEK)* to encrypt the session key that is attached to the encrypted data. Multiple subgroup keys can be used as *KEKs* to fulfil desired subgroup composition. Using the encrypting function $E(\cdot)$ and decrypting function $D(\cdot)$, we can encrypt the data as follows:

$$\langle [i_{1j_1}, \dots, i_{rj_r}, E_{k_{i_1j_1}^{(t_1)}}(k'), \dots, E_{k_{i_rj_r}^{(t_r)}}(k')], E_{k'}(Data) \rangle \quad (5)$$

$$(i_1 \dots i_r \in \{1, 2, \dots, n-1\}; j_1 \dots j_r \in \{1, 2, \dots, n-1\}; t_1 \dots t_r \in \{1, 2, \dots, s\})$$

the receivers check the key chain list $\{i_1, \dots, i_r\}$ to see if they belong to the group. The checking method is straight forward: a receiver compares each element in the list with the j_k in the key chains she possesses; if and only if $j_r \geq j_k$, she belongs to the group; then she can hash the key $k_{i_rj_k}^{t_r}$ by $j_r - j_k$ times to derive key $k_{i_rj_r}^{t_r}$. To decrypt the data, she first decrypts the session key:

$$k' = D_{k_{i_rj_r}^{t_r}}(E_{k_{i_rj_r}^{t_r}}(k')) \quad (6)$$

and then uses k' to decrypt the received cypher data:

$$Data = D_{k'}(E_{k'}(Data)) \quad (7)$$

COROLLARY 6.1. *Using Algorithm 6.2, we can build s (Group size is n , $s = (n-1)/2$ and $s \in \mathcal{N}$) sets of key chains, and for those arbitrarily selected subgroup members, we can find b ($0 \leq b \leq s$) encryption relations that provide secure subgroup communication.*

PROOF. Theorem 5.8 specifies that in a fully meshed graph \mathcal{K}_n , where $n = 2s+1$, $s \in \mathcal{N}$, there will be s *Hamiltonian* cycles. Following Algorithm 6.2, we can form s sets of key chains. For any subgroup $\mathcal{S} \subseteq \mathcal{U}$, we can find at most b continuous path segments in s *Hamiltonian* cycles, where $b \leq s$. The encryption and decryption formats are given from Equation (5) to Equation (7). \square

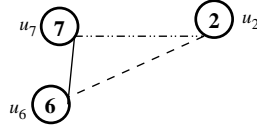


Fig. 7. Sub-graph \mathcal{K}_3

In *Example 6.1*, if we consider subgroup members u_2 , u_6 and u_7 , we cannot find any continuous path in three *Hamiltonian* cycles for these three group members. We prune graph \mathcal{K}_7 to \mathcal{K}_3 , shown in Fig. 7. When u_2 wants to send messages to subgroup $\{u_2, u_6, u_7\}$, u_2 sends out the cypher text as follows:

$$\langle [5_1, 4_1, E_{k_{5_1}^{(3)}}(k'), E_{k_{4_1}^{(2)}}(k')], E_{k'}(Data) \rangle$$

The encrypted session key is attached to the cypher text. Only u_6 and u_7 can decrypt the encrypted session key and thus can decrypt the message. In the worst case, the number of encryption/decryption operations is equal to the number of *Hamiltonian* cycles. It is important to note that this subgroup keying scheme can completely avoid *KDC* being involved in the subgroup’s communication, and the secure subgroup’s communication can be freely formed.

6.1.3 Lower bound and upper bound analysis of the number of encrypted key messages. In this section, we will do an analysis on the lower and upper bound on the number of encrypted key messages.

COROLLARY 6.2. *Consider when each group member has the same probability to be selected to form a desired subgroup with size $L = |\mathcal{S}|$. The lower bound of the number of transmitted encrypted keys is 1; the upper bound is L when $L \leq s$ and s when $L \geq s$ (where $s = (n - 1)/2$).*

PROOF. The proof of lower bound is straight forward. When all subgroup members are located consecutively within any of *Hamiltonian* cycles, the sender can use one key as the subgroup key.

We consider that the overall group can be described as a sorted list from member 1 to member n . We assume the sender to be u_i . In a *Hamiltonian* cycle, we define $d(u_i, u_j)$ to be the distance between two members u_i and u_j . That is, $d(u_i, u_j)$ is the minimum of the distances that counted in the increased order and decreased order, represented by $\vec{d}(u_i, u_j) = (j - i)$ and $\overleftarrow{d}(u_i, u_j) = (n + i - j)$, where $i < j$. Thus $d(u_i, u_j) = \min\{\vec{d}(u_i, u_j), \overleftarrow{d}(u_i, u_j)\}$, where $1 \leq d(u_i, u_j) \leq s$.

To build a subgroup with r subgroup members by using one subgroup key, sender u_i needs to find a key chain segment that can be one segment of any of the *Hamiltonian* cycles which contains exactly r subgroup members. Thus, the distance between two consecutive subgroup members in the key chain segment must be the same. Finding the upper bound on the number of subgroup keys for our scheme is equivalent to finding the maximum number of chain segments to set up the subgroup key. Obviously, when the subgroup size $L > s$, at least $L - s$ pair of subgroup members share the same chain segment with the sender. Hence, the upper bound must be less than or equal to s . To prove that the upper bound is s , we only need to create a scenario that needs s key chains. We assume that the sender is u_1 ;

then, the group of receivers is $\mathcal{S} \setminus \{u_1\} = \{u_2, \dots, u_{s-2}, u_s, u_{s+2}, \dots, u_{n-1}\}$, where $|\mathcal{S} \setminus \{u_1\}| = s$. Now, distances between u_1 and each receiver are: $d(u_1, u_2) = 1$, $d(u_1, u_{n-1}) = 2$, $d(u_1, u_4) = 3$, $d(u_1, u_{n-3}) = 4$, \dots , $d(u_1, u_s) = s$. Since each pair of the even index of users and the odd index of users are not contiguous (one from the increased order, another from the decreased order), they cannot use the subgroup key that is derived from the same *Hamiltonian cycle* and none of the key chain segments contains more than two members. Thus, in the worst case, the upper bound of the number of encrypted key messages is s . \square

6.1.4 *Average-case study.* In this section, we propose two group key searching algorithms and use the proposed algorithms to study the average case.

In our proposed algorithms, \mathcal{S} represents the desired subgroup that user $u_i \in \mathcal{S}$ wants to set up, where $\mathcal{S} \subset \mathcal{U}$ and $|\mathcal{S}| = L$. User u_i (the sender) is required to find r ($1 \leq r \leq L$) keys to encrypt session key k' ; then, use the session key to encrypt the message. We propose two searching algorithms to find the subgroup keys for two scenarios. They are: (1) *Addition algorithm*: when $|\mathcal{S}| \leq (|\mathcal{U}| + 1)/2$, we have $L = |\mathcal{S}|$; (2) *Eviction algorithm*: when $|\mathcal{S}| > (|\mathcal{U}| + 1)/2$, we have $L = |\mathcal{U}| - |\mathcal{S}|$. This is equivalent to evict L group members from the system.

We assume that the group members are indexed from 1 to n , where n is the overall group size. We use the following notations that are used in the proposed algorithms:

- \mathcal{S} : subgroup, where group member $i, \dots, j \in \mathcal{S}$. Particularly, we use i to represent the sender, others represent the receivers.
- $d(i, j)$: the distance between any two group members i and j .
- $chain(t)$: t^{th} key chain can be used to derive KEKs, where $1 \leq t \leq s$
- $chain(\hat{t})$: the sorted key chains in the decreased order by the *sizeof(chain(t))*
- $increase(t)$: the key chain t is created by increased searching.
- $decrease(t)$: the key chain t is created by decreased searching.
- $KEK^{(t)}$: KEK derived from key chain t .
- $indicator[j]$: a boolean array that identify the group members, where $j \in \mathcal{S}$ if and only if $indicator[j] == TRUE$.

We now present the addition algorithm:

ALGORITHM 6.3 ADDITION ALGORITHM.

<i>Addition Algorithm</i>	
<i>main function</i>	$create(chain(t))$ $chain(\hat{t}) = sort(chain(t))$ $derive_{KEKs}()$
<i>functions:</i>	$create(chain(t)) \quad \forall j \in \mathcal{S}, t = [(j - i) \bmod s] \text{ and } chain(t) = TRUE$ <i>begin 1: k := 1 to s</i> <i>if ((chain(k) == TRUE))</i>
	<i>...continued on next page</i>

...continued from previous page

```

begin 2: m := 1 to L - 1
  flag := 0
  if (indicator[(i + nk) mod n] == TRUE)
    increase(t)[m] = (i + nk) mod n; flag = 1
  if (indicator[(i + L - nk) mod n] == TRUE)
    decrease(t)[m] = (i + L - nk) mod n; flag = 1
  if (flag == 0) break
end begin 2
end begin 1

chain(t-hat)  sort key chains in a decreased order by the sizeof(chain(t))

derive_KEKs() begin: for each chain(t) in chain(t-hat)
  if (forall j in S \ {i} user j is marked)
    break
  if (exists j in (increase(t) || decrease(t)) j is unmarked)
    mark j;
  i_r := [chainID(i) - sizeof(decrease(t))] mod n;
  j_r := sizeof(increase(t)) + sizeof(decrease(t));
  KEK^(t) := k_{i_r j_r}^(t)
end begin

chainID(i)  begin: k:=1
  if [(kn + d(1, i)) mod t == 0]
    chainID(i) = (kn + d(1, i))/t + 1; break
  else k := k + 1; continue
end begin

```

After a group member executes the addition algorithm, s/he derives a set of subgroup keys $\{key_{i_r j_r}^{(t)}\}$. Using the message format formula presented in Equation 5, s/he can communicate with the desired group members. The addition algorithm involves three functions: $create(chain(t))$, $chain(\hat{t})$, and $derive_KEK()$. The operational complexities for these three functions are $O(sL)$, $O(s \log s)$, and $O(sL)$, respectively. Our simulation shows that the average number of operations for $create(chain(t))$ is approximately 2 ~ 4 times of subgroup size L , and the average number of operations for $derive_KEK()$ is approximately 1 ~ 2 times of subgroup size L . Our simulation is based on the group size $n = 503$ and the subgroup group size $L = 2, \dots, 251$.

When subgroup $\mathcal{S} > (n+1)/2$, we construct the subgroup by evicting $L = n - |\mathcal{S}|$ members from the system. We modify the addition algorithm (only $derive_KEK()$ function is modified) and present the eviction algorithm as follows:

ALGORITHM 6.4 EVICTION ALGORITHM.

<i>Eviction Algorithm</i>	
<i>main function</i>	<i>create(chain(t))</i> <i>chain(t̂) = sort(chain(t))</i> <i>derive_KKEKs()</i>
<i>functions:</i>	
<i>create(chain(t))</i>	<i>same as Algorithm 6.3</i>
<i>chain(t̂)</i>	<i>same as Algorithm 6.3</i>
<i>derive_KKEKs()</i>	<i>begin: for each chain(t) in chain(t̂)</i> <i>if (∀j ∈ S \ {i} user j is marked)</i> <i>break</i> <i>if (∃j ∈ (increase(t) decrease(t)) j is unmarked)</i> <i>mark j;</i> <i>i_r := [chainID(i) + sizeof(increase(t)) + 1] mod n;</i> <i>j_r := n - L - 1;</i> <i>KEK^(t) := k_{i_rj_r}^(t)</i> <i>end begin</i>
<i>chainID(i)</i>	<i>same as Algorithm 6.3</i>

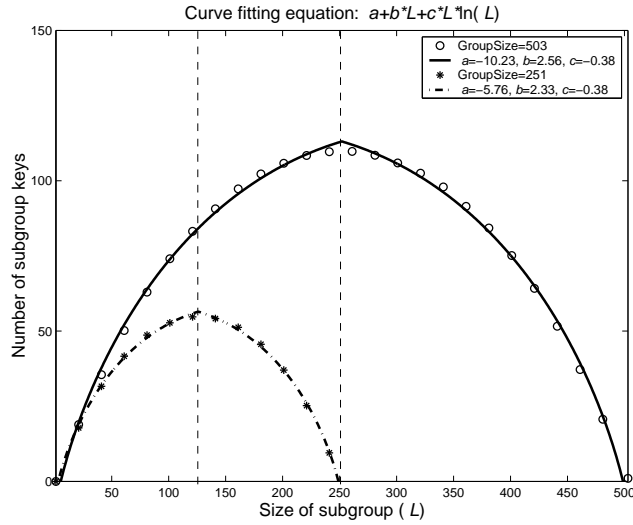
The complexity of the eviction algorithm is the same as addition algorithm.

Using either the addition or the eviction algorithm, each group member can derive the desired subgroup keys. We have simulated the addition and eviction subgroup key searching algorithms. Through regression fitting, we have found that the function $f(L) = a + bL + cL \ln(L)$ matches our simulation data. The simulation results and curve fitting function are shown in Fig. 8. Fig. 8(a) shows the simulation results for the group sizes 251 and 503. Fig. 8(b) shows the parameters of curve fitting functions for the group sizes ranged from 31 to 503; note that in this range of group size, parameter a varies from -2.027 to -10.2344 , parameter b from 1.8627 to 2.56477 , and parameter c from -0.4723 to -0.37534 . In particular, for a group size of 251, we have found that the parameters take the following values: $a = -5.76, b = 2.33, c = -0.38$. The complexity comparison studies are given in Section 7.5.

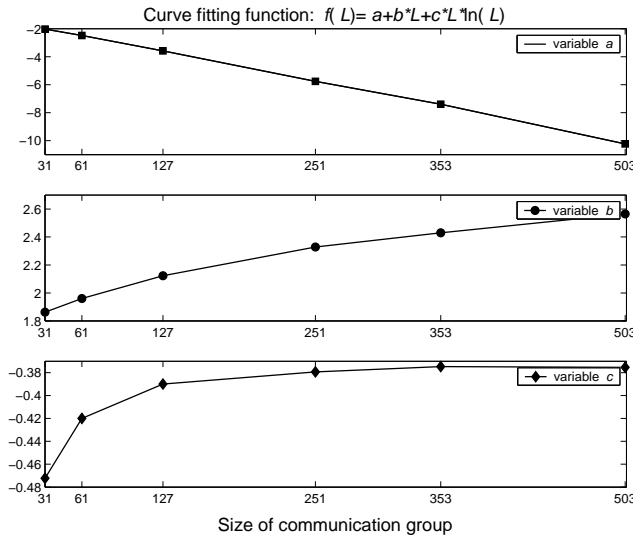
6.2 Unbalanced keying protocol

Unbalanced keying protocol is based on Lemma 5.6, in which the system is prone to give higher preference to one group member. This type of keying protocol is suitable for “one-to-many” communication, in which the selected member is the multicast center. Fig. 3 shows a $n \times (n - 1)$ permutation matrix. If we remove the first row, it is actually $(n - 1) \times (n - 1)$ balanced permutation matrix. Thus, we still can use the keying protocol proposed in Section 6.1 to analyze unbalanced keying protocol. Based on Lemma 5.6 and Lemma 5.7, we have the following corollary.

COROLLARY 6.3. *If there are n multicast listeners (n is prime and $n \geq 5$) and 1 multicast center, using Algorithm 6.1 and Algorithm 6.2, the number of distinct subgroups supported by $n(n - 1)/2$ key chains is $(n^3 - 4n^2 + 7n + 2)/2$, which is the*



(a) Curve fitting example



(b) Key messages function

Fig. 8. Curve fitting

number of subgroups a multicast center can set up.

PROOF. The number of subgroups formed by k of n group members is $\binom{n}{k}$, ($1 \leq k \leq n$). The number of subgroups formed by k of n group members by using Algorithm 6.1 and Algorithm 6.2 is $s * n = n(n - 1)/2$. When $k = \{1, n - 1, n\}$, we have $\binom{n}{k} < n(n - 1)/2$. Thus, there are subgroup overlapping. The number of

Table II. Storage requirement and k/g ratio of different keying schemes

		<i>UBS1</i> ($n \geq 3$)	<i>UBS2</i> ($n \geq 5$)	<i>BS1</i> ($n \geq 1$)	<i>BS2</i> ($n \geq 1$)
Storage requirement	<i>KDC</i>	$n - 1$	$\frac{n^2 - 3n + 2}{2}^\dagger$	n	$\frac{n^2 - n}{2}$
	Group member	$n - 1$	$\frac{n^2 - 3n + 2}{2}^\dagger$	n	$\frac{n^2 - n}{2}$
Number of subgroups		$n^2 - 3n + 4$	$\frac{n^3 - 7n^2 + 18n - 10}{2}^{\dagger\dagger}$	$\frac{n^2 - n + 2}{2}$	$\frac{n^3 - 2n^2 + 3n - 2}{4}$
k/g ratio		$\frac{n-1}{n^2-3n+4}$	$\frac{n^2-3n+2}{n^3-7n^2+18n-10}$	$\frac{2n}{n^2-n+2}$	$\frac{2n}{n^2-n+2}$

† : These equations are adjusted to $n - 1$ multicast listeners based on Lemma 5.6.

†† : This equation is adjusted to $n - 1$ multicast listeners from Corollary 6.3.

subgroups supported by $n(n - 1)/2$ key chains is given as follows:

$$\begin{aligned}
 & s * n * (n - 3) + \binom{n}{1} + \binom{n}{n-1} + \binom{n}{n} \\
 & = (n^3 - 4n^2 + 7n + 2)/2
 \end{aligned}$$

□

7. PERFORMANCE ASSESSMENT OF SECURE GROUP KEYING SCHEME

In this section, we analyze the performance issues related to the group keying scheme in regard to security, storage, and communication.

7.1 Security consideration

Our scheme requires a trusted-third party during the key predistribution phase, in our case – *KDC*. The strength of keying scheme depends on the used hash algorithms (such as *MD5* [Rivest 1992], *SHA-1* [NIST 1995], and so on).

In our case, *KDC* maintains the group member's *ID* and the corresponding group member's key sets. The key label $k_{i_0}^{(t)}$ can uniquely identify a group member, which can be used as group member's *ID*.

One-to-one communication within a group can be authenticated by using subgroup key. The receiver can identify the sender immediately, because only two communication participants can generate the subgroup key $k_{i_1}^{(t)}$. For the subgroup with more than two members, the authentication can only verify the message generated from the legitimate subgroup members.

7.2 Storage complexity and subgroup formation issues

We discuss four keying schemes based on Lemma 5.6, Corollary 6.3, and Lemma 5.7 (or Algorithm 6.1) and Algorithm 6.2. They are:

- *unbalanced scheme 1 (UBS1)*: $n - 1$ keys are allocated to each group member and one group member can set up subgroups more easily (directly) than other group members;
- *unbalanced scheme 2 (UBS2)*: $n(n - 1)/2$ keys are allocated to each group member and one group member can set up subgroups more easily (directly) than other group members;

- *balanced scheme 1 (BS1)*: n keys are allocated to each group member and each group member can set up the same number of subgroups directly;
- *balanced scheme 2 (BS2)*: $n(n - 1)/2$ keys are allocated to each group member and each group member can set up the same number of subgroups directly.

7.2.1 Storage complexity. Table II shows the storage requirements of the group with n members. Due to the derivative relations of a key chain, both the *KDC* and group members keep the same number of keys. This property is beneficial in reducing the storage overhead of the *KDC*. The storage complexity of *UBS1* and *BS1* is $O(n)$. The number of subgroups a user can directly derive is $O(n^2)$. Note that these two keying schemes cannot set up secure subgroup communication by using Corollary 6.1, since a group member does not have enough keys to derive all possible subgroup keys. The storage complexity of *UBS2* and *BS2* is $O(n^2)$. The number of subgroups a user can directly derive is $O(n^3)$. Note that these two keying schemes can set up secure subgroup communication that supports all possible subgroups by using Algorithm 6.3 or Algorithm 6.4.

7.2.2 Zero-delay group management. In our presented secure group keying schemes, no subgroup key distribution during the communication set-up is required as long as the *KDC* distributes the keys to each group member at the system initialization time. Using *USB2* and *BS2*, a group member can derive $O(n^3)$ number of subgroup keys directly. For those subgroup keys that can not be derived directly from pre-distributed keys, a group member can use self-managed subgroup-formation scheme as given from Equation (5) to Equation (7). In the broadcast environment, we assume each group member can receive every transmitted message in the broadcast channel. If the system has a synchronized mechanism, k' that is used the first time can be used as *KEK* for the rest of the subgroup communication session⁵. Thus, it is required to send multiple encrypted k' only within the first synchronized intervals of the subgroup communication session. If the system does not have a synchronized mechanism, it requires the sender to indicate which k' he had used before. Every group member needs to store previously received k' for future transmission, in which each k' has a time-to-live parameter associated with it.

7.3 Complexity analysis of group and subgroup key derivation

Recall that group and subgroup keys are derived from group member's key sets. Before the communication begins, a group member needs to know every other group member. This can be done at the time of a group's initial setup procedure when the *KDC* tells every group member the role of each group member. When a group member sends an encrypted message, it needs to attach group member's *ID* and key $ID - k_{i_j}^{(t)}$ along with the encrypted message. A receiver can quickly derive the proper subgroup key to decrypt the message.

Note that the longest key chain is of length n . From Lemma 5.2, when the subgroup size is k , the maximum required hash operations for a subgroup member is

⁵An election scheme is required when there are multiple first time subgroup session keys received within the same synchronized period. Discussion of the election scheme is outside the scope of this paper.

$k - 1$ and the derivation complexity is $O(k)$. Coppersmith and Jakobsson [Coppersmith and Jakobsson 2002; Jakobsson 2002] have shown that the generation of hash chain has a complexity of $O(\log_2 n)$. They proposed a “pebble” method that can be used on the most frequently used key chains. We assume that every subgroup is formed with equal probability $\frac{1}{2^{n-1}}$; this implies that the probability of forming a subgroup with size k is $\frac{\binom{n}{k}}{2^{n-1}}$. The average subgroup size is given by $\sum_{k=1}^n k \frac{\binom{n}{k}}{2^{n-1}}$. It is easy to show that the average length of derivative operations is $n/2$. By using the scheme proposed by Coppersmith and Jakobsson, it can also be shown that on the average, the number of hash operations is $\lceil \log_2 n - 1 \rceil$.

7.4 Ratio of number of keys to number of subgroups

In order to evaluate the efficiency of our keying scheme, we define the ratio of number of keys at a member to number of subgroups that can be directly generated by a member. We call it k/g ratio, which specifies the number of keys (or secrets) that are required to build a subgroup communication. Note that the ratio k/g is different from $\Delta k/\Delta g$ which was defined in Equation (1), which evaluates the incremental change of a key to the incremental change of number of subgroups. The ratio k/g evaluates the overall efficiency of a keying scheme.

Table II shows the k/g ratio of four difference keying schemes. Note that k/g ratios of four different keying schemes have the same complexity $O(n^{-1})$.

7.5 Comparative studies of secure group communication schemes

To our knowledge, there is no shared-key-based scheme designed specifically for *many-to-many* group communication; however, some of the *one-to-many* group communications schemes can be used for *many-to-many* group communication. We consider two such relevant *one-to-many* group communication schemes for comparison with our scheme: one comes from the information theory community (subset difference scheme [Naor et al. 2001]), and the other scheme from Internet community ((KOS) – OFT scheme [Sherman and McGrew 2003]). We have also considered a naïve scheme in this comparison.

7.5.1 Naïve scheme. The naïve scheme works as follows. During the *key pre-distribution phase*, a pairwise key is pre-distributed for each pair of group members; during the *group communication phase*, a group member u_i broadcast a encrypted message and only desired group members can decrypt the message. The message format is given as:

$$\langle [u_1, u_2, \dots, u_L, E_{k_{i1}}(k'), E_{k_{i2}}(k'), \dots, E_{k_{iL}}(k')], E_{k'}(M) \rangle$$

Upon receiving the message, a receiver u_j looks up the receiver list, locates the proper position to decrypt the session key k' by using the shared pairwise key k_{ij} , and then decrypts the message M .

7.5.2 Subset-difference scheme. The subset-difference scheme is an improved version of *BES* scheme [Fiat and Naor 1994]. During the keys pre-distribution phase, each group member is pre-distributed a set of secrets. In the group communication phase, the *KDC* broadcasts a message; only the legitimate group members can decrypt the message and then derive the group key. Naor et al [2001] proposed the

subset-difference scheme. In this scheme, a complete binary key tree is constructed and receivers are viewed as leaves. The collection of subsets S_1, \dots, S_w defined by this algorithm corresponds to subsets of the form “a group of receivers G_1 minus another group G_2 ”, where $G_2 \subset G_1$. The two groups G_1, G_2 correspond to leaves in two full binary subtrees. Therefore, a valid subset S is represented by two nodes in the tree (v_i, v_j) such that v_i is an ancestor of v_j that denoted as $S_{i,j}$. A leaf u is in $S_{i,j}$ if and only if it is in the subtree rooted at v_i but *not* in the subtree rooted at v_j ; in other words, $u \in S_{i,j}$ if and only if v_i is an ancestor of u but v_j is not. Using the key assignment scheme proposed by Noar et al, each group member is required to store $\frac{1}{2} \log^2 n + \frac{1}{2} \log n + 1$ keys, and the length of the broadcast message is at most $2L - 1$.

7.5.3 One-way function tree (OFT) scheme. The OFT scheme proposed a binary key tree structure. Group members are all located at the leaves in the key tree. Two keys are associated with each node of the key tree. Each internal node i has two children, namely $left(i)$ and $right(i)$. The node key for node i is represented by the following formula:

$$k_i = f(g(k_{left(i)}), g(k_{right(i)}))$$

where, function g is one-way, and f is a mixing function. Ancestors of a node are those nodes in the path from its parent node to the root. The set of ancestors of a node is called *ancestor set* and the set of siblings of the nodes in *ancestor set* are called *sibling set*. Each member receives the key (associated to its leaf node), its sibling’s blinded key (processed by one-way function g) and the blinded keys corresponding to each node in its *sibling set*. For a balanced tree, each member stores $\log_2 n + 1$ keys. When a node’s key changes, the new key must be encrypted with its two children’s key and the blinded key changed in a node has to be encrypted only with the key of its sibling node. Sherman and McGrew [2003] proposed an improved method to reduce the broadcast size and the computational effort of multiple additions and evictions. They call the corresponding operations as bulk addition and bulk eviction. Before the group operations, the key server constructs a *Combined Ancestor Tree (CAT)* who is on the tree of ancestors of the affected leaves. After a batch operation, the key server must broadcast the blinded secrets of all nodes on the CAT and the size of this broadcast is the size of the CAT. The lower bound and upper bound of the size of CAT s_L is given as:

$$2L - 1 + \lceil \log_2(n/L) \rceil < s_L < 2L - 1 + L \lceil \log_2(n/L) \rceil \quad (8)$$

where L is number of leaves in the CAT (number of added or deleted members).

7.5.4 Comparisons of many-to-many communication schemes. We note that most of the proposed secure group keying schemes are tailored for *one-to-many* communications [Dondeti et al. 2000]. In case of *many-to-many* communication, the key management overhead is proportional to the number subgroups. Moreover, setting up subgroup communication via trusted-third party (*KOS* and *BES*) will introduce additional communication delays between the subgroup members and centralized servers, making it difficult for realtime/delay-sensitive applications.

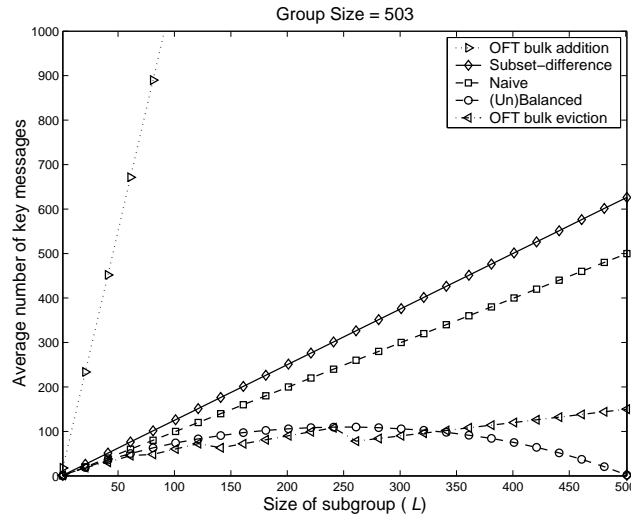
In Table III, we compare our proposed balanced scheme with two centralized group keying schemes (one from *KOS* – OFT scheme, another from *BES* – subset

Table III. Comparison Table of *many-to-many* secure group keying schemes

Scheme/ Feature	Group Management	Message size (bulk)		Storage	
		addition	eviction	KDC	member
OFT	Centralized	$s_L K + L(dK + I)$	$s_L K + LI$	$(2n - 1)K$	$(d + 1)K$
Subset-diff	Centralized	n/a	$(2L - 1)(K + I)$	$(2n - 1)K$	$(\frac{1}{2}d^2 + \frac{1}{2}d + 1)K$
Scheme/ Feature	Group Management	Group formation Message (bulk)		Storage	
				KDC	member
Naïve	Self-managed	$L(K + I)$		$\frac{n(n-1)}{2}K$	$(n - 1)K$
<i>BS2/UBS2</i>	Self-managed	$[a + bL + cL \ln(L)]K$		$\frac{n(n-1)}{2}K$	$\frac{n(n-1)}{2}K$

Notation for Table III

n	number of members in the group
I	number of bits in member id
d	depth of a key tree (for a balanced tree $d = \log_2 n$)
K	size of a key in bits
L	size of a subgroup
s_L	size of the CAT
a, b, c	constants specified in Fig. 8(b)

Fig. 9. Comparison of number of key messages for *many-to-many* communication schemes

difference scheme). In addition, we have constructed a naïve scheme, in which a unique key is distributed for each pair of group members. According to the OFT scheme proposed by Sherman and McGrew [2003], on average, the s_L is close to the lower bound for bulk addition and s_L is about 15% of the upper bound for bulk eviction (the lower/upper bound is shown in Equation 8). According to the authors [Naor et al. 2001], in the subset-difference scheme, the average-case is $1.25L$ for randomly selecting the group members. In our scheme, i.e., the (un)balanced scheme, the values of coefficients a, b, c for the curve fitting function are shown in Fig. 8, where $|\mathcal{U}| \leq 503$.

Fig. 9 shows that the communication overhead invoked by our scheme is the

minimum among the compared schemes. The most expensive operation is the bulk addition operation of the OFT scheme. Although, the bulk eviction operation is less expensive, the frequent group formations may involve both bulk addition and eviction. The communication overhead invoked by the subset-difference scheme is even higher than the naïve scheme. From this comparison study, we can infer that minimizing the storage overhead for group members comes at the expense of increased communication overhead.

8. CONCLUSION

In this paper, we have proposed a new secure group communication keying scheme for *many-to-many* group/sub-group communication. Our proposed scheme is suitable for applications that have the following three requirements:

- frequent subgroup set up for communications within a moderate size group without leaving the overall group
- the entire population change of the system is not prominent
- delay sensitive set-up for secure subgroup communication

We propose two schemes for secure group communication. Our proposed unbalanced keying scheme can be used in the multicast environment, in which a multicast center initiates most of the communication. We also present a balanced keying scheme that treats each group member equally where every group member can initiate the group/subgroup communication. Furthermore, our proposed balanced keying scheme requires $O(n^2)$ storage space, and more importantly, supports *many-to-many* secure subgroup communication. Our performance comparison studies show that the proposed schemes has minimal communication overhead of key set-up message.

There are two limitations of our proposed schemes. When the entire group formation is uncertain. This means the population of the system is unstable, and group members in the system keep changing continually. Then, this scenario can be considered as a situation where the population of a system is infinite and our proposed schemes are not designed to handle infinite population. The second limitation is that the group size is limited by storage capacity of group member's personal device. In order to solve this problem, we need to build multiple levels key-tree structure, in which multiple groups compose a global group. The multiple levels key-tree structure will be considered in a subsequent paper.

ACKNOWLEDGMENTS

The authors thank Lein Harn for his help and guidance when the authors first touched the research area of secure group communication. The authors also thank Amit Sinha for his feedback on the construction of the keying schemes. The authors also thank Jane Vogl and Balaji Krithikaivasan for proof reading of an earlier version of this paper. Finally, the authors would like to thank anonymous reviewers for their valuable, detailed comments that improve both the content and representation of this paper.

APPENDIX

We present some preliminary definitions of *permutation* theory are given in [Skiena 1990].

Definition A.1 Permutation. The rearrangement of elements in an ordered list \mathcal{S} into a one-to-one correspondence with \mathcal{S} itself, is also called an “arrangement number” or “order”. The number of permutations on a set of n elements is given by $n!$.

For example, there are $2! = 2 \cdot 1 = 2$ permutations of $\{1, 2\}$, namely $\{1, 2\}$ and $\{2, 1\}$, and $3! = 3 \cdot 2 \cdot 1 = 6$ permutations of $\{1, 2, 3\}$, namely $\{1, 2, 3\}$, $\{1, 3, 2\}$, $\{2, 1, 3\}$, $\{2, 3, 1\}$, $\{3, 1, 2\}$, and $\{3, 2, 1\}$.

Definition A.2 Permutation group. A permutation group is a finite group G whose elements are permutations of a given set and whose group operation is composition of permutations in G . Permutation groups have orders dividing $n!$.

Definition A.3 Permutation cycle. A permutation cycle a subset of a permutation whose elements trade places with one another. Permutations cycles are called “orbits” by Comtet ([1974], p. 256).

For example, in the permutation group $\{4, 2, 1, 3\}$, (143) is a 3-cycle and (2) is a 1-cycle. Here, the notation (143) means that starting from the original ordering $\{1, 2, 3, 4\}$, the first element is replaced by the fourth, the fourth by the third, and the third by the first, i.e., $1 \rightarrow 4 \rightarrow 3 \rightarrow 1$. In this paper, we use curly braces “ $\{\}$ ” to represent the permutation cycle, e.g., the permutation cycle $\{1, 4, 3\}$.

REFERENCES

- ALVES-FOSS, J. 2000. An efficient secure authenticated group key exchange algorithm for large and dynamic groups. In *Proceedings 23rd National Information Systems Security Conference (NISSC)*. National Institute of Standards and Technology, National Computer Security Center, Baltimore, MD, USA, 254 – 266.
- ATENIESE, G., STEINER, M., AND TSUDIK, G. 1998. Authenticated group key agreement and friends. In *Proceedings of the 5th ACM conference on Computer and communications security*. ACM Press New York, NY, USA, San Francisco, California, United States, 17 – 26.
- BALLARDIE, T. 1996. Scalable multicast key distribution. *RFC 1949*.
- BERKOVITS, S. 1992. How to broadcast a secret. *Advances in Cryptology: EUROCRPT '91, Lecture Notes in Computer Science 547*, 536 – 541.
- BLOM, R. 1985. An optimal class of symmetric key generation systems. In *EUROCRYPT'84, Lecture Notes in Computer Science*, vol. 209. Springer-Verlag, Paris, France, 335–338.
- BLUNDO, C. AND CRESTI, A. 1995. Space requirements for broadcast encryption. In *Advances in Cryptology: EUROCRYPT '94, Lecture Notes in Computer Science*. Springer-Verlag, New York, United States, 287–298.
- BLUNDO, C., MATTOS, L. A. F., AND STINSON, D. R. 1994. Multiple key distribution maintaining user anonymity via broadcast channels. *Journal of Computer Security* 3, 4, 309 – 323.
- BLUNDO, C., MATTOS, L. A. F., AND STINSON, D. R. 1996. Trade-offs between communication and storage in unconditionally secure schemes for broadcast encryption and interactive key distribution. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*. Springer-Verlag, Santa Barbara, California, USA, 387 – 400.
- BLUNDO, C., SANTIS, A. D., HERZBERG, A., KUTTEN, S., VACCARO, U., AND YUNG, M. 1993. Perfectly-secure key distribution for dynamic conferences. *Lecture Notes in Computer Science 740*, 471–486.

- BLUNDO, C., SANTIS, A. D., HERZBERG, A., KUTTEN, S., VACCARO, U., AND YUNG, M. 1998. Perfectly-secure key distribution for dynamic conferences. *Information and Computation* 146, 1, 1–23.
- BLUNDO, C., SANTIS, A. D., AND VACCARO, U. 1996. Randomness in distribution protocols. *Information and Computation* 131, 2, 111–139.
- BRISCOE, B. 1999. Marks: Zero side effect multicast key management using arbitrarily revealed key. In *Networked Group Communication (NGC'99)*. Springer-Verlag, Pisa, Italy.
- BURMESTER, M. AND DESMEDT, Y. 1995. A secure and efficient conference key distribution system. *Proceedings of Eurocrypt' 94, LNCS 950*, 275 – 286.
- BURMESTER, M. AND DESMEDT, Y. 1996. Efficient and secure conference-key distribution. In *Security Protocols Workshop*. Springer-Verlag, Cambridge, UK, 119 – 129.
- COMTET, L. 1974. *Advanced Combinatorics: The Art of Finite and Infinite Expansions*. Dordrecht, Boston, D. Reidel Pub. Co., Netherlands.
- COPPERSMITH, D. AND JAKOBSSON, M. 2002. Almost optimal hash sequence traversal. In *Finacial Cryptography*. InterVarsity Press, Southampton, Bermuda.
- DONDETI, L. R., MUKHERJEE, S., AND SAMAL, A. 1999. Survey and comparison of secure group communication protocols. Tech. rep., University of Nebraska-Lincoln.
- DONDETI, L. R., MUKHERJEE, S., AND SAMAL, A. 2000. A distributed framework for scalable secure many-to-many communication. In *Fifth IEEE Symposium on Computers and Communications*. Antibes-Juan les Pins, France.
- FIAT, A. AND NAOR, M. 1994. Broadcast encryption. In *CRYPTO'93*. Lecture Notes in Computer Science, vol. 773. Springer-Verlag New York, Inc., Santa Barbara, California, United States, 480–491.
- GONG, L. AND SHACHAM, N. 1994. Elements of trusted multicasting. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*. Fairfax, Virginia, 176–183.
- GONG, L. AND WHEELER, D. I. 1990. A matrix key-distribution scheme. *Journal of Cryptology* 2, 1, 51 – 59.
- HARARY, F. 1969. *Graph Theory*. Addison-Wesley Publishing Company, Inc.
- HARNEY, H. AND MUCKENHIRN, C. 1997. Group key management protocol (gkmp) architecture. *RFC 2094*.
- JAKOBSSON, M. 2002. Fractal hash sequence representation and traversal. In *IEEE International Symposium on Information Theory*.
- JUST, M., KRANAKIS, E., KRIZANC, D., AND VAN OORSCHOT, P. 1994. On key distribution via true broadcasting. In *Proceedings of the 2nd ACM Conference on Computer and communications security*. 81–88.
- KIM, Y., PERRIG, A., AND TSUDIK, G. 2000. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *ACM Conference on Computer and Communications Security*. 235–244.
- KIM, Y., PERRIG, A., AND TSUDIK, G. 2001. Communication-efficient group key agreement. Tech. rep., Department of Information and Computer Science, University of California, Irvine.
- KORJIK, V., IVKOV, M., MERINOVICH, Y., BARG, A., AND VAN TILBORG, H. C. A. 1995. A broadcast key distribution scheme based on block designs. In *IMA Conf.* 2–12.
- KUROSAWA, K., OKADA, K., AND SAKANO, K. 1995. Security of the center in key distribution schemes. *Advances in Cryptology: ASIACRYPT '94, Lecture Notes in Computer Science*.
- LEIGHTON, T. AND MICALI, S. 1994. Secret-key agreement without public-key cryptography. *Advances in Cryptology: CRYPTO '93, Lecture Notes in Computer Science* 773, 456 – 479.
- MITTRA, S. 1997. A framework for scalable secure multicasting. In *ACM SIGCOMM*. 277–288.
- MOYER, M. J., RAO, J. R., AND ROHATGGI, P. 1999. A survey of security issues in multicast communications. *IEEE Network* 13, 6, 12 – 23.
- NAOR, D., NAOR, M., AND LOTSPIECH, J. 2001. Revocation and tracing schemes for stateless receivers. *Lecture Notes in Computer Science* 2139, 41–62.
- NIST. 1995. Secure hash standard. *FIPS PUB 180-1*.

- RAFAELI, S. AND HUTCHISON, D. 2003. A survey of key management for secure group communication. *ACM Computing Surveys* 35, 3, 309–329.
- RIVEST, R. L. 1992. The md5 message-digest algorithm. *RFC 1321*.
- SHERMAN, A. T. AND MCGREW, D. A. 2003. Key establishment in large dynamic groups using one-way function trees. *IEEE Transactions on Software Engineering* 29, 5, 444–458.
- SKIENA, S. 1990. *Implementing Discrete Mathematics*. Addison-Wesley Publishing Company.
- SNOEYINK, J., SURI, S., AND VARGHESE, G. 2001. A lower bound for multicast key distribution. In *IEEE INFOCOM*. Vol. 1. 22–26.
- STEINER, M., TSUDIK, G., AND WAIDNER, M. 1996. Diffie-hellman key distribution extended to group communication. In *ACM Conference on Computer and Communications Security*. 31–37.
- STEINER, M., TSUDIK, G., AND WAIDNER, M. 1998. CLIQUES: A new approach to group key agreement. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98)*. IEEE Computer Society Press, Amsterdam, 380–387.
- STINSON, D. R. 1997. On some methods for unconditionally secure key distribution and broadcast encryption. *Designs, Codes and Cryptography* 12, 3, 215–243.
- STINSON, D. R. AND VAN TRUNG, T. 1998. Some new results on key distribution patterns and broadcast encryption. *Designs, Codes and Cryptography* 14, 3, 261–279.
- WALDVOGEL, M., CARONNI, G., SUN, D., WEILER, N., AND PLATTNER, B. 1999. The versa-key framework: Versatile group key management. *IEEE Journal on Selected Areas in Communications* 17, 9, 1614 – 1631.
- WALLNER, D. M., HARDER, E. J., AND AGEE, R. C. 1999. Key management for multicast: Issues and architectures. *RFC 2627*.
- WONG, C. K., GOUDA, M., AND LAM, S. S. 2000. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking* 8, 1, 16–30.