On the Performance of Group Key Agreement Protocols *

Yair Amir † Yongdae Kim ‡ Cristina Nita-Rotaru † Gene Tsudik ‡

Abstract

Group key agreement is a fundamental building block for secure peer group communication systems. Several group key agreement protocols were proposed in the last decade, all of them assuming the existence of an underlying group communication infrastructure.

This paper presents a performance evaluation of five notable key agreement protocols for peer groups, integrated with a reliable group communication system (Spread). They are: Centralized Group Key Distribution (CKD), Burmester-Desmedt (BD), Steer et al. (STR), Group Diffie-Hellman (GDH) and Tree-Based Group Diffie-Hellman (TGDH). The paper includes an in-depth comparison and analysis of conceptual results and is the first to report practical results in real-life local and wide area networks. Our analysis of these protocols' experimental results offers insights into their scalability and practicality.

Keywords: contributory group key agreement, secure communication, peer groups, group communication.

*This work was supported by grant F30602-00-2-0526 from The Defense Advanced Research Projects Agency.

†Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218, USA. Email: {yairamir, crisn}@cs.jhu.edu

‡Information and Computer Science Department, University of California, Irvine Irvine, CA 92697-3425, USA. Email: {kyongdae, gts}@ics.uci.edu

1 Introduction

The Internet is being increasingly used to support collaborative applications such as voice- and video-conferencing, white-boards, distributed simulations, as well as games, replicated servers and databases of all types. To be effective, these applications need supporting services, such as reliable and ordered message delivery as well as synchronization and fault-tolerance techniques. A reliable group communication system can provide an integrated platform containing such services, thus greatly simplifying the application development process and application complexity.

Since most communication over the Internet involves the traversal of insecure networks, basic security services – such as data secrecy, data integrity and entity authentication – are necessary for collaborative applications. These security services can be facilitated if group members share a common secret, which, in turn, makes group key management a fundamental service and a design challenge in secure and reliable group communication systems.

1.1 Key Agreement in Peer Groups

Several group key management approaches have been proposed in the last decade. (We stress that these are distinct from *multicast* key management which aims to minimize costs of key dissemination and re-keying in large, single-source multicast groups.) These approaches generally fall into three categories: 1) centralized, 2) distributed and 3) contributory.

Centralized group key management is conceptually simple as it involves a single entity (or a small set of entities) that generates and distributes keys to group members. We claim that centralized group key management is not appropriate for peer group communication since the central key server must be, at the same time, continuously available and present in every possible subset of a group in order to support continued operation in the event of arbitrary network partitions. Continuous availability can be addressed with fault-tolerance and replication techniques. Unfortunately, the omni-presence issue is impossible to solve in a scalable and efficient manner.

Distributed group key management is more suitable to peer group communication, especially over unreliable networks. It involves dynamically selecting a group member that acts as a key server. Although robust, this approach has a notable drawback in that it requires the key server to maintain long-term pairwise secure channels with all current group members in order to distribute group keys. Consequently, each time a new key server comes into play, significant costs must be incurred to set up these channels.

In contrast, *contributory group key agreement* requires each group member to contribute an equal share to the common group key (computed as a function of all members' contributions). This approach avoids the problems with the single points of trust and failure. Moreover, some contributory methods do not require the establishment of pairwise secret channels among group members.

As can be expected, the cost of group key management protocols is largely determined by two dominating factors: communication and computation. Typically, efficiency in one comes at the expense of the other. Protocols that distribute computation usually require more communication rounds or messages, whereas, protocols minimizing communication require more computational effort.

1.2 Focus

In recent previous work [1, 2] we demonstrated how provably secure, multi-round group key agreement protocols can be integrated with a reliable group communication system to obtain provably **fault-tolerant** group key agreement solutions. Specifically, we designed a robust contributory key agreement protocol resilient to any sequence of (possibly cascaded) events and prove that the resulting protocol preserved group communication membership and ordering guarantees. The protocol was based on Group Diffie-Hellman (GDH IKA.3) contributory key agreement [3] that generalizes the two-party Diffie-Hellman [4] key exchange.

In this paper we focus on the performance of key agreement protocols. To do so, we identified several notable protocols and integrated them with a reliable group communication system (Spread). Centralized Key Distribution (CKD) is a centralized key distribution scheme where one member of the group is dynamically chosen to act as a key server; Tree Based Group Diffie-Hellman (TGDH) combines a tree structure with the Diffie-Hellman key exchange algorithm to achieve a more computationally efficient protocol than GDH; STR trades off less communication for increased computation; Burmester-Desmedt (BD) distributes/minimizes computation by using more messages. All protocols provide security properties similar to those of GDH IKA.3.

The main contributions of this work are:

- Group key agreement framework that supports multiple protocols. This allows the system to assign different key agreement protocols to different groups.

- Detailed comparison of the conceptual performance of the five key agreement protocols with respect to communication and computation costs.

- In-depth analysis of practical results obtained in real-life experiments over both local and wide area networks. These results provide insights into the protocols' scalability and practicality.

The rest of the paper is organized as follows. Section 2 overviews related work. We then present Secure Spread, a secure group communication system and give a brief description of the key agreement protocols it supports in Section 4. Next, we analyze the conceptual costs of these protocols and present performance results. Finally, we summarize our work and identify possible directions for future work.

2 Related Work

In this section we summarize related work in two areas: group key management and reliable group communication.

2.1 Group Key Management

As indicated above, the focus of this work is on the performance of group key management protocols for collaborative, peer groups. Therefore, we consider only distributed key distribution and contributory key agreement protocols.

In looking at the available protocols, we are concerned mostly with the cost (performance) of the types of group key management operations that occur most often. It might seem, at the first glance, that a typical collaborative group scenario is: a group forms, exists for some time and then dissolves itself. If this was true, we would only need to consider the performance of the initial key agreement leading to the group's formation. Moreover, performance would not be of great concern since the protocol would be run only once or very infrequently in order to re-key the group. However, a typical collaborative group is formed incrementally and its population can mutate throughout its lifetime either because members join and leave or because of network connectivity changes.

We begin with the protocol proposed by Steer et al. [5] aimed at teleconferencing. As will be seen later, it is well-suited for adding new members as it takes only two rounds and two modular exponentiations. Member exclusion (re-keying following a member leave event), however, is relatively inefficient. Burmester and Desmedt [6] proposed an efficient protocol which takes only two rounds and three modular exponentiations per member to generate a group key. This protocol allows all members to re-compute the group key for any membership change with a constant CPU cost. However, it requires $2n$ broadcast messages which is expensive on a wide area network. Tzeng and Tzeng proposed an authenticated key agreement scheme based on secure multi-party computation [7]. This protocol uses two communication rounds, but each round consists of n simultaneous broadcast messages. Although the cryptographic mechanisms are quite elegant, the main shortcoming is the lack of perfect forward secrecy (PFS).

Steiner et al. addressed dynamic membership issues [3] in group key agreement as part of developing a family of Group Diffie Hellman (GDH) protocols based on straight-forward extensions of the two-party Diffie-Hellman protocol. GDH protocols are relatively efficient for member leave and group partition operations, but the merge protocol requires the number of rounds equal to the number of new (merging) members. Follow-on work by Kim et al. yielded a tree-based Diffie-Hellman (TGDH) protocol which is more efficient than GDH in both communication and computation [8].

In contrast to contributory group key agreement schemes, only few distributed group key distribution scheme have been proposed [1, 9, 10].

Very little has been done in the performance analysis of peer group key management. One exception is the recent work by Carman, Kruus and Matt [11]. It compares – via simulation – different group key agreement protocols in ad hoc, sensor network setting based on their power consumption.

2.2 Reliable Group Communication

Reliable group communication systems in LANs have a solid history beginning with ISIS [12] and followed by more recent systems such as Transis [13], Horus [14], Totem [15], and RMP [16]. These systems explored several different models of Group Communication such as Virtual Synchrony [17] and Extended Virtual Synchrony [18]. More recent work in this area focuses on scaling group membership to wide-area networks [19, 20].

Research in securing group communication is fairly new. The only implementations of group communication systems that focus on security (aside from ours) are the SecureRing [21] project at UCSB, the Horus/Ensemble work at Cornell [22, 23, 24] and the RAMPART system at AT&T [25]. The SecureRing system protects a low-level ring protocol by using cryptographic techniques to authenticate each transmission of the token and each data message received.

The Ensemble security architecture is the state-of-the-art in secure reliable group communication and addresses problems such as group keys and re-keying. It also allows application-dependent trust models and optimizes certain aspects of group key generation and distribution protocols. In comparison with our approach, Ensemble uses a different group key structure that is not contributory and provides a different set of security guarantees. Recent research on Bimodal-Multicast, Gossip-based protocols [26] and the Spinglass system has largely focused on increasing the scalability and stability of reliable group communication services in hostile environments such as wide-area and lossy networks by providing probabilistic guarantees about delivery, reliability, and group membership.

Rampart [25] is the first system providing atomic and reliable services in a model where some servers can get corrupted. The system builds the group multicast protocols over a secure group membership protocol.

Antigone from UMich [27] is a framework aimed to provide mechanisms which allow flexible application security policies. Antigone addresses four policy aspects: rekeying policy (defines what events trigger a rekey), membership awareness policy (determines the availability and accuracy of the membership information), process failure policy (defines the type of failures supported by the system and when available, the means of recovery) and access control policy (specifies the rights and potentially the responsibilities of the group members). The system implements group rekeying mechanisms in two flavors: session rekeying - all group members receive a new key, and session key distribution where the session leader transmits an existing session key.

3 Secure Spread Framework

The work presented in this paper evolved from integrating security services with the Spread wide-area group communication system. Specifically, multiple key agreement protocols were integrated resulting in the Secure Spread library. As its building blocks, our implementation uses the key agreement primitives provided by the Cliques group key agreement library. In this section we overview the Spread group communication system, the Cliques toolkit and the Secure Spread library.

3.1 Spread Group Communication System

Spread [28, 29, 30] is a group communication system for wide and local area networks. It provides reliability and ordering of messages (FIFO, causal, total ordering) and a membership service. The toolkit supports two different semantics: Extended Virtual Synchrony [18, 31] and View Synchrony [32, 33]. In this paper, and for our implementation we use only the View Synchrony semantics of Spread.

The system consists of a daemon and a library linked with the application. This architecture has many benefits, the most important for wide-area settings being the ability to pay the minimum possible price for different causes of group membership changes. A simple join or leave of a process translates into a single message, while a daemon disconnection or connection requires a full membership change. The process and daemon memberships correspond to the model of light-weight and heavy-weight groups [34].

Spread operates in a many-to-many communication paradigm, each member of the group can be both a sender and a receiver. It is designed to support small to medium groups, but can accommodate a large number of different collaboration sessions, each of which spans the Internet. This is achieved by using unicast messages over the wide-area network and routing them between Spread nodes on the overlay network. Spread scales well with the number of groups used by the application without imposing any overhead on network routers. Group naming and addressing is not a shared resource (as in IP multicast addressing), but rather a large space of strings which is unique to a collaboration session.

The Spread toolkit is publicly available (www.spread.org) and is being used by several organizations in both research and practical settings. The toolkit supports cross-platform applications and has been ported to several Unix platforms as well as Windows and Java environments.

3.2 Cliques Library

Cliques is a cryptographic toolkit that implements a number of key agreement protocols for dynamic peer groups. The toolkit assumes the existence of a reliable communication platform that transports protocol messages and provides ordering of messages, deals with group membership management, and performs all computations required to achieve a shared key in a group. The current implementation is built atop the popular OpenSSL library.

Currently, Cliques includes five group key agreement protocols: GDH, CKD, TGDH, STR and BD. Each is briefly mentioned below and discussed in more detail in Section 4.

GDH is a protocol based on group extensions of the two-party Diffie-Hellman key exchange [3] and provides fully contributory authenticated key agreement. GDH is fairly computation-intensive requiring $O(n)$ cryptographic operations upon each key change. It is, however, bandwidth-efficient.

CKD is a centralized key distribution with the key server dynamically chosen from among the group members. The key server uses pairwise Diffie-Hellman key exchange to distribute keys. CKD is comparable to GDH in terms of both computation and bandwidth costs.

TGDH combines a binary key tree structure with the group Diffie-Hellman technique [8]. TGDH seems to be efficient in terms of computation as most membership changes require $O(\log n)$ cryptographic operations.

STR [35] is a form of TGDH with a so-called "skinny" or imbalanced tree. It is based on the protocol by Steer et al. [5]. STR is more efficient than the above protocols in terms of communication; whereas, its computation costs for subtractive group events are comparable to those of GDH and CKD.

BD is a protocol proposed by Burmester-Desmedt [6], another variation of group Diffie-Hellman. BD is computation-efficient requiring a constant number of exponentiations upon any membership (group key) change. However, communication costs are significant.

All the protocols in the Cliques library provide key independence and perfect forward secrecy (PFS). Informally, key independence means that a passive adversary who knows any proper subset of group keys cannot discover any other (future or previous) group key. PFS means that a compromise of a member's long-term key cannot lead to the compromise of any short-term group keys.

Only outside intruders (both passive and active) are considered in Cliques. In this model, any entity who is not a current group member is considered an outsider[1]. Attacks coming from the inside of the group are not considered, as our focus is on the secrecy of group keys and the integrity of the group membership. Consequently, insider attacks are not relevant in this context since a malicious insider can always reveal the group key and/or its own private key(s) thus allowing for fraudulent membership authentication.

All the above protocols were proven secure with respect to passive outside (eavesdropping) attacks [3, 8, 35, 6]. Active outsider attacks consist of injecting, deleting, delaying and modifying protocol messages. Some of these attacks aim to cause denial of service and we do not address them. Attacks with the goal of impersonating a group member are prevented by the use of public key signatures, since every protocol message is signed by its sender and verified by all receivers. Other, more subtle, active attacks aim to introduce a known (to the attacker) or old key. These are prevented by the combined use of: timestamps, unique protocol message identifiers and sequence numbers which identify the particular protocol run.

3.3 Secure Spread Library

Secure Spread [36, 2] is a library that, along with the same reliable and ordered message dissemination and membership services as the Spread client library, provides security services such as data confidentiality and integrity.

The main added functionality in Secure Spread is as follows. Whenever the group membership changes, Secure Spread detects it and initiates the execution of a group key agreement protocol. It then detects the termination of the key agreement protocol and notifies the application about the membership change and the new key. In addition, Secure Spread encrypts and decrypts user data using the group key once a group is operational.

One major consideration in designing the library was modularity and flexibility. Secure Spread currently supports five key agreement protocols: BD, CKD, GDH, STR, TGDH (all described in detail below). The architecture of Secure Spread allows it to handle different key agreement algorithms for different groups. A client can be a member of different groups, each group with its own key agreement protocol.

[1] Any former or future member is also an outsider according to this definition.

4 Key Agreement Protocols in Secure Spread

In this section we present the key agreement protocols currently supported in Secure Spread which are the subject of our performance evaluation. The protocols were designed to accommodate different membership changes: join of a new member, leave of a member, network partition, and network merge. Due to lack of space, we provide a description only for merge and partition (multiple members leave), since join and leave can be seen as special cases of merge and partition, respectively.

4.1 GDH Protocol

Cliques GDH IKA.3 is a contributory key agreement protocol which is essentially an extension of the

Protocol GDH - merge: Assume that k members are added to a group of size n. The protocol runs as follows:

Step 1: M_n generates a new exponent N'_n, computes $g^{N_1 \ldots N_{n-1} N'_n}$, and unicasts the message to M_{n+1}.

Step $j+1$ for $j \in [1, k-1]$: New merging member M_{n+j} generates an exponent N_{n+j}, computes $g^{N_1 \ldots {N_n}' \ldots N_{n+j}}$ and forwards the result to M_{n+j+1}.

Step $k+1$: Upon receipt of the accumulated value, M_{n+k} broadcasts it to the entire group.

Step $k+2$: Upon receipt of the broadcast, every member M_i, $\forall i \in [1,\, n+k-1]$, computes $g^{N_1 \ldots {N_n}' \ldots N_{n+k-1}/N_i}$ and sends it back to M_{n+k}.

Step $k+3$: After collecting all the responses M_{n+k} generates a new exponent N_{n+k}, produces the set $S = \{g^{N_1 \ldots {N_n}' \ldots N_{n+k}/N_i} |\ \forall i \in [1,\, n+k-1]\}$ and broadcasts it to the group.

Step $k+4$: Upon receipt of the broadcast, every member M_i, $\forall i \in [1, n+k]$ computes the key
$K = (g^{N_1 \ldots {N_n}' \ldots N_{n+k}/N_i})^{N_i} = g^{N_1 \ldots {N_n}' \ldots N_{n+k}}$.

Figure 1. GDH - merge protocol

two-party Diffie-Hellman protocol. The basic idea is that the shared key is never transmitted over the network. Instead, a list of partial keys (that can be used by individual members to compute the group secret) is sent. One member of the group – group controller– is charged with the task of building and distributing this list. The controller is not fixed and has no special security privileges.

The protocol works as follows. When a merge event occurs (see Figure 1), the current group controller generates a new key token by refreshing its contribution to the group key and then passes the token to one of the new members. When the new member receives this token, it adds its own contribution and passes the token to the next new member[2]. Eventually, the token reaches the last new member. This new member, who is slated to become the new group controller, broadcasts the token to the group without adding its contribution. Upon receiving the broadcast token, each group member (old and new) factors out its contribution and unicasts the result (called a factor-out token) to the new group controller. The new group controller collects all the factor-out tokens, adds its own contribution to each of them, builds the list of partial keys and broadcasts it to the group. Every member can then obtain the group key by factoring in its contribution.

When some of the members leave the group (Figure 2), the group controller (who, at all times, is the most recent remaining group member) removes their corresponding partial keys from the list of partial keys, refreshes each partial key in the list and broadcasts the list to the group. Each remaining member can then compute the shared key. Note that if the current group controller leaves, the last remaining member becomes the group controller of the group.

[2]The new member list and its ordering is decided by the underlying group communication system; Spread in our case. The actual order is irrelevant to GDH.

Protocol GDH - leave: Assume that a set L of members is leaving a group of size n. The protocol runs as follows:

Step 1: The controller M_d generates a new exponent $N_d^{'}$, produces the set $S = \{g^{N_1 \dots N_d^{'}/N_i} | M_i \notin L\}$ and broadcasts it to the remaining group.

Step 2: Upon receipt of S, every remaining member M_i, $\forall i \notin L$ computes the key $K = (g^{N_1 \dots N_d^{'}/N_i})^{N_i} = g^{N_1 \dots N_d^{'}}$

Figure 2. GDH - leave protocol

4.2 CKD Protocol

CKD protocol is a simple centralized group key distribution scheme. The group key is not contributory, but it is always generated by one member, namely, the current group controller. [3]

The group controller establishes a separate secure channel with each current group member by using authenticated two-party Diffie-Hellman key exchange. Each such key stays unchanged as long as both parties (controller and the member) remain in the group. The controller is always the oldest member of the group.

Whenever group membership changes, the group controller generates a new secret and distributes it to the group using the long-term pairwise key (see Figure 3). In case of a merge, the controller, in addition, establishes a secure channel with each new member. When a partition occurs (i.e., multiple members leave), the controller also discards the long-term key it shared with each leaving member. A special case is the case when the group controller itself leaves the group. In this case, the oldest remaining member becomes the new group controller. Before distributing the new key, the new group controller must first establish secure channels with all of remaining group members.

Protocol CKD: Assume that k members are added to a group of size n. M_1 is the group controller. The protocol runs as follows:

Step 1: M_1 selects random $r_1 \bmod q$ (this selection is performed only once),

$M_1 \longrightarrow \{M_{n+j} \mid j \in [1,k]\} : \{g^{r_1} \bmod p \mid j \in [1,k]\}$

Step 2: For each $j \in [1,k]$, M_{n+j} selects random $r_{n+j} \bmod q$,

$M_1 \longleftarrow M_{n+j} : g^{r_{n+j}} \bmod p$

Step 3: M_1 selects a random group secret K_s and computes

$M_1 \longrightarrow M_i : K_s{}^{g^{r_1 r_i}} \bmod p \;\; \forall i \in [2, n+k]$

Step 4: From the broadcast message, every member can compute the group key.

Figure 3. CKD - merge protocol

4.3 TGDH Protocol

TGDH is an adaptation of key trees [37, 38] in the context of fully distributed, contributory group key agreement. TGDH computes a group key derived from the contributions of all group members using a binary tree.

The tree is organized in the following manner: each node $\langle l,v \rangle$ is associated with a key $K_{\langle l,v \rangle}$ and the corresponding blinded key $BK_{\langle l,v \rangle} = g^{K_{\langle l,v \rangle}} \bmod p$. The key at the root node is the group key shared by all members, and a key at the leaf node is the random session contribution by a group member. (Each leaf node is associated with a group member.) Every member knows all the keys on the path from its leaf node to the root as well as all blinded keys on the key tree.

The basic idea here is that every member can com-

[3] We use the term *current* to mean that, even in the CKD protocol suite, a controller can fail or be partitioned out thus causing the controller role to be reassigned to the oldest surviving member.

pute a group key when all blinded keys on the key tree are known. After any group membership event, every member unambiguously adds or removes some nodes related with the event, and invalidates all keys and blinded keys related with the affected nodes. A special group member – the sponsor– , then, takes on a role to compute keys and blinded keys and to broadcast the key tree[4]to the group. If a sponsor could not compute the group key, then the next sponsor comes into play. Eventually, some sponsor will compute the group key and all blinded keys, and broadcast the entire key tree to facilitate the computation of the group key by the the other members of the group.

$$\begin{array}{lcl} M_{s_1} & \xrightarrow{T_{s_1}(BK^*_{s_1})} & \\ & & \{M_1, ..., M_{n+k}\} \\ M_{s_2} & \xrightarrow{T_{s_2}(BK^*_{s_2})} & \end{array}$$

Round 1: request for merge by both groups

$$M'_s \quad \xrightarrow{\widehat{T}_{s'}(BK^*_{s'})} \quad \{M_1, ..., M_{n+k}\}$$

Round 2: update tree $T_{s'}$ to get $\widehat{T}_{s'}$ and broadcast it

Figure 4. TGDH - merge protocol

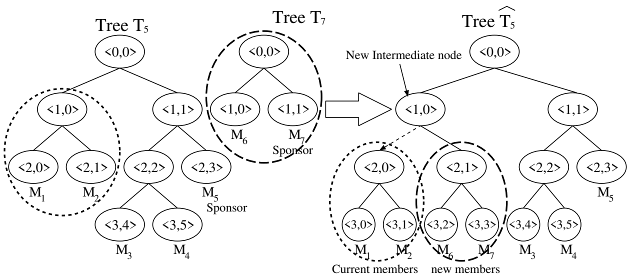

Figure 5. TGDH - merge operation

When a merge event happens (See Figure 4), each sponsor (the rightmost member of each group) broadcasts its tree information to the merging sub-group after refreshing its session random and blinded keys. Upon receiving this message, all members uniquely and independently determine the merge position of the two trees.[5]. As described above, all the keys and blinded keys on the path from the merge point to the root node are invalidated. The rightmost member of the subtree rooted at the merge point becomes the sponsor of the key update operation. The sponsor computes all the keys and blinded keys and broadcasts the tree with the blinded keys to all the other members. All members now have the complete set of blinded keys, which allows them to compute all keys on their key path. Figure 5 shows an example of the merge protocol. Members M_6 and M_7 are added to a group consisting of members M_1, M_2, M_3, M_4 and M_5.

[4]The keys are never broadcasted.

<u>Protocol TGDH - partition:</u> The protocol runs as follows:

for all $M_{s_i} \quad \xrightarrow{\widehat{T}_{s_i}(BK^*_{s_i})} \quad \{M_1, \ldots, M_n\}$

Round 1 to $h' + 1$: update tree T_{s_i} to get $\widehat{T}_{s_i}$

Figure 6. TGDH - partition protocol

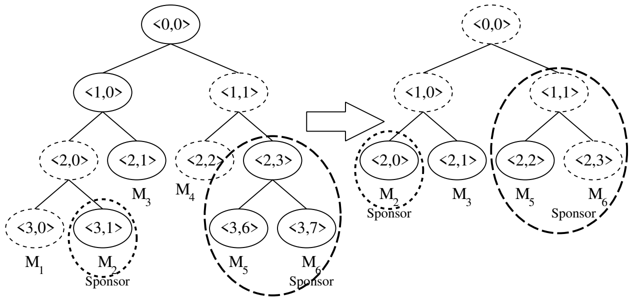

Figure 7. TGDH - partition operation

Following a partition, the protocol runs as follows (see Figure 6). In the first round, each remaining member updates its view of the tree by deleting all leaf nodes associated with partitioned members and (recursively) their respective parent nodes. To prevent re-use of an old group key, one of the remaining mem-

[5]Our heuristic is to choose the joining node as the rightmost "shallowest" node, which does not increase the height. For more details, see [8]

bers changes its key share. To this end, in the first protocol round, the shallowest rightmost sponsor changes its share. Each sponsor then computes the keys and blinded keys as far up the tree as possible, and, then, broadcasts the set of new blinded keys. Upon receiving the broadcast, each member checks whether the message contains a new blinded key. This procedure iterates until all members obtain the group key. Figure 7 shows a partition example where members M_1 and M_4 are removed from the group.

4.4 STR Protocol

STR is basically an "extreme" version of TGDH, where the key tree structure is completely imbalanced or stretched out. This protocol and its features are described in details in [35].

Like TGDH, the STR protocol uses a tree structure that associates the leaves with individual random session contributions of the group members. Every internal (non-leaf) node has an associated secret key and a public blinded key. The secret key is the result of a Diffie-Hellman key agreement between the node's two children. The group key is the key associated with the root node.

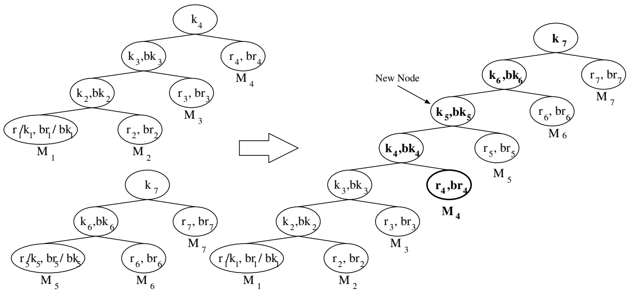

Figure 8. STR - merge operation

The merge protocol runs in two rounds. In the first round, each of the two sponsors (topmost members or right children of the respective root nodes in each tree) exchange their respective key trees containing all blinded keys after refreshing its session random and computing keys and blinded keys up to the root node. The highest-numbered member of the larger tree becomes the sponsor of the second round in the merge protocol (see Figure 8). Using the blinded session random of the other group, the sponsor computes every (key, blinded key) pair up to the intermediate node just below the root node. It then broadcasts the key tree with the blinded keys and blinded session random to the other members. All members now have the complete set of blinded keys which allows them to compute the new group key.

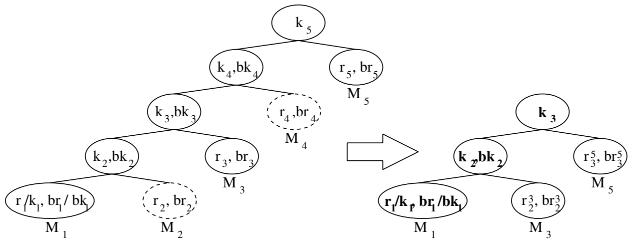

Figure 9. STR - partition operation

In a partition, the sponsor is the lowest-numbered remaining member. After deleting all leaving nodes (see Figure 9), the sponsor refreshes its session random, computes keys and blinded keys up the tree terminating with the root key. It then broadcasts the updated key tree containing only blinded values. Each member can compute then the group key.

4.5 BD Protocol

Unlike other protocols discussed thus far, the Burmester-Desmedt (BD) protocol [6] is independent of the type of group membership change. Furthermore, it has no sponsors, controllers or any other members charged with any special duties.

The main idea in BD is to distribute the computation among members, such that each member performs only three exponentiations. This is achieved by using two communication rounds, each of them consisting of n broadcasts. Figure 10 depicts the actual protocol.

<u>Protocol BD:</u> We assume that the group has n members. The protocol for all membership changes is identical, thus, we do not separately present the join and leave scenarios. The key computed as a result of this protocol is:
$K = g^{x_1 x_2 + x_2 x_3 + \ldots + x_{n-1} x_n} \bmod p$.
The protocol runs as follows:
Step 1: Each member M_i selects random r_i mod q, computes $Z_i = g^{r_1}$ mod p and multicasts the message to the group.
Step 2: Each member M_i, after receiving Z_{i-1} and Z_{i+1}, computes $Xi = (Z_{i+1}/Z_{i-1})^{r_i} = g^{r_{i+1} r_i - r_i r_{i-1}}$
and multicasts it to the group.
Step 3: Each member M_j, after receiving all $X_i, i \neq j$, computes $K = K_j = (Z_{j-1})^{n r_j} {X_i}^{n-1} \ldots X_{i+(n-2)}$ mod p.

Figure 10. BD protocol

5 Theoretical Evaluation

In this section, we analyze the conceptual costs of the five protocols presented above.

We first evaluate the time to compute a new group key when a membership change occurs. There are four events that can lead to a change in membership. The first two are determined by actions initiated by users: a new user wants to become a member of the group or a current member leaves the group. We refer to these events as **join**) and **leave**, respectively. Note that the latter can also happen when one member gets disconnected or simply crashes.

Another category of membership change events is related with the connectivity of the network. An unreliable network can split into disjoint components such that communication is possible within a component but not between components. For members in one component, it appears that the rest of the members have left. After the network fault heals, members previously in components can communicate again. From the group perspective, it appears as if a collection of new members are added to the group. We refer to these events as **partition** and **merge**, respectively.

From the conceptual perspective, we are interested in two cost aspects: the cost of communication (number of rounds, number and type of messages) and the cost of computation (number of exponentiations, signatures and verifications). Although the cost of communication in a modern high-speed LAN setting can appear negligible in comparison with the cost of, say, modular exponentiations, we discuss it nonetheless, since it becomes meaningful in LANs for protocols that trade off low computation for high communication costs. Of course, communication cost is very important in high-delay networks (e.g., WANs). Because of the distributed nature of group communication systems, we consider only serial cost of computation.[6] Thus, we stress that the number of cryptographic operations expressed in the table (for each protocol) is not the sum total for all participants.

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

The height of the key tree constructed by the TGDH protocol is denoted by h [7]. The actual cost of the

[6] Computation that needs to be processed strictly serial. Computation that can be processed in parallel is collapsed accordingly.

[7] Instead of fully balancing the key tree, TGDH uses best-effort approach: it tries to balance the key tree only upon an additive event. The height of the key tree, however, is smaller than $2 \log n$ (maximum group size: n) [8]. The tree can be better balanced when using the AVL tree management technique described in [23]. However, this will incur a higher communication cost for a leave

		Communication				Computation		
		Rounds	Messages	Unicast	Multicast	Exponentiations	Signatures	Verifications
GDH	Join	4	$n+3$	$n+1$	2	$n+3$	4	$n+3$
	Leave	1	1	0	1	$n-1$	1	1
	Merge	$m+3$	$n+2m+1$	$n+2m-1$	2	$n+2m+1$	$m+3$	$n+2m+1$
	Partition	1	1	0	1	$n-p$	1	1
TGDH	Join, merge	2	3	0	3	$\frac{3h}{2}$	2	3
	Leave	1	1	0	1	$\frac{3h}{2}$	1	1
	Partition	h	$2h$	0	$2h$	$3h$	h	h
STR	Join	2	3	0	3	7	2	3
	Leave, partition	1	1	0	1	$\frac{3n}{2}+2$	1	1
	Merge	2	3	0	3	$3m+4$	2	3
BD	Join	2	$2n+2$	0	$2n+2$	3	2	$n+3$
	Leave	2	$2n-2$	0	$2n-2$	3	2	$n+1$
	Merge	2	$2n+2m$	0	$2n+2m$	3	2	$n+m+2$
	Partition	2	$2n-2p$	0	$2n-2p$	3	2	$n-p+2$
CKD	Join	3	3	2	1	$n+2$	3	3
	Leave	1	1	0	1	$n-2$	1	1
	Merge	3	$m+2$	m	2	$n+2m$	3	$m+2$
	Partition	1	1	0	1	$n-p-1$	1	1

Table 1. Communication and Computation Costs

TGDH protocol depends on the tree height, the balancedness of the key tree, the insertion point of the joining tree (or node) and the locations of the leaving node(s). To err on the side of safety, we compute the worst case cost for the TGDH.

The number of modular exponentiations for STR upon a leave event is determined by the location of the deepest leaving node. We, therefore, compute the average cost, i.e. the case when the $\frac{n}{2}$-th node left the group. All other protocols, except TGDH and STR, show exact cost numbers.

Current implementations of TGDH and STR recompute a blinded key even though it has been computed already by the sponsor. This provides a form of key confirmation, since a user who receives a token from another member can check whether his blinded key computation is correct. This computation, however, can be removed for better efficiency, and we consider this optimization when counting the number of exponentiations.

operation.

The BD protocol has a hidden cost which is not reflected in Table 1. In Step 3 (see Figure 10), BD protocol has $n-1$ modular exponentiations with a small exponent. Though a single exponentiation takes a negligible amount of time, the sum of all the $n-1$ exponentiations is not negligible. For example, it requires 373 1024-bit modular multiplications, if modular exponentiation is implemented with the square-and-multiply algorithm. (OpenSSL uses Montgomery reduction and the sliding window algorithm to implement the modular exponentiation which is faster than simple square-and-multiply algorithm. However, the former requires almost the same time as the latter for small exponent.) Because of this hidden cost, it is difficult to compare the computational overhead of BD to the other protocols.

Join. All protocols except CKD require two communication rounds. CKD uses three rounds because the new member must first establish a secure channel (via

Diffie-Hellman key exchange) with the current group controller. The most expensive protocol in terms of communication is BD which uses n broadcast messages for each round. Other protocols use a constant number of messages, either two or three.

GDH and CKD are the most expensive in terms of computation, requiring a linear number of exponentiations relative to the group size. TGDH is comparatively efficient, scaling logarithmically in the number of exponentiations. STR, in turn, uses a constant number of modular exponentiations. BD requires the least modular exponentiations, but has the above hidden cost.

Leave. Table 1 shows that, for a leave event, the BD protocol is the most expensive from the communication point of view. The cost order between the CKD, GDH, STR and TGDH is determined strictly by the computation cost, since they all have the same communication cost (one round consisting of one message). Therefore, TGDH is best for handling leave events. STR, GDH and CKD scale linearly with the group size. We note that the cost of CKD is actually higher than the one listed in Table 1, because in the case when the controller leaves the group, the new group controller must establish secure channels with all group members. Since BD, again, has a hidden cost, it is hard to compare with other algorithms.

Merge. We first look at the communication cost. Note that GDH scales linearly with the number of the members added to the group in communication rounds, while BD, CKD, STR and TGDH are more efficient using a constant number of rounds. Since BD uses n messages for each round and CKD uses $m + 1$ messages, STR and TGDH are the most communication-efficient for handling merge events.

Looking at the computation costs, it seems that BD has the lowest cost: only three exponentiations. However, the impact of the number of small exponent exponentiations is difficult to evaluate. TGDH scales logarithmically with the group size, being more efficient than STR, CKD and GDH which scale linearly with both the group size and the number of new members added to the group.

Partition. Table 1 shows that the GDH, STR and CKD protocols are bandwidth efficient: only one round consisting of one message. BD is less efficient with two rounds of n messages each. Partition is the most expensive operation in TGDH, requiring a number of rounds bounded by the tree height.

As before, computation-wise it is difficult to compare BD with the other protocols because of its hidden cost in Step 3. TGDH requires a logarithmic number of exponentiations. GDH, STR, and CKD scale linearly with the group size.

6 Experimental Results

In this section we present, compare and evaluate the experimental costs of the five protocols discussed above. As mentioned earlier, four events can cause the group change: join, leave, partition and merge. Our results reflect only the two most common events: join and leave.

We measure the "total elapsed time" from the moment the group membership event happens until the moment when the group key agreement finished and the application is notified about the group change and the new key. This time includes all the communication and computation costs of the key agreement protocol as well as the cost of the membership service provide by the group communication system. In other words it represents the actual delay experienced by an application (or user) using the Secure Spread group communication system, when it performs a join or a leave operation to a group.

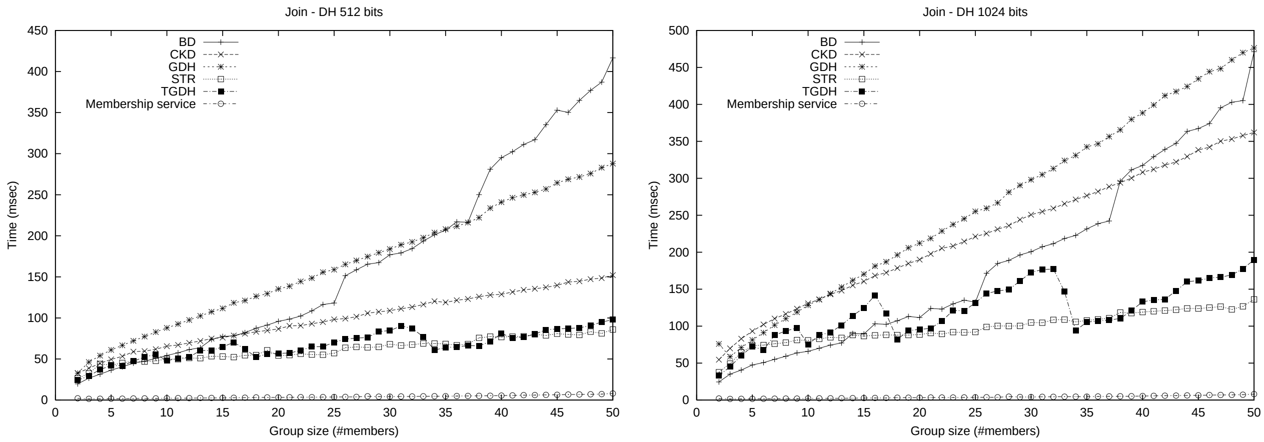

Figure 11. Join - average time (LAN)

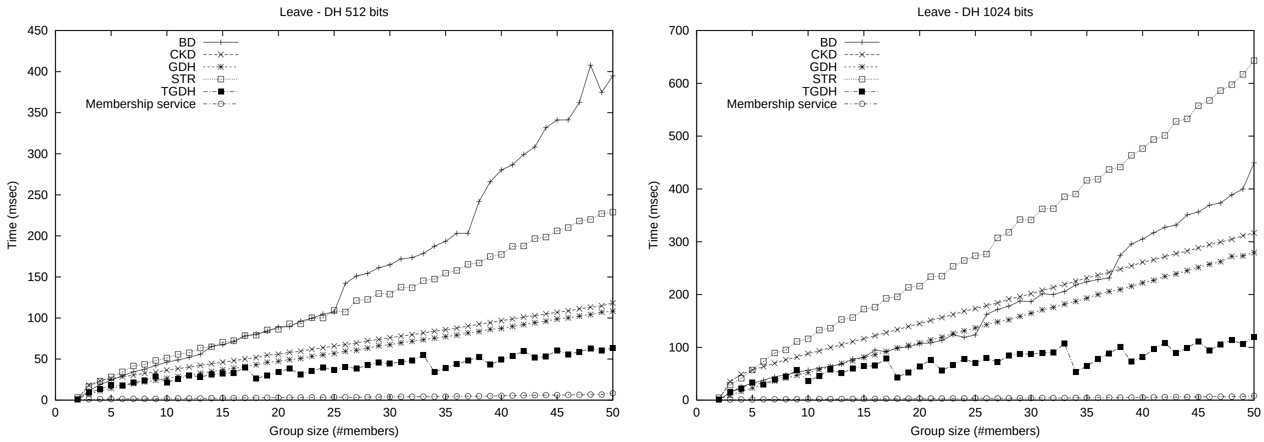

Figure 12. Leave - average time (LAN)

6.1 Experimental Results in LAN

In this section we present, compare and evaluate the performance of the five protocols discussed above in a local area network setting.

6.1.1 Testbed and Basic Parameters

The experimental testbed is a cluster of thirteen 666 MHz Pentium III dual-processor PCs running Linux. Each machine runs a Spread daemon. Group members are uniformly distributed on the thirteen machines. Therefore, more than one process can be running on a single machine (which is frequent in many collaborative applications).

Tests performed on our testbed show that the average cost of sending and delivering one Agreed multicast message is almost constant, ranging anywhere from 0.75 milliseconds to 0.92 milliseconds for a group size ranging from 2 to 50 members. Also, in a scenario (similar to a BD round) where each member of the group sends a broadcast message and receives all the $n-1$ messages from the rest of the members (n being the group size), the average cost is about 2 milliseconds for a group of 2 members and about 21 milliseconds for a group of size 50. The cost of the membership service (see Figures 11 and 12) is negligible with respect to the key agreement overhead, varying between 2 and 8 milliseconds for a group between 2 and 50 members.

We use RSA signatures for message origin and data authentication, although the Cliques toolkit supports any digital signature scheme implemented in OpenSSL library. This is because RSA signature verification is quite inexpensive and all group key agreement protocols described in this paper rely heavily on source authentication, i.e., most protocol messages must be verified by all receivers. If all processes are located on different CPU platforms, verification is performed in parallel. In practice, however, a CPU may have multiple group processes and expensive signature verification (e.g., as in DSA) noticeably degrades performance.

We used 1024-bit RSA signatures with the public exponent of 3 to reduce the verification overhead, although a quasi-standard in RSA parameter selection is $65,537$. This is because 1) there are no security risks for using 3 as a public exponent in RSA signature scheme [39], 2) BD and GDH require n simultaneous signature verifications, and 3) in our current topology, some machines can have multiple group member processes. On our hardware platform, the RSA sign and verify operations take 9.6 and 0.2 milliseconds, respectively.[8].

For the short-term group key, we use both 512- and 1024-bit Diffie-Hellman parameter p and 160-bit q. The cost of a single exponentiation is 1.7 and 5.3 milliseconds for a 512- and a 1024-bit modulus, respectively.

6.1.2 Test Scenarios

Each protocol has its peculiar features which we took into account by keeping experiments as similar and as simple as possible.

GDH and BD are both oblivious (insofar as cost) to the position of the joining or leaving member, i.e., all leave and all join operations cost the same in these protocols.

CKD can be expensive for a leave event if the leaving member is the current group controller. We take this into account by factoring in the $1/n$ probability of the group controller leaving the group.

Since the theoretical cost for leave in STR is the average cost, we tested the average case: this happens when the member leaving the group is the $n/2$-th member located in the middle of the STR key tree.

TGDH is the most difficult protocol to evaluate because its cost depends on the location of the leave or join node, tree height, and the balancedness of the tree. For a truly fair comparison, Secure Spread must be first run with TGDH for a long time (with a random sequence of joins and leaves) in order to generate a random-looking tree. The experiments for join and leave must then be conducted on this random tree. However, such tests are very difficult to perform, as mentioned above.

Therefore, we chose a simpler experimental setting by measuring join and leave costs on an artificially balanced TGDH key tree with n members. We note that for a random tree, the cost of join will be less expensive since the member will be joined closer to the root, while leave will be more expensive, but still less expensive than GDH [8].

6.1.3 Join Results

Figure 11 shows the total average time for a group to establish secure membership following a join of a new member.

In the graph on the left (512-bit modulus) it looks, overall, that STR outperforms other protocols. Closer inspection reveals that BD is actually the most efficient for small group sizes (less than 7 or so). Recall that BD involves only three full-blown exponenti-

[8]This is not surprising since OpenSSL uses the Chinese Remainder Theorem to speed up RSA signatures.

ations as opposed to STR's seven. However, BD has $(n+3)$ signature verifications, whereas STR only has 3. Furthermore, BD requires $O(n \log n)$ modular multiplications in Step 3 (to compute the key, see Figure 10). Finally, BD has two rounds of all-to-all broadcasts. Small group size makes all of these factors negligible. However, as the group size grows, BD deteriorates rapidly since both modular multiplications, RSA signature verifications and broadcasts add up. In fact, after passing the group size of thirty, BD becomes the worst performer if Diffie-Hellman parameter is 512. For 1024-bit modulus, GDH is the worst due to the sharp increase in modular exponentiation.

Another interesting observation about BD's performance in **ALL measurements** is that its cost roughly doubles as the group size grows in increments of 13. Recall that 13 is the number of machines used in the experiments. Because BD is fully symmetric, as soon as just one machine starts running one additional group member (process), the cost of BD doubles. Moreover, it can be noted that starting with the group size of 26, the performance degrades significantly. As mentioned before the machines we used are dual processors, so up to a group size of 26 it can be assumed that there is one client on one processor. For the other protocols this behavior is less obvious since in all of them, the most costly tasks are performed by a single member (controller or sponsor).

The graph on the right (1024-bit modulus) does not show the same deterioration in BD. It remains the best for very small groups up to 14 members. This is because the cost of exponentiations rises sharply from 512 to 1024 bits, while the cost of RSA signature verifications and broadcasts (which weighs BD down in the 512-bit case) is not felt nearly as much.

In both graphs, TGDH and STR are fairly close with the latter performing slightly better. Although the numbers in Table 1 show constant cost for STR, the measured cost increases slightly because: 1) a CPU may experience an increasing number processes as the number of members increases, and, 2) other minor overhead factors such as tree management. Conceptually, TGDH can never outperform STR in a join, since the latter's design includes the optimal case (i.e., join at the root) of the former. Experimental results, however, show that TGDH can sometimes outperform STR (see small dips in TGDH graph at around 18 and 34 members). This is because most members in a fully balanced TGDH tree compute two modular exponentiations in the last protocol round, as opposed to four in STR.

The difference between CKD and GDH comes from exponentiation and signature verifications: extra operations in GDH include n verifications, one RSA signature and one (DH) modular exponentiation. GDH and BD each have $n+3$ signature verifications, which is, as mentioned above, relatively expensive even with a 512-bit RSA modulus.

6.1.4 Leave Results

Figure 12 shows the average time for a group to establish secure membership upon a member leave event. In line with the conceptual evaluation, TGDH outperforms the rest, as it requires the fewest $(O(log\ n))$ modular exponentiations. This sub-linear behavior becomes particularly evident past the group size of 30. Note that for a random tree although leave will be more expensive, it will be less expensive than leave for the GDH protocol [8].

BD is the worst in 512-bit leave; recall that, in it, leave and join incur the same cost. STR, CKD and GDH all exhibit linear increase in cost. CKD and GDH are quite close while STR's linear factor is $2n$ which makes its slope steeper.

In case of 1024-bit modulus, STR is the most expensive protocol, since it involves (costlier in 1024- than in 512-bit case) modular exponentiations. TGDH exhibits the cost roughly twice that of 512-bit case

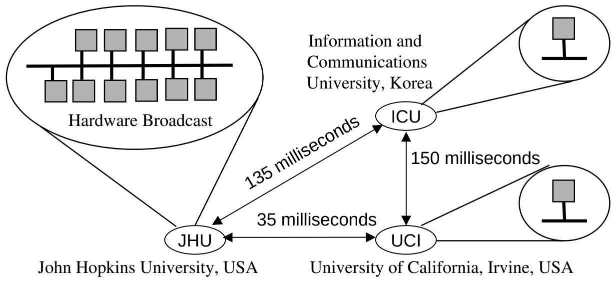

Figure 13. WAN testbed

and remains the leader. BD, however, is no longer the worst and, at least for small group sizes, (less than 37 or so) performs close to, or better than, GDH. Once again, we attribute this to the relatively cheap cost of RSA signature verifications in the commensurately small number of full-blown 1024-bit exponentiations in BD.

6.2 Experimental Results in WAN: An Extreme Case

In this section we present some preliminary results in a WAN environment with high-delays. As in the LAN experiments, we focused on evaluating the total time needed to perform a join or a leave operation, from the moment the group membership changes due to the join or leave of a member, till the moment a new key was computed and delivered to all members of the group. This time includes all communication and computation costs of the key agreement protocol as well as the cost of the membership service of the group communication system.

6.2.1 Testbed and Basic Parameters

Figure 13 presents the network configuration we used for our experiments on WAN.

To achieve the same computation distribution as for the LAN experiments, we used an experimental testbed of thirteen PCs running Linux: ten 666 MHz Pentium III dual-processor PCs, one 1.1 MHz Athlon and one 930 MHz Pentium III PCs, located as follows: first eleven machines at Johns Hopkins University (JHU), Maryland, one machine at University of California at Irvine (UCI) and one at the Information and Communications University (ICU), Korea. We uniformly distributed group members among the thirteen machines with more than one group member process running on a single machine. Each machine runs a Spread daemon. Approximate round-trip latencies (ping times) as reported by the ping program are: JHU - UCI 70 milliseconds, UCI - ICU 300 milliseconds and ICU - JHU 270 milliseconds.

The average delay of sending and delivering one Agreed multicast message depends on the sender's location. The actual delay (in milliseconds) is: sender at JHU – 392, sender at UCI – 328, and sender at ICU – 334. When each group member sends a broadcast message and waits to receive $n - 1$ messages from the rest of the group (similar to a BD communication round), the average cost is about 1000 milliseconds for a group of size 50.

It is important to notice that, in a LAN setting, the cost of the group membership service provided by the underlying group communication system is negligible with respect to the key agreement overhead, e.g., about 7 milliseconds vs. hundreds of milliseconds. However, this relative cost becomes significantly higher in a WAN setting. The cost of the membership service as it can be seen in Figure 14, varies between 400 and 670 milliseconds for a join operation and between 250 and 650 for a leave operation, for a group of 2 to 50 members.

As in the LAN experiments, we used RSA 1024-bit with the public exponent of 3 to compute message signatures. On our test PCs, the RSA sign and verify operations take $6.9 - 17.9$ and $0.2 - 0.4$ milliseconds, respectively, depending on the platform. For the short-term group key, we use 512-bit Diffie-Hellman parameter p and 160-bit q. [9] The cost of a single modular exponentiation is between 0.8 and 1.7 milliseconds.

6.2.2 Join Results

Figure 14 (left) presents our results for the average time required to establish a secure group membership when a new member joins the group. The graph also separately plots the cost of the insecure group membership service. The difference between the total time required by each protocol and the insecure group membership service cost, represents the overhead of the key agreement itself (both communication and computation).

The first observation is that the GDH protocol performs significantly worse than others. The main difference between GDH and the other protocols comes from communication. First, the number of rounds is greater than that of the other protocols as shown in Table 1. GDH requires 4 rounds while others require only 2 rounds. Second, a different factor comes from the round where all members factor out their contribution and then send the result to the group controller (see Section 4.1). Although theoretically this can be seen as a one round of n unicast messages, for the correctness and robustness of the key agreement protocol [2] these messages need to be in Agreed order with respect to the messages sent within the group. This raises the cost of such a message to the cost of an Agreed broadcast message. Lastly, all members but the controller are sending, while the group controller needs to receive and process $(n-1)$ messages, which increases the cost even more. In conclusion, the cost of the factor out message round is more expensive than the theoretical analysis shows and has a big impact on the performance of the protocol. This behavior was less obvious in a LAN setup because the communication cost in general is much smaller than the computation overhead.

The rest of the protocols are more or less in the same range, with BD becoming more expensive for a group size bigger than 30, while STR and TGDH show similar performance. It is interesting to notice that both STR and TGDH come closer to the BD performance. This is also because of the communication aspect of the protocols. As stated in Table 1 and Sections 4.4 and 4.3, both STR and TGDH have 2 rounds, out of which the first round consists of two "simultaneous" broadcasts. In our implementation, these broadcasts are not simultaneous, since to achieve ordered delivery of the messages, group communication systems use a mechanism, where a token is passed between participants and only the entity that has the token is allowed to send. Because in our particular WAN setup, there are three main sites, JHU, UCI and ICU (the cost of passing the token inside a site is significantly smaller than the cost between sites), the cost of a STR and TGDH scenario - 2 members are sending two broadcasts and all the members need to receive them - is close to the cost of a BD scenario - n members broadcast and all members need to receive $n-1$ messages. [10] BD deteriorates faster than other protocols due to the number of broadcast messages.

Though CKD has three rounds, two of them involve single-message unicasts. This helps CKD to remain competitive with respect to the other protocols.

We can clearly conclude that communication costs (the number of rounds and the numbers of messages sent in one round) of a group key agreement scheme affects severely its performance on the wide area network, particularly in one with high-delays as the one we used in our experiments.

[9] We intend to include results for 1024-bit Diffie-Hellman in the final submission.

[10] This is because if one member missed the token, it needs to wait for the token to pass the whole ring, while in the BD scenario if the token completes a cycle, no matter where it started, everybody succeeds to send.

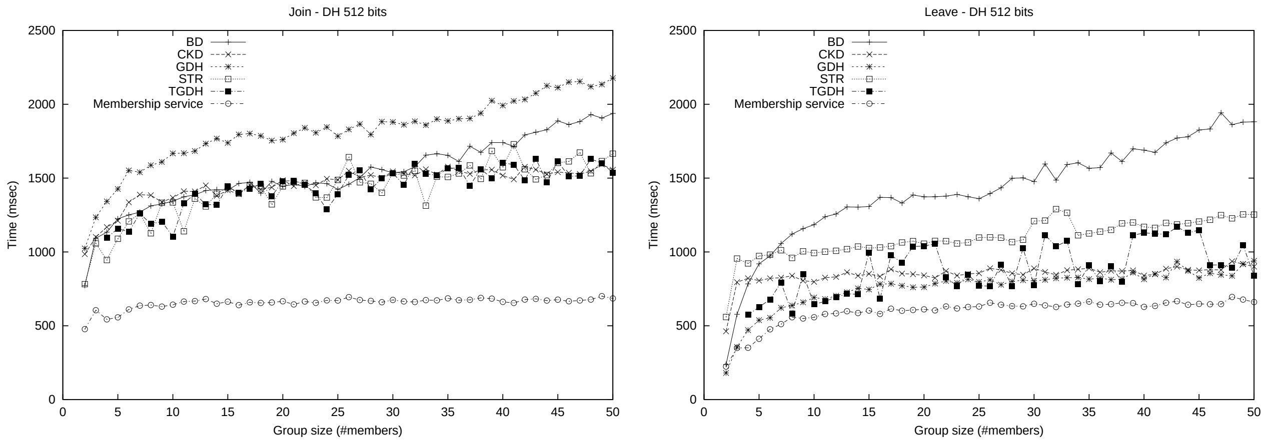

Figure 14. Join and leave - average time (WAN)

6.2.3 Leave Results

In case of leave (see Figure 14, right), BD is the most expensive protocol in our WAN setup, due to the two rounds on n broadcasts and its high computational cost.

GDH, CKD and TGDH require only a single broadcast, thus, they exhibit similar performance results. Although STR also requires only one broadcast, it has significantly higher computation cost with respect to the rest.

We observe that TGDH exhibits a behavior more dynamic than GDH and CKD. We attribute this to the fact that, in CKD and GDH, the controller (who does the bulk of computation and broadcasts) was running on a fixed machine. Whereas, in TGDH, the sponsor (who also does most of computation and broadcasts) was running on any of the 13 testbed machines. If tested with a fixed sponsor, we suspect that TGDH, GDH, CKD would have almost identical cost.

The number of rounds seems to be the most important factor in the performance of the protocols we investigated, in an 'extreme' WAN network.

6.3 Discussion

Following our experiments and their interpretation (as discussed above), we conclude that computation cost is most important in a low delay network and communication cost is most important in a high delay network.

In a LAN setting, TGDH is the best performing protocol overall. However, we also note that for small groups – no greater than, say, a dozen members – BD is a slightly better performer. Another factor in BD's favor is its simplicity: all operations are symmetric and are implemented via the same protocol with few data structures to manage. In contrast, TGDH involves some non-trivial tree management. (See [8] for details.)

An additional factor can skew the comparable performance of the evaluated protocols. TGDH was evaluated with a well-balanced tree. In a random (unbalanced) tree the join cost would have been less expensive since the joining node would have been inserted nearer the root node, while the leave cost would have been more expensive, but less expensive than GDH [8].

In a high-delay WAN setting, TGDH and CKD exhibited the best performance. Since TGDH has smaller computation overhead, we expect it to outperform CKD in a medium delay wide area network ($70 - 100$ milliseconds round-trip links).

From the above we conclude that TGDH is the best choice of a key agreement protocol for dynamic peer groups in both local and wide area networks.

7 Conclusions and Future Work

We presented a framework for cost evaluation of group key agreement protocols in a realistic network setting. The focus was on five notable group key agreement protocols integrated with a reliable group communication system (Spread). After analyzing the protocols' conceptual costs, we measured their behavior in both LAN and WAN settings. In particular, we presented and discussed the measurements for the elapsed time required to process two common group membership change events: join and leave. The results we presented indicate that TGDH is the protocol that will work best in both environments.

A few of items remain for future work. Experimenting with the protocols on a medium-delay (e.g., $70 - 100$ milliseconds round-trip) wide area network is of interest since communication and computation costs are expected to equalize, at least in theory. Finally, we also need to experiment with more complex group operations such as partition and merge.

8 Acknowledgements

The authors would like to thank Dang Nguyen Duc and Taekyoung Kwon for providing some of the machines used in our evaluation tests.

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," in *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, (Taipei, Taiwan), pp. 330–343, April 2000.

[2] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton, and G. Tsudik, "Exploring robustness in group key agreement," in *Proceedings of the 21th IEEE International Conference on Distributed Computing Systems,*, pp. 399–408, IEEE Computer Society Press, April 2001. An extended version is available as Tech. Rep. CNDS 2000-4.

[3] M. Steiner, G. Tsudik, and M. Waidner, "Key agreement in dynamic peer groups," *IEEE Transactions on Parallel and Distributed Systems*, August 2000.

[4] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 644–654, Nov. 1976.

[5] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener, "A secure audio teleconference system," *Advances in Cryptology – CRYPTO'88*, August 1990.

[6] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," *Advances in Cryptology – EUROCRYPT'94*, May 1994.

[7] W.-G. Tzeng and Z.-J. Tzeng, "Round-efficient conference-key agreement protocols with provable security," in *ASIACRYPT '2000*, Lecture Notes in Computer Science, (Kyoto, Japan), Springer-Verlag, December 2000.

[8] Y. Kim, A. Perrig, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collab-

orative groups," in *Proceedings of 7th ACM Conference on Computer and Communications Security*, pp. 235–244, ACM Press, November 2000.

[9] O. Rodeh, K. Birman, and D. Dolev, "Optimized group rekey for group communication systems," in *Proceedings of ISOC Network and Distributed Systems Security Symposium*, February 2000.

[10] L. Dondeti, S. Mukherjee, and A.Samal, "Disec: A distributed framework for scalable secure many-to-many communication," in *Proceedings of The Fifth IEEE Symposium on Computers and Communications (ISCC 2000)*, July 2000.

[11] D. W. Carman, P. S. Kruus, and B. J. Matt, "Constraints and approaches for distributed sensor network security," Tech. Rep. 00-010, NAI Labs, September 2000.

[12] K. P. Birman and R. V. Renesse, *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, March 1994.

[13] Y. Amir, D. Dolev, S. Kramer, and D. Malki, "Transis: A communication sub-system for high availability," *Digest of Papers, The 22nd International Symposium on Fault-Tolerant Computing Systems*, pp. 76–84, 1992.

[14] R. V. Renesse, K.Birman, and S. Maffeis, "Horus: A flexible group communication system," *Communications of the ACM*, vol. 39, pp. 76–83, April 1996.

[15] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. Agarwal, and P. Ciarfella, "The Totem single-ring ordering and membership protocol," *ACM Transactions on Computer Systems*, vol. 13, pp. 311–342, November 1995.

[16] B. Whetten, T. Montgomery, and S. Kaplan, "A high performance totally ordered multicast protocol," in *Theory and Practice in Distributed Systems, International Workshop*, Lecture Notes in Computer Science, p. 938, September 1994.

[17] K. P. Birman and T. Joseph, "Exploiting virtual synchrony in distributed systems," in *11th Annual Symposium on Operating Systems Principles*, pp. 123–138, November 1987.

[18] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal, "Extended virtual synchrony," in *Proceedings of the IEEE 14th International Conference on Distributed Computing Systems*, pp. 56–65, IEEE Computer Society Press, Los Alamitos, CA, June 1994.

[19] T. Anker, G. V. Chockler, D. Dolev, and I. Keidar, "Scalable group membership services for novel applications," in *Proceedings of the Workshop on Networks in Distributed Computing*, 1998.

[20] I. Keidar, K. Marzullo, J. Sussman, and D. Dolev, "A client-server oriented algorithm for virtually synchronous group membership in WANs," Tech. Rep. CS99-623, Univ. of California, San Diego, June 1999.

[21] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "The SecureRing protocols for securing group communication," in *Proceedings of the IEEE 31st Hawaii International Conference on System Sciences*, vol. 3, (Kona, Hawaii), pp. 317–326, January 1998.

[22] O. Rodeh, K. Birman, M. Hayden, Z. Xiao, and D. Dolev, "Ensemble security," Tech. Rep. TR98-1703, Cornell University, Department of Computer Science, September 1998.

[23] O. Rodeh, K. Birman, and D. Dolev, "Using AVL trees for fault tolerant group key management," Tech. Rep. 2000-1823, Cornell Univer-

sity, Computer Science; Tech. Rep. 2000-45, Hebrew University, Computer Science, 2000.

[24] O. Rodeh, K. Birman, and D. Dolev, "The architecture and performance of security protocols in the Ensemble Group Communication System," *ACM Transactions on Information and System Security*, To appear.

[25] M. K. Reiter, "Secure agreement protocols: reliable and atomic group multicast in RAMPART," in *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pp. 68–80, ACM, November 1994.

[26] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," Tech. Rep. TR99-1745, Department of Computer Science, Cornell University, May 1999.

[27] P. McDaniel, A. Prakash, and P. Honeyman, "Antigone: A flexible framework for secure group communication," in *Proceedings of the 8th USENIX Security Symposium*, pp. 99–114, August 1999.

[28] Y. Amir and J. Stanton, "The Spread wide area group communication system," Tech. Rep. 98-4, Johns Hopkins University, Center of Networking and Distributed Systems, 1998.

[29] Y. Amir, C. Danilov, and J. Stanton, "A low latency, loss tolerant architecture and protocol for wide area group communication," in *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 327–336, June 2000.

[30] Y. Amir, B. Awerbuch, C. Danilov, and J. Stanton, "Flow control for many-to-many multicast: A cost-benefit approach," Tech. Rep. CNDS-2001-1, Johns Hopkins University, Center of Networking and Distributed Systems, 2001.

[31] Y. Amir, *Replication using Group Communication over a Partitioned Network*. PhD thesis, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, 1995.

[32] A. Fekete, N. Lynch, and A. Shvartsman, "Specifying and using a partitionable group communication service," in *Proceedings of the 16th annual ACM Symposium on Principles of Distributed Computing*, (Santa Barbara, CA), pp. 53–62, August 1997.

[33] J. Schultz, "Partitionable virtual synchrony using extended virtual synchrony," Master's thesis, Department of Computer Science, Johns Hopkins University, January 2001.

[34] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 784–803, December 1997.

[35] Y. Kim, A. Perrig, and G. Tsudik, "Communication-efficient group key agreement," in *Proceedings of IFIP SEC 2001*, June 2001.

[36] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton, and G. Tsudik, "Exploring robustness in group key agreement," Tech. Rep. CNDS 2000-4, Johns Hopkins University, Center of Networking and Distributed Systems, 2000. http://www.cnds.jhu.edu/publications/.

[37] D. Wallner, E. Harder, and R. Agee, "Key Management for Multicast: Issues and Architectures." RFC 2627, June 1999.

[38] D. McGrew and A. Sherman, "Key establishment in large dynamic groups using one-way function trees." Manuscript, May 1998.

[39] D. Boneh, “Twenty years of attacks on the RSA cryptosystem,” *Notices of the American Mathematical Society (AMS)*, vol. 46, no. 2, pp. 203–213, 1999.