A Hybrid Group Key Management Protocol for Reliable and Authenticated Rekeying

Depeng Li and Srinivas Sampalli
(Corresponding author: Srinivas Sampalli)
Faculty of Computer Science, Dalhousie University
6050 University Avenue, Halifax, Nova Scotia B3H 1W5, Canada (Email: srini@cs.dal.ca)

(Received Mar. 3, 2006; revised and accepted Mar. 18, 2006)

Abstract

We present a hybrid group key management protocol that incorporates both a centralized and a contributory scheme for key management, and a tree-based Elliptic Curve Diffie-Hellman technique for group key updates. It combines the advantage of the centralized approach's efficiency and the contributory scheme's fault tolerance. In addition, our rekeying algorithm updates the group key in the presence of any sequence of node failures, node restorations, and membership changes. Furthermore, our scheme incorporates a reliable and authenticated rekeying message transport method. Performance analysis indicates that our protocol reduces computational costs and communication overhead as compared to other popular protocols.

Keywords: Authenticated multicast, group key management, periodic batch rekeying, reliability

1 Introduction

There has been a growing demand in the past few years for security in collaborative environments deployed for emergency services, as well as many applications in military, business, government and research organizations [5]. Many of these applications involve secure group communications. To protect group communication data against passive and active adversaries, providing confidentiality becomes one of the top concerns. To satisfy this requirement, a common and efficient solution is to deploy a symmetric group key shared by all group application participants. Whenever a member leaves or joins the group, or whenever a node failure or restoration occurs, the group key should be updated to provide forward and backward secrecy. Therefore, a key management protocol that computes the symmetric group key and forwards the rekeying messages to all legitimate group members is central to the security of the group application.

1.1 Centralized and Contributory Group Key Management Schemes

A number of group key management schemes have been proposed for network group applications. They can be broadly classified into two categories, namely, centralized [10, 12, 13, 18, 21, 22] and contributory [2, 4, 5, 6].

In a typical centralized group key management scheme, a trusted third party, known as the key server, is responsible to generate, to encrypt and to distribute the symmetric group key, auxiliary keys and individual keys to all other group members. It has the advantages of efficiency of the symmetric key encryption/decryption [10, 18, 21, 22]. However, it suffers from the following drawbacks. 1) Since all group secrets are generated and stored in one place, the key server could present itself as an attractive attack target for adversaries. 2) The key server can become the single point of failure/bottleneck. 3) The authentication technique for rekeying messages may result in high communication overheads and may not guarantee the reliable delivery of rekeying messages [17, 18].

In contrast, in contributory group key management schemes, every group member contributes to the group key generation. It has the advantage of fault-tolerance. However, for group membership changes, it lacks scalability in terms of computational cost. For example, Tree-based Group Diffie-Hellman (TGDH) [5] is one of the most efficient contributory group key agreement techniques. It still has the following drawbacks. 1) Every group member performs the expensive public key Diffie-Hellman key exponentiation $[1, O(\log_2 n)]$ times for every group membership update where n is the group size. 2) Every sponsor should sign and forward a large number of rekeying multicast messages to update a group key. This results in expensive communication overhead and computational costs.

Furthermore, previous well-known contributory group key managements [2, 4, 5, 6] do not address reliable multicasting of group rekeying messages. In addition, although previous approaches address the authentication of uni-

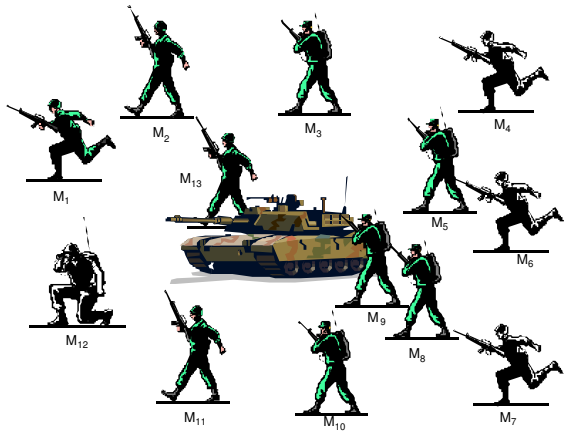

Figure 1: Battelfield scenario

cast rekeying messages, they do not include schemes for efficient amortized authentication of multicast messages. This is of importance because most of rekeying messages are forwarded via multicast services.

1.2 Motivation

In many secure group applications, a hybrid of centralized and contributory schemes may be required. In such cases, the group key management should be both efficient and failure-tolerant. We describe one example from a military scenario (Figure 1). A collection of wireless mobile devices carried by soldiers or a tank cooperate in relaying packets. In such a scenario, mobile nodes dynamically establish routes among themselves to form their own network "on the fly". However, all nodes except the one with the tank, have limited battery power and processing capacities. For the sake of power-consumption and computational efficiency, the tank can work as the key server while a centralized group key management scheme is deployed.

However, it is possible for the tank to be out of service during adverse conditions. Alternatively, some of the soldiers, for example M_4, M_6, and M_7 in the example, may not be able to be in contact with the tank because they are too far away from the tank, or, because M_5, the node to relay communication data, is out of service. In case of such emergencies, a straightforward solution is to launch a contributory group key agreement, for example, TGDH [5] or TECH [7], among the nodes which cannot be in contact with the centralized key server. But, the disadvantages are: 1) every node should install two sets of group key management schemes, 2) every node should store two sets of individual keys, auxiliary keys and group keys, 3) it takes up critical time to generate the new contributory group key, and 4) generating the new group key requires the additional computational cost and communication overhead.

1.3 Objective

In this paper, we propose a novel hybrid group key management protocol that removes the above drawbacks. In particular, our protocol enables fast switching between centralized and contributory schemes with minimal communication costs and computational overheads, and uses less number of keys. Furthermore, our contributory scheme reduces the number of rekeying messages and, therefore decreases the number of signing operations. The main contributions of our approach are given below.

1) *Hybrid Architecture:* We introduce a hybrid architecture in which the centralized and the contributory group key management schemes use a Logical Key Hierarchy (LKH)-based management technique to maintain one set of keys in a binary key tree. When the key server is online, the computationally efficient and reliable centralized management is deployed. While the key server is out of service for all group members or part of the group members and to handle network disconnections, we propose an efficient contributory scheme to update the group key shared by the group members that cannot connect with the key server.

2) *Contributory Rekeying:* Our protocol includes an efficient contributory rekeying scheme based on the dominating path concept to handle *J Join & L Leave* scenario for a binary key tree. It handles periodic batch rekeying, multi-group merging and group partitioning.

3) *Authentication of Rekeying Messages:* Our proposal signs multicast rekeying messages via Modified Signature Amortization using Information Dispersal Algorithm (M-SAIDA) for protection against active adversaries. A reliable function is also presented to guarantee that all rekeying messages are received by all group members.

The rest of this paper is organized as follows. We provide a literature review of previous works in Section 2. Requirements and background are introduced in Section 3. Our proposal including the hybrid architecture and the contributory rekeying is described in Section 4. How to authenticate rekeying messages without packet loss for both partial keys and authentication information is presented in Section 5. Performance analysis is presented in Section 6. Concluding remarks are given in Section 7.

2 Related Works

Various cryptographic techniques have been proposed to address the group key management problem (e.g. [15]). In this section, we review centralized and contributory group key managements and the efficient authentication of reliable rekeying messages.

Centralized schemes: A single entity is required to generate the group key, and then to distribute it to other group members. It deploys inexpensive approaches such as private-key encryption/decryption to encrypt rekeying messages. But the entire group will be affected if there is a problem with the centralized controller (e.g., the key server) although some previous proposals (e.g. [10, 12, 13, 18, 21, 22]) achieve good results. Logical Key Hierarchy [18] (LKH) uses a Key Distribution Center (KDC) to maintain a key tree. When the membership updates, new auxiliary and group keys are generated and encrypted with keys held by current group members. The rekeying messages will contain at most $2log_2n$ keys. Efficient Large-Group Key [10] (ELK) uses a hierarchical tree. This novel and sophisticated agreement applicable to large groups, refreshes its key tree in intervals for the group member join. Keygem [21, 22] follows a periodic batch rekeying method, where all members should have their time synchronized and agree on a rekeying period. Furthermore, they present a reliable multicast scheme to forward rekeying messages.

Contributory schemes: The group key is generated in a contributory fashion, where all members contribute their own shares to compute the group key. Unlike the centralized flavor, it is fault tolerant. But these schemes typically use expensive public key operations. According to the performance analysis and experiments [5, 6], the most efficient proposals are Tree-based Group Diffie-Hellman [5] (TGDH) and Skinny TRee [6] (STR).

TGDH takes advantage of the hierarchy of the binary tree. Each leaf node in the tree represents a group member, say, M_i. The internal nodes are used for the key management. The node key associated to the node, for example, (l, v), is $k_{(l,v)}$ and its blinded key $b_{(l,v)} = \alpha^{k(l,v)}$. For each internal node (l, v), its associated key $k_{(l,v)}$ is derived from its children's keys in the following way:

$$k_{(l,v)} = b_{(l+1,2v+1)}^{k(l+1,2v)} = \alpha^{k(l+1,2v)k(l+1,2v+1)}.$$

STR's group key structure is similar to that of TGDH. The new group member is always treated as the current root's sibling and a new root node is created which works as the former root and the new member's parent. It shows computational and communication efficiency for group membership additions but not always for group membership deletions. To satisfy frequent join/leave requests, individual rekeying focuses on the reduction of communication latency since the instant group key updating scheme is deployed. STR reduces the round number of the group key management.

Authentication: Previous contributory group key agreements propose a number of authentication techniques to protect rekeying messages against insider active attacks. For example, SA-GDH [1] authenticates GDH.3 [3] protocol. In TGDH [5] and STR [6], authenticated channels are assumed and rekeying messages are digitally signed by the sender with some efficiently strong public key signature methods such as DSA or RSA to guarantee the non-repudiation service. Digital signatures are computationally expensive. Simulation results of Wong and Lam [18] show that devoting all the processor time of a Pentium II 300 MHz machine can only generate 80 512-bit RSA signatures per second and that a digital signature operation is around two orders of magnitude slower than a key encryption using DES.

Some other authenticated group key agreements are also proposed: An ID-based authentication is proposed in [20] which does not need a dedicated server or group controller. Implicitly Certificated Public Keys method (ICPK) [11] modified by Shortened Digital Signatures Signcryption (SDSS) [23] is introduced to reduce the overhead of the certificate validation checking process and to improve computational efficiency.

Wong and Lam introduce the authentication scheme benefiting from the authentication tree with the usage of the amortized signature method. However, its communication overhead is large and every packet needs verification.

Reliability: To provide no-packet-loss service, Keygem [21, 22] deploys the FEC and ACK techniques. The Tornado code technique is appropriate when a large number (hundreds to thousands) of segments are being encoded as compared to IDA [16]. However, IDA has the advantage of less communication overhead.

3 Preliminaries

In this section, we briefly introduce security requirements for group key management, the Tree-based Elliptic Curve Diffie-Hellman (TECH) [7] and the *Marking* algorithm [21], which form the necessary background for our proposal.

3.1 Security Requirements

Two basic requirements of a secure group key management protocol are *Forward Secrecy* and *Backward Secrecy*.

Forward Secrecy: Previous group members who know contiguous subsets of old group keys must not be able to discover subsequent group keys after they leave the group.

Backward Secrecy: Current group members who know a contiguous subset of current group keys must not be able to discover preceding group keys.

Therefore, every group membership request such as *join/leave* should be processed immediately. However, the individual rekeying strategy introduces inefficiency and *out-of-sync* problems [21], especially for resource-limited networks. For example, suppose that a set of malicious users repeatedly request to join and then to leave the group immediately. Because each rekeying mes-

sage should be digitally signed to guarantee data source authentication, we find that frequent rekeying transactions introduce a number of signing operations which are heavyweight [18]. If there are *J Join & L Leave* operations during a rekeying interval, $J+L$ times signing operations are performed to maintain the key tree [18].

In contrast to individual rekeying, a periodic batch rekeying strategy [21] can alleviate the out-of-sync problem and to improve the efficiency. In this technique, all join and leave requests are processed in a batch at the end of each rekeying interval. It means that the group members that need to leave can stay longer and the new group members have to join later. In [21], the authors define the concept of a vulnerability window which is the period of time starting at the request of the join or leave and ending at the end of the rekeying interval. If the *vulnerability window* is too long, security can be compromised. Thus, periodic batch rekeying is a tradeoff between the group key security and the efficiency.

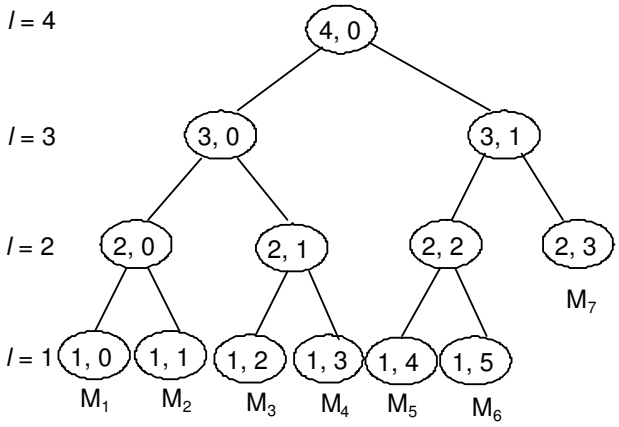

Figure 2: Binary key tree

3.2 Tree-based Elliptic Curve Diffie-Hellman Key Exchange [7]

Our protocol uses a Tree-based Elliptic Curve Diffie-Hellman (TECH) algorithm for updating the group key. Details of this algorithm can be found in our previous work [7]. For the sake of completeness, we provide a brief overview of the algorithm in the following. A binary tree **T** is a key tree in which every node can be denoted as $< h, i >$ where h is the height (level) of the node and i is the index of the node at level h. Thus, every node is uniquely identified. Each node $< h, i >$ is associated with a private key, $\boldsymbol{PV}_{<h,i>}$, and a public key, $\boldsymbol{PB}_{<h,i>}$. The $\boldsymbol{PB}_{<h,i>}$ is computed from the private key $\boldsymbol{PV}_{<h,i>}$, from Equation (1) below where $\boldsymbol{P}$ is a base point of an Elliptic Curve Equation $\boldsymbol{E}$, $\bullet$ is the scalar multiplication operation, and both $\boldsymbol{E}$ and $\boldsymbol{P}$ are shared by all group members and the key server in advance.

$$\boldsymbol{PB}_{<h,\boldsymbol{i}>} = \boldsymbol{PV}_{<h,\boldsymbol{i}>} \bullet \boldsymbol{P}. \quad (1)$$

Without loss of generality, we call $\boldsymbol{PV}_{<\boldsymbol{h},\boldsymbol{0}>}$ the *group key*. There are two kinds of nodes in a binary tree **T**. One is the leaf, $< h, i >$, which is associated with one and only one group member. We say that the leaf $< h, i >$ represents the group member M_i. The private key of one leaf is defined by the following rule, where $\boldsymbol{r_i}$ is a random integer generated by the group member M_i:

$$\boldsymbol{PV}_{<h,\boldsymbol{i}>>} = \boldsymbol{r_i} \bullet P.$$

The other is the intermediate node which has two children. It does not represent any group member but represents a sub-group in which every sub-group member hosts it. The intermediate node's private key is treated as the *sub-group key*. It can be calculated by the following rule where node $< h, i >$'s two children are $< h-1, 2i >$ and $< h-1, 2i+1 >$:

$$\begin{aligned} \boldsymbol{PV}_{<h,\boldsymbol{i}>} &= \mathrm{X_{co}}(\boldsymbol{PV}_{<h-1,\boldsymbol{2i}>} \bullet \boldsymbol{PB}_{<h-1,\boldsymbol{2i+1}>}) \\ &= \mathrm{X_{co}}(\boldsymbol{PV}_{<h-1,\boldsymbol{2i+1}>} \bullet \boldsymbol{PB}_{<h-1,\boldsymbol{2i}>}) \\ &= \mathrm{X_{co}}(\boldsymbol{PV}_{<h-1,\boldsymbol{2i}>} \bullet \boldsymbol{PV}_{<h-1,\boldsymbol{2i+1}>} \bullet P), \end{aligned}$$

where $\mathrm{X_{co}}$ is the x-coordinate of the point represented within the parentheses. The concepts *key-path*, *sibling path and key sub-path* are defined below.

Key path: A path starting at the leaf hosted by $\mathrm{M_i}$ and ending at the key tree's root. M_i hosts all nodes on its *key path*, namely, $\mathrm{KP_i}$. All $\boldsymbol{PBs}$ and $\boldsymbol{PVs}$ on the *key path*, $\mathrm{KP_i}$, are denoted as $\boldsymbol{PBs}^*$@$\mathrm{KP_i}$ and $\boldsymbol{PVs}^*$@$\mathrm{KP_i}$, respectively.

Sibling path: Siblings corresponding with every node on $\mathrm{M_i}$'s *key-path* constitute $\mathrm{M_i}$'s sibling path, namely, $\mathrm{SP_i}$. All $\boldsymbol{PBs}$ and $\boldsymbol{PVs}$ on the *sibling path*, $\mathrm{SP_i}$, are denoted as $\boldsymbol{PBs}^*$@$\mathrm{SP_i}$ and $\boldsymbol{PVs}^*$@$\mathrm{SP_i}$.

Key sub-path: For group member $\mathrm{M_i}$, a path starting at any node $\mathrm{N_x}$ and ending at any other node $\mathrm{N_y}$, on the key path $\mathrm{KP_i}$ is called a key sub-path. All $\boldsymbol{PBs}$ and $\boldsymbol{PVs}$ on *key sub-path*, $\mathrm{KSP_{i,x,y}}$, are denoted as $\boldsymbol{PBs}^*$@$\mathrm{KSP_{i,x,y}}$ and $\boldsymbol{PVs}^*$@$\mathrm{KSP_{i,x,y}}$, respectively.

In both the centralized and contributory group key management schemes, every group member, $\mathrm{M_i}$, should store $\boldsymbol{PVs}^*$@$\mathrm{KP_i}$, and $\boldsymbol{PBs}^*$@$\mathrm{SP_i}$. Every node also is aware of the entire structure of the key tree.

In addition, in our centralized group key management, the key server stores all $\boldsymbol{PBs}$ and $\boldsymbol{PVs}$ on the key tree **T**, similar to previous approaches.

Figure 2 shows an example of a key tree.

Node $< 3, 0 >$, is hosted by the group members, $\mathrm{M_1}, \mathrm{M_2}, \mathrm{M_3}$, and $\mathrm{M_4}$ which are represented by the leaves$< 1, 0 >, < 1, 1 >, < 1, 2 >$, and $< 1, 3 >$ respectively. We say that $\mathrm{M_1}, \mathrm{M_2}, \mathrm{M_3}$, and $\mathrm{M_4}$ construct sub-group

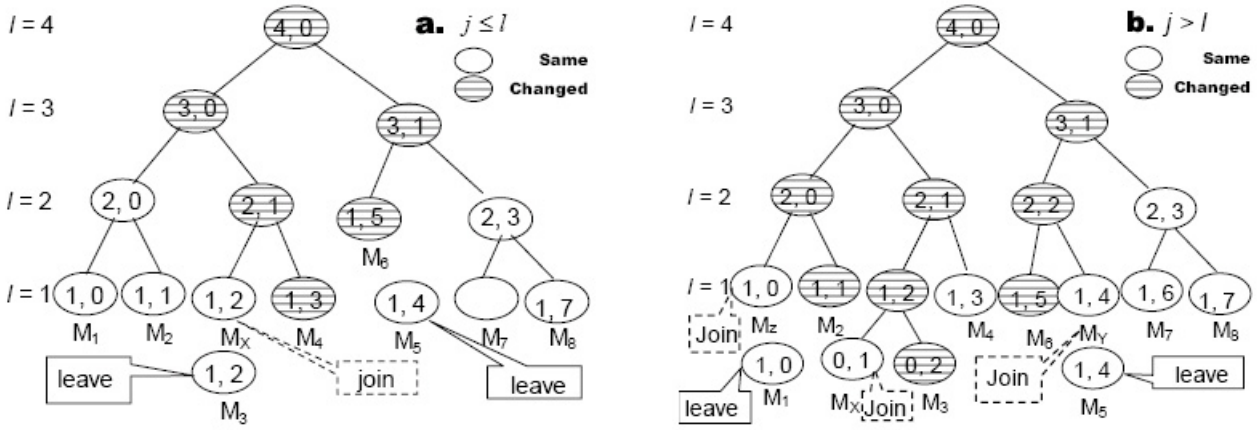

Figure 3: J Join/L Leave scenario handled by marking algorithm

$< 3,0 >$. Then, $PV_{<3,0>}$ is the *sub-group key* of sub-group $< 3,0 >$.

Group member M_2's *key-path:* $KP_2 = \{< 1,1 >, < 2,0 >, < 3,0 >, < 4,0 >\}$, its *sibling-path:* $SP_2 = \{< 1,0 >, < 2,1 >, < 3,1 >\}$. So, M_2 knows $\boldsymbol{PVs}^*@KP_i = \{PV_{<1,1>}, PV_{<2,0>}, PV_{<3,0>}, PV_{<4,0>}\}$, and $\boldsymbol{PBs}^*@SP_i = \{PB_{<1,0>}, PB_{<2,1>}, PB_{<3,1>}\}$.

3.3 Centralized Periodic Batch Rekeying - Marking Algorithm [21]

In [21], the authors introduce a periodic batch rekeying algorithm, called the Marking algorithm, for centralized group key management.

Consider a *J Join & L Leave* operation (J members need to join and L members need to leave the group), in a group G with n users, $\{M_1, M_2, \cdots, M_n\}$:

If $J \geq L$, the key tree is updated as follows: J of the$L$ departed nodes are replaced with the J join nodes following the one-to-one map, $\{M_i^{Depart} \rightarrow M_i^{Join}\}$. The remaining $L-J$ departed nodes' siblings will be promoted to their parents' position and their parents will be erased in the key tree. So, $\boldsymbol{PVs}$ and $\boldsymbol{PBs}$ of L key paths should be updated. Figure 3(a) demonstrates this procedure in which M_x joins, meanwhile M_3 and M_5 leave. M_x takes M_3's position while M_3 leaves and M_6 is promoted to its parents' position because M_5 leaves. $\boldsymbol{PVs}@M_4$ and $\boldsymbol{PVs}@M_6$ are updated by the sponsors M_4 and M_6 respectively.

If $J > L$, L new join group members take the places of the L departed members. The remaining $J-L$ new group members comprise a child key tree joining the key tree at the shallowest point. So, $\boldsymbol{PVs}$ and $\boldsymbol{PBs}$ of $L+1$ key paths should be updated. Figure 3(b) demonstrates this procedure in which M_x, M_y, and M_z join, meanwhile M_1 and M_5 leave. M_y and M_z take M_5 and M_1's positions respectively while M_5 and M_1 leave. M_x plays the role of M_3's sibling after M_x joins. $\boldsymbol{PVs}@M_2$, $\boldsymbol{PVs}@M_3$ and $\boldsymbol{PVs}@M_6$ are updated by the sponsors M_2, M_3 and M_6 respectively.

Note that in the centralized algorithm, the key server knows all keys in a key tree and therefore it is not hard for the key server to update the key tree for *J Join & L Leave* during a rekeying interval. However, in the contributory group key management, no group member knows all the keys on the key tree. So, how to efficiently update the group key in a batch requires all group members' cooperation. Furthermore, when a group is divided into several sub groups due to network partitions, or several sub groups merge into a super group due to network restorations, we need an efficient technique to process the *J Join & L Leave* operation.

4 Hybrid Group Key Management

In this section, we present the hybrid architecture that combines the advantages of the centralized approach's efficiency and the contributory scheme's fault tolerance. Furthermore, we present an efficient contributory rekeying scheme.

4.1 Hybrid Architecture

The basic idea behind our hybrid architecture is the following. If the key server is down (off-line), then the group key management is done using the contributory scheme. If the key server is on-line, then there are two possibilities. If all the group members are able access the key server (no partitioning of the group), then a centralized scheme is used. On the other hand, if the group is partitioned (some of the members are not able to access the key server), then we use a combination of the two schemes - the members with access to the key server use the centralized scheme while the others use the contributory scheme. Both of them follow the periodic tree-based Elliptic Curve Diffie-Hellman key management (TECH) to update the

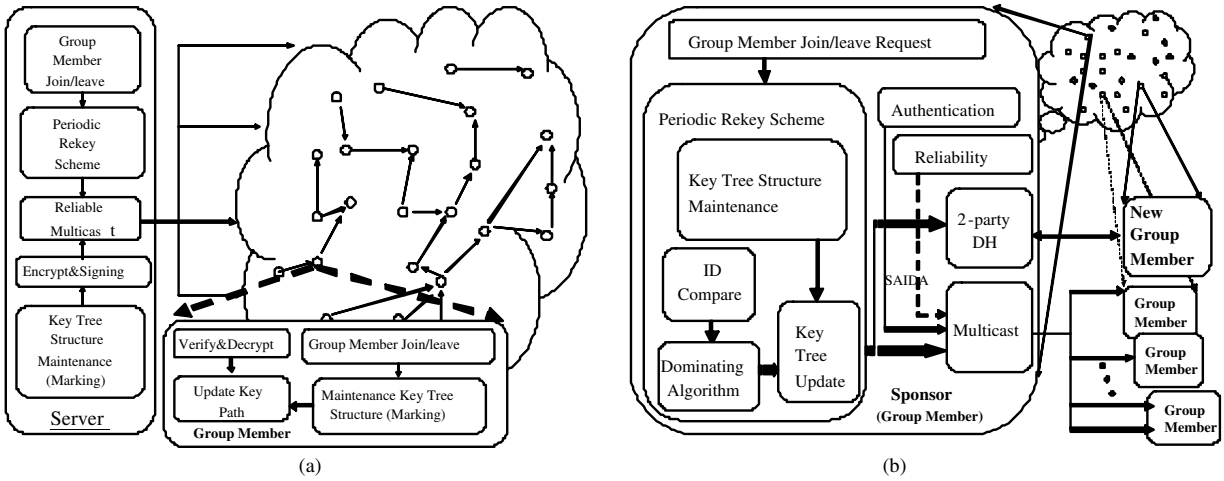

Figure 4: (a) Centralized group key management (b) Contributory group key management

public and private keys associated with the nodes on the binary key tree.

Figures 4(a) and (b) demonstrate the required components to implement the centralized and the contributory group key management schemes, respectively. As in [5, 6], we assume that every group member or the key server should be aware of any group membership changes based on *the group member join/leave request* component. Furthermore, the server and the members modify the key tree structure according to the *key tree structure maintenance component*.

In general, the centralized group key management administers the group key update for group membership changes. In Figure 4(a), the key server is responsible to update, encrypt and authenticate rekeying messages including individual keys, auxiliary keys, and group keys. Furthermore, it forwards the encrypted rekeying messages to other group members via the em reliable and authenticated multicast component. Every group member should utilize *the verify & decrypt* component to process the rekeying messages to update its key paths.

However, the centralized scheme cannot handle situations such as the key server failure or network partitions due to intermediate nodes being out-of-service or severe network congestions. This problem can be efficiently dealt with by our contributory scheme. When the key server loses its availability, all group members manage the key tree themselves in a contributory mode. Since centralized and contributory key managements use the same rekeying scheme, TECH, no rekeying operations are processed and no rekeying messages are forwarded to implement the switch from the centralized scheme to the contributory scheme.

Figure 4(b) shows the components of the contributory scheme, in which the *ID-Comparison and Dominating* algorithms are deployed to update the key tree. The sponsor's (i.e., group member selected by the algorithm to update and forward the keys on the key path [5, 6]) *reliable and authenticated multicast* component forwards to the other group members the updated public keys. Upon receipt, other group members calculate the new group key. As in previous contributory group key managements [5, 6], the sponsor should also process the *2-party DH key exchange* with the new group member.

4.2 Centralized Periodic Batch Group Rekeying

At the end of every periodic interval, the key server updates the key tree using the *Marking* algorithm [21] and TECH. Furthermore, like previous centralized group key managements [21], the key server will encrypt the updated keys with the private key associated with the leaf as the key and will multicast all encrypted messages after signing them. In order to save communication overhead and to guarantee that all rekeying messages arrive at all group member nodes, our centralized scheme utilizes M-SAIDA, which will be introduced in section 5, for multicast service.

4.3 Contributory Group Key Scheme

In our contributory scheme, three protocols, *contributory periodic rekeying*, *partitioning*, and *merging* protocols are proposed to process different *J Join & L Leave* scenarios. In all the three protocols, each group member utilizes the Marking algorithm [21] to select the key path to be updated and also to select the corresponding sponsor. Furthermore, we propose the *Dominating* algorithm to efficiently update the keys and forward rekeying messages. In the contributory periodic rekeying protocol, every group member will handle the *J Join & L Leave* scenario at the end of each periodic interval. When the group is divided into sub-groups, say due to network disconnections, the partitioning protocol will treat the members that cannot be in contact with the group as leaving members. In this

case, each group member will implement the *0 join & L leave* scenario. In a similar way, when sub-groups merge due to network re-connections, the merging protocol implements the *J join & 0 leave* scenario. For every sub-group, the group member hosting the leftmost key path is treated as the sponsor for the sub-group which generates the new session secret key, updates keys on its key path and multicast the updated keys.

As shown in Figures 3 and 4(b), the *ID comparison algorithm* is used to find intersections on the key path. We notice that n intersections divide the key path into $n+1$ key sub-paths. Furthermore, the *dominating* algorithm is proposed to decide which sponsor is responsible to update keys on each key sub-path and forward to other group members updated public keys.

We use an *ID comparison algorithm* similar to [24] to let every sponsor decide whether its key path intersects others (note: the root does not count).

Algorithm 1: ID comparison

Input: KP_k: key path for M_k; KP_j: key path for M_j;
Output: value i;
1: i = 1;
2: compare KP_k[i] with KP_j[i]
3: If they are the same, i = i+1, goto 2

For example, in Figure 3(b), i should be 2 with sponsors M_2 and M_3 as the input. Next we introduce the concept of a dominating key path.

Dominating key path: If two key paths intersect, we say that the right key path is dominated by the left key path. Therefore, the left key path is the dominating key and is responsible to update the overlapped nodes on the two key paths.

For example, in Figure 3(b), KP_2, the key path ofM_2, intersects KP_3, the key path of M_3, at $< 3, 0 >$. Because KP_2 is at the left of KP_3, KP_2 dominates KP_3. Therefore, M_2 should update and multicast the public keys of $< 3, 1 >$.

Let's now take a look at the *J Join & L Leave* scenario in which the *Marking algorithm* decides that k key paths should be updated. Without the consideration of the root of the key tree, a key path KP_i intersects $n-1$ other key paths, $KP_1, KP_2 \cdots KP_{i-1}, KP_{i+1} \cdots KP_{n-1}$, one by one from the leaf to the root, where $n < k$. Assume that the $n-1$ corresponding intersections are $< x_1, y_1 >, < x_2, y_2 > \cdots < x_{n-1}, y_{n-1} >$. The key path KP_i, is divided into the key sub-paths below.

$$KSP_{i,<1,i>,<x_1,y_1>}, KSP_{i,<x_1,y_1>,<x_2,y_2>} \cdots KSP_{i,<x_{n-1},y_{n-1}>,<h,0>}.$$

Next we propose the *dominating* algorithm which decides how to update and forward the keys on key sub-paths mentioned above. For example, in Figure 3(b):

Algorithm 2: *Dominating Algorithm*

Every Sponsor M_i**:**
1: updates $KSP_{i,<1,i>,<x_1,y_1>}$
2: if all public keys are on the key paths, dominated by M_i, are multicast:
3: repeat calculating PVs and PBs on its key path until it cannot continue
4: multicast updated PBs it calculated
5: else
6: wait for PBs sent from the sponsor whose key path is dominated by M_i's key path

All group members:
1: after receiving the PBs from all sponsors, update the PVs on its key path.

1st round:

Key path of M_3 is dominated by that of M_2.

M_3 multicasts $PB_{<0,2>} + PB_{<1,2>} + PB_{<2,1>}$

M_6 does not dominate others' key path. And others do not dominate M_6.

M_6 multicasts $PB_{<1,5>} + PB_{<2,2>} + PB_{<3,1>}$.

2nd round:

M_2 multicasts $PB_{<1,1>} + PB_{<2,0>} + PB_{<3,0>}$

All group members update the PBs and PVs on their key path and get the group key.

We notice that in our algorithm every sponsor forwards 1 message. In contrast, in TGDH, every sponsor forwards $[1, log_2 k]$ messages. Therefore, in total, all sponsors forward k messages for our contributory scheme and $2k-1$ messages for TGDH, where k is the number of sponsors.

5 Authenticated and Reliable Rekeying Message Multicast

In this section, we present our reliable and authenticated multicast algorithm known as M-SAIDA (Modified Signature Amortization Information Dispersal Algorithm). This is a modification of SAIDA proposed in [9]. In addition to the efficient signing feature, we also propose some techniques to guarantee the reliability for not only the digital signatures but also rekeying messages. Furthermore, benefiting from the dominating key path we introduced earlier, the number of signing operations is reduced as compared to previous popular schemes. Our algorithm satisfies the requirements of zero packet loss and computational efficiency.

To guarantee a reliable multicast service for authentication information, Park, Chong and Siegel [9] propose the Signature Amortization Information Dispersal Algorithm (SAIDA) to encode authentication information with Rabin's Information Dispersal Algorithm (IDA). Due to the no-packet-loss requirement, our rekeying transport protocol, outlined in Algorithm 3 below, M-SAIDA, encodes not only rekeying messages but also authentication information.

Algorithm 3: M-SAIDA

Sponsor M_i**:**

INPUT: Public keys $\mathrm{PB}_1, \mathrm{PB}_2 \cdots \mathrm{PB}_\mathrm{n}$,
OUTPUT: Public keys and signature encoded by Algorithm 4 IDA-Encode;.

1: Notation. ‖: concatenation; H = NULL; / * the hash value for $\mathrm{PB}_1, \mathrm{PB}_2 \cdots \mathrm{PB}_\mathrm{n}$ */
2: *for* $1 \leq i \leq n; i++$;
3: H = Hash(H ‖ $\mathrm{PB_i}$); /* Hash is an secure one-way-hash function such as SHA-256 */
4: *end for*
5: M_i multicast (IDA-Encode (PB_1), $\cdots$ IDA-Encode(PB_n), IDA-Encode ($\mathrm{Sign_{RSA}(H)}$));
6: /*RSA is used *because it is efficient in verification.*/

The Receiver $M_1 \cdots M_n$:

INPUT: Public keys and signature encoded by Algorithm 4 IDA-Encode;
OUTPUT: Public keys and signature;

1: ***for*** $1 \leq i \leq n; i++$;
2: IDA-Decode ($\mathrm{PB_i}$);
3: H = Hash(H ‖ $\mathrm{PB_i}$); /* Hash is an secure one-way-hash function such as SHA-256*/
4: ***end for***
5: ***Verify***$_{RSA}$ (H, IDA-Decode ($\mathrm{Sign_{RSA}(H)}$))

Algorithm 4 outlines on the implementation of the IDA which presents reliable transmission for data packets by introducing some amount of information redundancy. IDA splits the source data, for example, $\mathrm{PB_i}$, into n pieces, which are then encoded by the IDA algorithm. At the receiver end, the IDA can reconstruct $\mathrm{PB_i}$ after receiving any m pieces where $m < n$. However, guaranteeing zero packet loss comes at the cost of increased communication overhead. For example, for r public keys, assume every public key is 1024 bits. Therefore, n is 32, m can be 30. $1024 * r * n/m$ bits' data are sent over the network and at least $1024 * r$ bits' data are received. For example, in Figure 3(b):

1st round:

$\mathrm{M_3}$ multicasts (IDA-Encode ($\mathrm{PB_{<0,2>}}$), IDA-Encode($\mathrm{PB_{<1,2>}}$), (IDA-Encode ($\mathrm{PB_{<2,1>}}$), IDA-Encode ($\mathrm{Sign_{RSA}(H(PB_{<0,2>} \parallel (H(PB_{<1,2>} \parallel (H(PB_{<2,1>})))))}$);

$\mathrm{M_6}$ multicasts (IDA-Encode ($\mathrm{PB_{<1,5>}}$), IDA-Encode($\mathrm{PB_{<2,0>}}$), (IDA-Encode ($\mathrm{PB_{<3,1>}}$), IDA-Encode ($\mathrm{Sign_{RSA}(H(PB_{<1,5>} \parallel (H(PB_{<2,0>} \parallel (H(PB_{<3,1>})))))}$);

2nd round:

$\mathrm{M_2}$ multicasts (IDA-Encode ($\mathrm{PB_{<1,1>}}$), IDA-Encode($\mathrm{PB_{<2,0>}}$), (IDA-Encode ($\mathrm{PB_{<3,0>}}$), IDA-Encode ($\mathrm{Sign_{RSA}(H(PB_{<1,1>} \parallel (H(PB_{<2,0>} \parallel (H(PB_{<3,0>})))))}$);

All group members use IDA-Decode to decode messages and verify signatures. Hence they can update the PBs and PVs on their key path and get the group key.

We notice that, benefiting from Dominating algorithm and M-SAIDA, every sponsor signs once in our contributory scheme. In contrast, every sponsor signs $[1, \log 2k]$ times in TGDH. Therefore, in total, all sponsors signs k times for our contributory scheme and $2k - 1$ times for TGDH, where k is the number of sponsors. In our centralized scheme, the signing operation is performed once by the key server. Other group members do not need any signing operation.

6 Performance Evaluation

This section analyzes the computational cost and communication overhead for our centralized and contributory group key management schemes. The metrics we use to evaluate computational cost are the number of exponentiations and the number of signature generations for a sponsor and for the total group members. The communication overhead metric is the number of messages forwarded by all group members. In addition, the number of rounds is used to measure the group key updating time in a periodic interval in a merge operation or in a partition operation.

TGDH [5] and STR [6] have been known to be the most efficient contributory group key managements providing efficient rekeying schemes and authenticated multicast service to forward rekeying messages. Please refer to [2] for a detailed comparison with other contributory Diffie-Hellman based group key managements. However, our scheme deploys Elliptic Curve Diffie-Hellman (ECDH) which is more lightweight as compared to regular DH used by TGDH and STR. To show that not all of our performance gain is from ECDH, we compare our centralized and contributory schemes with TGDH and STR by deploying regular DH rather than ECDH.

In Table 1, we summarize our centralized and contributory group key managements, TGDH and STR. The current group size is denoted by N and height of the key tree for TGDH and our protocol is h. For a merge protocol, the number of sub-groups is k and the number of group members in k merging sub-groups is m. For a partition protocol, the number of leaving members is p. For TGDH and our proposal, the overhead is varied according to the balance of the key tree and the join or leave member's

Algorithm 4: IDA

The Sender Party A: IDA-Encode

INPUT: a block of data C_j

OUTPUT: encoded vectors $T_1, T_2 \cdots T_n$

(1) Split C_j into N/m pieces where N=n/8:

$C_j = (C_1, \cdots, C_m), (C_{m=1}, \cdots, C_{2m}), \cdots, (C_{N-m-1}, \cdots, C_N) where\ C_i : byte$

$R_i = (C_{(i-1)m+1}, \cdots, C_{im}), where i < N/m$

(2) Process C_j: following the specification of IDA [6], choose n vectors, $A_i = (a_{i1}, \cdots, a_{im}), 1 \le i \le n$, let every subset of m different vectors are linearly independent. Then, process C_j:

$$T_i = A_i \cdot (R_1, R_2, \cdots, R_{N/m}) = (a_{i1}, \cdots, a_{im}) \cdot \begin{pmatrix} C_1, C_{m+1}, \cdots C_{N-m+1} \\ \cdot \\ \cdot \\ \cdot \\ C_m, C_{2m}, \cdots C_N \end{pmatrix} where 1 \le i \le n \quad (2)$$

(3) Send $T_1, T_2 \cdots T_n$ to the receiver.

The Receiver Party B: IDA-Decode

INPUT: encoded vectors $T_1, T_2 \cdots T_m$

OUTPUT: a block of data C_j

(1) Assume that the receiver receives $T_1, T_2 \cdots T_m$

$T_1 = A_1 \cdot R_1, A_1 R_2, \cdots, A_1 \cdot R_{N/m}$

$T_2 = A_2 \cdot R_1, A_2 R_2, \cdots, A_2 \cdot R_{N/m}$

$\cdots$

$T_m = A_m \cdot R_1, A_m R_2, \cdots, A_m \cdot R_{N/m}$

(2) Prepare for the calculation of R_1

Based on $T_1 \cdots T_m$, and Formula (2), we can get:

$$A'g \begin{pmatrix} C_1 \\ \cdot \\ \cdot \\ \cdot \\ C_m \end{pmatrix} = \begin{pmatrix} A_1 g R_1 \\ \cdot \\ \cdot \\ \cdot \\ A_m g R_1 \end{pmatrix} where\_A' = \begin{pmatrix} a_{11} \cdots a_{1m} \\ \cdots \\ \cdots \\ \cdots \\ a_{m1} \cdots a_{mm} \end{pmatrix}$$

(3) Since A' is invertible, we can calculate R_1:

$$R_1 = \begin{pmatrix} C_1 \\ \cdot \\ \cdot \\ \cdot \\ C_m \end{pmatrix} = \begin{pmatrix} a_{11} \cdots a_{1m} \\ \cdots \\ \cdots \\ \cdots \\ a_{m1} \cdots a_{mm} \end{pmatrix}^{-1} \begin{pmatrix} A_1 g R_1 \\ \cdot \\ \cdot \\ \cdot \\ A_m g R_1 \end{pmatrix}$$

(4) Repeat step 3, we can calculate $R_2 \cdots R_{N/m}$.

(5) Reconstruct C_j:

$C_j = R_1 \parallel R_2 \cdots \parallel R_{N/m} where \parallel denotes\ concatenation.$

Table 1: Computational cost and communication overhead

		Communication overhead		Computational cost				
		Total		Main sponsor		Total		
		Rounds	Messages	Exponentiation	Signatures	Exponentiation	Signatures
TGDH	J join& L leave	$2J+L$	$2J+L$	$2h(J+L)$	$J+L$	$(2n-1)(J+L)$	$2J+L$
TGDH	Merge	$\log 2k+1$	2k	2h	$\log 2k+1$	$2(h-\log 2k)k+(2k-1)$	2k
TGDH	$min(\log 2p+1,h)$	$min(2h,2p)$	2h	$min(\log 2p+1,h)$	$2(h-\log 2p)p$ $+(2p-1)$	min(2h,2p)	
STR	J join & L leave	2J+L	2J+L	4J+(3n/2+2)L	J+L	(2n+2)J+(3n/2+2)L	2J+L
STR	Merge	2	k+1	3m+1	2	(n+m)m+3m+1	k+1
STR	Partition	1	1	3n/2+2	1	(n-1) (3n/4+1) +3n/2+2	1
Our Contr.	J join&L leave	$min(\log 2L+1,h)$	L	2h	1	$2(h-\log 2L)L+2L-1$	k
Our Contr.	Merge	$\log 2k+1$	k	2h	1	$2(h-\log 2k)k+(2k-1)$	k
Our Contr.	Partition	min(log2p+1, h)	p	2h	1	$2(h-\ log2p)p+(2p-1)$	p
Our Centr.1	J join&L leave	1	1	2h*	1	$2(h-\log 2L)L+(2L-1)+2hn$	1
Our Centr.1	Merge	1	1	2h*	1	$2(h-\log 2k)k+(2k-1)+2hn$	1
Our Centr.1	Partition	1	1	2h*	1	$2(h-\log 2p)p+(2p-1)+2hn$	1

* Encryption/decryption instead of the modular exponentiation is required.
1: In centralized scheme, key server which is in charge of the key tree update is not treated as a sponsor.

location. Our performance analysis for them is based on the worst scenario. But for the leaving group membership for STR, we compute the average case, in which the $(n/2)^{th}$ node leave, to follow the same analysis as that in [6].

Our centralized group key management is as same as keygem [21] with the difference that keygem updates the keys with random key generations but ours updates the keys following TECH. Furthermore, the key tree's degree can be any optimized number but ours is 2. Please refer to [21] for a detailed performance analysis between keygem and other centralized group key managements.

In the following, we introduce the worst-case scenario for the tree-based group key managements such as TGDH and our two schemes. We also compare the performance and study the overhead of the reliable multicast.

6.1 The Worst-case Scenario

As demonstrated in Figure 5(a), the worst-case scenario occurs when the sponsors are evenly distributed on the tree leaf nodes. As a consequence, the overlapped public and private key updates are least in number [21]. Figure 5(b) shows how to update public keys on different key paths and how to multicast them by corresponding sponsors for the dominating algorithm.

6.2 Performance Comparison

J join & L leave: As seen from Table I, our two schemes are comparatively efficient in terms of number of rounds, number of messages, number of exponentiations and number of signing operations because they take advantage of periodic batch rekeying. STR requires the most computational cost. Both STR and TGDH demand the most communication overhead. But TGDH and STR, which implement individual rekeying, can provide forward and backward security with 0 vulnerability window [21].

Merge: Our contributory and centralized schemes require less cost as compared to TGDH and STR in terms of number of messages and computational cost. STR needs the most number of exponentiation operations and TGDH requires the most number of signing operations. Our centralized scheme and STR use a constant number of rounds.

Partition: In terms of other computational metrics, our centralized and contributory schemes are more efficient. TGDH demands the most communication overhead and computation cost. Our centralized scheme and STR are more bandwidth saving and they require a constant number of signing operations.

The performance evaluation earlier shows us that our centralized group key management scheme is the most efficient among these four schemes. However the drawback is the requirement of a key server.

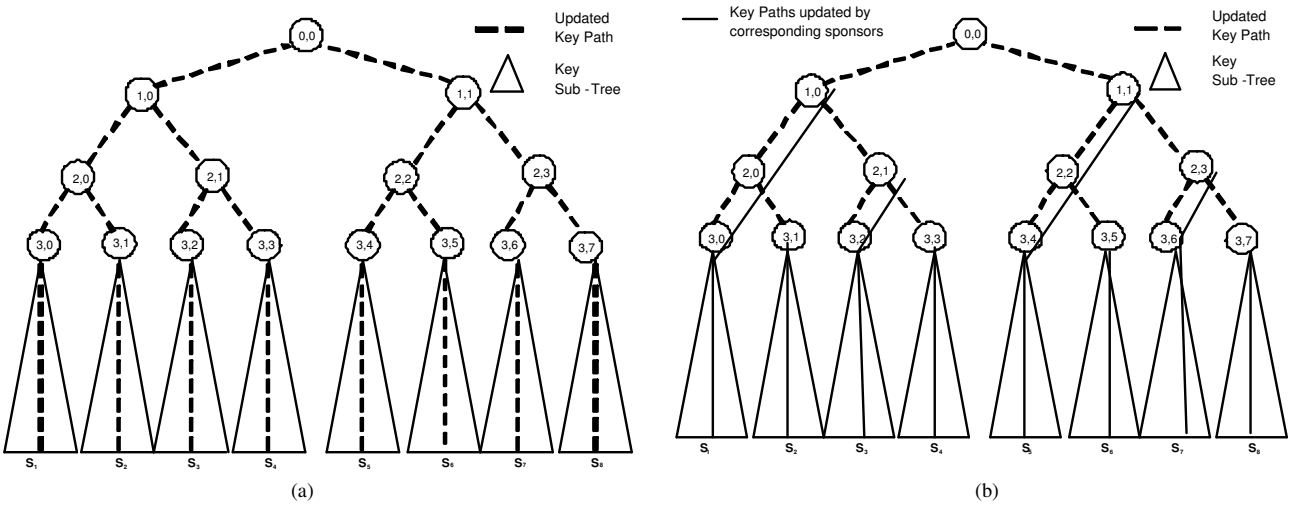

Figure 5: (a) Worst-cases scenario (b) Dominating key path

6.3 Reliability

Unlike other contributory group key managements which assume the reliable multicast service, our proposal M-SAIDA guarantees all rekeying messages arriving other members without packet loss and with the running complexity, $O(n^2)$.

Since the actual packet loss follows the burst rather than the independent model, Yajnik et al. [19] introduce the 2-state Markov chain (2-MC) loss model and Miner et al. [8] introduce the Biased Coin Toss (BCT) loss model. Both of them can accurately model bursty loss patterns. Park, Chong and Siegel [9] analyze the authentication probability, $P_r\{P_i \text{ verifiable} \mid P_i \text{ is received}\}$ of SAIDA using the two loss models. The result suggests that making the block size large could decrease the communication overhead if relatively long verification delays are tolerated. Unlike TGDH which multicasts single public keys frequently, our group key management always multicasts all the updated public keys on the entire key path within one message. Hence, the message size is large and higher sending delays are tolerable. Therefore, the SAIDA method suits our proposal well.

6.4 Authentication

The communication cost of M-SAIDA is:

$$\begin{aligned} NUM_{key\_overhead} &= (n/m) * (NUM_{key} + (H + s)) \\ NUM_{signature} &= 1. \end{aligned}$$

The cost for Wong-Lam's authentication tree is:

$$\begin{aligned} NUM_{key\_overhead} &= NUM_{key} + (Hhn + s) \\ NUM_{signature} &= 1, \end{aligned}$$

where NUM is the number; s: signature size; H: hash size; h is the height of the key tree Notice that Wong-Lam's method cannot guarantee no-packet-loss.

7 Conclusion

Security in group communications is an important and evolving field. Group key management requiring both efficiency and failure-tolerance over resource-limited networks is a challenging task. This paper presents the design and specification of a hybrid architecture which includes a contributory and a centralized group key management based on one set of keys. It takes advantage of centralized scheme's efficiency and contributory scheme's failure-tolerance. Furthermore, no computational and communication cost are involved to switch between the two schemes. We also propose a contributory rekeying scheme which can handle the periodic batch rekeying, merge and partition protocols efficiently as well as cooperate with the centralized scheme. Finally, we propose a multicast protocol that has zero packet loss (reliable) and authenticated. Our performance analysis shows that our technique reduces the number of rounds, the number of messages, the number of exponentiations and the number of signing operations as compared to other popular key management schemes.

References

[1] G. Ateniese, M. Steiner, and G. Tsudik, "New multiparty authentication services and key agreement protocols," *IEEE Journal of Selective Areas Communication*, vol. 18, no. 4, pp. 628-639, 2000.

[2] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton, and G. Tsudik, "Secure group communication using robust contributory key agreement," *IEEE Tranactions On Parallel and Distributed Systems*. vol. 15, no. 5, pp. 468-480, May 2004.

[3] G. Ateniese, M. Steiner, and G. Tsudik, "Authenticated group key agreement and friends," in *Proceed-*

ings of ACM Conference on Computer and Communication Security (CCS'98), pp. 17-26, 1998.

[4] E. Bresson, O. Chevassut, D. Pointcheval, and J. J. Quisquater, "Provably authenticated group Diffie-Hellman key exchange," in *8th ACM Conference on Computer and Communication Security (CCS'01)*, pp. 255-264, 2001.

[5] Y. Kim. A. Perrigm, and G. Tsudik. "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *7th ACM Conference on Computer and Communications Security (CCS'00)*, pp. 235-24, Nov. 20004.

[6] Y. Kim. A. Perrig, and G. Tsudik, "Group key Agreement efficient in communication," *IEEE Transactions on Computers*, vol. 53, no. 7, pp. 905-921, July 2004.

[7] D. Li and S. Sampalli, "An efficient group key establishment in location-aided mobile ad hoc networks," in *The 2nd ACM Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous (PE-WASUN'05)*, pp. 57-64, 2005.

[8] S. Miner and J. Staddon, "Graph-based authentication of digital streams," in *Proceedings of IEEE symposium on Research in Security and Privacy*, pp. 232-246, 2001.

[9] J. M. Park, E. K. P. Park, and H. J. Siegel, "Efficient multicast stream authentication using erasure codes," *ACM Transactions On Information and System Security*, vol 6, no. 2, pp. 258-285, May 2003.

[10] A. Perrig, D. Song, and J. D. Tygar. "ELK, A new protocol for efficient large-group key distribution," in *Proceeding of the IEEE Symposium on Security and Privacy (IEEE S&P)*, pp. 247-262, 2001.

[11] K. H. Rhee, Y. H. Park, and G. Tsudk, "A group key management architecture for mobile ad-hoc networks," *Journal of Information science and engineering*, vol 21, pp. 415-428, 2005.

[12] S. Setia, S. Koussih, and S. Jajodia, "Kronos: A scalable group re-keying approach for secure multicast," in *Proceedings of the IEEE Symposium on Security and Privacy (IEEE S&P)*, pp. 215 - 228, May 2000.

[13] A. T. Sherman and D. A. Mcgrew, "Key establishment in large dynamic groups using one-way function trees," *IEEE Transactions on Software Engineering*, vol. 29, no. 5, pp. 444-458, May 2003.

[14] M. Steiner, G. Tsudik, and M. Waidner, "Key agreement in dynamic peer groups," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 8, pp. 769-780, Aug. 2000.

[15] Y. Sun and K. J. Ray Liu, "Securing dynamic Membership Informationin Multicast Communications," in *Proceedings of IEEE INFOCOM*, Mar. 2004.

[16] H. Weatherspoon, C. Wells, P. Eaton, B. Zhao, and J. Kubiatowicz. *Silverback: A Global-Scale Archival System*, Technical Report UCB/CSD-01-1139, Computer Science Division, University of California, Berkeley, CA.

[17] C. K. Wong and S. S. Lam. "Digital signatures for flows and multicasts," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 502-513, Aug. 1999.

[18] C. K. Wong, G. Gouda, and S. S. Lam, "Secure group communications using key graphs," *IEEE/ACM Transactions on Networking*, vol. 8, no. 1, pp. 16-30, Feb. 2000.

[19] M. Yajnik, S. Moon, and D. Towsley, "Measurement and modeling of the temporal dependence in package loss," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM '99)*, pp. 345-352, 1999.

[20] W. H. Yang and S. P. Shieh, "Secure key agreement for group communications," *International Journal of Network Management*, vol. 11, no. 6, pp. 365-374, 2001.

[21] Y. R. Yang, X. S. Li, X. B. Zhang, and S. S. Lam, "Reliable group rekeying: A performance analysis," in *Proceeding of ACM SIGCOMM'01*, pp. 27-38, Aug. 2001.

[22] X. B. Zhang, S. S. Lam, D. Y. Lee, and Y. R. Yang, "Protocol design for scalable and reliable group rekeying," *IEEE/ACM Transactions on Networking*, vol. 11, no. 6, pp. 908-922, Dec. 2003.

[23] Y. Zheng, "Shortened digital signatures, signcryption and compact and unforgeable key agreement schemes," *Submission to IEEE P1363a: Standard Specifications for Public-Key Cryptography*, 1998.

[24] X. K. Zou, B. Ramamurthy, and S. S. Magliveras, *Secure Group Communication over Data Networks*, Springer, 2005.

Depeng Li received the Bachelor and Master degrees in Computer Science from Shandong University, Jinan, P. R. China. He is currently working toward the Ph.D. degree in Computer Science at Dalhousie University, Halifax, Canada. His research interests are in the areas of network security, peer-to-peer group communication and performance evaluation.

Srinivas Sampalli is a Professor and 3M Teaching Fellow in the Faculty of Computer Science, Dalhousie University, Halifax, Nova Scotia, Canada. He has been actively researching in the area of security and quality of service in wireless and wireline networks. Specifically, he has been involved in research projects on protocol vulnerabilities, security best practices, risk mitigation and analysis, and the design of secure networks. He is currently the principal investigator for the Wireless Security project sponsored by Industry Canada. Dr. Sampalli has received many teaching awards including the 3M Teaching Fellowship, Canada's most prestigious teaching acknowledgement.