



A communication–computation efficient group key algorithm for large and dynamic groups

Shanyu Zheng, David Manz, Jim Alves-Foss *

Department of Computer Science, Center for Secure and Dependable Systems, University of Idaho, P.O. Box 441010, Moscow, ID 83844-1010, United States

Received 3 June 2005; received in revised form 21 December 2005; accepted 9 March 2006

Responsible Editor: G. Schaefer

Abstract

The management of secure communication among groups of participants requires a set of secure and efficient operations. In this paper we extend existing work to present a Communication–Computation Efficient Group Key Algorithm (CCEGK) designed to provide both efficient communication and computation, addressing performance, security and authentication issues of CCEGK. Additionally, we compare CCEGK with three other leading group key algorithms, EGK, TGDH, and STR. An analytical comparison of all algorithms revealed eight similar methods: add, remove, merge, split, mass add, mass remove, initialize, and key refresh. Comparing the cost in terms of communication and computation, we found CCEGK to be more efficient across the board.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Security; Group key management; Group communications; Communication complexity; Cryptographic protocols

1. Introduction

With the advent of new arenas such as wireless ad-hoc and low powered distributed computing and communication devices, designers of group key encryption algorithms can no longer ignore communication in favor of computation or vice versa. In some environments the power cost of communication may be sufficiently high to warrant low cost communication protocols, whereas in other

environments computation cost may be the dominant feature. Consequently, this paper introduces the Communication–Computation Efficient Group Key protocol (CCEGK), which is an extension of EGK [1] and TGDH [2,3].

The Communication–Computation Efficient Group Key Algorithm (CCEGK) is a group key management algorithm based upon two preceding group key management algorithms, EGK [1] and TGDH [2,3]. By extending this previous work, CCEGK considerably improves both communication and computation costs of their related operations. Furthermore CCEGK fully implements, as detailed in this paper, several methods that are not

* Corresponding author. Tel.: +1 208 885 5196; fax: +1 208 885 6840.

E-mail address: jimaf@csds.uidaho.edu (J. Alves-Foss).

described by other authors in the literature. For instance, CCEGK fully implements an initialization operation while the TGDH and STR [4,5] algorithms do not. Next CCEGK presents two mass leave operations, mass leave-balanced, and mass leave-imbanced (detailed later), while the TGDH algorithm only details a mass leave-imbanced. The Add operation in CCEGK is distinct from TGDH and similar to EGK; each time the controller sends a message, it sends a message with only one key size, as opposed to TGDH which sends all blinded keys in the group. Our merge method differs from TGDH in a similar manner. Next TGDH and STR's partition (split) operation could be better understood as a mass leave operation; since it is not detailed in literature it will not be compared further. Lastly TGDH and STR do not implement a balance operation, and CCEGK does detail a balance operation later in this paper.

There are similarities as well. CCEGK's leave and key refresh operations are the same operation as in TGDH, while join and initialize are almost identical to those of EGK and other operations are similar. While CCEGK adopts ideas from both TGDH and EGK, the focus on computation and communication efficiency warrants a distinct algorithm. Table 1 details the differences and similarities between CCEGK and TGDH, and CCEGK and EGK.

The goal of this paper is to present the full details of every operation of CCEGK in a standardized manner. This paper provides the following contributions.

- We fully describe the main key management operations for CCEGK, including initialization, add, mass add, merge, leave, mass leave, partition, and key refresh. This set of operations provides a basis for full comparison of competing group key protocols.
- Some group key protocols, such as our proposed CCEGK protocol, do not rebalance the binary key tree upon every operation. Therefore, we introduce two separate rebalance schemes, which are not expensive in either computation or communication costs. The rebalance operation is

not documented in the literature for EGK, TGDH or STR even though it can greatly affect the overall performance of the protocols over time as the trees become more unbalanced.

- We provide a theoretical analysis of the costs of both computation and communication for each operation for each of the compared protocols. This is beneficial since a complete treatment of all of these operations is not available in the literature. A separate recent study by Challal and Seba reviews a large number of group key protocols, but only compare the initialization costs [6]. The papers introducing STR [4,5] and TGDH [2,3], and companion performance paper [7] do not address initialization and partition operations at all (their partition operation is effectively a mass leave operation). We provide this missing information and also illustrate a point of contention with their published results.
- Beyond the theoretical comparison, we provide for experimental comparison of the performance of CCEGK, STR, TGDH, and EGK. In our experiment we created a framework to simulate these four contributory group key protocols and compared their communication and computation costs.

The remainder of this paper is organized as follows. In Section 2, we provide a historical and contextual background for our research. In Section 3, we provide an overview of the terminology and basic concepts used in this paper. This is followed in Section 4 by a discussion of our proposed protocol, CCEGK. In Section 5, we discuss management of our group key tree in the form of selective rebalancing. We follow this with a discussion of message authentication in Section 6. In Sections 7 and 8 we then provide a summary of comparisons between CCEGK and other protocols that we have conducted.

2. Background

In 1976, Diffie and Hellman introduced a two party key exchange protocol, DH, that allows two

Table 1
Operation comparison with CCEGK

	Key refresh	Initialization	Join	Mass join	Merge	Leave	Mass leave
EGK	NA	Similar	Similar	Different	Different	Different	Different
TGDH	Same	NA	Different	Different	Different	Same	Different

participants to create a private key [8] through the use of publicly exchanged messages. This protocol is now at the heart of many two-party secure communication protocols, including SSL [9] and SSH [10]. When we log in to a secure web site, our machine exchanges information with the web server to create a secure session key. That key is used to encrypt subsequent communication, protecting confidential information such as credit card numbers. With the increasing use of network technologies, there are several applications that would greatly benefit from an extended group key exchange algorithm that provides the same protection as DH, but for groups of participants. These applications include conference calls, distributed computation, white-boards and distributed databases, among many others. To ensure secure and reliable communication in these applications, there have been several attempts to create efficient group key protocols for large and dynamic groups based on the DH algorithm [11–15].

These approaches can be divided into two categories of key management, centralized and contributory [1,16,17,7]. In centralized key management, a single member creates keys and securely transmits them to all other group members. The drawback is that a centralized approach requires a trusted sponsor who generates and distributes the keys, leaving a single point of failure. The trusted sponsor cannot be continuously available to support operations in the event of an arbitrary network partition, so this approach is not appropriate for reliable group communication. Some authors have suggested using a distributed approach where the centralized management is distributed among many hosts, increasing the system's fault tolerance and reliability, but still requiring a set of trusted hosts.

The contributory group key management approach is based on the idea that each group member will directly contribute to key management and key generation (each member contributes to the entropy of the key). This alleviates the problem of a single point of failure and trust in a centralized group key management system. In the late 1990s there were several group key protocols concurrently introduced, such as EGK [1], TGDH [2,3], and STR [4,5], based on contributory group key management to improve the costs of communication and computation. EGK, TGDH and STR utilize a binary-tree structure to provide an efficient number of message exchanges, but they provided different algorithms for the key and group management protocols.

EGK and TGDH focus more on the cost of computation than on that of communication. STR focuses on improving the cost of communication more than the cost of computation, in keeping with Becker and Willie's original work on efficient communication in group key protocols [15].

3. Group key exchange and management background

The purpose of this paper is truly two-fold: first to introduce a new group-key agreement protocol based on an underlying two-party key exchange and second, to compare our protocol with other protocols that have appeared in the literature. To that aim, we constructed a simulator to evaluate the efficiency of the algorithms, as described by their authors. Additionally, we have analytically compared their algorithmic performance. Before we introduce our protocol, we first provide a summary of the basic concepts and terminology of group key protocols used throughout this paper.

3.1. Public two-party key exchange

A two-party key exchange protocol is a protocol that permits two entities with no prior shared secrets to publicly exchange information so that at the end of the protocol they both possess the same shared secret, and any eavesdropper listening in on the communication is unable to obtain that same secret. After the protocol runs, the participants can derive a session key from the shared secret. Throughout this paper we will use the term group key or secret key to refer to this shared secret and its derivative keys. Diffie and Hellman [8] introduced the first such public two-party key exchange algorithm, which we call DH. There are other two-party key exchange protocols with cryptographic strength comparable to DH. For the purposes of this paper we will use DH as a place holder for any of these two-party protocols. As we first introduced with EGK [1], we can use any of these two-party protocols at any point in CCEGK, changing protocols for each pair of groups communicating and even between rounds. The replaceability of these protocols is unique to the binary-tree-based group key protocols.

3.2. Diffie–Hellman protocol with indistinguishable security

Let g , p and q be publicly known values, where p and q are large primes and where $p = 2q + 1$. g is a

generator of the group $\mathbb{Q}\mathbb{R}_p$ that is a subgroup of \mathbb{Z}_p^* and the order is $|\mathbb{Q}\mathbb{R}_p| = q$. Let Alice and Bob choose secret private values x_A and x_B from $\mathbb{Z}_q = \{0, \dots, q-1\}$, respectively. From these secret private values, they create their corresponding public values (*blinded keys*), r_A and r_B , and send them to each other.

$$\text{Alice} \rightarrow \text{Bob} : r_A = g^{x_A} \bmod p$$

$$\text{Bob} \rightarrow \text{Alice} : r_B = g^{x_B} \bmod p$$

Bob and Alice then compute a shared secret key using their partner's public value and their own private value:

$$K = (r_A)^{x_B} \bmod p = (r_B)^{x_A} \bmod p = g^{x_A x_B} \bmod p$$

An eavesdropper can know p , q , g , $g_a^x \bmod p$, and $g_b^x \bmod p$ and still not be able to compute the shared key K . Furthermore, this Diffie–Hellman protocol is based on the *Decisional Diffie–Hellman* assumption. An eavesdropper can not distinguish between random keys in $\mathbb{Q}\mathbb{R}_p$ and keys that were generated by the above algorithm.

3.3. Group key protocols

The class of group key protocols we examine are those based on the utilization of underlying two-party key exchange algorithms such as the DH algorithm described above. The idea behind these protocols is that the group is arranged in a binary tree hierarchy. Although other structures exist, the binary tree hierarchy has provided for the most efficient algorithms to date. At the base of the tree are pairs of nodes, each representing a group of size one. During the algorithm, two groups pair up and communicate their blinded keys to one another, forming a single shared key. These groups then form a new group and use the shared key to generate a new secret group key which is used to generate the new blinded key for the next merging operation. This style of tree-based algorithm appeared in the literature in the early 2000s as different research groups concurrently focused on this topic. The first suggestion of a tree-based approach was by Becker and Willie [15], but they did not develop a full protocol around the concept. Protocols emerged later, including EGK [1], TGDH [2,3], and STR [4,5], all based on contributory group key management to improve the cost of communication and computation.

3.4. Terminology

3.4.1. Group key management operations

Throughout this paper we will address several group key management operations used by group key systems. The operations we describe are the following:

Initialization Operation: This is the initial creation of the group key and organization of the key management infrastructure.

Join: This operation brings a new member into the existing group.

Mass join (Mass add): This operation allows many new members to be added to an existing group simultaneously when these new members have not already formed a group of their own.

Merge (Group fusion): This operation, as opposed to mass join, is used when another group is combined with the existing group to become a new group.

Leave: This operation is used to remove a member from the group.

Mass leave: This operation is used when multiple members are simultaneously removed from the existing group.

Split (Partition, or group fission): This operation, different from mass leave, occurs when a single group is divided into two or more component groups.

Key refresh: To prevent the secret key from becoming stale, it should be changed. Moreover, to prevent an adversary from breaking in, we should refresh the original key and generate a new secret key periodically.

3.4.2. Secrecy terms

In the cryptographic protocol literature, there are several goals related to the management of group keys with respect to membership changes in the group. These goals are:

Group key secrecy: Any generated group key is indistinguishable in polynomial time from a random number. This is called the “Decision Diffie–Hellman Problem” (DDH) [18]. For the protocols discussed in this paper, the cryptographic strength of the generated group key is solely dependent upon the strength of the underlying two-party protocols. That said, we will not address the DDH problem further in this paper.

Weak backward secrecy: This ensures that new members cannot decrypt and understand previous messages that were sent prior to their joining the group.

Backward secrecy: This ensures that a passive adversary who knows a contiguous subset of group keys can not discover preceding group keys.

Weak forward secrecy: This ensures that the leaving members can no longer decrypt and understand messages after they have left the group.

Forward secrecy: This ensures that a passive adversary who knows a contiguous subset of previously used group keys can not discover subsequent group keys.

Perfect forward secrecy: This is a more strict version of the above. The active or passive attacker can additionally have a long-term secret key or session key of a current or past group member and still not compromise the communication.

Key independence: This requires that anyone who knows a true subset of the group keys, or the private keys corresponding to a true subset of the blinded keys, is unable to guess an additional group key.

3.4.3. Evaluation metrics

In our evaluation of the performance of the algorithms we evaluate many costs. This includes communication and computation costs for the evaluated operations. The values of these costs are depicted in Table 2.

Number of rounds: This is a generic time unit used to compare the number of steps taken in different

operations. The algorithms often require synchronization between rounds; therefore, this number becomes important when taking synchronization time into account.

Number of unicast messages: This is the sum of the messages sent by each member to another single member in the group in the operation. This number is useful for determining total communication and is important if many or all nodes are on the same network collision domain, thus forcing these messages to be sent sequentially and not in parallel.

Number of broadcast messages: This is the sum of the messages sent by each member to all the other members in the group for the operation. Since the messages go to all members of the group, it greatly affects total communication costs depending upon the underlying network topology.¹

Number of messages: This is the sum of all unicast messages and broadcast messages. We use this number to determine the total time of communication in an underlying broadcast network.

Number of sequential exponentiations: During an operation there will be a series of computationally expensive cryptographic operations (such as modular exponentiation used in the DH algorithm). The algorithms in the literature often require the results of one cryptographic operation prior to the execution of another. This metric represents the worst case scenario, the longest sequence of dependencies of these costly cryptographic operations in the operation.

Number of signatures: This is the sum of digital signatures used in every round. In every round the sponsor sends one digital signature.

Number of verifications: Given that each message needs to be verified, the number of verifications is equal to the number of messages; however, several verifications can occur in parallel, so we need to concern ourselves with the number of sequential verifications – the maximum number of verifications that must occur during an operation.

In addition, we define the *sponsor* as the group member elected to coordinate group activities. In our prior work, we have used the term *group controller* for this designation; however, sponsor seems to be a more fitting appellation.

Table 2

Symbols used in metrics

n	Number of group members
N	Number of merging, joining or leaving members or subgroups
r	Denotes a random integer
$\langle l, i \rangle$	The i th node at level l in the binary key tree
$K_{\langle l, i \rangle}$	The secret key of the node $\langle l, i \rangle$
$bK_{\langle l, i \rangle}$	Blinded key of the node $\langle l, i \rangle$, e.g., $\alpha^{K_{\langle l, i \rangle}} \bmod q$
bK_i	Blinded key of the i th group member.
M_i	The i th group member, $i \in [1, n]$
K_i	Secret key of the member M_i
α	An exponentiation base, which is a prime number based on DH
q	Order of the algebraic group (large prime number for DH)
BT	The binary key tree

¹ The existing group key literature assumes a fully connected broadcast network.

4. CCEGK protocol

In this section, we discuss our communication–computation efficient group key (CCEGK) algorithm. This algorithm is based on EGK, which we presented a few years ago [1], and adopts conventions and a few operations from TGDH [2,3]. EGK was originally created with the concept of rapid addition of new members and low rekeying costs. TGDH and STR adopt a more balanced approach but also use tree-based group keying. CCEGK is an attempt to merge these two concepts.

Having extended EGK and TGDH, CCEGK is based on a binary tree structure that assumes a consistent world view among all group members. Specifically, we make the following assumptions which are consistent with those made in the group-key literature:

1. All members know the key tree structure, an ordered list of each other’s identities, and their initial position in the tree.
2. All participants can unambiguously determine their group sponsor.
3. Every member sees the same sequence of group key operations.

To implement a consistent world view, TGDH and STR use View Synchrony (VS), a simple specification that allows processes to synchronize on specific views [19]. According to the authors of TGDH and STR, VS is essential for any fault tolerant group key agreement protocol. Therefore, as with TGDH and STR, we assume an implementation that supports the semantics of VS.

These assumptions are straightforward but have sweeping ramifications that are necessary to facilitate current group key exchange algorithms. It is understandable that the current group key exchange algorithms would make use of these assumptions, as the cost of communication for a traditional wired network is fairly small. However, less traditional networks, including wireless, will need a reassessment of these assumptions; we leave that discussion to a future paper.

Given these assumptions, a set of nine operations emerge from the group key field. Almost every proposed protocol is presented with split, merge, add, leave, mass add, and mass leave. However, initialize, rebalance, and key refresh are potential exceptions, as we indicate below. In this section we define each of these nine operations as they are implemented in

CCEGK. We also provide the CCEGK communication and computation costs for each operation. For completeness we include digital signature and verification costs if they are implemented in the protocols. We leave discussion of signatures and signature verification to Section 6. In Sections 7 and 8 we provide theoretical and experimental comparisons of the CCEGK costs to those of EGK, STR and TGDH.

Throughout the remainder of this section, we chose to use terminology and conventions established in the TGDH papers since they have been published in venues with a wider distribution than the original EGK paper.

4.1. Initialization operation

The *initialization* operation is the group genesis algorithm. In this algorithm each node pairs up with its neighbor, if available, to establish a new shared key. If no neighbor is available (as in an incomplete binary tree), the single node will behave as an atomic group. The new group then repeats the pairing with other groups until there is only one group. The basic operation occurs as follows:

1. Suppose that there is a collection of entities: $\{M_1, \dots, M_n\}$ (it is not important whether n is equal to 2^r). From assumption 1, we know all members can sort their identities in some order. Each member represents the sorted set of identities as the labels for the sponsors of a binary tree.
 - (a) In round 1, each node M_i generates a DH key pair consisting of a secret key K_i and a corresponding blinded key $bK_i = \alpha^{K_i} \bmod p$. Each node M_i unicasts a message including its blinded key to its sibling. M_i performs a two-party Diffie–Hellman (DH)² with its sibling to calculate a new DH key pair with the private key and the blinded key. Each node has a single parent and single sibling. We choose the rightmost node as the sponsor of the group (if n is not even, the last node does nothing in the first round; we treat it as a group. The same condition is in every round). We treat every new group as a new node, so the new number of nodes is $\lceil \frac{n}{2} \rceil$.

² Recall that although we use DH in this paper, any equivalent two-party public key algorithm can be used in place of DH. The choice of algorithm can be based on the pair of nodes involved in the exchange.

- (b) In subsequent rounds, each member of the group determines the new group private and blinded keys. The sponsor of each group broadcasts a message including its group's new blinded key to all of the members of its sibling group. Every member then performs a two-party DH with the blinded key of its sibling group to generate a new group with a new DH key pair. We again choose the rightmost node as the sponsor of this new group.
- (c) We repeat the above process until round $h = \lceil \log_2 n \rceil$ (h is the height of the key tree). At this point we have a single group, G , which includes all the members, each sharing the group secret key $K_{(0,0)}$.

The costs of the CCEGK initialization operation are summarized in Table 3.

4.2. Join operation

The *join* operation occurs when a new joining member sends a join request to the group. The procedure for the join operation is as follows:

1. Suppose that the group has n members: $\{M_1, \dots, M_n\}$. The new member, M_{n+1} , broadcasts a message including its own blinded key to every member of the group. Simultaneously the sponsor of the original group (the shallowest rightmost leaf) unicasts the group's blinded key to M_{n+1} .
2. The group and the new node create a new group. The new root key node has two children: the root node of the original tree on the left and M_{n+1} on the right. Every member can now compute the

group key because all original members only need the new member's blinded key, and the new member M_{n+1} needs only the blinded group key of the original group.

The costs of the join operation are summarized in Table 4. We assume that before the join operation, the sponsor of the original group and a new joining member generate their own blinded key and therefore do not include these costs in the table.

CCEGK, as do EGK and STR, always joins at the root of the tree, resulting in potentially far fewer sequential exponentiations. TGDH will join at the root only if the tree is a full tree, in an attempt to keep the tree more balanced. STR, on the other hand always has a skinny tree, where every internal node has one child that is a leaf.

For purposes of comparison, the left-hand side of Fig. 1 shows a sample tree taken from Fig. 2 of the TGDH paper [3]. In this example, member M_4 is joining the group $BT1$. In [3] the new node is added to the bottom of the tree. In this example, the sponsor M_3 performs the following actions:

1. generates a new root node and a new member node.
2. sets the new root node as the parent node of the original root node and the new member node, denoted as node $\langle 0,0 \rangle$.

After these actions, all members in the original group know $bK_{(1,1)}$; and the new member M_4 knows $bK_{(1,0)}$, and if it has previously computed its own blinded key, then every member can compute the new group key in one round with one sequential exponentiation.

Table 3
Costs of initialization operation

Communication				Computation		
Rounds	Messages	Unicast	Broadcast	Sequential exponentiations	Signatures	Verifications
h	$2n - 2$	n	$n - 2$	$2h - 2$	h	$2n - 2$

Table 4
Costs of the join operation

Communication				Computation		
Rounds	Messages	Unicast	Broadcast	Sequential exponentiations	Signatures	Verifications
1	2	1	1	1	1	2

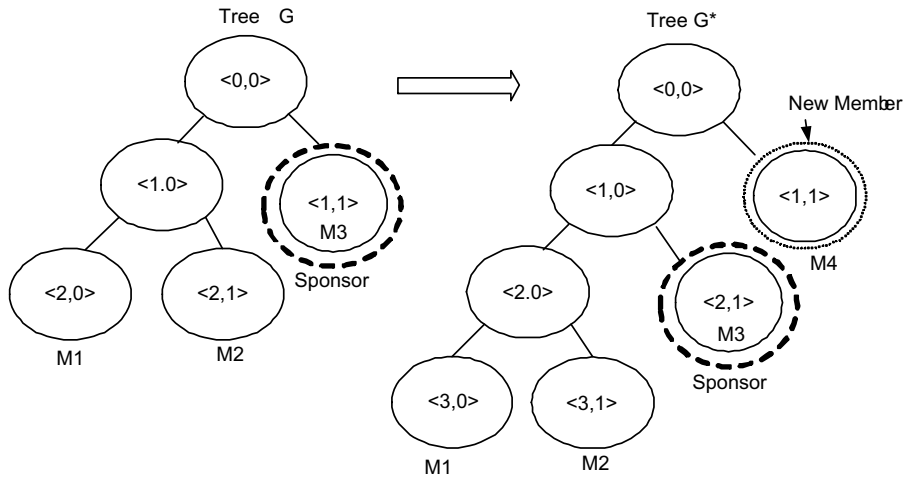


Fig. 1. Tree updates from the join operation.

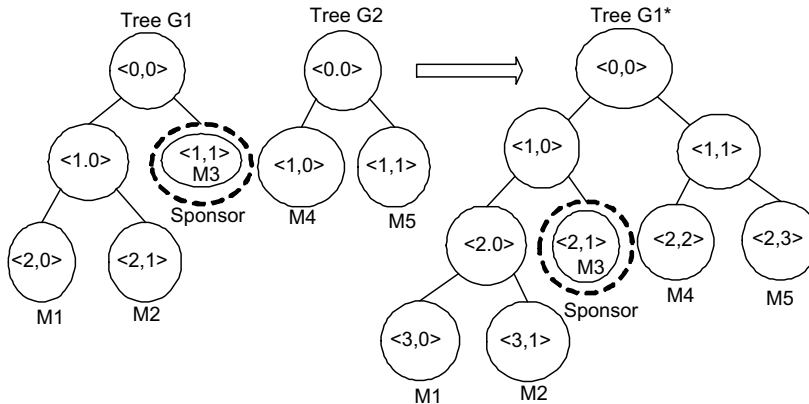


Fig. 2. Tree updates from the merge operation.

4.3. Merge operation

The *merge* operation occurs when two or more subgroups wish to merge into one group. The sponsor announces the merge event for these subgroups. These subgroups receive the notification and they perform DH exchanges to establish a new group key. The procedure of the two-subgroup merge operation is as follows:

1. Suppose we have two groups G_1 and G_2 in the first round. The sponsor of each subgroup (the shallowest rightmost leaf) broadcasts a merge request message including the blinded key of the group to the members of the other group.
2. G_1 and G_2 create a new group. The new root key node has two children: the root node of G_1 on the

left and the root node of G_2 on the right. Every member in the new group can compute the group key because all the members in G_1 know the blinded group key of G_2 and all the members in G_2 know the blinded group key of G_1 .

Fig. 2 shows an example of Group $G_1 = \{M_1, M_2, M_3\}$ merging with Group $G_2 = \{M_4, M_5\}$, where the sponsor is M_3 . Since all members in G_1 know $bK_{(1,1)}$, and all members in G_2 know $bK_{(1,1)}$, every member can compute the group key in one round. The costs of the two-subgroup merge operation are summarized in Table 5.

We can extend the merge operation to an N -subgroup merge operation: the sponsor of every subgroup broadcasts the blind key of its own group key to every member, including the group members

Table 5
Costs of merge operation

Communication				Computation		
Rounds	Messages	Unicast	Broadcast	Sequential exponentiations	Signatures	Verifications
1	N	0	N	N	1	N

and the members of the other subgroups. Every member knows his position in the key tree. The sponsor of the new group internally determines the iterative merger of the group as above, calculating the blinded keys up the path to the root. It then broadcasts these to all members who can then internally rekey. So in an N -subgroup merge operation we only need one round, N messages, and N sequential exponentiations.

CCEGK, as with EGK, merges the two roots of the trees, resulting in lower cost but a potentially more unbalanced tree. TGDH merges the shallower tree to the deeper one, or at the root, whichever creates the shallowest final tree. STR adds one tree to the bottom of the other.

4.4. Mass join (N -member join) operation

The *mass join* operation occurs when two or more joining members send join requests to the group sponsor. The sponsor then announces the mass join event to the current group and the joining members. The current group and these members receive the notification and they perform DH exchanges to establish a new group key. EGK [1] put forward two ways to implement this mass join. In this paper we still use these two methods and supplement them with a new method.

1. Mass join-iterative [1]: treats N members as N separate single member joins.
2. Mass join-merge [1]: initialize the N entities to form their own independent group, then merge this group with the original group to generate a new group.
3. Mass join-simultaneous: The sponsor of the current group and N joining members broadcast a message including their blinded key to every member. Due to VS we know that every member has received this set of blinded keys before starting the mass-join operation. The sponsor then calculates all the blinded keys created as if these members were being added sequentially and broadcasts the keys to all members. Each member can determine the same view of the system

and can calculate his position in the key tree; every member can generate the group key of the new group. In this method, we use one round, $N + 1$ messages, and $2(N - 1)$ sequential exponentiations.

The costs of the mass join operation are summarized in Tables 6 and 7.

Mass join-iterative is most expensive among these three methods in the communication and computation costs. In communication costs, mass join-simultaneous is more efficient than mass join-merge, whereas in computation costs, mass join-merge is more efficient than mass join-simultaneous. In this paper, we use the mass join-simultaneous method when we compare the costs of communication and computation in the mass join operations of the other protocols.

Table 6
Communication cost of mass join

Method	Communication			
	Rounds	Messages	Unicast	Broadcast
Mass join-iterative	N	$2N$	N	N
Mass join-merge	$h' + 1$	$2N$	N	N
Mass join-simultaneous	1	$N + 1$	0	$N + 1$

h' is the height of the key tree that is created by N joining members, $h' = \lceil \log_2 N \rceil$.

Table 7
Computation cost of mass join

Method	Computation		
	Sequential exponentiations	Signatures	Verifications
Mass join-iterative	$2N - 1$	N	$2N$
Mass join-merge	$2h'$	$h' + 1$	$2N$
Mass join-simultaneous	$2(N - 1)$	1	$N + 1$

h' is the height of the key tree that is created by N joining members, $h' = \lceil \log_2 N \rceil$.

4.5. Leave operation

The *leave* operation occurs when a group member sends a leave request to the group. The sponsor announces the leave event to the remaining group members, who then do DH exchanges to create a new shared key. In EGK we required a leave operation to fully reinitialize the tree. To reduce costs, the procedure of the CCEGK leave operation is very similar to that in TGDH [2,3]:

1. Suppose that we have n members in the group, and a group member M_i wants to leave the group. The sponsor is the shallowest rightmost sponsor of the subtree rooted at M_i 's sibling node. In the leave operation, every member updates its key tree by deleting the sponsor corresponding to M_i . The former sibling of M_i replaces M_i 's parent node. The sponsor picks a new secret key and computes all keys on its key path up to the root.
2. After the sponsor generates all the keys, it broadcasts a message including all the blinded keys to the other members in the group. After the other group members receive the message, they can compute the group key for each node on up the new tree.

Fig. 3 shows an example of member M_3 leaving from a group where the sponsor is M_2 .

The costs of the leave operation in the worst case are shown in Table 8. The total sequential exponentiations in the leave operation depend on the position of the leaving member. For purposes of evaluation, we will only consider the worst case in the leave operation.

Since our join and merge operations add to the root node, our tree can become very unbalanced. At some point the cost of the above leave operation may be greater than the cost of initializing a new group. Under such cases, instead of the above operation, we will invoke a rebalance operation (see Section 5).

4.6. Mass leave (N -member leave) operation

The *mass leave* operation occurs when two or more group members send leave requests to the group. The sponsor announces the mass leave event to the remaining group members. The remaining group members receive this notification and do DH exchanges to create a new group key. There are two ways to implement the mass leave operation, depending on the number of leaving group members. Suppose that we have n members in the

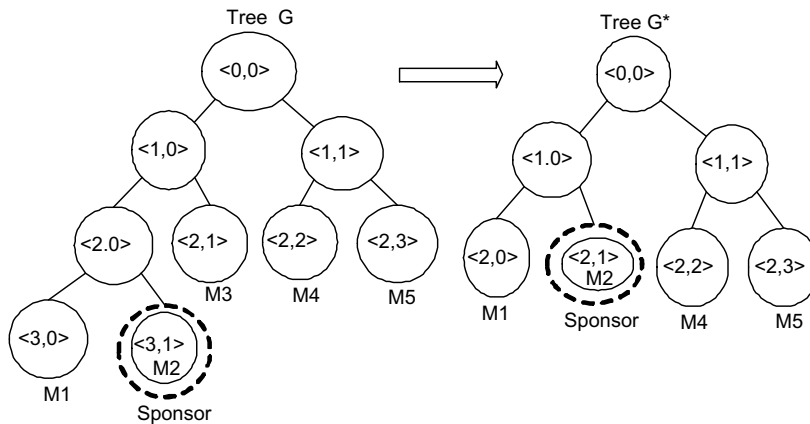


Fig. 3. Tree updates from the leave operation.

Table 8
Costs of leave operation

Communication				Computation		
Rounds	Messages	Unicast	Broadcast	Sequential exponentiations	Signatures	Verifications
1	1	0	1	$3h - 3$	1	1

group and N members will leave the group. The two methods are as follows:

- (I) *Mass leave-balanced*: When the number of leaving group members is very large, it is more efficient to reconstruct the key tree using the initialization operation in Section 4.1. The total number of rounds is $\lceil \log_2(n - N) \rceil$; the total number of messages is $2(n - N - 1)$. The sequential exponentiations are $2\lceil \log_2(n - N) \rceil$. In this method, the updating key tree becomes balanced. Consequently, a later rebalance scheme is not needed.
- (II) *Mass leave-imbalanced*: When there are fewer leaving group members, it is more efficient to use the mass leave-imbalanced method. The procedure is as follows:
 1. The sibling of every leaving node replaces its parent node. Every remaining member updates its key tree by deleting all leaving members. For every leaving node, we determine its sibling node and its parent node and replace its parent node with its sibling node. Then every sponsor, after deleting every leaving node, uses the same criteria as described in the leave operation. We choose the shallowest rightmost sponsor as the sponsor in the new key tree. To prevent reusing the old group key, the shallowest rightmost sponsor in the array picks a new secret key. We update the order number and the level of every node in the new tree. We put every sponsor into two initial empty arrays: A , called sponsor array, and B , called intermediate node array.

2. The elements in A calculate their secret keys and blinded keys on their key-path as far as possible in parallel. For example, an element $\langle k, m \rangle$ in B can calculate its secret keys and blinded keys up to the node $\langle l, i \rangle$. We check whether $\langle l, i \rangle$ is already in B . If not, we insert the node $\langle l, i \rangle$ into B , and remove $\langle k, m \rangle$ from B . And so on for the other elements in B . After that, we check whether elements in B have an offspring-ancestor relationship. If they do, we will remove all the offspring from B . Now, for every element $\langle l, i \rangle$ in B , we will search the nodes in A which are the offspring of $\langle l, i \rangle$, keep the shallowest rightmost node in A and remove the other offspring from A . Now every element in A will broadcast all of its blinded keys to all the members in the new balanced tree. After receiving the messages, all the members calculate the blinded keys and secret keys on their key-path as far as possible in parallel.
3. We iterate Step 2 until the number of elements in B is 1 and the element is the root node.

In the second method, the updating key tree is still imbalanced. Because rounds, messages and sequential exponentiations in the mass leave operation depend on the position of the leaving members, we will consider the worst case in the mass leave operation. In the worst case, we need $\min(\lceil \log_2 N \rceil + 1, h)$ rounds, $\min(2N, n - N)$ messages and $3h - 3$ sequential exponentiations (note that in general h is larger than $\lceil \log_2(n - N) \rceil$). Fig. 4 shows an example of members M_1 and M_4 leaving from a group using the second method.

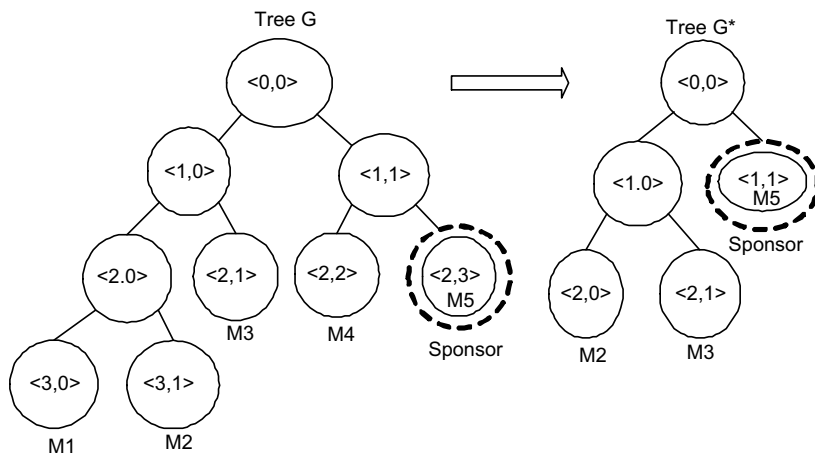


Fig. 4. Tree updates from the mass-leave operation.

Table 9
Communication cost of mass leave

Method	Communication			
	Rounds	Messages	Unicast	Broadcast
Balanced	h	$2(n - N - 1)$	$n - N$	$n - N - 2$
Imbalanced	$\min(\lceil \log_2 N \rceil + 1, h')$	$\min(2N, \lceil n - N \rceil)$	0	$\min(2N, \lceil n - N \rceil)$

Table 10
Computation cost of mass leave

Method	Computation		
	Sequential exponentiations	Signatures	Verifications
Balanced	$2h$	h	$2(n - N - 1)$
Imbalanced	$3h' - 3$	$\min(\lceil \log_2 N \rceil + 1, h')$	$\min(2N, \lceil n - N \rceil)$

h and h' are the height of updating key tree after N members leave the group using the mass leave-balanced and mass leave-imbalanced schemes respectively. $h = \lceil \log_2(n - N) \rceil$ and in general h' is larger than h .

The worst-case costs of the mass leave operation are summarized in Tables 9 and 10.

From Tables 9 and 10, we know the number of messages and verifications in mass leave-balanced are greater than those in mass leave-imbalanced. Total sequential exponentiations in mass leave-balanced are smaller than those in mass leave-imbalanced, in both the average and worst case. When $N \geq \frac{n}{2}$, the number of rounds and signatures in mass leave-balanced are smaller than those in mass leave-imbalanced. Otherwise, the number of rounds and signatures in mass leave-balanced are larger than those in mass leave-imbalanced.

4.7. Partition operation

The *partition* operation occurs when the current group will split into two or more subgroups. The sponsor announces the partition event to every subgroup. Every subgroup receives this notification and performs DH exchanges to create a new group key. The partition operation is similar to the mass leave operation; it can apply two different methods: *partition-balanced* and *partition-imbalanced* to the partition operation corresponding to mass-leave-balanced and mass-leave-imbalanced methods. Suppose that we have n members in the group and the group will be split into N separate subgroups G_1, G_2, \dots, G_N . For every subgroup, we treat the group members of the other subgroups as leaving members. Therefore, the partition operation can be treated as an N -mass leave operation. Note that the N -mass leave operations are simultaneous. The total number of rounds is the largest of the rounds

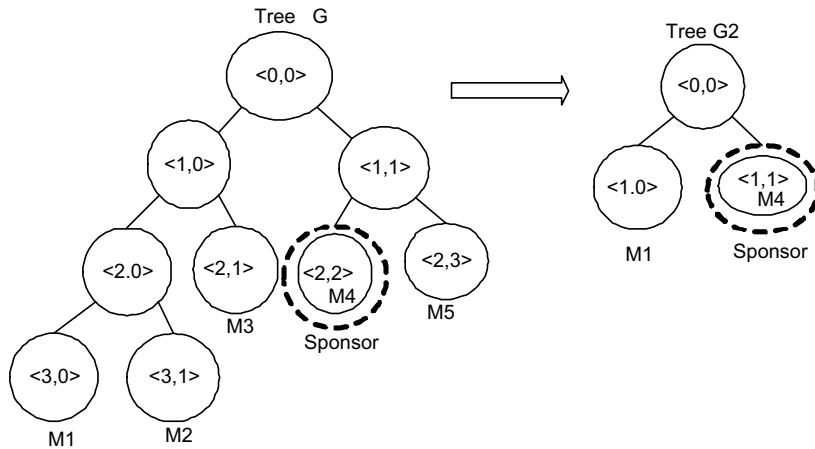
from the separate N -mass leave operations, and the number of sequential exponentiations is the maximum sequential exponentiations among the N -mass leave operations. The number of messages is the largest of all the messages in every mass leave operation. Figs. 4 and 5 are examples of splitting two subgroups from the original group. The first of these figures is identical to that of the mass-leave operation. The two subgroups are $G_1\{M_2, M_3, M_5\}$ and $G_2\{M_1, M_4\}$, respectively.

Suppose the total size of leaving members for subgroup G_i is P_i . h_i and h'_i are the height of the updating key tree in the i th subgroup ($1 \leq i \leq N$) in partition-balanced and partition-imbalanced, respectively (note that in general $h'_i > h_i$). The worst case costs of the partition operation are summarized in Tables 11 and 12. Since this is a partition operation and not a mass leave, we consider the costs of reforming both subgroups in contrast to the costs published by Kim et al. [2,3] which only look at the costs of one of the subgroups.

4.8. Key refresh

To avoid exposure of the group's secret key, we should refresh it periodically. The group's sponsor picks a new secret key and computes its keys and blinded keys up to the root of the tree. After computing the group key, it broadcasts a message including all the blinded keys to the other members. Upon receiving the message, the other members can calculate their keys and blinded keys.

Because sequential exponentiations depend on the position of the group sponsor, we will consider

Fig. 5. Tree updates for subgroup G_2 in the partition operation.Table 11
Communication cost of partition

Method	Balanced	Imbalanced
Rounds	$\max(h_i)$	$\max(\min(\lceil \log_2 P_i \rceil + 1, h'_i))$
Messages	$\max(2(n - P_i - 1))$	$\max(\min(2P_i, n - P_i))$
Unicast	$\max(n - P_i)$	0
Broadcast	$\max(n - P_i - 2)$	$\max(\min(2P_i, n - P_i))$

the worst case costs in the key refresh operation, summarized in Table 13.

5. Rebalance scheme

After numerous add, merge, leave, mass leave, and partition operations, the key tree can become quite unbalanced. So, when the key tree reaches a certain imbalance point, usually caused by leave, mass leave, or partition operations, we should rebal-

ance the tree and treat the rebalance operation as a leave, mass leave, or partition operation. The imbalance point can vary depending on network and efficiency requirements. One possible rule of thumb is to rebalance during a leave operation when the cost of rebalancing exceeds that of leaving. We discuss two ways to implement the rebalance scheme on a tree of size n :

1. Choose a subtree as the basic tree. To rebalance we have two requirements, the height, h , of the basic tree is $\lceil \log_2 n \rceil$ and the basic tree is more densely populated than the other subtrees of the same height. The height of the basic tree is easy to calculate, and it is also easy to determine which subtree is the most dense by counting the number of members. After we determine the basic tree, we choose the shallowest rightmost sponsor in the basic tree as the sponsor in the

Table 12
Computation cost of partition

Method	Computation		
	Sequential exponentiations	Signatures	Verifications
Balanced	$\max(2h_i)$	$\max(h_i)$	$\max(h_i)$
Imbalanced	$\max(3h'_i) - 3$	$\max(\min(\lceil \log_2 P_i \rceil + 1, h'_i))$	$\max(\min(2P_i, n - P_i))$

Table 13
Costs of key refresh

Communication				Computation		
Rounds	Messages	Unicast	Broadcast	Sequential exponentiations	Signatures	Verifications
1	1	0	1	$3h - 3$	1	1

new balanced tree. We treat every group member which is not in the basic tree as a separate entity and insert them into the basic tree from the shallowest level down to the deepest level. After we finish inserting all the entities, we update the order number and the level of every node in the new tree. The height of the new key tree is h and the new key tree becomes more balanced, if $\log_2 n$ is an integer, the tree is a full binary tree.

2. Alternatively, we divide the original tree into several subtrees. The height of every subtree is at most h . The division procedure is as follows:
 - (a) We choose a subtree with the largest number of members and height no more than h . We split off that subtree.
 - (b) We repeat Step (a) to split every remaining subtree, until the height of every remaining subtree is no more than h . Note that in the end the heights of some subtrees may be less than h .
 - (c) We sort these subtrees from highest height to lowest height and from largest to smallest group size. We denote the largest subtree as the basic tree. We treat every subtree as an entity. We choose the shallowest rightmost node of every entity as the sponsor of the entity. The sponsor of the basic tree determines the insertion position for every entity and informs the other subtree sponsors of the positions in the basic tree. Our insertion strategy is as follows: we insert the subtrees sorted from highest to lowest height into the nodes from the shallowest level down to the deepest level. If the joining entities increase the height of the basic tree, they join to the root node of the basic tree. After we finish inserting all the entities, we update the order number and the level of every node in the new tree.

Picking one of the above methods, we perform the following actions:

1. We put every sponsor into two initial empty arrays: A , the sponsor array, and B , the intermediate node array. The nodes are sorted by depth order. The elements in A calculate their secret keys and blinded keys on their key-path as far as possible in parallel. For example, an element $\langle k, m \rangle$ in B can calculate its secret keys and blinded keys up to the node $\langle l, i \rangle$. We check

whether the node $\langle l, i \rangle$ is already in B ; if not, we insert the node $\langle l, i \rangle$ into B . We then remove $\langle k, m \rangle$ from B , and so on for the other elements in B . After that, we check whether some elements in B have an offspring-ancestor relationship. If so, we will remove all the offspring from B . Now for every element $\langle l, i \rangle$ in B , we will search the nodes in A which are the offspring of $\langle l, i \rangle$, keep the shallowest rightmost node, and remove the other offspring from A . Now every element in A broadcasts all its blinded keys to all the members in the new balanced tree. After receiving the messages, all the members calculate the blinded keys and secret key on their key-path as far as possible in parallel.

2. We iterate Step 1 until the number of elements in B is one and that element is the root node.

Suppose the number of total entities including the basic tree is N , and (using the second method) the height of the new rebalanced tree is h' . In the first method, in the worst case, we need $\min(-\log_2 N + 1, h)$ rounds, $2N$ messages, and $3h - 3$ sequential exponentiations. The advantage of the first way is that it can let the tree become a full binary tree, and the number of rounds and sequential exponentiations are very small. The drawback, however, is that N may be very large because we treat every group member that initially is not in the basic tree as an entity and they could be quite numerous. The total communication cost is quite expensive when N is very large. In the second method, in the worst case, we need $\min(\lceil \log_2 N \rceil + 1, h')$ rounds, $\min(2N, n - N)$ messages, and $3h' - 3$ sequential exponentiations. The advantage of the second method is that N is much smaller than that of the first method. Although the height of the key tree might be increasing, it increases slowly. Therefore, the number of the rounds and sequential exponentiations are still small, but the number of messages is much smaller than that of the first method. The drawback is that the new tree is still unbalanced. So, when N is very small, we shall use the first method. If N becomes very large, we will consider using the second method. Note that in EGK, the key tree will be rebalanced in the leave, mass leave, and partition operations by reconstructing the new key tree. Although the number of rounds is h , the communication and computation costs are very expensive because the number of rounds is $2n$, as is the number of verifications. In TGDH after the merge, leave, mass leave and partition operations,

the key tree also becomes unbalanced but they do not provide a rebalance scheme, rather they rely on the heuristic choices of insertion points for joins and merges to maintain a partial balance. In STR, the key tree is the most unbalanced compared to EGK, TGDH, and CCEGK, and STR does not provide a rebalance scheme.

Fig. 6 shows an example of the first rebalance scheme. We perform the following actions:

1. We will choose the subtree whose root node is $\langle 2, 0 \rangle$ as a basic tree and M_4 as the group controller. Suppose at first, the sponsor array A and the intermediate node array B are empty. Node $\langle 2, 1 \rangle$ is the sibling of $\langle 4, 1 \rangle$, and $\langle 1, 1 \rangle$ is the sibling of $\langle 3, 1 \rangle$. The key tree is updated where $\langle 5, 0 \rangle$ becomes $\langle 3, 0 \rangle$ in the new tree, and so on. We put $\langle 3, 3 \rangle$, $\langle 2, 2 \rangle$, and $\langle 2, 3 \rangle$ into A and B .
2. In the first round, $\langle 3, 3 \rangle$, $\langle 2, 2 \rangle$ and $\langle 2, 3 \rangle$ calculate their blinded keys and secret keys. Node $\langle 3, 3 \rangle$ can calculate the secret keys $K_{\langle 2, 1 \rangle}$ and $K_{\langle 1, 0 \rangle}$ and blinded keys $bK_{\langle 2, 1 \rangle}$ and $bK_{\langle 1, 0 \rangle}$ because it knows the blinded keys $bK_{\langle 3, 2 \rangle}$ and $bK_{\langle 2, 0 \rangle}$. We put $\langle 1, 0 \rangle$ into B and remove $\langle 3, 3 \rangle$. Node $\langle 2, 2 \rangle$ can calculate the $K_{\langle 1, 1 \rangle}$ and $bK_{\langle 1, 1 \rangle}$ because it knows $bK_{\langle 2, 3 \rangle}$, we remove $\langle 2, 2 \rangle$ from B and insert $\langle 1, 1 \rangle$ into B . We remove $\langle 2, 3 \rangle$ from B , because in A , $\langle 2, 2 \rangle$ and $\langle 2, 3 \rangle$ are the offspring of $\langle 1, 1 \rangle$ in B . We keep $\langle 2, 2 \rangle$ in A and remove $\langle 2, 3 \rangle$ from A , then $\langle 3, 3 \rangle$ and $\langle 2, 2 \rangle$ broadcast their message to

every member. Every member can calculate its blinded keys and secret keys.

3. We remove $\langle 1, 0 \rangle$ and $\langle 1, 1 \rangle$ from B and insert $\langle 0, 0 \rangle$ into B . We remove $\langle 3, 3 \rangle$ from A . Node $\langle 2, 2 \rangle$ broadcasts all the blinded keys to every member, every member can calculate the group key.

6. Authentication

Although the algorithm of CCEGK based on DH is secure and not easy to break, the entire system is vulnerable if the keys are not securely distributed. Therefore, we should implement an authentication algorithm in CCEGK. Our protocol will authenticate any operation, such as an addition of new members, a leaving group member, or mass leaving group members. We suppose that it is acceptable for this authentication to use public-key authentication. There are several ways to authenticate a group key exchange, such as centralized authentication, implicit authentication, and pairwise authentication [1,20,12,21,22]. The authentication of CCEGK can use any of the above authentication schemes. Centralized authentication is based on a digital signature and a public key infrastructure. A group may assign a single trust sponsor to perform authentication for the whole group, which will occur in the first step of the key

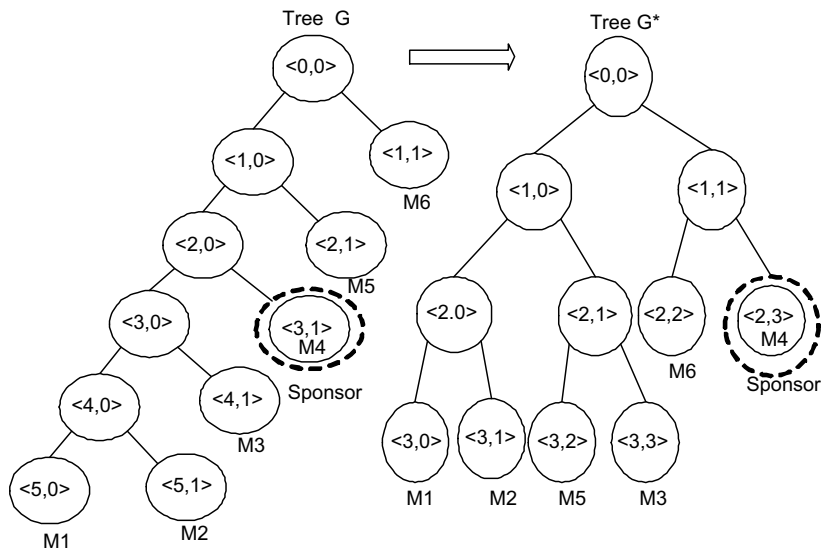


Fig. 6. Tree operations from the rebalance scheme.

exchange algorithm. Every group member sends its public key PK_i using an authenticated message. The sponsor validates the authenticity of the senders and sends a message to all authenticated group members. The drawback of centralized authentication is that the certificates are to be exchanged, so they consume bandwidth. Implicit authentication authenticates the sponsor and trusts the sponsor to have authenticated group members. This method directly maps to authenticated two-party key exchange algorithms in which the group members authenticate the blinded key transmitted by their partner. In pairwise authentication, every member encrypts a message with the group key and then broadcasts it in an authenticated message to all group members. If any of the authentications fails, the group key is discarded and a new key is generated without the unauthenticated member. In this paper, in order to compare the computation cost between the four protocols, for convenience and because TGDH and STR use the centralized authentication scheme, we suppose CCEGK uses the centralized authentication scheme (EGK sup-

ports centralized, implicit, and pairwise authentication and gives the algorithms in detail in [1]).

7. Theoretical comparison of four protocols

In this section, we analyze the complexity of the four protocols CCEGK, EGK, TGDH, and STR. Complexity includes the cost of communication (such as number of rounds, number of messages, number of direct messages, number of broadcast messages) and the cost of computation (number of sequential exponentiations, signatures, and verifications).

Because communication cost is important in high-delay networks, we cannot ignore it. A protocol is needed to balance the cost. Tables 14 and 15 summarize the costs of communication and computation for every operation among four protocols in the worst case situation.

The number of current group members is n ; the number of mass join members, mass leave members and merging groups (including the original group) is N and the number of merging members is m . We use

Table 14
Table in comparison of communication

Protocols		Communication			
		Rounds	Messages	Unicast	Broadcast
CCEGK	Initialization	h	$2n - 2$	n	$n - 2$
	Join	1	2	1	1
	Mass join	1	$N + 1$	0	$N + 1$
	Merge	1	N	0	N
	Leave	1	1	0	1
	Mass leave	$\min(h' + 1, h)$	$\min(2N, n - N)$	0	$\min(2N, n - N)$
EGK	Initialization	h	$2n - 2$	0	$2n - 2$
	Join	1	2	0	2
	Mass join	$h' + 1$	$2N$	0	$2N$
	Merge	N	$2N - 2$	0	$2N - 2$
	Leave	h	$2(n - 1)$	0	$2(n - 1)$
	Mass leave	h	$2(n - N)$	0	$2(n - N)$
TGDH	Initialization	h	$2n - 2$	0	$2n - 2$
	Join	2	3	0	3
	Mass join	$h' + 1$	$2N$	0	$2N$
	Merge	$h' + 1$	$2N$	0	$2N$
	Leave	1	1	0	1
	Mass leave	$\min(h' + 1, h)$	$\min(2N, n - N)$	0	$\min(2N, n - N)$
STR	Initialization	$n - 1$	$2n - 2$	0	$2n - 2$
	Join	2	3	0	3
	Mass join	2	$N + 2$	0	$N + 2$
	Merge	2	$N + 1$	0	$N + 1$
	Leave	1	1	0	1
	Mass leave	1	1	0	1

Table 15
Table in comparison of computation

Protocols		Computation		
		Sequential exponentiation	Signatures	Verifications
CCEGK	Initialization	$2h - 2$	h	$2(n - 1)$
	Join	1	1	1
	Mass join	N	1	$N + 1$
	Merge	$N - 1$	1	N
	Leave	$3h - 3$	1	1
	Mass leave	$3h - 3$	$\min(h' + 1, h)$	$\min(2N, n - N)$
EGK	Initialization	$2h - 2$	h	$2n - 2$
	Join	1	1	2
	Mass join	$2h'$	$h' + 1$	$2N$
	Merge	$2N$	N	$2N - 2$
	Leave	$2h$	h	$2(n - 1)$
	Mass leave	$2h$	h	$2(n - N)$
TGDH	Initialization	$2h - 2$	h	$2(n - 1)$
	Join	$3h - 3$	2	3
	Mass join	$3h - 3$	$h' + 1$	$2N$
	Merge	$3h - 3$	$h' + 1$	$2N$
	Leave	$3h - 3$	1	1
	Mass leave	$3h - 3$	$\min(h' + 1, h)$	$\min(2N, n - N)$
STR	Initialization	$2(n - 1)$	$n - 1$	$2(n - 1)$
	Join	4	2	3
	Mass join	$3N + 1$	2	$N + 2$
	Merge	$3m + 1$	2	$N + 1$
	Leave	$\frac{3n}{2} + 2$	1	1
	Mass leave	$\frac{3n}{2} + 2$	1	1

h to denote the height of the updating key tree and define h' as $\lceil \log_2 N \rceil$. It is noted that Kim et al. do not implement the initialization operation in TGDH and STR. The total number of messages and total verifications in the mass leave operation of TGDH are inconsistent with each other in [3] and [7]. In this paper, we implement the initialization operation in TGDH and STR, and we use a modified value explained in Appendix A that seems to better fit their description of their algorithm; see Appendix A for further details. It is important that we see how these protocols operate when multiple operations are executed. In Section 8 we therefore compare the average costs of multiple instances of these operations to get a better feel by experimental simulation.

8. Experimental comparison

8.1. Simulation setup

Amir et al. compare the performance of each single operation among GDH3.0, BD, CKD, TGDH, and STR for different group sizes [7]. The drawback

of their simulations is that they only compared the costs of single operations and did not consider the costs of combined operations. For example, Amir et al. report that for the join results of their simulation, STR outperforms other protocols [7]. However, for the leave operation, TGDH outperforms other protocols. From these results, it is difficult to determine that TGDH is more efficient than STR because we do not know what happens if we run join and leave operations together.

In our simulation, we consider the average costs of combined operations among EGK, TGDH, STR, and CCEGK. We combine the operations join, leave, mass join, mass leave, merge, and partition. In our tests we assume that each operation occurs with a fixed probability, independent of the other operations. We have generated ten scenarios which indicate different relative probabilities for these operations.

8.2. Test scenarios

The costs of communication and computation in each protocol depend on a number of factors. We

design our simulations to take into account some of these factors. For EGK, for example, the communication and computation costs do not depend on the position(s) of the joining or leaving member(s) because in the joining operations it adds the new member(s) or merging groups to the root of the original key tree, and in many others it reconstructs a new balanced binary tree. The communication and computation costs of TGDH and CCEGK depend on the position(s) of the leaving or joining member(s), tree height, and the balancing of the tree. STR's communication and computation costs depend on the position(s) of leaving member(s), and the size of merging group members. Based on the above analysis, we restricted our simulations as follows:

- For EGK, TGDH, and CCEGK, we first build a random, unbalanced binary tree (not using an initialization operation) to create the original tree structure that could possibly exist at some point in time.
- For STR, we obey the specified rules to create the binary tree using its initialization operation.
- Since any fault-tolerant group key agreement protocol requires a VS implementation [3], we do not include the overhead costs of this implementation in this paper.

We initially conducted our experiments with initial group sizes 200, 600, and 1000. We saw that the relative performance of the protocols was the same in these groups, so we only report the results for groups of 600 in this paper. The combined operations are defined for ten scenarios as depicted in Table 16. The scenarios are sorted with decreasing probabilities of single member joins relative to numbers of leave operations. To summarize, our experimental configuration was:

Table 16
Ten scenarios in the simulation

Scenario	Join	Leave	Mass join	Mass leave	Merge	Split
1	95	5	0	0	0	0
2	85	5	2	3	2	3
3	75	10	4	3	4	4
4	65	20	4	3	4	4
5	55	25	6	4	5	3
6	45	30	7	5	6	7
7	35	35	8	7	7	8
8	25	40	9	9	8	9
9	15	50	9	9	8	9
10	5	55	10	10	9	11

- total operations per run: 100.
- initial group size for every group operation: 600.
- result presented are averages over 10 runs.
- during each merge operation, we merge two subgroups to a new group.
- during a partition operation, the original group is divided into two subgroups of equal size. After doing a partition, we randomly choose one of the new subgroups as the current group.
- for leave, mass leave, and partition experiments, the leaving members are chosen randomly over all members.
- for mass join or mass leave, we randomly choose from 2 to 10 members to join or leave.

In each of the group operations, we compute the averages for phases, messages, sequential exponentiations, signatures, and verifications. Since average verifications are equal to average messages, and average signatures are equal to average phases, we use the same graphs to describe them.

8.3. Result of simulation without a CCEGK rebalance

Fig. 7 shows average phases, Fig. 8 demonstrates average messages, and Fig. 9 presents average sequential exponentiations when the size of each original group is 600 and without using a rebalance scheme.

In Fig. 8 as the number of join operations decreases and all other operations increase, the average messages of EGK increase quickly – this is due to the requirement of EGK to reform the tree with the leaving operations. However, average messages of STR remain fairly constant while average messages of TGDH and CCEGK increase steadily. In the long run, average messages of STR are the smallest, followed by CCEGK, TGDH, and EGK.

In Fig. 9 as the number of join operations decrease and all other operation increase, we find the average sequential exponentiations of EGK, TGDH, and CCEGK increase steadily while average sequential exponentiations of STR are increasing sharply. In the long run, average sequential exponentiations of EGK are the smallest, followed by TGDH, CCEGK, and STR.

8.4. Result of simulation with a CCEGK rebalance

From the above results without a rebalance scheme, we now consider the use of the CCEGK rebalance over scenarios 5–10. After we execute

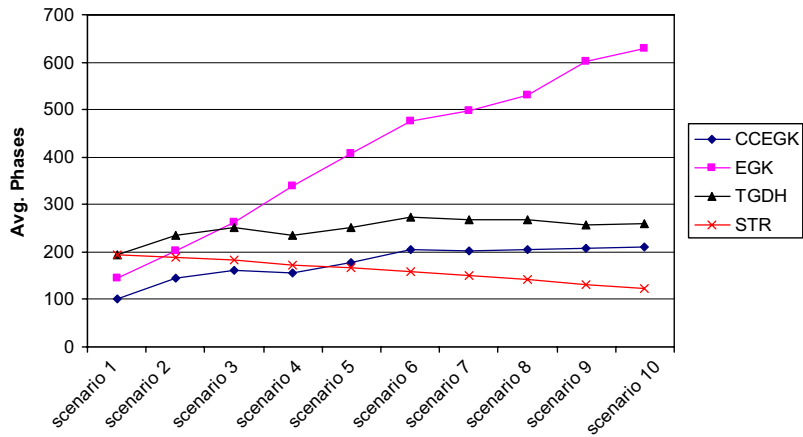


Fig. 7. Average phases without rebalance scheme for four protocols.

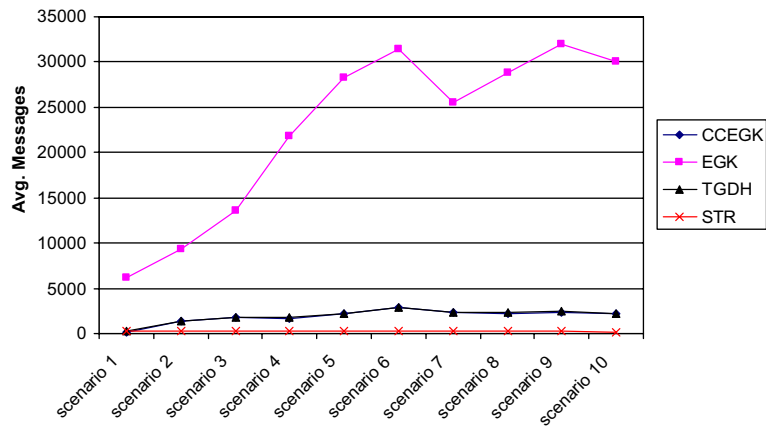


Fig. 8. Average messages without rebalance scheme for four protocols.

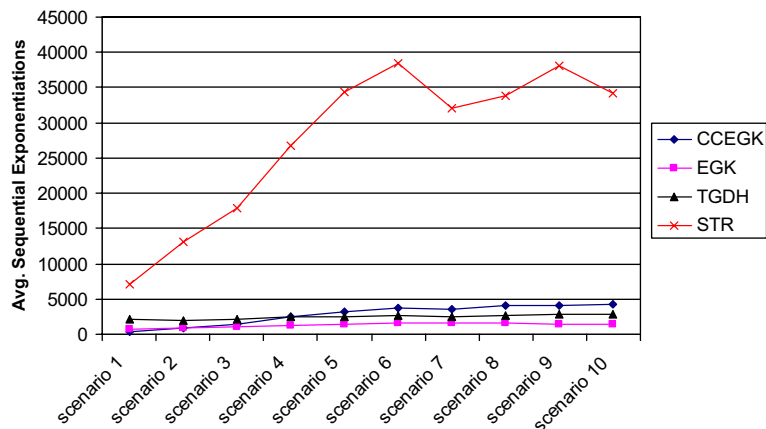


Fig. 9. Average sequential exponentiations without rebalance scheme for four protocols.

about 40 operations, if a partition operation occurs, we implement the rebalance scheme. Fig. 10 shows

average phases, Fig. 11 demonstrates average messages, and Fig. 12 presents average sequential

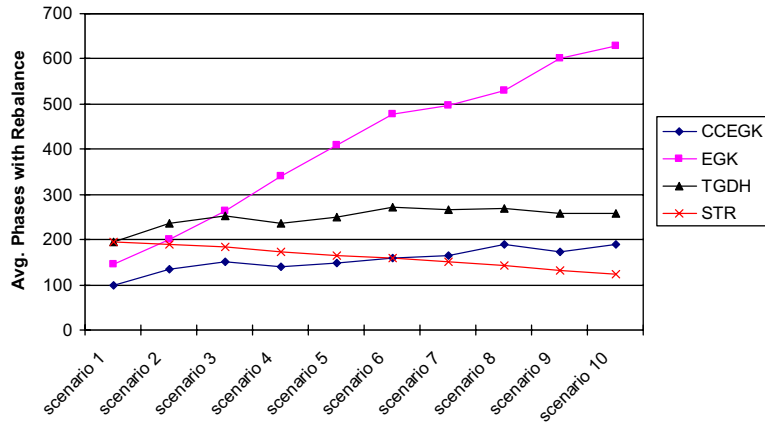


Fig. 10. Average phases with rebalance scheme for four protocols.

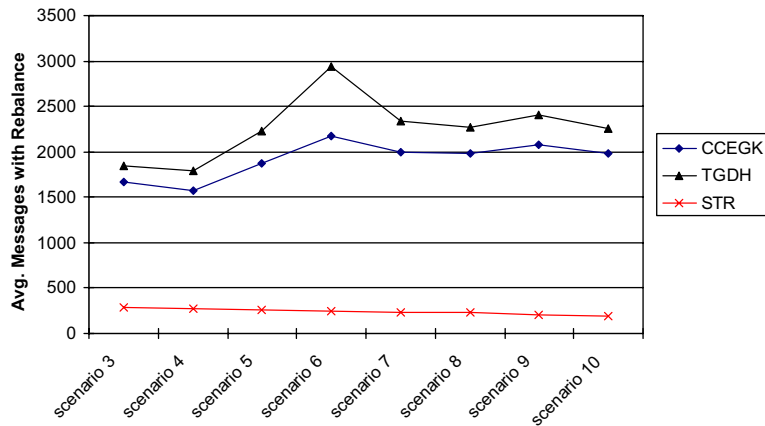


Fig. 11. Average messages with rebalance scheme for best three protocols.

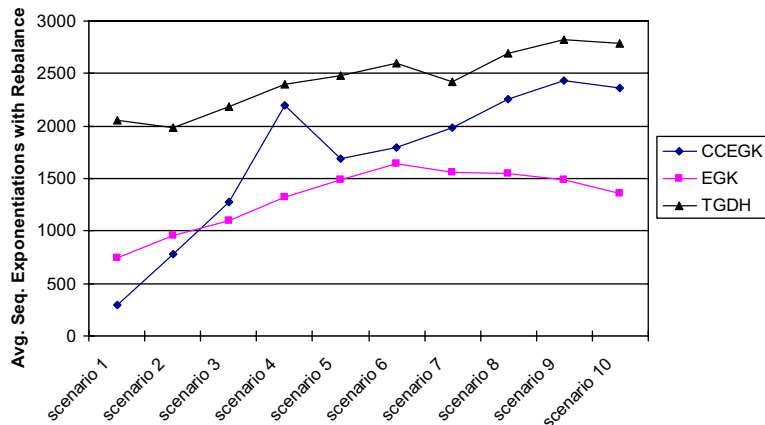


Fig. 12. Average sequential exponentiations with rebalance scheme for best three protocols.

exponentiations when the size of each original group is 600 and with using a rebalance scheme.

In Fig. 10, as the number of join operations decreases and all other operations increase the average phases in STR decrease steadily. However, average phases of CCEGK and TGDH rise gradually while those of EGK rise sharply. Average phases of CCEGK with a rebalance scheme are a little smaller than its average phases without a rebalance scheme because the rebalance scheme makes its group members more symmetrically distributed, which reduces the average phases when we do a mass leave or partition operation. In the long run, average phases of STR and CCEGK are almost equal and the smallest, followed by TGDH and EGK.

In Fig. 11, as the number of join operations decreases and all other operations increase, the average messages of STR remain fairly constant while average messages of TGDH and CCEGK increase steadily. Average messages of CCEGK are smaller than those of TGDH because the rebalance scheme makes group members equally distributed in the binary tree and needs less messages when we do a mass leave or partition operation. In the long run, average messages of STR are the smallest, followed by CCEGK, TGDH, and EGK.

In Fig. 12, as the number of join operations decreases and all other operations increase, the average sequential exponentiations of EGK, TGDH, and CCEGK steadily increase during the first half of the scenarios, and then they gradually decrease. Average sequential exponentiations of CCEGK are smaller than those of TGDH because the binary tree of CCEGK becomes more balanced and reduces sequential exponentiations when we do a leave, mass leave, or partition operation. In the long run, average sequential exponentiations of EGK are the smallest, followed by CCEGK, TGDH, and STR.

9. Conclusion and further work

In this paper, we introduced a new communication-computation efficient group key algorithm (CCEGK) for dynamic and large groups, derived from EGK and TGDH. We describe the implementation of every operation, analyze and evaluate the complexity of every operation by theoretical methods. In the end, we compared the performance of every operation between CCEGK and the other three notable group key algorithms, EGK, TGDH,

and STR. Overall, CCEGK appears to have better performance characteristics across the board when considering communication and computation costs. In the future, we will consider different scenarios and combinations of operations to evaluate the relative costs of the algorithms. We will also look at the specific costs of phases and message lengths during these simulations to get a more precise idea of the impact of these features of the algorithms. We would like to consider further the expansion of CCEGK specifically for an ad-hoc wireless network, and to extend CCEGK to use Elliptic Curve encryption. Both areas have had less attention to date and warrant more exploration.

Acknowledgements

This material is based on research sponsored by AFRL and DARPA under agreement number F30602-02-1-0178. The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL and DARPA or the US Government.

Appendix A

We include this Appendix because TGDH and STR do not implement the initialization operation.

A.1. Initialization operation in TGDH

In every operation of TGDH (e.g., add, merge, leave, and mass leave), every message uses the broadcast scheme, so we implement the initialization operation in TGDH using a broadcast scheme. The initialization algorithm is as follows:

Suppose that there is a collection of entities: $\{M_1, \dots, M_n\}$

1. In Round 1, each node M_i generates a DH key pair consisting of a secret key K_i and a corresponding blinded key $bK_i = \alpha^{K_i} \bmod p$. Each node M_i broadcasts a message including its blinded key to its sibling. M_i performs a two-party Diffie Hellman (DH) with its sibling to calculate a new DH key pair with the private key and the blinded key. Each node has a single

- parent and single sibling. We choose the rightmost sponsor as the sponsor of the group. We treat every new group as a new node. So the new number of nodes is $\lceil \frac{n}{2} \rceil$.
- In Round 2, each sponsor of each group broadcasts a message including all the blinded keys to the members of the sibling group. Every member performs a two-party DH with the blinded key of its sibling group to generate a new group with a new DH key pair. We choose the rightmost sponsor as the sponsor of this new group.
 - We repeat Step 3 until round $h = \lceil \log_2 n \rceil$ (where h is the height of the key tree). We create a group G including all the members with the group secret key $K_{(0,0)}$.

The costs of the initialization operation in TGDH are summarized in Table A.1.

A.2. Initialization operation in STR

In every operation of STR (e.g., add, merge, leave, and mass leave), every message uses the broadcast scheme. Therefore we implement the initialization operation in STR using the broadcast scheme. The initialization algorithm is as follows:

Suppose that there is a collection of entities: $\{M_1, \dots, M_n\}$

- In Round 1, M_1 and M_2 generate a DH key pair consisting of a secret key and a corresponding blinded key. M_1 and M_2 perform the normal 2-party DH; their key is $K_{(1,2)} = \alpha^{K_1 * K_2} \text{ mod } p$.
- In Round 2, M_3 wants to join the existing group (which includes M_1 and M_2). M_3 starts to broadcast messages including its blinded key to the existing group. The group controller M_1 replies

by sending its blinded keys to M_3 . M_1 and M_2 can both compute the new group key. M_3 can also compute the same group key.

- Repeat Step 2 until M_n wants to join the existing group. We create a group G including all the members with the group secret key $K_{(0,0)}$.

The costs of the initialization operation in STR are summarized in Table A.2.

A.3. Alternative solution for TGDH mass leave

In the mass leave operation in TGDH, [3] gave the result of total messages as $\min(2N, \lceil \frac{n}{2} \rceil)$ and the result of total verifications as $\min(2N, \lceil \frac{n}{2} \rceil)$, however, we believe these results are incorrect (or we believe these results to be in conflict with their described algorithm). We will give a counterexample to verify our claim, and then we give our solution. Fig. A.1 shows an example of members $M_1, M_4, M_7, M_{10}, M_{13}, M_{16}, M_{19}$, and M_{22} leaving from a group using the mass leave operation algorithm in TGDH, where the sponsor is M_{23} .

- In the first step the sponsors are as follows: M_2 is the sponsor after deleting M_1 ; M_5 after deleting M_4 ; M_8 after deleting M_7 ; M_{11} after deleting M_{10} ; M_{14} after deleting M_{13} ; M_{17} after deleting M_{16} ; M_{20} after deleting M_{19} ; and M_{23} after deleting M_{22} . Node $\langle 4,0 \rangle$ can calculate the secret key $K_{(3,0)}$ and blinded key $bK_{(3,0)}$. Because it does not know the blinded key $bK_{(3,1)}$, it cannot calculate the secret key $K_{(2,0)}$. Node $\langle 4,2 \rangle$ can calculate the secret key $K_{(3,1)}$ and blinded key $bK_{(3,1)}$. Because it does not know the blinded key $bK_{(3,0)}$, it cannot calculate the secret key $K_{(2,0)}$. Node $\langle 4,4 \rangle$ can calculate the secret key

Table A.1
Costs of initialization operation

Communication				Computation		
Rounds	Messages	Unicast	Broadcast	Sequential exponentiations	Signatures	Verifications
h	$2n - 2$	0	$2n - 2$	$2h - 2$	h	$2n - 2$

Table A.2
Communication Cost of Initialization Operation

Communication				Computation		
Rounds	Messages	Unicast	Broadcast	Sequential exponentiations	Signatures	Verifications
$n - 1$	$2(n - 1)$	0	$2(n - 1)$	$2(n - 1)$	$n - 1$	$2(n - 1)$

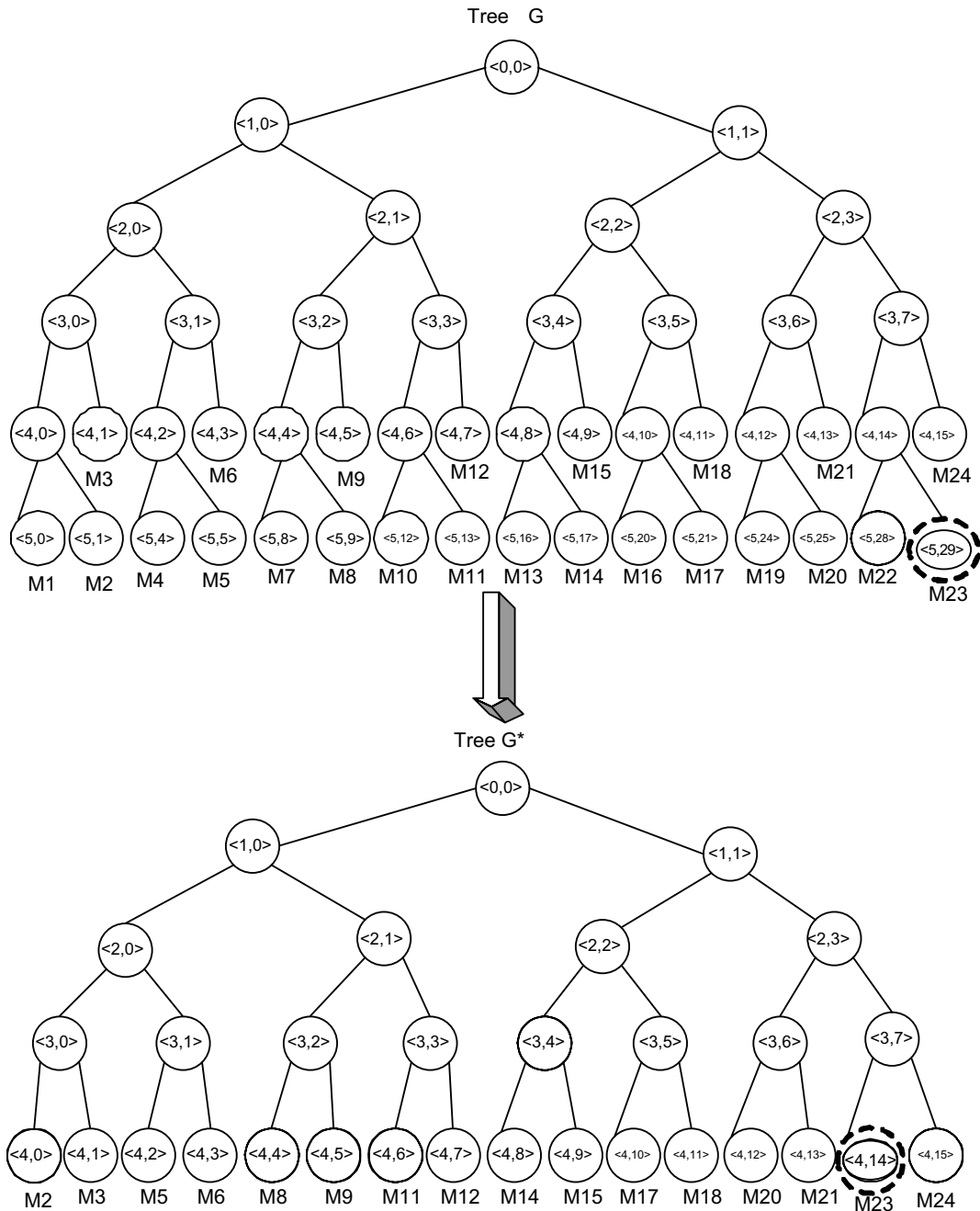


Fig. A.1. A Counterexample of mass Leave operation in TGDH.

$K_{(3,2)}$ and blinded key $bK_{(3,2)}$. Because it does not know the blinded key $bK_{(3,3)}$, it cannot calculate the secret key $K_{(2,1)}$. Node $\langle 4,6 \rangle$ can calculate the secret key $K_{(3,3)}$ and blinded key $bK_{(3,3)}$. Because it does not know the blinded key $bK_{(3,2)}$, it cannot calculate the secret key $K_{(2,1)}$. Node $\langle 4,8 \rangle$ can calculate the secret key

$K_{(3,4)}$ and blinded key $bK_{(3,4)}$. Because it does not know the blinded key $bK_{(3,5)}$, it cannot calculate the secret key $K_{(2,2)}$. Node $\langle 4,10 \rangle$ can calculate the secret key $K_{(3,5)}$ and blinded key $bK_{(3,5)}$. Because it does not know the blinded key $bK_{(3,4)}$, it cannot calculate the secret key $K_{(2,2)}$. Node $\langle 4,12 \rangle$ can calculate the secret key $K_{(3,6)}$ and

blinded key $bK_{(3,6)}$. Because it does not know the blinded key $bK_{(3,7)}$, it cannot calculate the secret key $K_{(2,3)}$. Node $\langle 4, 14 \rangle$ picks up a new secret key $K_{(4,14)}$ and calculates the secret key $K_{(3,7)}$ and blinded key $bK_{(3,7)}$. Because it does not know the blinded key $bK_{(3,6)}$, it cannot calculate the secret key $K_{(2,3)}$. After that, $\langle 4, 0 \rangle$, $\langle 4, 2 \rangle$, $\langle 4, 4 \rangle$, $\langle 4, 6 \rangle$, $\langle 4, 8 \rangle$, $\langle 4, 10 \rangle$, $\langle 4, 12 \rangle$, and $\langle 4, 14 \rangle$ broadcast their blinded keys to every other group member.

2. In Step 2, the sponsors are as follows: $\langle 4, 2 \rangle$ becomes the sponsor in the branch of $\langle 2, 0 \rangle$; $\langle 4, 6 \rangle$ in the branch of $\langle 2, 1 \rangle$; $\langle 4, 10 \rangle$ in the branch of $\langle 2, 2 \rangle$; $\langle 4, 14 \rangle$ in the branch of $\langle 2, 3 \rangle$. Node $\langle 4, 2 \rangle$ can calculate the secret key $K_{(2,0)}$ and blinded key $bK_{(2,0)}$, but cannot calculate the secret key $K_{(1,0)}$ because it does not know the blinded key $bK_{(2,1)}$. Node $\langle 4, 6 \rangle$ can calculate the secret key $K_{(2,1)}$ and blinded key $bK_{(2,1)}$, but cannot calculate the secret key $K_{(1,0)}$ because it does not know the blinded key $bK_{(2,0)}$. Node $\langle 4, 10 \rangle$ can calculate the secret key $K_{(2,2)}$ and blinded key $bK_{(2,2)}$, but cannot calculate the secret key $K_{(1,1)}$ because it does not know the blinded key $bK_{(2,3)}$. Node $\langle 4, 14 \rangle$ can calculate the secret key $K_{(2,3)}$ and blinded key $bK_{(2,3)}$, but cannot calculate the secret key $K_{(1,1)}$ because it does not know the blinded key $bK_{(2,2)}$. After that, $\langle 4, 2 \rangle$, $\langle 4, 6 \rangle$, $\langle 4, 10 \rangle$, and $\langle 4, 14 \rangle$ broadcast their blinded keys to every other group member.
3. In Step 3 the sponsors are as follows: $\langle 4, 6 \rangle$ becomes the sponsor in the branch of $\langle 1, 0 \rangle$ and $\langle 4, 14 \rangle$ in the branch of $\langle 1, 1 \rangle$. Node $\langle 4, 6 \rangle$ can calculate the secret key $K_{(1,0)}$ and blinded key $bK_{(1,0)}$, but cannot calculate the secret key $K_{(0,0)}$ because it does not know the blinded key $bK_{(1,1)}$. Node $\langle 4, 14 \rangle$ can calculate the secret key $K_{(1,1)}$ and blinded key $bK_{(1,1)}$, but cannot calculate the secret key $K_{(0,0)}$ because it does not know the blinded key $bK_{(1,0)}$. After that, $\langle 4, 6 \rangle$ and $\langle 4, 14 \rangle$ broadcast their blinded keys to every group member. Now every group member can calculate the secret key $K_{(0,0)}$.

From the above actions, we know we need $8 + 4 + 2 = 14$ messages. But in [3], they state that TGDH needs $\min(2 * 8, \lceil \frac{2d}{2} \rceil) = 12$, so their conclusion does not agree with their supposition. We put forward a solution that reflects their algorithm. Suppose the number of leaving members and the number of original group members are N and n respectively, so the number of the remaining group members is $n - N$. Our solution is as following:

Case 1: $N < \lceil \frac{n-N}{2} \rceil$. Because in the worst case in the key tree there are at most $N < \lceil \frac{n-N}{2} \rceil$ paths with empty blinded keys after N members leave, in this situation in the first round the total number of broadcasted messages is N . Because in every round the total number of messages is reduced by half, the total number is

$$\lim_{n \rightarrow \infty} \sum_{m=0}^n \left(\frac{N}{2^m} \right) = 2N.$$

Case 2: $N = \lceil \frac{n-N}{2} \rceil$. Because in the worst case in the key tree there are at most $\lceil \frac{n-N}{2} \rceil = K$ paths with empty blinded keys after N members leave, in this situation in the first round the total number of broadcasted messages is N or $\lceil \frac{n-N}{2} \rceil$. Because in every round the total number of messages is reduced by half, the total number is

$$\begin{aligned} \lim_{n \rightarrow \infty} \sum_{m=0}^n \left(\frac{N}{2^m} \right) &= \lim_{n \rightarrow \infty} \sum_{m=0}^n \frac{\lceil \frac{n-N}{2} \rceil}{2^m} = 2N \\ &= n - N. \end{aligned}$$

Case 3: $N > \lceil \frac{n-N}{2} \rceil$. Because in the worst case in the key tree there are at most $\lceil \frac{n-N}{2} \rceil < N$ paths with empty blinded keys after N members leave, in this situation in the first round the total number of broadcasted messages is $\lceil \frac{n-N}{2} \rceil$. Because in every round the total number of messages is reduced by half, the total number is

$$\lim_{n \rightarrow \infty} \sum_{m=0}^n \frac{\lceil \frac{n-N}{2} \rceil}{2^m} = n - N.$$

So, when N group members leave from the group, the total number of messages is $\min(2N, \lceil n - N \rceil)$ and the total number of verifications is $\min(2N, n - N)$.

References

- [1] J. Alves-Foss, An efficient secure authenticated group key exchange algorithm for large and dynamic groups, in: Proc. 23rd National Information Systems Security Conference, 2000, pp. 254–266.
- [2] Y. Kim, A. Perrig, G. Tsudik, Simple and fault-tolerance key agreement for dynamic collaborative groups, in: Proc. of 7th ACM Conference on Computer and Communications Security, 2000, pp. 235–244.
- [3] Y. Kim, A. Perrig, G. Tsudik, Tree-based group key agreement, ACM Transactions on Information and System Security 7 (1) (2004) 60–96.

- [4] Y. Kim, A. Perrig, G. Tsudik, Communication-efficient group key agreement, in: Proc. of IFIP SEC 2001, 2001.
- [5] Y. Kim, A. Perrig, G. Tsudik, Group key agreement efficient in communication, IEEE Transactions on Computers 53 (7) (2004) 905–921.
- [6] Y. Challal, H. Seba, Group key management protocols: A novel taxonomy, International Journal of Information Technology 2 (1) (2005) 105–118.
- [7] Y. Amir, Y. Kim, C. Nita-Rotaru, G. Tsudik, On the performance of group key agreement protocols, ACM Transactions on Information and System Security 7 (3) (2004) 457–488.
- [8] W. Diffie, M. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (6) (1976) 644–652.
- [9] A. Freier, P. Kartion, P. Kocher, The SSL Protocol: Version 3.0, Netscape Communications, Inc, 1996.
- [10] T. Ylonen, SSH – secure login connections over the Internet, in: the Sixth USENIX Unix Security Symposium, 1996, pp. 37–42.
- [11] M. Burmester, Y. Desmedt, A secure and efficient conference key distribution system, in: Advances in Cryptology – EUROCRYPT’94, 1994, pp. 275–286.
- [12] M. Burmester, Y. Desmedt, Efficient and secure conference key distribution, in: Security Protocols: International Workshop, 1996, pp. 119–129.
- [13] M. Just, S. Vaudenay, Authenticated multi-party key agreement, Tech. Rep. SCS-TR-96-04, Carleton University, Computer Science Department, Ottawa, Canada, 1996.
- [14] M. Steiner, G. Tsudik, M. Waidner, Diffie-Hellman key distribution extended to groups, in: Third ACM Conference on Computer and Communications Security, ACM Press, 1996, pp. 31–37.
- [15] K. Becker, U. Willie, Communication complexity of group key distribution, in: Proc. 5th Conference on Computers and Communication Security, 1998, pp. 1–6.
- [16] M. Steiner, G. Tsudik, M. Waidner, Key agreement in dynamic peer groups, Tech. rep., Information Sciences Institute, January 1999.
- [17] Y. Amir, Y. Kim, C. Nita-Rotaru, G. Tsudik, On the performance of group key agreement protocols, in: Proc. 22nd IEEE Conference on Distributed Computing Systems, 2002, pp. 263–264.
- [18] D. Boneh, The Decision Diffie-Hellman problem, in: Proceedings of the Third Algorithmic Number Theory Symposium, LNCS, vol. 1423, Springer-Verlag, 1998, pp. 48–63.
- [19] A. Fekete, N. Lynch, A. Shvartsman, Specifying and using a partitionable group communication service, in: Proc. ACM PODC ’97, 1997, pp. 53–62.
- [20] G. Ateniese, M. Steiner, G. Tsudik, Authenticated group key agreement and friends, in: Proc. 5th ACM Conference on Computer and Communications Security, 1998, pp. 17–26.
- [21] M. Just, S. Vaudenay, Authenticated multi-party key agreement, in: Advances in Cryptology – ASIACRYPT ’96, 1996, pp. 36–49.
- [22] A. Perrig, Efficient collaborative key management protocols for secure autonomous group communication, in: Proc. CryptTEC ’99, 1999, pp. 192–202.



Shanyu Zheng is a PhD student in the Department of Computer Science at the University of Idaho, USA. He is a research associate with the Center for Secure and Dependable Systems. His research interests include cryptography, network security, and wireless systems, with an emphasis on group key management in wired and wireless networks. Before that, he received his BS, MS in computer science from Sichuan University, China, in 1999 and 2002, respectively.



David Manz is an MS student in the Department of Computer Science at the University of Idaho. During the summer of 2002 he participated in a Research Experience for Undergraduates at the University of Idaho. There he was exposed to security research, and decided that graduate school was for him. He finished up his B.S. degree from the Robert D. Clark Honors College at the University of Oregon in 2003. He is

currently researching group key encryption at the University of Idaho; where he received his MS degree in 2005 and is continuing his Ph.D studies.



Jim Alves-Foss is the Director of the University of Idaho’s Center for Secure and Dependable Systems and is an associate professor of Computer Science. He received his BS degree in Mathematics and Computer Science and Physics from the University of California at Davis in 1987, and the MS and PhD degrees in Computer Science from the University of California at Davis in 1989 and 1991 respectively. His main research

interests are in the design and analysis of secure distributed systems, with a focus on formal methods and software engineering.