Group Key Agreement Efficient in Communication

Yongdae Kim, *Member*, *IEEE*, Adrian Perrig, *Member*, *IEEE*, and Gene Tsudik, *Member*, *IEEE*

Abstract—In recent years, collaborative and group-oriented applications and protocols have gained popularity. These applications typically involve communication over open networks; security thus is naturally an important requirement. Group key management is one of the basic building blocks in securing group communication. Most prior research in group key management focused on minimizing computation overhead, in particular minimizing expensive cryptographic operations. However, continued advances in computing power have not been matched by a decrease in network communication delay. Thus, communication latency, especially in high-delay long-haul networks, increasingly dominates the key setup latency, replacing computation delay as the main latency contributor. Hence, there is a need to minimize the size of messages and, especially, the number of rounds in cryptographic protocols. Since most previously proposed group key management techniques optimize computational (cryptographic) overhead, they are particularly impacted by high communication delay. In this work, we discuss and analyze a specific group key agreement technique which supports dynamic group membership and handles network failures, such as group partitions and merges. This technique is very communication-efficient and provably secure against hostile eavesdroppers as well as various other attacks specific to group settings. Furthermore, it is simple, fault-tolerant, and well-suited for high-delay networks.

Index Terms—Security, group key agreement, group communication, communication complexity, cryptographic protocols.

1 Introduction

Secure group communication is an increasingly popular research area having received much attention in recent years. Since most group communication takes place over the wide-open expanse of the Internet, security is a major concern. The fundamental security challenge revolves around secure and efficient group key management. Centralized key management methods (key distribution) are appropriate for 2-party (e.g., client-server or peer-to-peer) communication as well as for large multicast groups. However, many collaborative group settings (e.g., conferencing, white-boards, shared instruments, and command-and-control systems) require distributed key management techniques.

The majority of research in group key agreement (one way of implementing distributed group key management) was mainly concerned with increasing the security while minimizing cryptographic computation cost. It has been long held as an incontrovertible fact that heavy-weight computation—such as large number arithmetic that forms the basis of many modern cryptographic algorithms—is the greatest burden imposed by security protocols. However, the continuing increase in computation power of modern workstations speed up the heavy-weight cryptographic operations. For example, four years ago, a top-of-the-line RISC workstation performed a 512-bit modular exponentiation in around 24 ms. Four years later, an 850 MHz Pentium III PC (priced at one-fifth of the old RISC workstation) performs the same operation in under 1 ms.

In contrast, communication latency has not improved appreciably. Network devices and communication lines have become significantly faster and cheaper. The communication (especially via the Internet) has become both accessible and affordable, which resulted in drastic increase in the demand for network bandwidth. While computation power and bandwidth are increasing, network delay has the lower bound dictated by the speed of light.

Consequently, the half-around-the-world packet round trip delay is likely to remain constant (at least for terrestrial communication). In addition, interplanetary networking is not too far off in the future. Consider, for instance, the communication delay with a Mars Rover or other space exploration device. More concretely, collaborative work groups where the members are dispersed across continents will expect considerable communication latency and would thus benefit from protocols that minimize communication rounds. Similarly, group teleconferences are becoming increasingly popular.

The bottleneck shift from computation to communication latency prompts us to look at cryptographic protocols in a different light: allowing more liberal use of cryptographic operations while attempting to reduce the communication overhead. The latter includes both round and message complexity. Communication overhead is especially relevant in a peer group setting since group members can be spread throughout a large network, e.g., the global Internet.

We consider a protocol first proposed by Steer et al. in 1988 [27]. It is one of the first group key agreement

- *Y. Kim is with the Computer Science and Engineering Department, University of Minnesota-Twin Cities, 200 Union St. SE, Minneapolis, MN 55455. E-mail: kyd@cs.umn.edu.*
- *A. Perrig is with the Electrical and Computer Engineering Department, Carnegie Mellon University, B204 Hamerschlag Hall, 5000 Forbes Ave., Pittsburgh, PA 15213. E-mail: perrig@cmu.edu.*
- *G. Tsudik is with the Computer Science Department, University of California at Irvine, 458 CS Bldg., Irvine, CA 92697-3425. E-mail: gts@ics.uci.edu.*

Manuscript received 7 Jan. 2003; revised 20 Aug. 2003; accepted 3 Sept. 2003. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 118097.

protocols. This protocol extends the 2-party Diffie-Hellman key exchange and supposes the formation of a secure **static group**. This protocol—referred to as STR (short for **S**kinny **TR**ee) hereafter—involves heavy computation and communication requirements: $O(n)$ communication rounds and $O(n)$ cryptographic operations are necessary to establish a shared key in a group of n members. We extend it to deal with dynamic groups and network failures in a communication-efficient manner. Concretely, we construct an entire group key management protocol suite that is particularly efficient in a WAN environment where network delay is high.

The remainder of this paper is organized as follows: Section 2 explains our assumptions and requirements for the reliable group communication system over wide area network and cryptographic requirements of group key agreement schemes. Notations used in the rest of this paper are introduced in Section 3 and the actual protocol suite is described in Section 4. Section 5 considers the security, complexity, and implementation issues and performance of STR is discussed in Section 6. The summary of related work appears in Section 7 and conclusions appear in Section 8. Security argument of the proposed protocols is provided in the Appendix.

2 Reliable Group Communication and Group Key Agreement

In this section, we set the stage for the rest of the paper with a brief overview of the notable features of reliable group communication and group key agreement.

As noted earlier, many current collaborative and distributed applications require a reliable group communication platform. In addition, many group communication applications require security services which are built atop secure group key management. This dependency is mutual since practical group key agreement protocols themselves rely on the underlying group communication semantics for protocol message transport and strong membership semantics. Implementing multiparty and multiround cryptographic protocols without such support is foolhardy as, in the end, one winds up reinventing reliable group communication tools.

2.1 Reliable Group Communication Semantics

Many modern collaborative and distributed applications require a reliable group communication platform. Current reliable group communication toolkits generally provide one (or both) of two strong group communication semantics: Extended Virtual Synchrony (EVS) [22] and View Synchrony (VS) [15]. Both semantics guarantee that: 1) group members see the same set of messages between two sequential group membership events and 2) the sender's requested message order (e.g., FIFO, Causal, or Total) is preserved. VS offers a stricter guarantee than EVS: Messages are delivered to all recipients in the same membership as viewed by the sender application when it originally sent the message. In the context of this paper, we require the underlying group communication to provide VS. However, we stress that VS is needed for the sake of fault tolerance and robustness; the security of our protocols is in no way affected by the lack of VS. More details on the interaction of key agreement protocols and reliable group communication are addressed in [1].

2.2 Communication Delay

Due to the reliable group communication platform, network delay is amplified by the necessary acknowledgments between the group members. The speed of light puts a lower bound on the minimum network delay. For example, a laser pulse that travels through a fiber optic cable takes ≈ 10 ms to travel from New York to San Francisco, ≈ 21 ms from Paris to San Francisco, and ≈ 40 ms from London to Sydney. In practice, networks today are about 3 to 4 times slower than the lower bound.

To put this into perspective, an 850MHz Pentium III PC performs a single 512-bit modular exponentiation (one of the most expensive, but most basic public key primitives) in under 1 ms. Moreover, the speed of computers continues to increase. Comparing this with the WAN network delay, it is clear that reducing the number of communication rounds is much more important in the long run (for an efficient group key agreement scheme) than reducing the computation overhead.

2.3 Group Key Agreement

A comprehensive group key agreement solution must handle adjustments to group secrets subsequent to all membership change operations in the underlying group communication system. The following membership changes are considered: We distinguish among single and multiple member operations. We also distinguish between additive and subtractive member operations. Single member changes include member join or leave and multiple member changes include group merge and group partition.

- **Join** occurs when a prospective member wants to join a group.
- **Leave** occurs when a member wants to leave (or is forced to leave) a group. There might be different reasons for member deletion such as voluntary leave, involuntary disconnect, or forced expulsion. We believe that group key agreement must only provide the tools to adjust the group secrets and leave the rest up to the local security policy.
- **Partition** occurs when a group is split into smaller groups. A group partition can take place for several reasons, two of which are fairly common:
  1. Network failure—this occurs when a network event causes disconnectivity within the group. Consequently, a group is split into fragments, some of which are singletons while others (those that maintain mutual connectivity) are subgroups.
  2. Explicit (application-driven) partition—this occurs when the application decides to split the group into multiple components or simply exclude multiple members at once.
- **Merge** occurs when two or more groups merge to form a single group (a group merge may be voluntary or involuntary):

1. Network fault heal—this occurs when a network event causes previously disconnected network partitions to reconnect. Consequently, groups on all sides (and there might be more than two sides) of an erstwhile partition are merged into a single group.
2. Explicit (application-driven) merge—this occurs when the application decides to merge multiple preexisting groups into a single group. (The case of simultaneous multiple-member addition is not covered.)

At first glance, events such as network partitions and fault heals might appear infrequent and dealing with them might seem to be a purely academic exercise. In practice, however, such events are common owing to network misconfigurations and router failures. Moser et al. present compelling arguments in support of these claims [22]. Hence, dealing with group partitions and merges is a crucial component of group key agreement.

In addition to the aforementioned membership operations, periodic refreshes of group secrets are advisable so as to limit the amount of ciphertext generated with the same key and to recover from potential compromises of members' contributions or prior session keys.

2.4 Cryptographic Properties

In this section, we summarize the desired properties for a secure group key agreement protocol. Following the model of [18], we define four such properties:

Definition 1.

- Group Key Secrecy *guarantees that it is computationally infeasible for a passive adversary to discover any group key.*
- Forward Secrecy (not to be confused with Perfect Forward Secrecy or PFS) *guarantees that a passive adversary who knows a contiguous subset of old group keys cannot discover subsequent group keys.*
- Backward Secrecy *guarantees that a passive adversary who knows a contiguous subset of group keys cannot discover preceding group keys.*
- Key Independence *guarantees that a passive adversary who knows any proper subset of group keys cannot discover any other group key not included in the subset.*

The relationship among the properties is intuitive. Backward and Forward Secrecy properties (often called Forward and Backward Secrecy in the literature) assume that the adversary is a current or a former group member. The other properties additionally include the cases of inadvertently leaked or otherwise compromised group keys.

Our definition of group key secrecy allows partial leakage of information. Therefore, it would be more desirable to guarantee that any bit of the group key is unpredictable. For this reason, we prove a decisional version of group key secrecy in Appendix A. In other words, the decisional version of group key secrecy guarantees that it is computationally infeasible for a passive adversary to **distinguish** any group key from random number.

Other, more subtle, active attacks aim to introduce a known (to the attacker) or old key. These are prevented by the combined use of: sender information, timestamps, unique protocol message identifiers, and sequence numbers which identify the particular protocol run.

All protocol messages include the following attributes:

- sender information: name of the sender, or, equivalently, signer.
- group information: unique name of the group.
- membership information: names (and other information) of current group members.
- protocol identifier: protocol being used (fixed as "STR").
- message type: unique message identifier for each protocol message.
- key epoch: strictly increasing counter. Whenever a new membership event occurs, each member increments key epoch. If two groups G_1 and G_2 merge, the resulting epoch is:

$$epoch_{new} = max(epoch_{G_1}, epoch_{G_2}) + 1.$$

  Key epoch is the same across all current group members. If a group member receives a protocol message with a smaller than current epoch, it terminates the protocol (suspected replay).
- time stamp: current time. Loose time synchronization among group members is assumed.

We assume that a group member rejects any message which does not match its expectations. Since all messages are signed, we also assume PKI for all protocol parties. Since no other long-term secrets or keys are used, we are not concerned with Perfect Forward Secrecy (PFS) as it is achieved trivially.

In this paper, we do not assume key authentication to be part of group key management. All communication channels are thus considered public but authentic. The latter means that all messages are digitally signed by the sender with some sufficiently strong public key signature method such as DSA or RSA (and using a long-term private key).[1] All receivers are required to verify signatures on all received messages and check the aforementioned fields. Consequently, our security model is different from some recent related work [9], [10] that does not assume authentic channels.

3 Notation

We use the following notation throughout the rest of this paper:

1. Furthermore, as discussed above, all protocol messages are assumed to contain: 1) sender/group information, 2) a prototol identifier (i.e., STR here) to distinguish among multiple protocols, 3) a unique message identifier to distinguish among messages within a protocol, and 4) a key epoch identifier to capture the instance of the protocol.

n, N	number of protocol parties (group members)
i, j	group member indices: $i, j \in \{1, \ldots, N\}$
M_i	i-th group member; $i \in \{1, \ldots, N\}$
r_i	M_i's session random (secret key of leaf node M_i)
br_i	M_i's blinded session random, i.e. $\alpha^{r_i} \bmod p$
k_j	secret key shared among $M_1 ... M_j$
bk_j	blinded k_j, i.e. $\alpha^{k_j} \bmod p$
p	large prime number
α	exponentiation base
$\mathsf{N}_{\langle j \rangle}$	key-tree node j
$\mathsf{IN}_{\langle l \rangle}$	Internal key-tree node at level l
$\mathsf{LN}_{\langle i \rangle}$	Leaf node associated with member M_i
$T_{\langle i \rangle}$	key-tree of member M_i
$BT_{\langle i \rangle}$	key-tree of member M_i including all of its blinded keys

Fig. 1 shows an example of an STR key tree. The tree has two types of nodes: leaf and internal. Each leaf node is associated with a specific group member. An internal node $\mathrm{IN}_{(i)}$ always has two children: another (lower) internal node $\mathrm{IN}_{(i-1)}$ and a leaf node $\mathrm{LN}_{(i)}$. The exception is $\mathrm{IN}_{(1)}$ which is also a leaf node corresponding to M_1. (Note that, consequently, $r_1 = k_1$.)

Each leaf node $\mathrm{LN}_{(i)}$ has a *session random* r_i chosen and kept secret by M_i. The blinded version thereof is $br_i = \alpha^{r_i} \bmod p$. Every internal node $\mathrm{IN}_{(j)}$ has an associated secret key k_j and a public blinded key (bkey) $bk_j = \alpha^{k_j} \bmod p$. The secret key k_i $(i > 1)$ is the result of a Diffie-Hellman key agreement between the node's two children (k_1 is an exception and is equal to r_i.), which is computed recursively as follows:

$$k_i = (bk_{i-1})^{r_i} \bmod p = (br_i)^{k_{i-1}} \bmod p = \alpha^{r_i k_{i-1}} \bmod p \text{ if } i > 1.$$

The group key in Fig. 1 is the key associated with the root node: $k_4 = \alpha^{r_4 \alpha^{r_3 \alpha^{r_2 r_1}}}$.

We note that the root (group) key is never used directly for the purposes of encryption, authentication, or integrity. Instead, such special-purpose subkeys are derived from the root key, e.g., by applying a cryptographically secure hash function to the root key. All bkeys bk_i are assumed to be public.

The basic key agreement protocol is as follows: We assume that all members know the structure of the key tree and their initial position within the tree. (There are many ways to order members unambiguously.) Furthermore, each member knows its session random and the blinded session randoms of all other members. The two members M_1 and M_2 can first compute the group key corresponding to $\mathrm{IN}_{(2)}$. M_1 computes:

$$\begin{aligned} k_2 &= (br_2)^{r_1} \bmod p = \alpha^{r_1 r_2} \bmod p, & bk_2 &= \alpha^{k_2} \bmod p \\ k_3 &= (br_3)^{k_2} \bmod p, & bk_3 &= \alpha^{k_3} \bmod p \\ &\ldots \\ k_N &= (br_N)^{k_{N-1}} \bmod p. \end{aligned}$$

Next, M_1 broadcasts all bkeys bk_i with $1 \le i \le N-1$. Armed with this message, every member then computes k_N as follows. (As mentioned above, members M_1 and M_2 derive the group key without additional broadcasts.) Any M_i (with $i > 2$) knows its session random r_i and bk_{i-1} from the broadcast message. Hence, it can derive $k_i = bk_{i-1}^{r_i} \bmod p$. It can then compute all remaining keys recursively up to the group key from the public blinded session randoms: $k_i = br_i^{k_{i-1}} \bmod p$ $(i \le N)$.

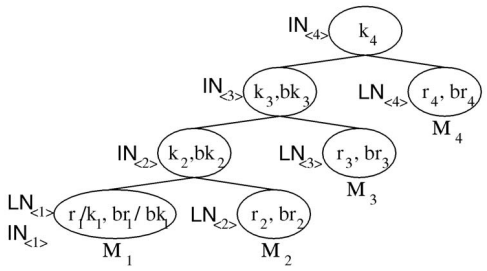


Fig. 1. Example STR key tree.

Following every membership change, all members independently update the key tree. Since we assume that the underlying group communication system provides *view synchrony* (see Section 2.1), all members who correctly execute the protocol recompute an identical key tree after any membership event. The following proposition describes the minimal requirement for a group member to compute the group key:

Proposition 1. *If all members know the blinded session randoms of all other members, at least two members can compute the group key.*

This follows directly from the recursive definition of the group key. In other words, both M_1 and M_2 (the members at the lowest leaf nodes) can obtain the group key by computing pairwise keys recursively and using blinded session randoms of other members.

Proposition 2. *Any member can compute the group key, if it knows: 1) its own secret share, 2) the bkey of its sibling subtree, and 3) blinded session randoms of members higher in the tree.*

Proof. This also follows from the definition of the group key. To compute the group key, member M_i needs 1) r_i, 2) bk_{i-1}, and 3) $br_{i+1}, br_{i+2}, \ldots, br_N$. □

The protocols described below benefit from a special role (called sponsor) assigned to a certain group member following each membership change. A sponsor reduces communication overhead by performing "housekeeping" tasks that vary depending on the type of membership change. The criteria for selecting a sponsor are described below.

4 STR Protocols

We now describe the protocols that make up the STR key management suite: join, leave, merge, and partition. All protocols share a common framework with the following features:

- Each group member contributes an equal share to the group key; this share is kept secret by each group member.

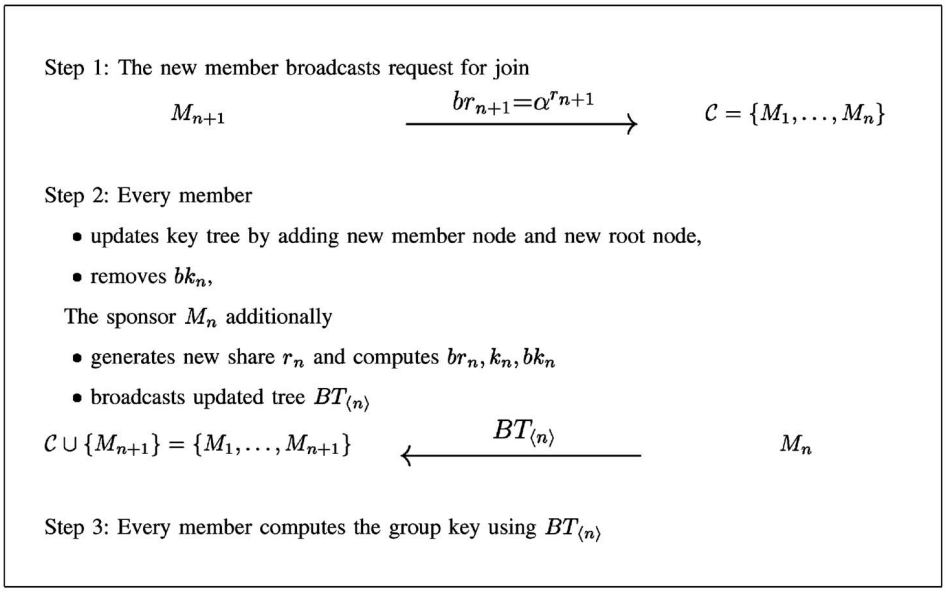


Fig. 2. JOIN protocol.

- The group key is computed as a function of all current group members' shares.
- As the group grows, new members' shares are factored into the group key while the remaining members' shares (except for sponsor which changes its session randomly to provide key independence) stay unchanged.
- As the group shrinks, departing members' shares are removed from the new group key and at least one remaining member changes its share.
- All protocol messages are signed by the sender, i.e., we assume an authenticated broadcast channel.
- In a join or a merge, sponsor is associated with the topmost leaf node of each key tree.
- In a leave or a partition, sponsor is located immediately below the deepest leaving node.

4.1 Join

We assume the group has n users, $\{M_1, \ldots, M_n\}$, when the group communication system announces the arrival of a new member. Both the new member and the prior group members receive this notification simultaneously. As shown in Fig. 2, the new member M_{n+1} broadcasts a join request message that contains its own bkey bk_{n+1} (which is the same as its blinded session random br_{n+1}). Upon receiving this message, the current group's sponsor M_n refreshes its session random, computes br_n, k_n, bk_n, and sends the current tree $BT_{(n)}$ to M_{n+1} with all bkeys.

Next, each member M_i increments $n = n + 1$ and creates a new root key node $\mathrm{IN}_{(n)}$ with two children: the root node $\mathrm{IN}_{(n-1)}$ of the prior tree T_i on the left and the new leaf node $\mathrm{LN}_{(n)}$ corresponding to the new member on the right. Note that every member can compute the group key (see Proposition 2) since:

- All existing members only need the new member's blinded session random.
- The new member needs the blinded group key of the prior group.

In a join operation, the sponsor is always the topmost leaf node, i.e., the most recent member in the current group. Fig. 3 shows an example of a new member M_5 joining a group. To provide forward secrecy, the sponsor M_4 updates its session random r_4.

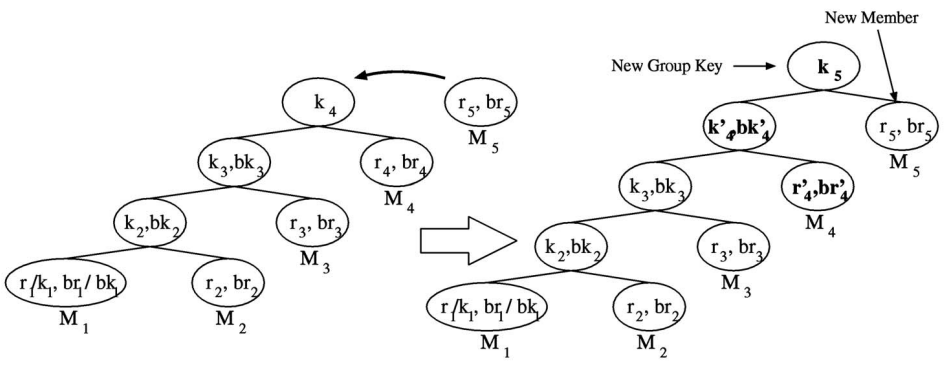


Fig. 3. Tree update in JOIN.

Step 1: Every member

- updates key tree as described above,
- removes all keys and bkeys from the sponsor node to the root node

The sponsor M_s additionally

- generates new share and computes all (keys, bkeys)
- and broadcasts updated tree $BT_{\langle s \rangle}$

$\mathcal{C} - \{M_d\}$ ← $BT_{\langle s \rangle}$ — M_s

Step 2: Every member computes the group key using $BT_{\langle s \rangle}$

Fig. 4. LEAVE Protocol.

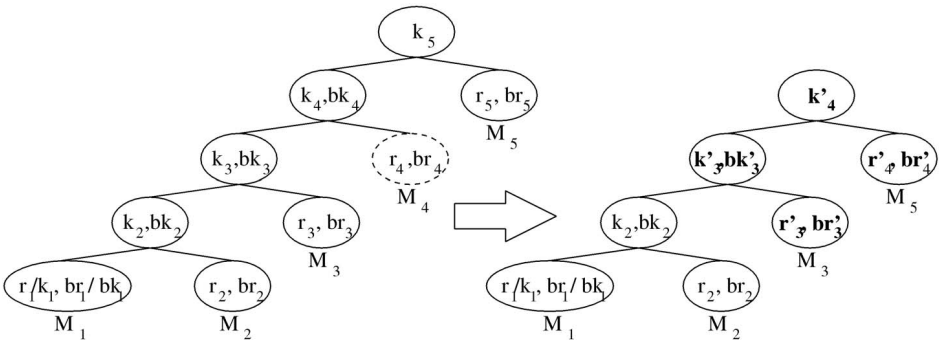


Fig. 5. Tree update in LEAVE.

As described, JOIN takes two communication rounds and five cryptographic operations to compute the new group key (four by the sponsor and two by everyone else.) As will be discussed in Section 5.1.2, the JOIN protocol provides backward secrecy.

4.2 Leave

We again have a group of n members when a member M_d $(d \leq n)$ leaves the group. If $d > 1$, the sponsor M_s is the leaf node directly below the leaving member, i.e., M_{d-1}. Otherwise, the sponsor is M_2. Upon hearing about the leave event from the group communication system, each remaining member updates its key tree by deleting the nodes $\mathrm{LN}_{(d)}$ corresponding to M_d and its parent node $\mathrm{IN}_{(d)}$. The nodes above the leaving node are also renumbered. The former sibling $\mathrm{IN}_{(d-1)}$ of M_d is promoted to replace (former) M_d's parent. The sponsor M_s selects a new secret session random, computes all keys (and bkeys) just below the root node, and broadcasts $BT_{(s)}$ to the group. This information allows all members (including the sponsor) to recompute the new group key. Fig. 4 describes the leave protocol in detail.

Fig. 5 shows that if member M_4 leaves the group, other members delete the leaving node along with its parent. Then, the sponsor M_3 picks its new session random r_3, computes br'_3, k'_3, bk'_3, and broadcasts the updated tree BT_4. Upon receiving the broadcast, all members (including M_3) compute the group key k_4. Note that M_4 cannot compute the group key (even though it knows all bkeys) since its session random is no longer a part thereof.[2]

The LEAVE protocol takes one communication round and involves a single broadcast. The cryptographic cost varies depending upon two factors: 1) the position of the departed member and 2) the position of the remaining member needing to compute the new key.

The total number of serial cryptographic operations in the leave protocol can be expressed as (assuming n is the original group size):

- $2(n-d)+1+(n-d)+1 = 3n-3d+2$ when $d > 2$.
- $3n-7$ when $d = 1, 2$.

In the worst case, M_1, M_2, or M_3 leaves the group. The cost for this leave operation is equal to $3n-7$. The expected leave cost is $3(n/2)+2$.

The LEAVE protocol provides forward secrecy since a former member cannot compute the new key owing to the sponsor's changing the session random. The protocol also provides key independence since knowledge of the new key cannot be used to derive the previous keys; this is, again, due to the sponsor refreshing its session random. For details of key independence, see Section 5.1.2.

4.3 Partition

A network fault (or severe congestion) can cause a partition of the group. To the remaining members, this actually

2. r_5 and br_5 are renumbered and are denoted as r'_4 and br'_4, respectively.

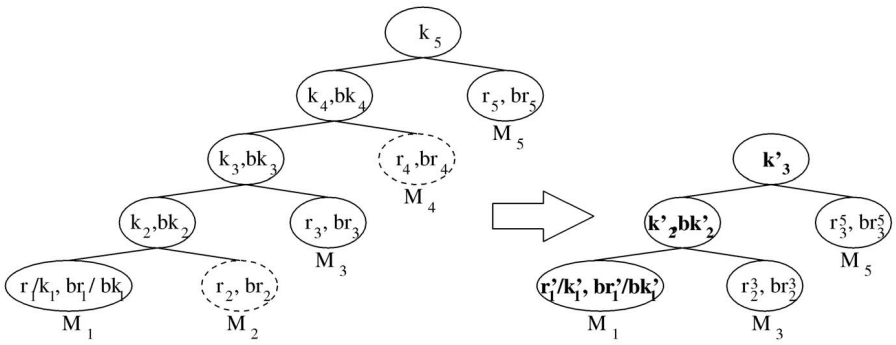


Fig. 6. Tree update in PARTITION.

appears as a concurrent leave of multiple members. With a minor modification, the LEAVE protocol can handle multiple leaving members in a single round. The only difference is in sponsor selection. In case of a partition, the sponsor is the leaf node directly below the lowest-numbered leaving member. (If M_1 is the lowest-numbered leaving member, the sponsor is the lowest-numbered surviving member.)

After deleting all leaving nodes, the sponsor M_s refreshes its session random (key share), computes keys and bkeys going up the tree—as in the plain leave protocol—terminating with the computation of $\alpha^{k_{n-1}} \bmod p$. It then broadcasts the updated key tree $BT_{(s)}$ containing only blinded values. Each member (including M_s) can now compute the group key.

Fig. 6 shows an example where the sponsor deletes all nodes of leaving members and computes all necessary keys and bkeys in the first round. In this example, M_1 is the sponsor since M_2 left the group. After picking a new session random r_1, the sponsor computes k_2 and $\alpha^{k_2} \bmod p$, and broadcasts the whole tree. Upon receiving this message, every member can compute the new group key k_3. Note that session randoms and blinded session randoms are renumbered as in the leave protocol.

The computation and communication complexity of this protocol is identical to that of the leave protocol. The same holds for its security properties.

4.4 Merge

We now describe the merge protocol. We assume that, as in the join case, the communication system simultaneously notifies all group members (in all groups) about the merge event. Moreover, reliable group communication toolkits typically include a list of all members that are about to merge in the merge notification. More specifically, we require that each member be able to distinguish the group it was in from the group that it is merging with. This assumption is not unreasonable, e.g., it is satisfied in SPREAD [1].

It is natural to graft the smaller tree atop the larger tree. If any two trees are of the same height, we can use any unambiguous ordering to decide which group joins which. (For example, lexicographical order of the identifiers of the respective sponsors.) When merging two trees, the lowest-numbered leaf of the smaller tree becomes the right child of a new intermediate node. The left child of the new intermediate node becomes the root of the larger tree.

Using this technique recursively, we can merge multiple trees. *k*-ary merge protocol is shown in Fig. 7.

In the first round of the merge protocol, all sponsors (members associated with the topmost leaf node in each tree) exchange their respective key trees containing all **blinded session randoms**.[3] The highest-numbered member of the largest tree becomes the sponsor of the second round in the merge protocol. After refreshing its session random, this sponsor computes every (key, bkey) pair up to the intermediate node just below the root node using the blinded session randoms of the other group members. It then broadcasts the key tree with **the bkeys and blinded session randoms** to the other members. All members now have the complete set of bkeys, which allows them to compute the new group key.

Fig. 8 shows an example of merging two trees. After the merge notification, the sponsors M_4 and M_7 broadcast their key trees containing all blinded session randoms. Upon receiving these broadcast messages, every member in both groups reconstructs the key tree. The smaller tree with three members is placed on top of the large tree with four members. Every member generates a new intermediate node $\mathrm{IN}_{(5)}$ and makes it the parent of the old root node $\mathrm{IN}_{(4)}$ of the larger tree and the previous leftmost leaf node $\mathrm{LN}_{(5)}$. Both intermediate nodes $\mathrm{IN}_{(1)}$ and $\mathrm{IN}_{(2)}$ of the previous smaller tree then need to be renumbered as $\mathrm{IN}_{(6)}$ and $\mathrm{IN}_{(7)}$, respectively. The new intermediate node $\mathrm{IN}_{(5)}$ also becomes the child of the previous lowest intermediate node $\mathrm{IN}_{(6)}$. Using the previous blinded group key at $\mathrm{IN}_{(4)}$ of the larger group and blinded session random br_5 and br_6, the sponsor in the second round, M_4, computes all intermediate keys and bkeys $(k_4, bk_4, k_5, bk_5, k_6, bk_6)$ except the root node. Finally, it broadcasts $BT(4)$ that contains all bkeys and blinded session randoms up to $\mathrm{IN}_{(6)}$.[4] Upon receipt of the broadcast, every member can compute the group key.

In summary, the merge protocol runs in two communication rounds.

5 Discussion

We now discuss security, efficiency, and other practical issues related to STR key management.

5.1 Security

As discussed earlier in the paper, the main security requirements of group key agreement are: group key

3. Bkeys do not need to be exchanged this time.
4. In fact, it need not broadcast unchanged bkeys, $\{bk_1, bk_2, bk_3\}$.

Step 1: Each sponsor M_{s_i} in T_{s_i} for $i \in [1, k]$

- broadcasts tree $BT_{\langle s_i \rangle}$

M_{s_i} $\xrightarrow{BT_{\langle s_i \rangle}}$ $\cup_{i=1}^{k} \mathcal{C}_i$

Step 2: Every member

- updates key tree by merging all trees,
- removes all keys and bkeys from the sponsor node,

The sponsor M_s (additionally)

- generates new share r_s and computes br_s,
- computes all keys and bkeys from its parent to the node just below root,
- broadcasts updated tree $BT_{\langle s \rangle}$

$\cup_{i=1}^{k} \mathcal{C}_i$ $\xleftarrow{BT_{\langle s \rangle}}$ M_s

Step 3: Every member computes the group key using $BT_{\langle s \rangle}$

Fig. 7. MERGE Protocol.

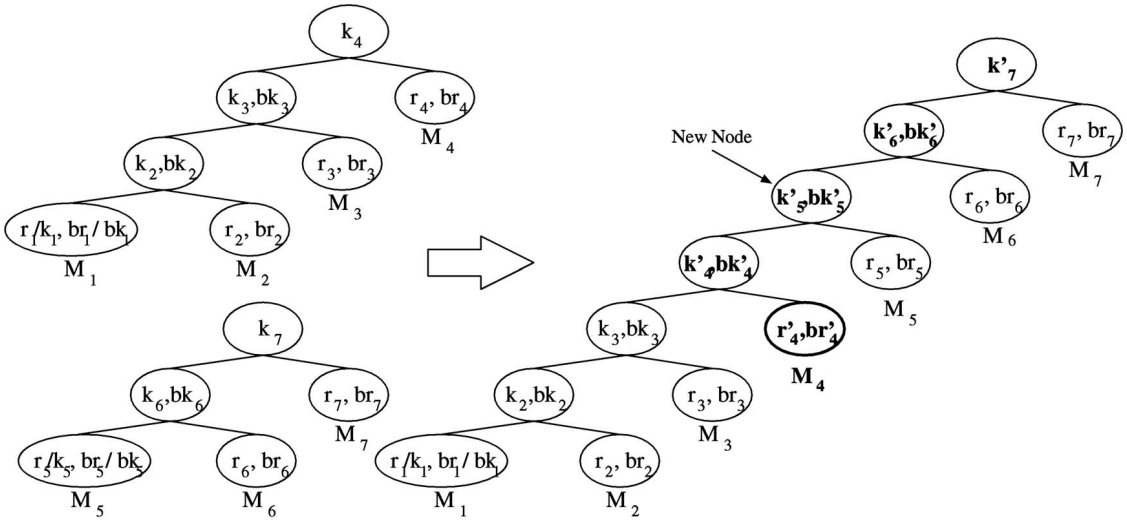


Fig. 8. Tree update in MERGE.

secrecy, forward/backward secrecy, and key independence. In this section, we prove that STR provides those four security requirements.

5.1.1 Group Key Secrecy

Before considering group key secrecy, we briefly examine key freshness. Every group key is *fresh* since at least one member in the group generates a new random key share for every membership change.[5] The probability that the new group key is the same as any old group key is negligible due to the bijectiveness of the $(f \circ g)$ function.

5. Recall that insider attacks are not our concern. This excludes the case when an insider intentionally generates nonrandom numbers.

We note that the root (group) key is never used directly for the purposes of encryption, authentication, or integrity. Instead, special-purpose subkeys are derived from the this key, e.g., by applying a cryptographically secure hash function, i.e., $H(group\ key)$ is used for such applications.

As discussed in Section 2.4, decisional group key secrecy is more meaningful if subkeys are derived from a group key. Decisional group key secrecy of STR protocol is related to the imbalanced tree decision Diffie-Hellman assumption mentioned in Section 2.2. This assumption ensures that there is no information leakage other than public bkey information.

We can also derive the subkeys based on the Shoup's hedge technique [26] as follows: Compute the key as: $H(group\ key) \oplus \mathcal{H}(group\ key)$, where $\mathcal{H}$ is a random oracle.

It follows that, in addition to the security in the standard model based on the imbalanced Tree Decision Diffie-Hellman assumption, the derived key is also secure in the random oracle model [6] based on the imbalanced Tree Computational Diffie-Hellman assumption.

5.1.2 Key Independence

We now give an informal proof that STR satisfies forward and backward secrecy or, equivalently, key independence. In order to show that STR provides key independence, we only need to show that the former (prospective) member's *view* of the current tree is exactly the same as the passive adversary's *view*. This is because the advantage of the former (prospective) member is the same as the passive adversary and the view of the passive adversary does not reveal any information about the group key by Theorem 3.

We first consider backward secrecy, which states that a new member who knows the current group key cannot derive any previous group keys. Let M_{n+1} be the new member. The sponsor for the join event changes its session random and, consequently, the root key of the current key tree is changed. Therefore, the *view* of M_{n+1} with respect to the prior key trees is exactly the same as the *view* of an outsider. Hence, the new member does not gain any advantage compared to a passive adversary.

This argument can be easily extended to a merge of two or more groups. When a merge happens, the sponsor at the top leaf node of the largest tree changes its session random. Therefore, each member's *view* on other member's tree is exactly the same as the *view* of a passive adversary. This shows that the newly merged member has exactly the same advantage about any of the old key tree as a passive adversary.

Now, we consider forward secrecy, meaning that a passive adversary who knows a contiguous subset of old group keys cannot discover subsequent group keys. Here, we consider partition and leave at the same time. Suppose M_d is a former group member who left the group. Whenever subtractive event happens, the sponsor located immediately below the deepest leaving leaf node refreshes its session random and, therefore, all keys known to leaving members will be changed accordingly. Therefore, M_d's *view* is exactly the same as the *view* of the passive adversary.

This proves that STR provides a decisional version of key independence.

5.1.3 Other Security Properties

As discussed in Section 2.4, all protocol messages consist of sender information, group information, membership information, message type, key epoch, and time stamp. We also assumed that receiver rejects any message that does not match its expectation and all channels are authentic (i.e., all messages are signed). Therefore, we claim that STR provides implicit key authentication.

Furthermore, the independence of the session key from any long-term keys guarantees PFS. Finally, the loss of a group key does not endanger any other session. Therefore, STR is secure against a known key attack.

5.2 Practical Considerations

5.2.1 Protocol Unification

Although described separately in Section 4, the four STR operations (join, leave, merge, and partition) actually represent different strands of a single protocol. We justify this claim with an informal argument below.

Obviously, join and leave are special cases of merge and partition, respectively. We observed that merge and partition can be collapsed into a single protocol since, in either case, the key tree changes and the remaining group members lack some number of bkeys that prevents them from computing the new root key. In a partition, the remaining members (in any surviving group fragment) reconstruct the tree where some bkeys are missing. In the case of a merge, let us suppose that k groups (Tree T_1 through T_k) are merging. After the first round of the merge protocol, all members reconstruct the new tree unambiguously and independently where all bkeys from the sponsor node up to the root node are missing, similarly to the partition protocol. The sponsor in merge is located at the topmost leaf node of the highest key tree. As discussed in Sections 4.4 and 4.3, every member reconstructs the key tree after a partition and a merge in one and two rounds, respectively.

From these outlines of the merge and partition protocol, we can find some similarities:

- Whenever new membership event happens, all *current* group members first reconstruct the key tree.
- The resulting key tree has missing bkeys from the parent node of the sponsor to the root node, as well as the sponsor's blinded session random.
- The sponsor generates new session random and computes all keys and bkeys from its parent node up to the node just below the root node. It then broadcasts the whole key tree containing only bkeys and blinded session randoms.
- Using the broadcast message, any member can compute the group key.

This apparent similarity between partition and merge allows us to combine the protocols stemming from all membership events into a single, unified protocol. Fig. 9 shows the pseudocode. The incentive for this is threefold. First, unification allows us to simplify the implementation and minimize its size. Second, the overall security and correctness are easier to demonstrate with a single protocol. Third, we can now claim that (with a slight modification) the STR protocol is self-stabilizing and fault-tolerant as discussed below.

5.2.2 Cascaded Events

Since network disruptions are random and unpredictable, it is natural to consider the possibility of so-called *cascaded membership events*. (In fact, cascaded events and their impact on group protocols are often considered in group communication literature, but, alas, not often enough in the security literature.) A cascaded event occurs, in its simplest form, when one membership change occurs while another is being handled. Event here means any of: join, leave, partition, merge, or a combination thereof. For example, a

```
 1  receive msg (msg type = membership event)
 2  construct new tree
 3  while there are missing bkeys
 4    if ((I can compute any missing keys and I am the sponsor) ||
 5        (sponsor computed a key))
 6      while(1)
 7        compute missing (key, bkey) pairs
 8        if (I cannot compute)
 9          break
10        endif
11      if (others need my information)
12        broadcast new bkeys
13      endif
14    endif
15  endwhile
```

Fig. 9. Unified protocol pseudocode.

partition can occur while a prior partition is being dealt with, resulting in a cascade of size two. In principle, cascaded events of arbitrary size can occur if the underlying network is highly volatile.

As discussed before, STR protocol requires at most two rounds. One might wonder why robustness against cascaded failure is important for only a 2-round protocol. We give a couple of examples that illustrate (potential) failure of the STR protocol.

- Suppose a network partition breaks a group $\mathcal{G}$ into groups $\mathcal{G}_1$ and $\mathcal{G}_2$. The sponsor $M_{\mathcal{G}_1}$ needs to compute missing keys and bkeys. While computing these keys, another partition breaks $\mathcal{G}_1$ into two other groups $\mathcal{G}_1^1$ (containing $M_{\mathcal{G}_1}$) and $\mathcal{G}_1^2$. Based on the partition protocol description, the members in group $\mathcal{G}_1^2$ still wait for the message from $M_{\mathcal{G}_1}$ to process the previous partition.
- Suppose a merge event happens whereby groups $\mathcal{G}_1$ and $\mathcal{G}_2$ form a single group $\mathcal{G}$. The sponsors $M_{\mathcal{G}_1}$ and $M_{\mathcal{G}_2}$ in each group broadcast their tree information. In the next round, while a sponsor computes the missing bkeys, a member M_1 originally in group $\mathcal{G}_1^1$ leaves the group. If the leaving member is the sponsor, the STR protocol cannot proceed for every other member is waiting for the message from this member.

The protocols described above cannot cope with these situations. However, we can modify the protocol in Fig. 9 to handle such cascaded events.

We claim that the STR protocol is self-stabilizing, i.e., robust against cascaded network events. This is quite rare as most multiround cryptographic protocols are not geared toward handling of such events. In general, self-stabilization is a very desirable feature since lack thereof requires extensive and complicated protocol "coating" to either 1) shield the protocol from cascaded events or 2) harden it sufficiently to make the protocol robust with respect to cascaded events (essentially, by making it reentrant).

The high-level pseudocode for the self-stabilizing protocol is shown in Fig. 10. The changes from Fig. 9 are minimal (lines 15-18 are added).

6 Performance Anaysis and Communication Efficiency

6.1 Performance Comparison

We analyze both communication and computation costs for the join, leave, merge, and partition protocols. In doing so, we focus on the number of: rounds, messages, and serial exponentiations. We distinguish among serial and total measures. The serial measure assumes parallelization within each protocol round and represents the greatest cost incurred by any participant in a given round. The total measure is the sum of all participants' costs in a given round.

We compare STR protocols to TGDH which has been known to be most efficient in both communication and computation. For a detailed comparison with other group key agreement protocols such as GDH.3 [28], BD (Burmester-Desmedt) [11] can be found in [2].

Table 1 summarizes the communication and computation costs of both protocols. The numbers of current group members, merging members, merging groups, and leaving members are denoted as: n, m, k, and p, respectively.

The height of the key tree constructed by the TGDH protocol is h. The overhead of the TGDH protocol depends on the tree height, the balancedness of the key tree, the location of the joining tree, and the leaving nodes. In our analysis, we assume the worst-case configuration and list the worst-case cost for TGDH.

The number of modular exponentiations for a leave event in STR depends on the location of the deepest leaving node. We thus compute the average cost, i.e., the case when the $\frac{n}{2}\text{th}$ node leaves the group. For all other events and protocols, exact costs are shown.

In the current implementations of TGDH and STR, all group members recompute bkeys that have already been computed by the sponsors. This provides a weak form of key confirmation since a user who receives a token from another member can check whether his bkey computation is correct. This computation, however, can be removed for better efficiency and we consider this optimization when counting the number of exponentiations.

It is clear that the computation cost of STR is fairly high: $O(m)$ for merge and $O(n)$ for subtractive events. However,

```
 1  receive msg (msg type = membership event)
 2  construct new tree
 3  while there are missing bkeys
 4    if ((I can compute any missing keys and I am the sponsor) ||
 5        (sponsor computed a key))
 6      while(1)
 7        compute missing (key, bkey) pairs
 8        if (I cannot compute)
 9          break
10        endif
11      if (others need my information)
12        broadcast new bkeys
13      endif
14    endif
15    receive msg
16    if (msg type = membership event)
17      construct new tree
18    endif
19  endwhile
```

Fig. 10. Self-stabilizing protocol pseudocode.

as mentioned in Section 1, this high cost becomes negligible when STR is used in a high-delay wide-area network. Evidence to support this claim can be found in [2].

6.2 Lower Bound for Dynamic Group Key Agreement

In [5], Becker and Wille proved the lower bound for communication complexity of static group key agreement, i.e., how n group members share a common group key without considering subsequent additive/subtractive events. When assuming bradcast channel, they prove the following theorem:

Theorem 1 (Becker and Wille). *Let $\mathcal{P}$ be a static group key agreement protocol for n parties allowing broadcasts.*

1. *For the number of messages $\mathrm{m}(\mathcal{P})$ required by $\mathcal{P}$, it holds that $\mathrm{m}(\mathcal{P}) \geq n$.*
2. *For the number of rounds $\mathrm{r}(\mathcal{P})$ required by $\mathcal{P}$, it holds that $\mathrm{r}(\mathcal{P}) \geq 1$.*

However, it is commonly assumed that at least two rounds are required for group key agreement.

Assumption 1. *Let $\mathcal{P}$ be a static group key agreement protocol for n parties allowing broadcasts when $n > 3$. Using the same notation above, $\mathrm{r}(\mathcal{P}) \geq 2$.*

Indeed, finding an one-round group key agreement is a well-known open problem [8]. When group size is 3, there exists one round group key agreement based on Bilinear map using Weil paring [17]. This work shows that we can design one round group key agreement protocol for any n, if a multilinear map exists. Unfortunately, the existence of a multilinear map is unknown [8].

TABLE 1
Communication and Computation Costs

		Communication		Computation
		Round	Message	Exponentiation
TGDH	Join	2	3	$3h-3$
	Leave	1	1	$3h-3$
	merge	$\lceil \log_2 k \rceil + 1$	$2k$	$3h-3$
	Partition	$min(\lceil \log_2 p \rceil + 1, h)$	$min(2p, \lceil \frac{n}{2} \rceil)$	$3h-3$
STR	Join	2	3	4
	Leave	1	1	$\frac{3n}{2}+2$
	Merge	2	$k+1$	$3m+1$
	Partition	1	1	$\frac{3n}{2}+2$

Based on Theorem 1 and Assumption 1, we can easily find the bound for communication complexity of dynamic group key agreement.

Theorem 2 (Communication complexity of dynamic group key agreement). *Let $\mathcal{P}$ be a static group key agreement protocol for n ($n > 3$) parties allowing broadcasts.*

- *For any subtractive events $\mathrm{r}(\mathcal{P}) \geq 1$ and $\mathrm{m}(\mathcal{P}) \geq 1$, when the number of remaining group members is greater than 2.*
- *For any additive events $\mathrm{r}(\mathcal{P}) \geq 2$ and $\mathrm{m}(\mathcal{P}) \geq k$, when k groups are merging.*[6]

Proof (Sketch). In a contributory group key agreement, group key is determined by participating entities contribution. Furthermore, to provide key independence, each group key should be independent from the previous keys/future group keys. In other words, for any additive/subtractive events, at least one member in the group has to change its random secret. Therefore, at least one message (and one round) is required to let others know about this change. This provides rough lower bounds of communication for both additive/subtractive events:

$$\mathrm{r}(\mathcal{P}) \geq 1 \text{ and } \mathrm{m}(\mathcal{P}) \geq 1. \tag{1}$$

Now, let us tighten the bound based on each event. In the case of subtractive events, we are done by the rough bound described in (1).

So, the remainder of this proof will focus on finding a tighter lower bound for additive events. We will consider only the merge of k groups since join is a special case of merge when one group has only one user. One of the most important observations for merge is that the merge of k groups can be seen as a static group key agreement of k members. If this is the case, then we are done since our lower bounds for additive events are the same as those for static group key agreement provided in Theorem 1 and Assumption 1.

Now, it remains to show that the merge of k groups is equivalent to the static group key agreement of k members. Since there are k groups merging, it is obvious that at least k messages need to be exchanged to share each group information. This is because the group key is a function of all group members' contribution and each group information contains current group members' contribution. In fact, the merge of k groups can be seen as a group formation of k members whose session random is the current group key sk and the blinded key is $g^{sk} \pmod p$, where the blinded key is never known to other group members. Therefore, lower bounds of communication for additive events are equvalent to those of static group key agreement. Consequently, any additive events of group key agreement (when k groups are involved) requires $\mathrm{r}(\mathcal{P}) \geq 2 \text{ and } \mathrm{m}(\mathcal{P}) \geq k$. □

From Theorem 2, the communication costs of STR are near optimal (it requires one more message than the optimal protocol does). However, it can be easily modified to achieve optimal communication efficiency: When a merge even happens, a partition is chosen unambiguously (such as the partition that has a group member whose alphabetical order precedes all other members). All sponsors in other partition send tree information to the partition ($k-1$ messages). Upon receiving these messages, the sponsor in the partition can compute all of the required blinded keys and it broadcasts the whole key tree containing only blinded keys (one more message). Finally, all members can compute the group key.

This protocol has optimal communication costs: k messages and two rounds. However, this has an obvious drawback: When the group including the sponsor has only one member, whole $n-1$ blinded keys need to be recomputed. On the other hand, if we can choose a highly populated partition, we can save a number of modular exponentiation. Therefore, in the first round of merge, the sponsor in every partition sends their tree information (k messages) and the sponsor in the biggest group will act as the sponsor to broadcast a new set of bkeys. Note that the number of the round is more sensitive for the performance of multiround multiparty protocol than the number of the message as shown in [2].

7 Related Work

Group key management protocols come in three different flavors: contributory key agreement protocols, centralized, decentralized group key distribution scheme, and server-based key distribution protocols. Since the focus of this work is to provide a common key to the dynamic peer group, we only consider the first two below.

7.1 Group Key Agreement Protocols

We begin by first summarizing the early (and theoretical) group key agreement protocols which did not consider dynamic membership operations and only supported group formation.

The earliest attempt to obtain contributory group key agreement built upon 2-party Diffie-Hellman (DH) is due to Ingemarsson et al. (called ING) for teleconferencing [16]. In the first round of ING, every member M_i generates its session random N_i and computes α^{N_i}. In the subsequent rounds k to $n-1$, M_i computes $K_{i,k} = (K_{i-1 \bmod n, k-1})^{N_i}$, where K_{i-1} is the message received from M_{i-1} in the previous round $k-1$ when n is the number of group members. The resulting group key is of the form:

$$K_n = \alpha^{N_1 N_2 N_3 \ldots N_n}.$$

The ING protocol is inefficient:

1. Every member has to start synchronously,
2. $n-1$ rounds are required to compute a group key,
3. It is hard to support dynamic membership operations due to its symmetricity, and
4. n sequential modular exponentiations are required.

Another group key agreement developed for teleconferencing was proposed by Kim et al. [18]. This protocol (called TGDH, for Tree-based Group Diffie-Hellman) is of

6. Clearly, this also covers the case of a single member joining the group, thus k is equal to 2.

particular interest since its group key structure is similar to that of STR.

TGDH is well-suited for member leave operation since it takes only one round and $log(n)$ modular exponentiations. Member addition, however, is relatively costly since—in order to keep the key tree balanced—the sponsor performs $log(n)$ exponentiations. Also, in the event of partition, as many as $log(n)$ rounds may be necessary to stabilize the key tree. This is where STR offers a clear advantage.

Burmester and Desmedt construct an efficient protocol (called BD) which takes only two rounds and three modular exponentiations per member to generate a group key [11]. This efficiency allows all members to recompute the group key for any membership change by performing this protocol. However, according to [28], most (at least half) of the members need to change their session random on every membership event. The group key in this protocol is different from STR and TGDH:

$$K_n = \alpha^{N_1N_2+N_2N_3+...+N_nN_1}.$$

A shortcoming of BD is the high communication overhead. It requires $2n$ broadcast messages and each member needs to generate two signatures and verify $2n$ signatures.

Becker and Wille analyze the minimal communication complexity of contributory group key agreement in general [5] and propose two protocols: *octopus* and *hypercube*. Their group key has the same structure as the key in TGDH. For example, for eight users, their group key is:

$$K_n = \alpha^{(\alpha^{r_1r_2}\alpha^{r_3r_4})(\alpha^{r_5r_6}\alpha^{r_7r_8})}.$$

The Becker/Wille protocols handle join and merge operations efficiently, but the member leave operation is inefficient. Also, the *hypercube* protocol requires the group to be of size 2^n (for some integer n); otherwise, the efficiency slips.

Asokan and Ginzboorg look at the problem of small-group key agreement, where the members do not have previously set up security associations [3]. Their motivating example is a meeting where the participants want to bootstrap a secure communication group. They adapt password authenticated DH key exchange to the group setting. Their setting, however, is different from ours since they assume that all members share a secret password, whereas we assume a PKI where each member can verify any other members authenticity and authorization to join the group.

Tzeng and Tzeng propose an authenticated key agreement scheme that is based on secure multiparty computation [29]. This scheme also uses $2 \cdot N$ broadcast messages. Although the cryptographic mechanisms are quite elegant, a shortcoming is that the resulting group key does not provide perfect forward secrecy (PFS). If a long-term secret key is broken and/or published, all previous and future group keys (where that key was used) are also revealed.

Steiner et al. first address dynamic membership issues [4], [28] in group key agreement and propose a family of Group Diffie Hellman (GDH) protocols based on straightforward extensions of the two-party Diffie-Hellman. GDH provides contributory authenticated key agreement, key independence, key integrity, resistance to known key attacks, and perfect forward secrecy. Their protocol suite is fairly efficient in leave and partition operation, but the merge protocol requires as many rounds as the number of new members to complete key agreement.

Perrig extends the work of one-way function trees (OFT, originally introduced by McGrew and Sherman [20]) to design a tree-based key agreement scheme for peer groups [23]. However, this work does not consider group merges and partitions.

7.2 Decentralized Group Key Distribution Protocols

Decentralized group key distribution protocols can be preferred to contributory group key agreement protocols since they rely on an inexpensive symmetric key encryption technique. However, all group key distribution schemes assume a secure channel that is, in practice, implemented by public key cryptosystem (e.g., Diffie-Hellman). Furthermore, they require the leader to establish multiple secure two-party channels between itself and other group members in order to securely distribute the new key. Maintaining such channels in dynamic groups can be expensive since setting up each channel involves a separate two-party key agreement. When a group is dynamic, the amortized number of secure channel becomes $O(n^2)$. Another disadvantage is the reliance on a single entity to generate good (i.e., cryptographically strong, random) keys.

The first decentralized group key distribution scheme is due to Caronni et al. [12]. They propose efficient protocols for small-group key agreement and large-group key distribution. Unfortunately, their scheme for autonomous small group key agreement is not collusion resistant.

Dondeti et al. modified OFT (One-way Function Tree) [20] to provide dynamic server election [14]. This protocol has the same key tree structure and uses similar notations (e.g., keys, blinded keys). Other than the expensive maintainence of secure channels described above, this protocol has expensive communication cost: Even for single join and leave, this protocol can take $O(h)$ rounds to complete when h is the height of the key tree. The authors do not consider merge and partition event and also implementation. One advantage different from others is that their group key does not depend on a single entity.

Rodeh et al. [24] propose a decentralized group key distribution protocol extended from the LKH protocol [30]. It tolerates network partitions and other network events. Even though this approach cannot help incurring basic disadvantages discussed above, the authors reduce the communication and computational cost. In addition, the authors use the AVL tree to provide provable and efficient tree height.

8 Conclusion

In this paper, we described a provably secure contributory group key agreement protocol (STR) optimized for communication. STR supports all dynamic peer group operations: join, leave, merge, and partition. Furthermore, it easily handles cascaded (nested) membership events and network failures.

Assuming that Moore's Law continues to hold, the computational cost of cryptographic operations will gradually decrease. Eventually, communication latency, which

has a lower-bound dictated by the speed of light, will dominate the cost of computation in determining the running time of group key agreement protocols. STR is already the most efficient group key agreement protocol over high-delay wide-area networks; it will become more advantageous as processor speeds increase.

APPENDIX A

DECISIONAL IMBALANCED GROUP DIFFIE-HELLMAN PROBLEM

A.1 2-party Decision Diffie-Hellman Problem

Our proofs require a specific group setting. In this section, we introduce a specific group (G) and define the 2-party Decision Diffie-Hellman (DDH) problem on G.

Let k be a security parameter and n be an integer. All algorithms run in probabilistic polynomial time with k and n as inputs.

For concreteness, we consider a specific group G:

On input k, algorithm *gen* chooses, at random, a pair (q, α), where q is a $2k$-bit value[7] and q and $p = 2q + 1$ are both prime. Before introducing G, we first consider a group $\widehat{G}$, which is a group of squares modulo prime p. The group can be described more precisely as follows: Consider an element α which is a square of a primitive element $\widehat{\alpha}$ of multiplicative group $\mathbb{Z}_p^*$, i.e., $\alpha = \widehat{\alpha}^2$. (Without loss of generality, we may assume $\alpha < q$.) Then, group $\widehat{G}$ can be represented as:

$$\widehat{G} = \left\{\alpha^i \bmod p \mid i \in [1, q]\right\}.$$

An attractive variation of this group is to represent the elements by the integers from 0 to $q-1$. The group operation is slightly different: Let a function f be defined as

$$f(x) = \begin{cases} x & \text{if } x \leq q \\ p - x & \text{if } q < x < p. \end{cases}$$

Using this f function, we can introduce the group G as

$$G = \left\{f(\alpha^i \bmod p) \mid i \in \mathbb{Z}_q\right\}.$$

Group operation on group G is defined as $a \cdot b = f(a \cdot b \pmod p)$, where $a, b \in G$.

Proposition 3. *Let $g(x) = \alpha^x \bmod p$. Then, the function $f \circ g$ is a bijection from $\mathbb{Z}_q$ to $\mathbb{Z}_q$.*

Proof. To see this, suppose $f \circ g(x) = f \circ g(y)$. Then, this can be written and $f(X) = f(Y)$, where integer $X = \alpha^x \bmod p$ and $Y = \alpha^y \bmod p$. Now, we can have four different cases:

- $X \leq q, Y \leq q$: In this case, $f(X) = X$ and $f(Y) = Y$ and, hence, $X = Y$. Now, we have an equation $\widehat{\alpha}^{2(x-y)} = 1 \bmod p$. Since $\widehat{\alpha}$ is a generator for $\mathbb{Z}_p^*$, its order (i.e., $2q$) has to divide $2(x-y)$. This implies that q has to divide $x - y$ and, finally, $x = y$ since $0 < x, y \leq q$.
- $X > q, Y > q$: In this case, $f(X) = p - X$ and $f(Y) = p - Y$ and, hence, $X = Y$. The rest is the same as above.
- $X \leq q, Y > q$: This case is impossible, since $(\frac{X}{p}) = 1$ and $(\frac{p-Y}{p}) = -1$ since $p \equiv 3 \bmod 4$ and $X = p - Y$.
- $X > q, Y \leq q$: This is also impossible by similar reasoning.

Therefore, $f \circ g$ is an injection. It is also a surjection since the sizes of the domain and codomain are the same. □

7. In order to achieve the security level 2^{-k}, the group size should be at least 2^{2k} [25].

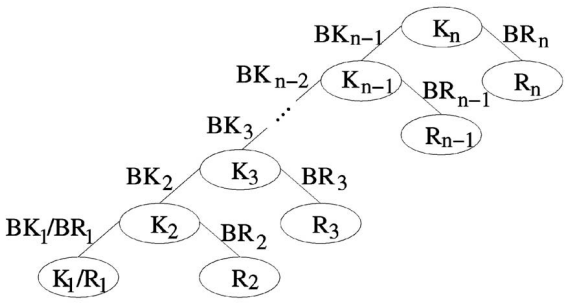


Fig. 11. Notations for Imbalanced Tree.

Proposition 4. *When a distribution r is uniform and random in G, $f \circ g(r)$ is still uniform and random in G since $f \circ g$ is bijective.*

Groups of this type are also considered by Chaum [13]. It is generally assumed that DDH is intractable in these groups [7]. More concretely, the **2-party Decision Diffie-Hellman assumption on group** G is that, for all polynomial time attackers $\mathcal{A}$, for all polynomials $Q(k)$ $\exists k_0$ $\forall k > k_0$, for $X_0 := N_1 N_2$ and $X_1 := N_3$ with $N_1, N_2, N_3 \in_{\mathcal{R}} G$ uniformly chosen and, for a random bit b, the following equation holds:

$$\left|Prob[\mathcal{A}(1^k; G; \alpha; \alpha^{N_1}; \alpha^{N_2}; \alpha^{X_b}) = b] - 1/2\right| < 1/Q(k).$$

A.2 Decisional Imbalanced Tree Group Diffie-Hellman Problem

We start with the easier problem where a key tree is completely imbalanced. Fig. 11 shows the structure and the notation for the imbalanced tree.

For $(q, \alpha) \leftarrow gen(k), n \in \mathbb{N}$ and $X = (R_1, R_2, \ldots, R_n)$ for $R_i \in G$ and an imbalanced key tree IT with n leaf nodes which correspond to R_i, we define the following random variables:

- K_i: ith level key
- BK_i: ith level blinded key, i.e., $\alpha^{K_i} \bmod p$
- R_i: ith level session random chosen uniformly $\in_{\mathcal{R}} G$, where G is the group mentioned in the previous section. For $i = 1$, $R_i = K_i$.
- BR_i: ith level blinded session random, i.e., $\alpha^{R_i} \bmod p$. For $i = 1$, $BR_i = BK_i$.
- K_i is recursively defined as follows:

$$K_i = \alpha^{K_{i-1} R_i} = BK_{i-1}^{R_i} = BR_i^{K_{i-1}},$$

K_i and R_i are secret, and BK_i and BR_i are public. In Section 4, BK_i and BR_is will be publicly available, while K_i will be known to group members and R_i will be known only to a single member. The root node K_i will be used as a group key.

For $(q, \alpha) \leftarrow gen(k), n \in \mathbb{N}$ and $X = (R_1, R_2, \ldots, R_n)$ for $R_i \in G$ and an imbalanced key tree IT with n leaf nodes which correspond to N_i, we can define public and secret values collectively as below:

$$
\begin{aligned}
&view(q, \alpha, n, X.IT) := \{BK_i \mid 1 \le i \le n\} \cup \{BR_i \mid 1 \le i \le n\} \\
&= \{\alpha^{K(i,X,IT)} \bmod p \mid 1 \le i \le n-1\} \cup \{BR_i \mid 1 \le i \le n\} \\
&= \{\alpha^{\alpha^{R_1 R_2}}, \alpha^{\alpha^{R_3 \alpha^{R_1 R_2}}}, \ldots, \alpha^{\alpha^{R_{n-1} \ldots \alpha^{R_1 R_2}}}\} \cup \{\alpha^{R_1}, \alpha^{R_2}, \ldots, \alpha^{R_n}\} \\
&K(q, \alpha, n, X, IT) := \alpha^{K_{n-1} R_n}.
\end{aligned}
$$

Since (q, α) are obvious from the context, we omit them in *view*() and *K*(). Also, for simplicity, we sometimes use K_n instead of $K(n, X, IT)$. The $view(n, X, IT)$ represents all public information, and the root secret key is $K(n, X, IT)$. Let the following two random variables be defined by generating $(q, \alpha) \leftarrow gen(k)$ and choosing X randomly from G:

- $A_n := (view(n, X, IT), y)$ and
- $D_n := (view(n, X, IT), K_n)$.

The operator "$\approx_{poly}$" denotes polynomial indistinguishability as in [28].

Proposition 5. *Let K and R be l-bit strings such that R is a random and K is a Diffie-Hellman key. We say that K and R are **polynomially indistinguishable** if, for all polynomial time distinguishers, A, the probability of distinguishing K and R is smaller than $(\frac{1}{2} + \frac{1}{Q(k)})$, for all polynomial $Q(l)$.*

The following is a main lemma (induction argument) for the DITGDH problem.

Lemma 1. *If the DDH assumption holds and $A_{n-1} \approx_{poly} D_{n-1}$, then $A_n \approx_{poly} D_n$.*

Proof. Assume that there exists a polynomial algorithm that can distinguish between A_n and D_n. We will show that this algorithm can be used to distinguish A_{n-1} and D_{n-1} or solve the 2-party DDH problem.

Consider the following equations when $X_1 = (R_1, R_2, \ldots, R_{n-1})$ and IT_1 is a subtree rooted at the left child of the root node:

$$
\begin{aligned}
&A_n := (view(n, X, IT), y) \\
&= (view(n-1, X_1, IT_1), BK_{n-1}, BR_n, y) \\
&= (view(n-1, X_1, IT_1), \alpha^{K(n-1,X_1)}, \alpha^{R_n}, y) \\
&B_n := (view(n-1, X_1, IT_1), \alpha^r, \alpha^{R_n}, y) \\
&C_n := (view(n-1, X_1, IT_1), \alpha^r, \alpha^{R_n}, \alpha^{rR_n}) \\
&D_n := (view(n, X, IT), K(n, X, IT)) \\
&= (view(n-1, X_1, IT_1), BK_{n-1}, BR_n, \alpha^{K(n-1,X_1,IT_1)R_n}) \\
&= (view(n-1, X_1, IT_1), \alpha^{K(n-1,X_1,IT_1)}, \alpha^{R_n}, \alpha^{K(n-1,X_1,IT_1)R_n}).
\end{aligned}
$$

Since we can distinguish A_n and D_n in polynomial time, we can distinguish at least one of (A_n and B_n) or (B_n and C_n) or (C_n and D_n).

- A_n and B_n: Suppose one can distinguish A_n and B_n in polynomial time. We will show that this distinguisher $\mathcal{A}_{AB_n}$ can be used to solve the DITGDH problem with height $n-1$. Suppose we want to decide whether $P'_{n-1} = (view(n-1, X', IT'), r')$ is an instance of the DITGDH problem or r' is a random number. To solve this problem, we generate a random number r'' and compute $\alpha^{r''}$. Using P'_{n-1} and $(r''; , \alpha^{r''})$ pair, we can generate a distribution

  $$P_n = (view(n-1, X', IT'), \alpha^{r'}, \alpha^{r''}, y),$$

  where $y \in_{\mathcal{R}} G$. Now, we put P_n as an input of $\mathcal{A}_{AB_n}$. If P_n is an instance of A_n (B_n), then P'_{n-1} is an instance of D_{n-1} (A_{n-1}) by Proposition 4, respectively.
- B_n and C_n: Suppose we can distinguish B_n and C_n in polynomial time. We will show that this distinguisher $\mathcal{A}_{BC_n}$ can be used to solve the 2-party DDH problem in group G. Note that α^r is an independent variable from $view(n-1, X_1, T_1)$. Suppose we want to test whether $(\alpha^a, \alpha^b, \alpha^c)$ is a DDH triple or not. To solve this problem, we generate a key tree T' of height $n-1$ with distributions X'. Now, we generate a new distribution:

  $$P_n = (view(n-1, X_1, T_1), \alpha^a, \alpha^b, \alpha^c).$$

  If P_n is an instance of B_n (C_n), then $(\alpha^a, \alpha^b, \alpha^c)$ is a valid (invalid) DDH triple, respectively.
- C_n and D_n: Suppose one can distinguish C_n and D_n in polynomial time. We will show that this distinguisher $\mathcal{A}_{CD_n}$ can be used to solve the DITGDH problem with height $n-1$. Suppose we want to decide whether $P'_{n-1} = (view(n-1, X', IT'), r')$ is an instance of the DITGDH problem or r' is a random number. To solve this problem, we generate a random number r'' and compute $\alpha^{r''}$. Using P'_{n-1} and $(r'', \alpha^{r''})$ pair, we generate a distribution:

  $$P_n = (view(n-1, X', IT'), \alpha^{r'}, \alpha^{r''}, \alpha r' r'').$$

  Note that we can compute $\alpha^{r'r''}$ since we know $\alpha^{r'}$ and r''. Now, we put P_n as an input of $\mathcal{A}_{CD_n}$. If P_n is an instance of C_n (D_n), then P'_{n-1} is an instance of D_{n-1} (A_{n-1}) by Proposition 4. □

Lemma 2. *If the DDH assumption holds, then $A_3 \approx_{poly} D_3$.*

The proof is similar to the above. The only difference is that we can break the 2-party DDH asumption using $\mathcal{A}_{AB_3}$ or $\mathcal{A}_{CD_3}$.

Using induction and Lemmas 1 and 2, the following theorem can be easily proven.

Theorem 3 (DITGDH problem). *If the 2-party DDH problem is hard, then DITGDH is also hard.*

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their helpful and insightful comments. The work of Yongdae Kim and Gene Tsudik was supported in part by grants F30602-00-2-0526 and DABT63-97-C-0031 from the US Defense Advanced Research Projects Agency (DARPA). The work of Adrian Perrig was supported in part by the Carnegie Mellon University Center for Computer and Communications Security under grant DAAD19-02-1-0389 from the US Army Research Office and by gifts from the Bosch Corporation. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of the funding organizations. An early version of this paper appeared, in part, in [19].

REFERENCES

[1] Y. Amir, G. Ateniese, D. Hasse, Y. Kim, C. Nita-Rotaru, T. Schlossnagle, J. Schultz, J. Stanton, and G. Tsudik, "Secure Group Communication in Asynchronous Networks with Failures: Integration and Experiments," *Proc. IEEE Int'l Conf. Distributed Computing Systems,* Apr. 2000.

[2] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik, "On the Performance of Group Key Agreement Protocols," Technical Report CNDS-2001-5, The Johns Hopkins Univ., http://www.cnds.jhu.edu/pub/papers/cnds-2001-5.pdf, 2001.

[3] N. Asokan and P. Ginzboorg, "Key-Agreement in Ad-Hoc Networks," *Proc. Nordsec '99,* 1999.

[4] G. Ateniese, M. Steiner, and G. Tsudik, "Authenticated Group Key Agreement and Friends," *Proc. Fifth ACM Conf. Computer and Comm. Security,* pp. 17-26, Nov. 1998.

[5] C. Becker and U. Wille, "Communication Complexity of Group Key Distribution," *Proc. Fifth ACM Conf. Computer and Comm. Security,* Nov. 1998.

[6] M. Bellare and P. Rogaway, "Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols," *Proc. First ACM Conf. Computer and Comm. Security,* 1993.

[7] D. Boneh, "The Decision Diffie-Hellman Problem," *Proc. Third Algorithmic Number Theory Symp.,* pp. 48-63, 1998.

[8] D. Boneh and A. Silverberg, "Applications of Multilinear Forms to Cryptography," *Contemporary Math.,* to appear.

[9] E. Bresson, O. Chevassut, and D. Pointcheval, "Provably Authenticated Group Diffie-Hellman Key Exchange—The Dynamic Case," *Proc. Advances in Cryptology (ASIACRYPT 2001),* C. Boyd, ed., 2001.

[10] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater, "Provably Authenticated Group Diffie-Hellman Key Exchange," *Proc. Eighth ACM Conf. Computer and Comm. Security,* P. Samarati, ed., Nov. 2001.

[11] M. Burmester and Y. Desmedt, "A Secure and Efficient Conference Key Distribution System," *Proc. Advances in Cryptology (EUROCRYPT '94),* A. De Santis, ed., 1995.

[12] G. Caronni, M. Waldvogel, D. Sun, N. Weiler, and B. Plattner, "The VersaKey Framework: Versatile Group Key Management," *IEEE J. Selected Areas in Comm.,* vol. 17, no. 9, Sept. 1999.

[13] D. Chaum, "Zero-Knowledge Undeniable Signatures," *Proc. Advances in Cryptology (EUROCRYPT '90),* I.B. Damgard, ed., pp. 458-464, May 1991.

[14] L. Dondeti, S. Mukherjee, and A. Samal, "Disec: A Distributed Framework for Scalable Secure Many-to-Many Communication," *Proc. Fifth IEEE Symp. Computers and Comm. (ISCC 2000),* July 2000.

[15] A. Fekete, N. Lynch, and A. Shvartsman, "Specifying and Using a Partionable Group Communication Service," *Proc. 16th Ann. ACM Symp. Principles of Distributed Computing,* pp. 53-62, Aug. 1997.

[16] I. Ingemarsson, D.T. Tang, and C.K. Wong, "A Conference Key Distribution System," *IEEE Trans. Information Theory,* vol. 28, no. 5, pp. 714-720, Sept. 1982.

[17] A. Joux, "A One Round Protocol for Tripartite Diffie-Hellman," *Proc. Fourth Int'l Algorithmic Number Theory Symp. (ANTS-IV),* pp. 385-393, July 2000.

[18] Y. Kim, A. Perrig, and G. Tsudik, "Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups," *Proc. Seventh ACM Conf. Computer and Comm. Security,* pp. 235-244, Nov. 2000.

[19] Y. Kim, A. Perrig, and G. Tsudik, "Communication-Efficient Group Key Agreement," *Information Systems Security, Proc. 17th Int'l Information Security Conf. IFIP SEC '01,* 2001.

[20] D. McGrew and A. Sherman, "Key Establishment in Large Dynamic Groups Using One-Way Function Trees," manuscript, May 1998.

[21] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography.* CRC Press, 1997.

[22] L. Moser, Y. Amir, P. Melliar-Smith, and D. Agarwal, "Extended Virtual Synchrony," *Proc. IEEE Int'l Conf. Distributed Computing Systems,* pp. 56-65, June 1994.

[23] A. Perrig, "Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication," *Proc. CrypTEC '99,* pp. 192-202, 1999.

[24] O. Rodeh, K. Birman, and D. Dolev, "Optimized Rekey for Group Communication Systems," *Proc. Symp. Network and Distributed Systems Security (NDSS '00),* pp. 37-48, Feb. 2000.

[25] V. Shoup, "Lower Bounds for Discrete Logarithms and Related Problems," *Proc. Advances in Cryptology (EUROCRYPT '97),* W. Fumy, ed., pp. 256-266, 1997.

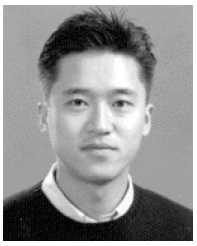
[26] V. Shoup, "Using Hash Functions as a Hedge against Chosen Ciphertext Attacks," *Proc. Advances in Cryptology (EUROCRYPT 2000),* B. Preneel, ed., pp. 275-288, 2000.

[27] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener, "A Secure Audio Teleconference System," *Advances in Cryptology (CRYPTO '88),* pp. 520-528, Aug. 1988.

[28] M. Steiner, G. Tsudik, and M. Waidner, "Cliques: A New Approach to Group Key Agreement," *IEEE Trans. Parallel and Distributed Systems,* Aug. 2000.

[29] W.-G. Tzeng and Z.-J. Tzeng, "Round-Efficient Conference-Key Agreement Protocols with Provable Security," *Proc. Advances in Cryptology (ASIACRYPT 2000),* Dec. 2000.

[30] C. Wong, M. Gouda, and S. Lam, "Secure Group Communications Using Key Graphs," *Proc. ACM SIGCOMM '98 Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm.,* pp. 68-79, 1998, *ACM SIGCOMM Computer Comm. Rev.,* vol. 28, no. 4, Oct. 1998.



Yongdae Kim received the PhD degree from the Computer Science Department at the University of Southern California (USC) in May 2002. He is an assistant professor in the Department of Computer Science and Engineering and a member of the Digital Technology Center (DTC) at the University of Minnesota, Twin Cities, Minneapolis. From 1993 to 1998, he was a research staff member in the Electronics and Telecommunication Research Institute (ETRI), Korea. From January 2001 until July 2002, he worked as a research scientist at the University of California at Irvine. His academic interests include network security and cryptography. He is a member of the IEEE. More information about his research is available at http://www.cs.umn.edu/~kyd.

Adrian Perrig earned the PhD degree in computer science from Carnegie Mellon University and spent three years during his PhD studies at the University of California at Berkeley. He received the BS degree in computer engineering from the Swiss Federal Institute of Technology in Lausanne (EPFL). He is an assistant professor in electrical and computer engineering, engineering and public policy, and computer science at Carnegie Mellon University. His research interests revolve around building secure systems and include network security, security for sensor networks and mobile applications. He is a member of the IEEE. More information about his research is available at: http://www.ece.cmu.edu/~adrian.

Gene Tsudik received the PhD degree in computer science from the University of Southern California (USC) in 1991; his dissertation focused on access control in internetworks. He is a professor in the Computer Science Department at the University of California, Irvine (UCI). He has been active in the area of internetworking, network security, and applied cryptography since 1987. Before coming to UCI in 2000, he was a project leader at the IBM Research, Zurich Laboratory (1991-1996) and USC Information Science Institute (1996-2000). Over the years, his research interests included: internetwork routing, firewalls, authentication, mobile network security, secure e-commerce, anonymity, secure group communication, digital signatures, key management, ad hoc network routing, and, more recently, database privacy and secure storage. Some of his notable research contributions include: Inter-Domain Policy Routing (IDPR), IBM Network Security Program (KryptoKnight), IBM Internet Keyed Payment (iKP) protocols, Peer Group Key Management (CLIQUES), and Mediated Cryptographic Services (SUCSES). He has published more than 80 refereed publications and seven patents. He is currently serving as associate dean of research and graduate studies in the School of Information and Computer Science at UCI. He is a member of the IEEE.