Group Key Agreement Efficient in Communication ⋆

Yongdae Kim[1], Adrian Perrig[2], and Gene Tsudik[3]⋆⋆

[1] Computer Science and Engineering Department
University of Minnesota, Twin Cities, MN, USA.
kyd@cs.umn.edu
[2] Electrical and Computer Engineering Department
Carnegie Mellon University, Pittsburgh, PA, USA.
perrig@cmu.edu
[3] Department of Information and Computer Science,
University of California at Irvine, CA, USA
gts@ics.uci.edu

Abstract. In recent years, group-oriented applications and protocols have been gaining popularity. Such applications typically involve communication over open networks where security is an important concern. 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 due mostly to expensive cryptographic operations. Communication cost has been treated as a secondary concern. This has been (and perhaps still is) a reasonable strategy, however, certain changes are looming on the horizon.

In particular, recent dramatic advances in computing power motivate a priority shift. As computation becomes faster, communication speeds do not enjoy similar advances and communication latency, especially in high-delay long-haul networks increasingly dominates protocol costs, replacing in some cases computation as the main latency factor. Hence, there is a need to minimize the number of messages, their size and the number of rounds in cryptographic protocols.

Since most previously proposed group key management techniques optimize cryptographic overhead, they are particularly impacted by high communication delays. 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. Our technique is communication-efficient and provably secure against hostile attacks. Furthermore, it is simple, fault-tolerant and well-suited for high-delay networks.

⋆ An early version of this paper has appeared, in part, in [19].
⋆⋆ Contact author.

1 Introduction

Security is crucial for many currently popular distributed and collaborative applications and protocols that operate in a dynamic network environment and communicate over insecure networks, e.g., the global Internet. Basic security services needed in such dynamic peer group settings are largely the same as in point-to-point communication: data secrecy, data integrity and entity authentication. These services cannot be obtained without secure and efficient group key management techniques.

Most prior research in group key management has been mainly concerned with increasing the security while minimizing cryptographic computation costs. 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 continuously improving computational power of modern PCs and workstations directly translates into more efficient cryptographic operations. More specifically, rapid advances in computing have resulted in drastic improvements in large integer arithmetic, which is the most important overhead factor in public key cryptography. For example, only four years ago, a top-of-the-line RISC workstation performed a 512-bit modular exponentiation in around 24 ms. Now, four years later, a 1.2 GHz Pentium III PC (priced at $1/5$-th of the old RISC workstation) performs the same operation well 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 both the computing power and the communication bandwidth are increasing, the communication delay retains a 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, inter-planetary networking is not too far off in the future. Consider, for instance, communication delay with a Mars Rover or some other space exploration device.

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.

In this paper, we focus on group key management and consider a technique initially proposed by Steer, et al. in 1988 [26]. It essentially extends the well-known 2-party Diffie-Hellman key exchange and provides for the formation of a secure but **static** group. This method (actually, a protocol) is hereafter referred to as STR; short for **S**kinny **TR**ee. We extend it to obtain provable security and to cope with dynamic groups and network failures in a communication-efficient manner. More concretely, we construct a provably secure group key management suite which is particularly efficient, and even optimal, in some respects, in a high-delay WAN environment. We support our claims by contrasting STR with other contributory group key management techniques.

The remainder of this paper is organized as follows. Section 2 presents our assumptions, requirements for the reliable group communication system and necessary cryptographic features of group key agreement. Notation is introduced in Section 3 and the actual protocols are described in Section 4. Section 5 discusses the security, complexity, and implementation issues. The performance of STR is evaluated in Section 6 and a summary of related work appears in Section 7. The paper concludes with a brief recap in Section 8 and detailed security arguments are provided in Appendix A.

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

multi-party and multi-round 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) [21] 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

Since we assume a reliable group communication platform, network delay is amplified by the necessary acknowledgments between the group members. The speed-of-light imposes 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 these lower bounds.

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 cryptographic primitives) in under 1 ms. Moreover, the speed of computation continue to increase. In the long run, it seems natural that, for group key management reducing the number of communication rounds is more important than reducing the computation overhead.

2.3 Group Key Agreement

A comprehensive group key agreement method must handle adjustments to group secrets subsequent to all membership change operations in the underlying group communication system. Membership changes can involve single or multiple members. Single member changes include member join or leave, and multiple member changes include group merge and group partition. They can also be additive or subtractive in nature: join and merge are additive, while partition and leave are subtractive.

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 sub-groups.
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 pre-existing groups into a single group. (The case of simultaneous multiple-member addition is not covered.)

At the 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 [21]. 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 Section A. In other words, 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 messages in our protocol include:

- sender information: name of the sender, or, equivalently, signer
- group information: unique name of the group
- membership information: the name and some information of the current group members
- protocol identifier: the name of the protocol being used (fixed here for STR)
- message type: unique message identifier for each protocol message
- key epoch: strictly increasing number. Whenever new membership event happens, every member increases epoch by 1. For example, when two groups G_1 and G_2 merge, the resulting epoch is one greater than the maximum of two epochs in each group. In other words, $epoch_{new} = 1 + Max(epoch_{G_1}, epoch_{G_2})$. Epoch is synchronized over current group members. If a current group member sends a message with smaller epoch, each group key agreement protocol terminates since the sender has smaller epoch that means this message can be replayed.
- time stamp: current time. We assume loose time synchronization among group members.

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 entities in our group key agreement protocols. 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 and Basic Protocol Features

The notation used in this paper is shown on the left side of Figure 1. The right side of Figure 1 illustrates an example of the STR key tree. Each leaf node is associated with a specific group member. An internal node $\mathsf{IN}_{\langle i \rangle}$ always has two children: another (lower) internal node $\mathsf{IN}_{\langle i-1 \rangle}$ and a leaf node $\mathsf{LN}_{\langle i+1 \rangle}$. The exception is $\mathsf{IN}_{\langle 1 \rangle}$ which is also a leaf node corresponding to M_1. (Note that, consequently, $r_1 = k_1$.)

Each leaf node $\mathsf{LN}_{\langle i \rangle}$ maintains its own group key contribution, called a *session random* (r_i), which it generates and keeps secret. The blinded version thereof is: $br_i = \alpha^{r_i} \bmod p$. Each internal (non-leaf) node in key tree $\mathsf{IN}_{\langle j \rangle}$ 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. It is computed recursively as:

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

The group (root) key 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 key is never used directly for the purposes of encryption, authentication or integrity. Instead, such special-purpose sub-keys 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

[1] Furthermore, as discussed above, all protocol messages are assumed to contain: 1) sender/group information, 2) a protocol 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. $\alpha^{r_i} \mod p$
k_j	secret key shared among M_1...M_j
bk_j	blinded k_j, i.e. $\alpha^{k_j} \mod p$
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 view of M_i
$BT_{\langle i \rangle}$	key tree view of M_i including all blinded keys

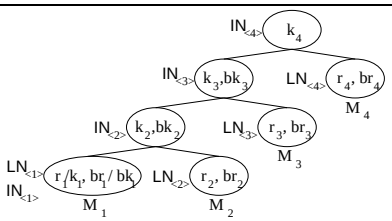


Fig. 1. Notation and STR key tree example.

corresponding to $\mathsf{IN}_{\langle 2 \rangle}$. M_1 computes:

$$k_2 = (br_2)^{r_1} \bmod p = \alpha^{r_1 r_2} \bmod p, bk_2 = \alpha^{k_2} \bmod p \tag{2}$$

$$k_3 = (br_3)^{k_2} \bmod p \qquad bk_3 = \alpha^{k_3} \bmod p \tag{3}$$

$$\ldots = \ldots \tag{4}$$

$$k_N = (br_N)^{k_{N-1}} \bmod p \tag{5}$$

Next, M_1 broadcasts all bkeys bk_i with $1 \leq i \leq N-1$. 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: $k_i = {br_i}^{k_{i-1}} \bmod p$ ($i \leq N$).

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 2. *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 3. *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.*

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.
- 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 who changes its session random 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. 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 $BT_{\langle n \rangle}$ to M_{n+1} with all bkeys.

Step 1: New member broadcasts join request

M_{n+1} $\xrightarrow{br_{n+1}=\alpha^{r_{n+1}}}$ $\mathcal{C} = \{M_1, \ldots, M_n\}$

Step 2: Each member
- updates key tree by adding new member node and new root node,
- removes bk_n,

sponsor M_n additionally
- generates new share r_n and computes br_n, k_n, bk_n
- broadcasts updated tree $BT_{\langle n \rangle}$

$\mathcal{C} \cup \{M_{n+1}\} = \{M_1, \ldots, M_{n+1}\}$ $\xleftarrow{BT_{\langle n \rangle}}$ M_n

Step 3: Each member computes group key using $BT_{\langle n \rangle}$

a. Join Protocol

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 $BT_{\langle s \rangle}$

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

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

b. Leave Protocol

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: Each 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: Each member computes group key using $BT_{\langle s \rangle}$

c. Merge Protocol

Fig. 2. STR Protocol Suite

Next, each member M_i increments $n = n + 1$ and creates a new root node $\mathsf{IN}_{\langle n \rangle}$ with two children: 1) root node $\mathsf{IN}_{\langle n-1 \rangle}$ of the prior tree T_i on the left and 2) new leaf node $\mathsf{LN}_{\langle n \rangle}$ corresponding to the new member on the right. Note that every member can compute the group key (see Proposition 3) 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. Figure 3a shows an example of a new member M_5 joining a group. The sponsor M_4 updates its session random ${r_4}^2$.

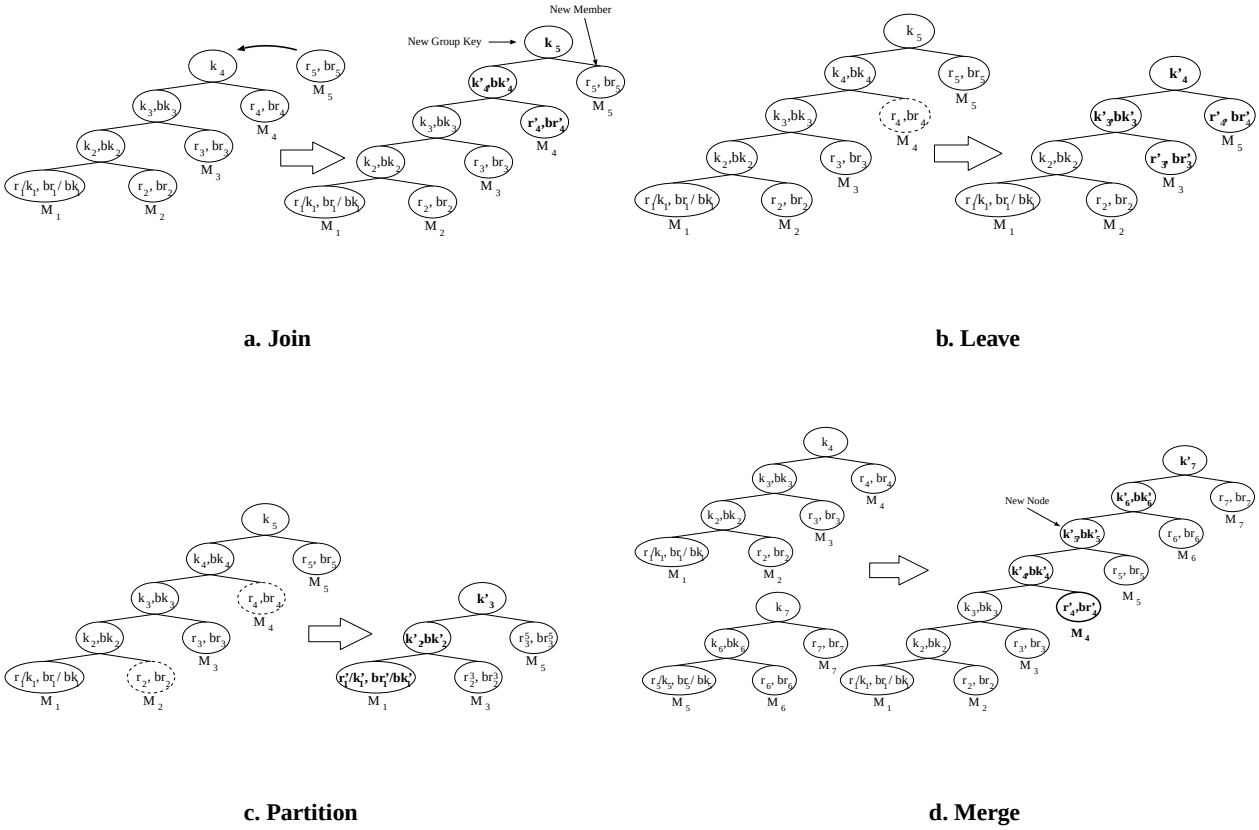
As described, JOIN takes two communication rounds and five cryptographic operations to compute the new group key (four by the sponsor and two by everyone else.) As will be discussed in Section 5.1.2, the JOIN protocol provides backward and forward secrecy.

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 $\mathsf{LN}_{\langle d \rangle}$ corresponding to M_d and its parent node $\mathsf{IN}_{\langle d \rangle}$. The nodes above the leaving node are also renumbered. The former sibling $\mathsf{IN}_{\langle d-1 \rangle}$ 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 $BT_{\langle s \rangle}$ to the group. This information allows all members (including the sponsor) to recompute the new group key. Figure 4.1b describes the leave protocol in detail.

Figure 3b shows what happens when 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 $BT_{\langle 4 \rangle}$. Upon receiving the broadcast, all members

[2] To provide forward secrecy



a. Join **b. Leave**

c. Partition **d. Merge**

Fig. 3. Tree Update Examples

(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 part thereof[3].

The 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 is $(3n - 3d + 2)$ if $d > 2$ and $(3n - 7)$ if $d \leq 2$. The latter is the worst case when M_1, M_2 or M_3 leaves the group. However, the expected leave cost (based on $d = n/2$) is $3(n/2) + 2$.

Leave 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 appears as a concurrent leave of multiple members. With a minor modification, Leave 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 $BT_{\langle s \rangle}$ containing only blinded values. Each member (including M_s) can now compute the group key.

Figure 3c 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

[3] r_5 and br_5 are renumbered, and are denoted as $r_4{}'$ and $br_4{}'$, respectively.

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 Figure 4.1c.

In the first round of the merge protocol, all sponsors (members associated with topmost leaf node in each tree) exchange their respective key trees containing all **blinded session randoms**[4]. 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.

[4] Bkeys do not need to be exchanged this time.

Figure 3d 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 large tree with four members. Every member generates a new intermediate node $\mathsf{IN}_{\langle 5\rangle}$ and makes it the parent of the old root node $\mathsf{IN}_{\langle 4\rangle}$ of the larger tree and the previous leftmost leaf node $\mathsf{LN}_{\langle 5\rangle}$. Both intermediate nodes $\mathsf{IN}_{\langle 1\rangle}$ and $\mathsf{IN}_{\langle 2\rangle}$ of the previous smaller tree then need to be renumbered as $\mathsf{IN}_{\langle 6\rangle}$ and $\mathsf{IN}_{\langle 7\rangle}$, respectively. The new intermediate node $\mathsf{IN}_{\langle 5\rangle}$ also becomes the child of the previous lowest intermediate node $\mathsf{IN}_{\langle 6\rangle}$. Using the previous blinded group key at $\mathsf{IN}_{\langle 4\rangle}$ 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 $BT_{\langle 4\rangle}$ that contains all bkeys and blinded session randoms up to $\mathsf{IN}_{\langle 6\rangle}$[5]. 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 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[6]. The probability that new group key is the same as any old group key is negligible due to bijectiveness of $(f \circ g)$ function.

[5] In fact, it need not broadcast unchanged bkeys, $\{bk_1, bk_2, bk_3\}$.

[6] Recall that insider attacks are not our concern. This excludes the case when an insider intentionally generates non-random numbers.

We note that the root (group) key is never used directly for the purposes of encryption, authentication or integrity. Instead, special-purpose sub-keys 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 sub-keys are derived from a group key. Decisional group key secrecy of STR protocol is related to imbalanced tree decision Diffie-Hellman assumption mentioned in Section A.2. This assumption ensures that there is no information leakage other than public bkey information. We can also derive sub-keys based on the Shoup's hedge technique [25] and compute them 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 11.

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, 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 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 KKA.

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 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 similar 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. Figure 4a 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 take into account the occurence of so-called *cascaded membership events*. 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 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.

```
 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
```

```
15    receive msg           \* this line replaces line 15 above *\
16    if (msg type = membership event)
17      construct new tree
18    endif
19  endwhile
```

Fig. 4. Unified protocol pseudocode (top) and self-stabilizing additions (bottom)

As discussed before, STR protocol requires at most two rounds. One might wonder why robustness against cascaded failure is important for 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 $\mathcal{G}_2$ to 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 STR protocols as described above cannot cope with such situations. However, we can easily modify the unified STR protocol to handle cascaded events. The pseudocode additions for the self-stabilizing protocol are shown in Figure 4b. The changes from Figure 4a are minimal.

We can now claim that STR is self-stabilizing, i.e., robust against cascaded network events. This feature is quite rare since most multi-round cryptographic protocols are not equipped to handle 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 re-entrant).

6 Performance Analysis and Communication Effi ciency

This section assesses the computational and communication overhead costs incurred by STR protocols.

6.1 Performance Comparison

We analyze both communication and computation costs for join, leave, merge and partition protocols. In doing so, we focus on the number of: rounds, messages, serial exponentiations, signature

generations, and signature verifications. Note that we use RSA signatures for message authentication since RSA is particularly efficient in verification. 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 that has been known to be most efficient in both communication and computation. For detailed comparison with other group key agreement protocols such as GDH.3 [27], BD (Burmester-Desmedt) [11] can be found at [2].

Table 6.1 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 $\frac{n}{2}$-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 computation cost of STR is fairly high: $O(m)$ for merge and $O(n)$ for subtractive events. However, 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].

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

Table 1. Communication and Computation Cost Summary

6.2 Communication Lower Bounds

In [5], Becker and Wille proved the lower bound for communication complexity of static group key agreement (where the goal is for n group members to share a key without considering subsequent membership changes). Assuming a broadcast channel, they prove the following theorem:

Theorem 4 (Becker and Wille). *Let $\mathcal{P}$ be a static group key agreement protocol for n parties allowing broadcasts.*

1. *If $\mathsf{m}(\mathcal{P})$ is the number of messages required by $\mathcal{P}$, then: $\mathsf{m}(\mathcal{P}) \geq n$*
2. *If $\mathsf{r}(\mathcal{P})$ is the number of rounds required by $\mathcal{P}$, then: $\mathsf{r}(\mathcal{P}) \geq 1$*

However, it is commonly assumed that at least 2 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 as above, $\mathsf{r}(\mathcal{P}) \geq 2$*

Indeed, secure one-round group key agreement is a well-known open problem [8]. When the group size is 3, there exists a one-round group key agreement protocol (due to Joux) based on a Bi-linear map using Weil parings [17]. The same result also shows that a one-round group key agreement protocol for any n can be obtained if a multi-linear (n-linear) map exists. Unfortunately, the existence of a multi-linear map is quite uncertain [8].

Based on Theorem 4 and Assumption 1, we can easily come up with the lower bound for communication complexity of dynamic group key agreement.

Theorem 5 (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 event $\mathsf{r}(\mathcal{P}) \geq 1$ and $\mathsf{m}(\mathcal{P}) \geq 1$, when the number of remaining group members is greater than 2.*
- *For any additive event $\mathsf{r}(\mathcal{P}) \geq 2$ and $\mathsf{m}(\mathcal{P}) \geq k$, when k groups are merging.*[7]

Proof (Sketch). In contributory group key agreement, a group key is determined by the contributions of participating members. To provide key independence, each group key should be unrelated

[7] of course, this includes join of a single member. In this case, $k = 2$.

to both previous and future group keys. In other words, for each membership change event, at least one member in the group has to change its contribution. Therefore, at least one message (and one round) is required to let others know about the change. This provides a communication lower bound for all events: $\mathsf{r}(\mathcal{P}) \geq 1$ and $\mathsf{m}(\mathcal{P}) \geq 1$

This bound can be tightened for additive events. We consider only a merge event of k groups, since join is a special case of a merge. It is important to note that a merge of k groups can be seen as static group key agreement of k members. If so, the STR lower bound for additive events is same as those for static group key agreement captured by Theorem 4 and Assumption 1.

It remains to show that a merge of k groups is equivalent to the static group key agreement of k members. Since there are k distinct groups, at least k messages need to be exchanged to communicate each group's (updated) information. In fact, a merge of k groups can be seen as a group formation with k members where each "member's" contribution is its current group key sk and the corresponding blinded key is $g^{sk} \pmod p$ where the blinded key is never revealed to other group members. Therefore, the communication lower bound for additive events is the same as that of static group key agreement. Consequently, any additive event requires: $\mathsf{r}(\mathcal{P}) \geq 2$ and $\mathsf{m}(\mathcal{P}) \geq k$.

□

From Theorem 5, it follows that the communication costs of STR are near optimal. For additive events it requires one more message than the optimal number. However, it can be easily modified to achieve optimal communication efficiency: when a merge event occurs, one of the merging groups is chosen unambiguously, e.g., the group that has the highest lexicographically named member. All sponsors in other groups send their key tree information to this chosen group. This takes $(k-1)$ messages. Upon receiving these messages, the sponsor in the chosen group computes all required blinded keys, and broadcast the new key tree. This takes one message resulting in k messages total. Finally, all members compute the group key.

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 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 earlist 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 fist 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: $K_n = \alpha^{N_1 N_2 N_3 ... N_n}$

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 Treee-based Group Diffie-Hellman) is of 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 re-compute the group key for any membership change by performing this protocol. However, according to [27], 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: $K_n = \alpha^{N_1 N_2 + N_2 N_3 + ... + N_n N_1}$

A notable shortcoming of BD is the high communication overhead. It requires $2n$ broadcast messages and each member needs to generate 2 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, in a group of eight, the key is: $K_n = \alpha^{(\alpha^{\alpha^{r_1 r_2} \alpha^{r_3 r_4}})(\alpha^{\alpha^{r_5 r_6} \alpha^{r_7 r_8}})}$.

The Becker and Wille protocols handle join and merge operations efficiently, but the 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, et al. 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 multi-party computation [28]. 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, 27] in group key agreement and propose a family of Group Diffie Hellman (GDH) protocols based on straight-forward 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 [22]. 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 inexpensive symmetric key encryption technique. However, all group key distribution schemes assume 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, 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.

First decentralized group key distribution scheme is due to Waldvogel 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 same key tree structure and uses similar notations (e.g. keys, blinded keys). Other than expensive maintenance 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. [23] propose a decentralized group key distribution protocol extended from LKH protocol [29]. It tolerates network partitions and other network events. Even though this approach cannot help incurring basic disadvantages discussed above, authors reduce the communication and computational cost. In addition, authors use AVL tree to provide provable and efficient tree height.

8 Conclusion

In this paper we described STR, a provably secure contributory group key agreement protocol optimized for communication. Assuming that Moore's Law continues to hold, the computation cost of cryptographic operations will continue to 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 network, and its advantage will improve as the speed of processors increases. STR supports all dynamic group operations: join, leave, merge, and partition. Furthermore, it easily handles cascaded network failures.

References

1. Y. Amir, et al. Secure group communication in asynchronous networks with failures: Integration and experiments. In *IEEE International Conference on Distributed Computing Systems*, April 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, Johns Hopkins University, *http://www.cnds.jhu.edu/pub/papers/cnds-2001-5.pdf*, 2001.
3. N. Asokan and P. Ginzboorg. Key-agreement in ad-hoc networks. In *Nordsec'99*, 1999.
4. G. Ateniese, M. Steiner, and G. Tsudik. Authenticated Group Key Agreement and Friends. In *5th ACM Conference on Computer and Communications Security*, Nov. 1998.
5. C. Becker and U. Wille. Communication complexity of group key distribution. In *5th ACM Conference on Computer and Communications Security*, Nov. 1998.
6. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, Nov. 1993.
7. D. Boneh. The Decision Diffie-Hellman problem. In *3rd Algorithmic Number Theory Symposium*, LNCS 1423, pages 48–63. Springer-Verlag, 1998.
8. D. Boneh and A. Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics, American Mathematical Society*, 2002.
9. E. Bresson, O. Chevassut, and D. Pointcheval. Provably authenticated group Diffie-Hellman key exchange — the dynamic case. In *IACR ASIACRYPT 2001*, LNCS, 2001.
10. E. Bresson, O.. Chevassut, D. Pointcheval, and J. Quisquater. Provably authenticated group diffie-hellman key exchange. In *8th ACM Conference on Computer and Communications Security*, Nov. 2001.
11. M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *IACR EUROCRYPT 1994*, LNCS 950, 1995. final version of proceedings.
12. G. Caronni, et al. The VersaKey framework: Versatile group key management. *IEEE Journal on Selected Areas in Communications*, 17(9), Sept. 1999.
13. D. Chaum. Zero-knowledge undeniable signatures. In *IACR EUROCRYPT 1990*, LNCS 473, May 1991.
14. L. Dondeti, S. Mukherjee, and A.Samal. Disec: A distributed framework for scalable secure many-to-many communication. In *5th IEEE Symposium on Computers and Communications*, July 2000.
15. A. Fekete, N. Lynch, and A. Shvartsman. Specifying and using a partionable group communication service. In *16th ACM Symposium on Principles of Distributed Computing*, Aug. 1997.
16. I. Ingemarsson, D. Tang, and C. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, 28(5):714–720, Sept. 1982.
17. A. Joux. A one-round protocol for tripartite Diffie-Hellman. In *4th International Algorithmic Number Theory Symposium*, LNCS 1838, Springer-Verlag.
18. Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *7th ACM Conference on Computer and Communications Security*, Nov. 2000.
19. Y. Kim, A. Perrig, and G. Tsudik. Communication-efficient group key agreement. In *17th International Information Security Conference (IFIP SEC'01)*, June 2001.

20. D. McGrew and A. Sherman. Key establishment in large dynamic groups using one-way function trees. Manuscript, May 1998.
21. L. Moser, Y. Amir, P. Melliar-Smith, and D. Agarwal. Extended virtual synchrony. In *IEEE International Conference on Distributed Computing Systems*, June 1994.
22. A. Perrig. Efficient collaborative key management protocols for secure autonomous group communication. In *CrypTEC '99*, pages 192–202, 1999.
23. O. Rodeh, K. Birman, and D. Dolev. Optimized rekey for group communication systems. In *Symposium on Network and Distributed Systems Security*, Feb. 2000.
24. V. Shoup. Lower bounds for discrete logarithms and related problems. In *IACR EUROCRYPT 1997*, LNCS 1233, Springer-Verlag, 1997.
25. Victor Shoup. Using hash functions as a hedge against chosen ciphertext attacks. In *IACR EUROCRYPT 2000*, LNCS 1807, Springer-Verlag, 2000.
26. D. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A secure audio teleconference system. In *IACR CRYPTO 1988*, LNCS 403, Springer Verlag, Aug. 1988.
27. M. Steiner, G. Tsudik, and M. Waidner. Cliques: A new approach to group key agreement. *IEEE Transactions on Parallel and Distributed Systems*, Aug. 2000.
28. W. Tzeng and Z. Tzeng. Round-efficient conference-key agreement protocols with provable security. In *IACR ASIACRYPT 2000*, LNCS 727, Springer-Verlag, Dec. 2000.
29. C. Wong, M. Gouda, and S. Lam. Secure group communications using key graphs. In *ACM SIGCOMM'98*, Appeared in ACM SIGCOMM Computer Communication Review, 28(4), Oct. 1998.

A Decisional Imbalanced Group Diffi e-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 algorithm 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 [8], 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. This 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: $\widehat{G} = \{\alpha^i \bmod p \mid i \in [1, q]\}$.

An attractive variation of this group is the representation of 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 $f(x)$, we can define the group G as: $G = \{f(\alpha^i \bmod p) \mid i \in \mathbb{Z}_q\}$. The group operation is then defined as $a \cdot b = f(a \cdot b \pmod p)$, where $a, b \in G$.

Proposition 6. *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 $X = Y$. Rests are the same as above.

[8] In order to achieve the security level 2^{-k}, the group size should be at least 2^{2k} [24].

– $X \leq q, Y > q$**:** This case is impossible, since $\left(\frac{X}{p}\right) = 1$ and $\left(\frac{p-Y}{p}\right) = -1$ 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 domain and co-domain are the same.

Proposition 7. *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 such 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 by formulating the problem underlying the security of the STR protocols. We refer to it as the *Decisional Imbalanced Tree Group Diffie-Hellman Problem* or DITGDH.

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

– K_i: i-th level key

– BK_i: i-th level blinded key, i.e. $\alpha^{K_i} \bmod p$

– R_i: i-th 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: i-th level blinded session random, i.e. $\alpha^{R_i} \bmod p$. For $i = 1$, $BR_i = BK_i$.

– K_i is (recursively) defined as:
$K_i = \alpha^{K_{i-1}R_i} = BK_{i-1}^{R_i} = BR_i^{K_{i-1}}$

K_i and R_i are secret, and BK_i and BR_i are public. In Section 4, BK_i and BR_i's are public while K_i is known only to group members, R_i is known only to a single member M_i and the root node K_i denotes the 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:

$$view(q, \alpha, n, X.IT) := \{BK_i \mid 1 \leq i \leq n\} \cup \{BR_i \mid 1 \leq i \leq n\} \tag{6}$$

$$= \{\alpha^{K(i,X,IT)} \bmod p \mid 1 \leq i \leq n-1\} \cup \{BR_i \mid 1 \leq i \leq n\} \tag{7}$$

$$= \{\alpha^{\alpha^{R_1 R_2}}, \alpha^{\alpha^{R_3 \alpha^{R_1 R_2}}}, \ldots, \alpha^{\alpha^{R_{n-1} \ldots \alpha^{R_1 R_2}}}\} \cup \{\alpha^{R_1}, \alpha^{R_2}, \ldots, \alpha^{R_n}\} \tag{8}$$

$$K(q, \alpha, n, X, IT) := \alpha^{K_{n-1} R_n} \tag{9}$$

Since (q, α) are obvious from the context, we omit them in *view*() and *K*(). Also, for simplicity, we use K_n instead of $K(n, X, IT)$. The notation $view(n, X, IT)$ 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:

$$A_n := (view(n, X, IT), y) \quad \text{and} \quad D_n := (view(n, X, IT), K_n)$$

The operator "$\approx_{poly}$" denotes polynomial indistinguishability as in [27].

Proposition 8. *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 9. *If 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:

$$A_n := (view(n, X, IT), y) = (view(n-1, X_1, IT_1), BK_{n-1}, BR_n, y) \quad (10)$$

$$= (view(n-1, X_1, IT_1), \alpha^{K(n-1,X_1)}, \alpha^{R_n}, y) \quad (11)$$

$$B_n := (view(n-1, X_1, IT_1), \alpha^{r}, \alpha^{R_n}, y) \quad (12)$$

$$C_n := (view(n-1, X_1, IT_1), \alpha^{r}, \alpha^{R_n}, \alpha^{rR_n}) \quad (13)$$

$$D_n := (view(n, X, IT), K(n, X, IT)) = (view(n-1, X_1, IT_1), BK_{n-1}, BR_n, \alpha^{K(n-1,X_1,IT_1)R_n}) \quad (14)$$

$$= (view(n-1, X_1, IT_1), \alpha^{K(n-1,X_1,IT_1)}, \alpha^{R_n}, \alpha^{K(n-1,X_1,IT_1)R_n}) \quad (15)$$

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 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 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 7, 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 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 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 7.

Lemma 10. *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 assumption using $\mathcal{A}_{AB_3}$ or $\mathcal{A}_{CD_3}$.

Now, using induction and Lemmas 9 and 10, the following theorem can be easily proved.

Theorem 11 (DITGDH problem). *If the 2-party DDH problem is hard, then DITGDH is also hard.*