On the Performance of Group Key Agreement Protocols

YAIR AMIR
Johns Hopkins University
YONGDAE KIM
University of Minnesota, Twin Cities
CRISTINA NITA-ROTARU
Purdue University
and
GENE TSUDIK
University of California, Irvine

Group key agreement is a fundamental building block for secure peer group communication systems. Several group key management techniques were proposed in the last decade, all assuming the existence of an underlying group communication infrastructure to provide reliable and ordered message delivery as well as group membership information. Despite analysis, implementation, and deployment of some of these techniques, the actual costs associated with group key management have been poorly understood so far. This resulted in an undesirable tendency: on the one hand, adopting suboptimal security for reliable group communication, while, on the other hand, constructing excessively costly group key management protocols.

This paper presents a thorough performance evaluation of five notable distributed key management techniques (for collaborative peer groups) integrated with a reliable group communication system. An in-depth comparison and analysis of the five techniques is presented based on experimental results obtained in actual local- and wide-area networks. The extensive performance measurement experiments conducted for all methods offer insights into their scalability and practicality. Furthermore, our analysis of the experimental results highlights several observations that are not obvious from the theoretical analysis.

This work was supported by grant F30602-00-2-0526 from the Defense Advanced Research Projects Agency (DARPA).
A two-page abstract related to this work was published at ICDCS 2002 [Amir et al. 2002].
Authors' addresses: Y. Amir, Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218, USA; email: yairamir@cs.jhu.edu; Y. Kim, Department of Computer Science and Engineering, University of Minnesota—Twin Cities, MN 55455, USA; email: kyd@cs.umn.edu. (Part of this work has been done when he was working at University of California, Irvine.); C. Nita-Rotaru, Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA; email: crisn@cs.purdue.edu. (Part of this work has been done when she was a graduate student at Johns Hopkins University.); G. Tsudik, Information and Computer Science Department, University of California, Irvine, CA 92697-3425, USA; email: gts@ics.uci.edu.

Categories and Subject Descriptors: C.2.4 [**Distributed Systems**]: Distributed Applications; D.4.5 [**Operating Systems**]: Reliability; D.4.6 [**Operating Systems**]: Security and Protection; D.4.8 [**Operating Systems**]: Performance

General Terms: Algorithms, Design, Performance, Reliability, Security, Theory

Additional Key Words and Phrases: Group Key Management, Secure Communication, Peer Groups, Group Communication

1. INTRODUCTION

The Internet is increasingly being 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 open networks, basic security services—such as data privacy, integrity, and authentication—are necessary for collaborative applications. These services, in turn, are impossible without secure, robust, and efficient group key management. Clearly, key management is the most basic security service, since, without it, secure communication is basically impossible. In the context of collaborative groups, besides the design of the individual security building blocks, group security policy is a critical component. A comprehensive set of definitions and requirements of security policies in groups is presented in Harney et al. [2001].

1.1 Key Agreement in Peer Groups

A number of group key management techniques have been proposed in the past. They 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 via a pair-wise secure channel established with each group member. We view centralized group key management as inappropriate for secure peer group communication, since a 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 by using fault-tolerance and replication techniques. Unfortunately, the omni-presence issue is difficult to solve in a scalable and efficient manner. We note, however, that the centralized approach works well in one-to-many multicast scenarios since a trusted third party (or a set thereof) placed at, or very near, the source of communication, can support continued operation within an arbitrary partition as long as it includes the source. This is appropriate since most one-to-many settings only

aim to offer continued secure operation within a single partition containing the source.

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 distribution server. Although robust, this approach has a notable drawback in that it requires a key server to maintain long-term pair-wise secure channels with all current group members in order to distribute group keys. Some schemes take advantage of data structures to minimize the number of encryption and messages that must be generated when the key changes. When a new key server is selected all these data structures also need to be recreated.

In contrast, *contributory group key agreement* requires every group member to contribute an equal share to the common group secret, computed as a function of all members' contributions. These protocols are appropriate for dynamic peer groups. This approach avoids the problems with the single point(s) of trust and failure. Moreover, some contributory methods do not require establishing pair-wise secret channels among group members. Also, unlike most group key distribution protocols, they offer strong key management security properties such as key independence and perfect forward secrecy (PFS) [Menezes et al. 1996]. Recent research on authenticated group key agreement protocols provides stronger security guarantees against active attackers [Bresson et al. 2001a, 2001b]. More detailed discussion can be found in Kim [2002].

We note that many centralized and distributed key management protocols (such as the Logical Key Hierarchy (LKH) Protocol [Wallner et al. 1999; Wong et al. 2000], LKH+ [Caronni et al. 1999], One-Way Function Tree protocol [Sherman and McGrew 2003], and Centralized Flat Table [Caronni et al. 1999], to name just a few) rely on symmetric encryption to distribute group keys, as opposed to contributory protocols which rely on modular exponentiations. Therefore, they do not provide PFS. However, such protocols scale to large groups and have a lighter overhead than contributory ones.

The cost of group key management is 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, while protocols minimizing communication require more computational effort.

1.2 Focus

Our previous work [Amir et al. 2004] showed 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 proved that the resulting protocol preserved group communication membership semantics and ordering guarantees. The resulting protocol was based on Group Diffie–Hellman (GDH IKA.2) contributory key agreement [Steiner et al.

2000] that generalizes the two-party Diffie–Hellman key exchange to groups [Diffie and Hellman 1976]. The technique used in Amir et al. [2004] can be adapted for any multiround key management protocol.

Key management has a great impact not only on the security and fault tolerance of the overall system, but also on its performance. For this reason, in this work we focus on the *performance analysis* of group key management protocols. Moreover, we focus on contributory key agreement protocols because of their strong security properties. We began by identifying several notable techniques and integrating them with a reliable group communication system [Amir and Stanton 1998]. In addition to GDH, we analyze four other protocols: Centralized Key Distribution (CKD) is a centralized scheme where one group member group is dynamically chosen to act as a key server; Tree-Based Group Diffie–Hellman (TGDH) blends a logical tree structure with Diffie–Hellman key exchange to achieve a protocol more efficient in computation than GDH; Skinny Tree (STR) trades off lower communication overhead for increased computation; Burmester–Desmedt (BD) distributes and minimizes computation by using more messages and many-to-many broadcasts. All protocols offer similar security properties, including group key independence and 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 keys.

The main contributions of this work are:

—A peer group key management framework that supports multiple protocols, allowing assignment of different key agreement protocols to different groups.
—A detailed theoretical performance analysis of the five notable group key agreement methods with respect to communication and computation costs.
—An in-depth experimental evaluation obtained from live experiments with various types of group membership changes over both local- and wide-area networks. These results provide valuable insights into the protocols' scalability and practicality. Our experiments show that, in practice, the actual costs of group key management cannot be trivially extrapolated from the theoretical analysis (see Section 6).
—A taxonomy of application scenarios for secure group communication systems and a mapping between broad application classes and appropriate group key management protocols.

Organization: The rest of this paper is organized as follows. First, we overview related work in Section 2. Then, in Sections 3 and 4 we present Secure Spread, the framework we used in our experiments and briefly describe the five key agreement protocols we evaluate. Next, we analyze the conceptual costs (in Section 5) of these protocols and present performance results gathered on both local- and wide-area networks (in Section 6). We conclude in Section 7 with the discussion of various peer group applications and appropriate key management techniques.

2. RELATED WORK

Our work is closely related to two distinct areas: group key management and reliable group communication. This section provides an overview of related work in each of the two areas.

2.1 Group Key Management

As noted 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. At the first glance, it might appear that a typical collaborative group scenario is as follows: a group forms, functions for some time, and then dissolves itself. If this were 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 because the protocol would be invoked only once or very infrequently in order to rekey the group. However, a typical collaborative group is formed incrementally and its population can mutate throughout its lifetime due either to members joining and leaving or to network connectivity changes.

We begin with the STR protocol proposed by Steer et al. in 1989 [Steer et al. 1990] and originally aimed at teleconferencing. As will be seen later, STR is well suited for adding new members as it takes only two rounds and two modular exponentiations. However, member exclusion (rekeying following a member leave event) is relatively inefficient. Burmester and Desmedt [1994] 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 recompute the group key for any membership change with a constant CPU cost. The distribution in computation is obtained at the cost of using $2n$ broadcast messages which is expensive on a wide-area network. Tzeng and Tzeng proposed an authenticated key agreement scheme based on secure multiparty computation [Tzeng and Tzeng 2000]. 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 PFS.

Steiner et al. addressed dynamic membership issues [Steiner et al. 2000] in group key agreement as part of developing a family of GDH protocols based on straightforward 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 TGDH protocol, which is more efficient than GDH in both communication and computation [Kim et al. 2000].

Kronos [Setia et al. 2000] has performed an in-depth performance analysis on CKD protocols. The authors show that refreshing group key periodically by aggregating membership events provides better efficiency than refreshing

group key for every membership event. In general, this can be true for peer group scenario. However, the former provide weaker security than the latter. Note that our focus is on group key management over peer group, while their focus is on centralized key management.

2.2 Reliable Group Communication

Reliable group communication systems in LANs have a solid history beginning with ISIS [Birman and Renesse 1994] and followed by more recent systems such as Transis [Amir et al. 1992], RMP [Whetten et al. 1994], Totem [Amir et al. 1995], and Horus [Renesse et al. 1996]. These systems explored several different group communication models such as Virtual Synchrony [Birman and Joseph 1987] and Extended Virtual Synchrony [Moser et al. 1994]. More recent work in this area focuses on scaling group membership to wide-area networks [Anker et al. 1998; Keidar et al. 2000].

Research in securing group communication is relatively new. Previous implementations of group communication systems that focus on security are the Rampart system at AT&T [Reiter 1994], the SecureRing [Kihlstrom et al. 1998] project at UCSB, and the Horus/Ensemble work at Cornell [Rodeh et al. 2001, 2002]. Rampart [Reiter 1994] is the first system providing atomic and reliable services in a model where some servers can get corrupted. The system builds group multicast protocols over a secure group membership protocol. 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 system is the state of the art in secure reliable group communication. It allows application-dependent trust models and optimizes certain aspects of the group key generation and distribution protocols. Ensemble achieves data confidentiality by using a shared group key obtained by means of group key distribution protocols. In comparison with our approach, although efficient, the scheme does not provide forward secrecy, key independence and PFS.

Some other approaches focus on building highly configurable dynamic distributed protocols. Cactus [Hiltunen and Schlichting 1996] is a framework that allows the implementation of configurable protocols as composition of microprotocols. Survivability of the security services is enhanced by using redundancy [Hiltunen et al. 2001].

Another toolkit that can be used to build secure group oriented applications is Enclaves [Gong 1997]. It provides group control and communication, and data confidentiality using a shared key. The group utilizes a CKD scheme where a group leader selects a new key every time the group changes and securely distributes it to all members of the group. The main drawback of the system is that it does not address failure recovery when the leader fails.

Antigone [McDaniel et al. 1999] is a framework aimed to provide mechanisms which allow flexible application security policies. It implements group rekeying mechanisms in two flavors: (1) session rekeying whereby all

members receive a new key, and (2) session key distribution whereby the session leader transmits an existing session key. Both schemes have problems: distributing the same key when the group membership changes negates any notion of PFS, while the session rekeying mechanism cannot recover from a leader's failure.

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 key agreement primitives provided by the Cliques key management 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 [Amir and Stanton 1998] is a group communication system for wide- and local-area networks. It provides reliability and message ordering (FIFO, causal, agreed/total ordering) as well as a membership service. The toolkit supports two different semantics: Virtual Synchrony [Fekete et al. 1997; Schultz 2001] and Extended Virtual Synchrony [Moser et al. 1994]. In this paper, and for our implementation, we use only the former.

Spread consists of a server and a library linked with the application. The process and server memberships correspond to the model of light-weight and heavy-weight groups [Floyd et al. 1997]. This approach amortizes the cost of expensive distributed protocols, since these protocols are executed only by a relatively small number of servers, as opposed to (a much larger number of) all clients.

Spread operates in a many-to-many communication paradigm, where each member of the group can be both a sender and a receiver. Although designed to support small- to medium-size groups, Spread can accommodate a large number of different collaboration sessions, each spanning the Internet. Spread scales well with the number of groups used by the application without imposing any overhead on network routers. The Spread toolkit is publicly available and is being used by several organizations for both research and practical projects. The toolkit supports cross-platform applications and has been ported to several Unix platforms as well as Windows and Java environments.

3.2 Cliques Toolkit

Cliques is a cryptographic toolkit that supports a menu of key management techniques for dynamic peer groups. It performs all computations required to achieve a shared key in a group and is built atop the popular OpenSSL library [OpenSSL Project team 1999]. The toolkit assumes the existence of a reliable group communication platform to transport and order protocol messages as well as to maintain group membership.

Currently, Cliques includes five group key agreement protocols: GDH, CKD, TGDH, STR, and BD. All these protocols provide key independence and PFS. We briefly overview each protocol below and discuss them in more detail in Section 4.

GDH is a protocol based on group extensions of the two-party Diffie–Hellman key exchange [Steiner et al. 2000] and provides fully contributory key agreement. GDH is fairly computation-intensive, requiring $O(n)$ cryptographic operations for each key change. It is, however, bandwidth efficient. *CKD* is a centralized key distribution technique with the key server dynamically chosen among the group members [Amir et al. 2004]. The key server uses pair-wise 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 GDH technique [Kim et al. 2000, 2004b]. TGDH is efficient in terms of computation as most membership changes require $O(\log n)$ cryptographic operations. *STR* [Kim et al. 2001, 2004a] is a special case of TGDH with a so-called "skinny" or imbalanced tree. It is based on the protocol by Steer et al. [1990]. STR is more efficient than the above protocols in terms of communication; whereas, its computation costs are comparable to those of GDH and CKD. BD is another variation of GDH proposed by Burmester and Desmedt [1994]. It is efficient in computation, requiring a constant number of exponentiations (three) upon any membership change. However, communication costs are significant.

As far as security, only protection against outside adversaries (both passive and active) is considered in Cliques. In this model, any entity who is not a current group member is considered an outsider.[1] Attacks emanating from within the group are not considered, as the focus is on the secrecy of group keys and the integrity of group membership. Consequently, insider attacks are not germane because a malicious insider can always reveal the group key and/or its own private key(s), thus allowing for fraudulent membership.

All of the aforementioned protocols are proven secure with respect to passive outsider (eavesdropping) attacks [Steiner et al. 2000; Kim et al. 2000, 2001; Burmester and Desmedt 1994]. 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 aiming to impersonate a group member are prevented by the use of public key signatures: 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 identifying the particular protocol instance (group key epoch).

3.3 Secure Spread Library

Secure Spread [Amir et al. 2001] library is a client library providing data confidentiality and integrity, in addition to reliable and ordered message

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

dissemination and membership services. The library uses Spread as communication infrastructure and Cliques library primitives for group key management. A client who wants to communicate securely is required to connect to a server and then join a group before proceeding with the communication. The library provides an API allowing a client to connect/disconnect to a server, join/leave a group, and send/receive messages.

The Virtual Synchrony model enables the use of a shared group view-specific key to encrypt client data, since the receiver is guaranteed to have the same view as the sender and, hence, the same key.

The main functionality of Secure Spread is to encrypt/decrypt client data using a group shared key and to ensure the freshness of this key. The group key is computed using a key agreement protocol and refreshed every time the group membership changes. Secure Spread detects group membership changes and initiates the execution of a group key agreement protocol, ensuring its correct execution. When it terminates, Secure Spread notifies the application about the membership change and the new key.

The architecture of Secure Spread allows using different key agreement algorithms for different groups. A client can be a member of different groups, each group managing keys using its own key agreement protocol. The initial creator of the group decides on the specific key agreement protocol for the group. Once selected, the key agreement protocol cannot be modified throughout the group's lifetime.

Secure Spread currently supports all five key agreement protocols supported by the Cliques library: BD, CKD, GDH, STR, and TGDH. This allows us to evaluate and compare these protocols in the same common framework. The selection of the key management protocol is a component of the group policy. When such a policy is in place, group members must reach agreement on the policy.

We note that our framework might not be the best architecture for peer group applications, as it addresses only client security. Therefore, additional measures must be taken to protect the server-to-server communication. In this paper, we are not trying to determine the right (or the best) architecture for secure group communication in general. We acknowledge that it is a very important topic and we addressed it in previous work [Amir et al. 2003; Nita-Rotaru 2003].

4. KEY AGREEMENT PROTOCOLS IN SECURE SPREAD

In this section, we overview the key agreement protocols currently supported in Secure Spread which are the subject of our performance evaluation.

With the exception of the BD protocol, all other protocols we consider optimize their cost by taking advantage of the nature of the group change. A group can change because a single member joins or leaves, or a set of members leave or join simultaneously. The latter can be caused by changes in network connectivity, such as network partitions and network reheals. We refer to such events as partition and merge. For brevity's sake, we only describe how each of the five protocols handle merge and partition events, since single-member join and leave events can be seen as special cases of merge and partition, respectively.

Assume that k members are added to a group of size n.
Step 1: M_n generates a new exponent $r'_n \in \mathbb{Z}_q$, computes $g^{r_1 \ldots r_{n-1} r'_n} \bmod p$, 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 $r_{n+j} \in \mathbb{Z}_q$, computes $g^{r_1 \ldots r_n{}' \ldots r_{n+j}} \bmod p$ 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 $g^{r_1 \ldots r_n{}' \ldots r_{n+k-1}/r_i} \bmod p$ and sends it back to M_{n+k}.
Step $k+3$**:** After collecting all the responses M_{n+k} generates a new exponent r_{n+k}, produces the set $S = \{g^{r_1 \ldots r_n{}' \ldots r_{n+k}/r_i} \bmod p | \forall i \in [1,\ n+k-1]\}$ 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 $K = (g^{r_1 \ldots r_n{}' \ldots r_{n+k}/r_i})^{r_i} \bmod p = g^{r_1 \ldots r_n{}' \ldots r_{n+k}} \bmod p$.

Fig. 1. GDH—Merge protocol.

4.1 GDH Protocol

GDH is a contributory group key agreement protocol which generalizes upon the well-known 2-party Diffie–Hellman key exchange. The basic idea is that the shared key is never transmitted over the network (in the clear or otherwise). Instead, a list of partial keys, one for each member, is generated and sent. Upon receipt, each member uses its partial key to compute the group secret. One particular member of the 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 runs as follows. When a merge event occurs (see Figure 1), the current controller generates a new key token by refreshing its contribution to the group key and passes the token to one of the new members. When the new member receives the 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 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 controller. The new controller collects all the factor-out tokens and adds its own contribution to each. Every thus modified factor-out token (to which the new controller added its contribution) represents a partial key. Once all the partial keys are computed, the list of partial keys is broadcast to the group. Every member obtains the group key by factoring in its contribution into the corresponding partial key in the broadcasted list.

When some of the members leave the group (Figure 2), the 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

[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.

Assume that a set L of members is leaving a group of size n.
Step 1: The controller M_d generates a new exponent $r_d' \in \mathbb{Z}_q$, produces the set $S = \{g^{r_1 \cdots r_d{}'/r_i} \bmod p | M_i \notin L\}$ 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 $K = (g^{r_1 \cdots r_d{}'/r_i})^{r_i} = g^{r_1 \cdots r_d{}'} \bmod p$

Fig. 2. GDH—Partition protocol.

Assume that k members are added to a group of size n. M_1 is the group controller.
Step 1: M_1 selects random $r_1 \in \mathbb{Z}_q$ (this selection is performed only once),
$M_1 \longrightarrow \{M_{n+j} \mid j \in [1,k]\} : g^{r_1} \bmod p$
Step 2: For each $j \in [1,k]$, M_{n+j} selects random $r_{n+j} \in \mathbb{Z}_q$,
$M_1 \longleftarrow M_{n+j} : g^{r_{n+j}} \bmod p$
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.

Fig. 3. CKD—Merge protocol.

then compute the shared key. Note that if the current controller leaves, the last remaining member becomes the controller of the group.

4.2 CKD Protocol

The CKD protocol is a simple group key management scheme. In CKD, the group key is not contributory; it is always generated by the current controller.[3]

The 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 regular group member) remain in the group. The controller is always the oldest member of the group. (The oldest member is picked in order to reduce expensive establishment of pair-wise secure channels necessary upon each controller change.) Whenever group membership changes, the controller generates a new secret and distributes it to every member encrypted under the long-term pair-wise key it shares with that member. In case of a join or merge event (see Figure 3), the controller initially establishes a secure channel with each incoming member.

Note that an efficient symmetric cipher can be used to securely distribute the group key. However, the security would differ from that of our group key agreement protocol which relies solely on the Decision Diffie–Hellman assumption [Boneh 1998] and the Discrete Logarithm problem [Menezes et al. 1996]. Therefore, to provide equivalent level of security, we encrypt the group key via modular exponentiation.

When a partition occurs (see Figure 4), in addition to refreshing and distributing the group key, the controller discards the long-term key it shared with each leaving member. A special case is when the controller itself leaves the

[3]We use the term *current* to capture that, even in CKD, a controller is a regular member which can fail or become partitioned out. Thus, the controller role must be reassigned to another member.

Assume that a set L of members is leaving a group of size n.
Step 1: The controller M_1 selects a random group secret K_s and computes $M_1 \longrightarrow M_i : {K_s}^{g^{r_1 r_i}} \bmod p$, $M_i \notin L$.
Step 2: From the broadcast message, every member can compute the group key.

Fig. 4. CKD—Partition protocol.

group. In this case, the oldest remaining member becomes the new controller. Significant additional cost is incurred, since, before distributing the new key, the new controller must first establish a secure channel with every remaining group member.

4.3 TGDH Protocol

TGDH is an adaptation of key trees [Wallner et al. 1999; Sherman and McGrew 2003] in the context of fully distributed, contributory group key agreement. TGDH computes a group key derived from individual contributions of all group members using a logical binary key tree [Kim et al. 2000, 2004b].

The key tree is organized as follows: each node $\langle l, v \rangle$ is associated with a key $K_{\langle l,v \rangle}$ and a corresponding blinded key $BK_{\langle l,v \rangle} = g^{K_{\langle l,v \rangle}} \bmod p$. The root is associated with the group and each leaf—with a distinct member. The root key represents the group key shared by all members, and a leaf key represents the random contribution by of a group member. Each internal 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. Every member knows all keys on the path from its leaf node to the root as well as all blinded keys of the entire key tree.

The protocol relies on the fact that every member can compute a group key if it knows all blinded keys in the key tree.[4]

Following every group membership change, each member independently and unambiguously modifies its view of the key tree. Depending on the type of the event, it adds or removes tree nodes related to the event, and invalidates all keys and blinded keys related with the affected nodes (always including the root node). As a result, some nodes may not be able to compute the root key by themselves. However, the protocol guarantees that at least one member can compute at least one new key corresponding to either an internal node or to the root. Every such member (called a sponsor) computes all keys and blinded keys as far up the tree as possible and then broadcasts its key tree (only blinded keys) to the group. If a sponsor cannot compute the root key, the protocol guarantees the existence of at least one member which can proceed further up the tree, and so on. After at most two rounds (in case of a merge) or $\log(n)$ rounds (in case of a worst-case partition), the protocol terminates with all members computing the same new group (root) key.

After a partition, the protocol operates as follows. First, each remaining member updates its view of the tree by deleting all leaf nodes associated with the

[4]Actually, this is sufficient but not necessary: a member does not need to know **all** blinded keys to compute the root key, however, as discussed in [Kim et al. 2000], knowing all is useful for handling merge events.

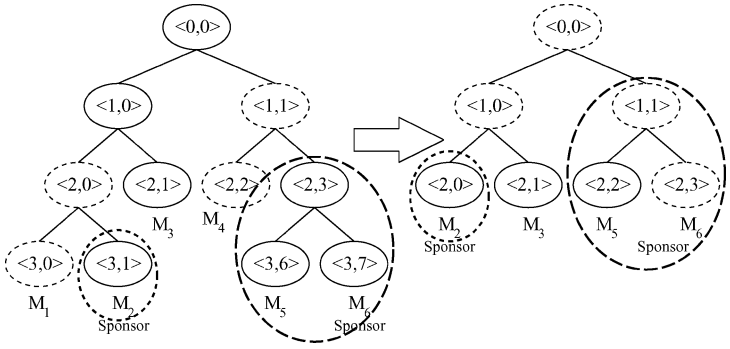

Fig. 5. TGDH—Partition operation.

partitioned members and (recursively) their respective parent nodes. To prevent reuse of old group keys, one of the remaining members (the shallowest rightmost sponsor) changes its key share. Each sponsor computes all keys and blinded keys as far up the tree as possible and then broadcasts its view of the key tree with the 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 new group key. Figure 5 shows an example where members M_1 and M_4 are partitioned out of the group.

When a merge happens, the sponsor (the rightmost member of each merging group) broadcasts its tree view to the merging subgroup after refreshing its key share (leaf key) and recomputing all affected blinded keys. Upon receiving this message, all members uniquely and independently determine the merge position of the two trees.[5]

As described above, all keys and blinded keys on the path from the merge point to the root are invalidated. The rightmost member of the subtree rooted at the merge point becomes the sponsor. The sponsor computes all keys and blinded keys and broadcasts the key tree (with only blinded keys) to the group. All members now have the complete set of new blinded keys which allows them to compute all keys on their key path. Figure 6 shows an example of the merge protocol. Members M_6 and M_7 are added to the group consisting of members $M_1, M_2, \ldots, M_5$.

4.4 STR Protocol

The STR protocol [Steer et al. 1990; Kim et al. 2004a] is an "extreme" version of TGDH with the underlying key tree completely unbalanced or stretched out. In other words, the height of the key tree is always $(n-1)$, as opposed to (roughly) $\log(n)$ in TGDH. All other features of the key tree are the same as in TGDH.

After a partition, the sponsor is defined as the member corresponding to the leaf node just below the lowest leaving member. After deleting all leaving nodes (see Figure 7), the sponsor refreshes its key share, computes all (key, blinded key) pairs up to the level just below the root node. Finally, the sponsor

[5]We choose the merge node as the rightmost "shallowest" node, which does not increase the tree height. For more details see Kim et al. [2000].

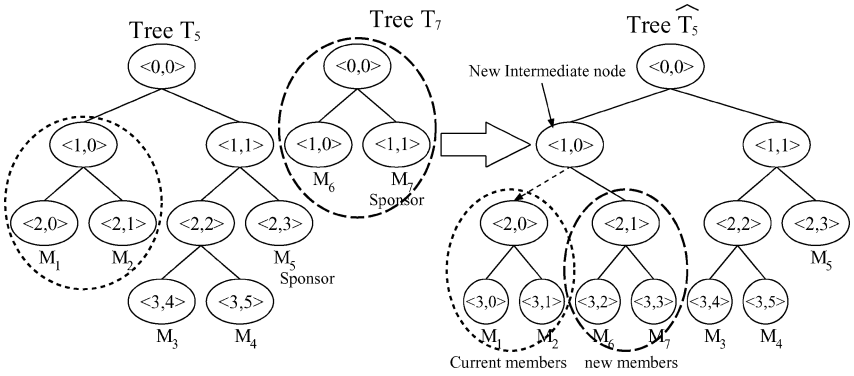

Fig. 6. TGDH—Merge operation.

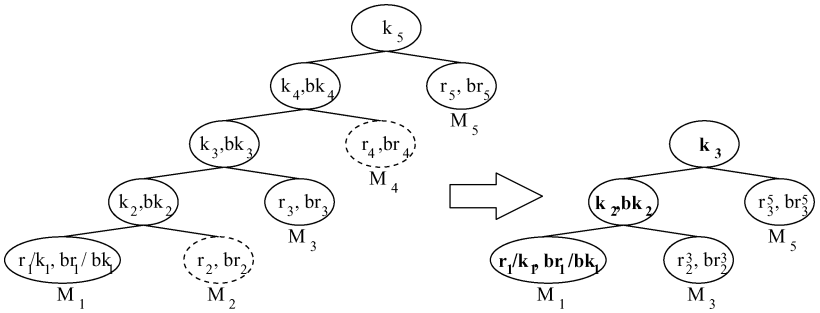

Fig. 7. STR—Partition operation.

broadcasts the updated key tree thus allowing each member to compute the new group key.

STR merge runs in two rounds. In the first round, each sponsor (topmost leaf node in each of the two merging tree) first refreshes its key share and computes the new root key and root blinded key. Then, the sponsors exchange their respective key tree views containing all blinded keys. The topmost leaf of the larger tree becomes the sole sponsor in the second round in the protocol (see Figure 8). Using the blinded keys from the key tree it received in the first round, the sponsor computes every (key, blinded key) pair up to the level just below the root node. It then broadcasts the new key tree to the entire group. All members now have the complete set of blinded keys which allows them to compute the new group key.

4.5 BD Protocol

Unlike other protocols discussed thus far, the BD protocol [Burmester and Desmedt 1994] is stateless. Therefore, the same key management protocol is performed regardless of the type of group membership change. Furthermore, BD is completely decentralized and has no sponsors, controllers, or any other members charged with any special duties.

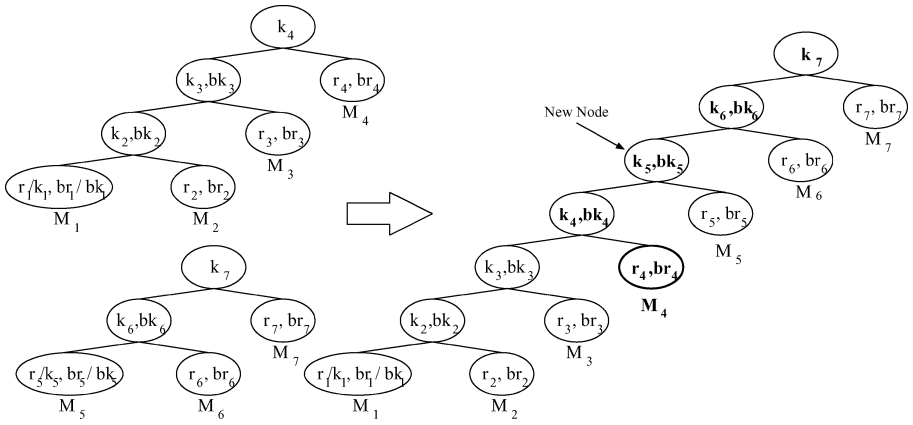

Fig. 8. STR—Merge operation.

Assume a group of size n.
Step 1: Each member M_i selects random $r_i \in \mathbb{Z}_q$, computes $Z_i = g^{r_i} \bmod p$ and broadcasts the message to the group.
Step 2: Each member M_i, after receiving Z_{i-1} and Z_{i+1}, computes $Xi = (Z_{i+1}/Z_{i-1})^{r_i} = g^{r_{i+1}r_i - r_i r_{i-1}} \bmod p$ and broadcasts it to the group.
Step 3: Each member M_j, after receiving all $X_i, i \neq j$, computes $K = K_j = (Z_{j-1})^{nr_j} X_i^{n-1} ... X_{i+(n-2)} = g^{r_1 r_2 + r_2 r_3 + ... + r_{n-1} r_n} \bmod p$.

Fig. 9. BD protocol.

The main idea in BD is to distribute the computation among members, such that each member performs only three exponentiations. This is performed in two communication rounds, each consisting of n broadcasts. Figure 9 depicts the protocol.

5. COST EVALUATION

In this section, we estimate and analyze the costs of the five protocols presented above. We first evaluate the time to compute a new group key when a membership change occurs. Four types of events can lead to a change in group membership. The first two are the single-member join and leave events. A join is always voluntary, whereas, a leave can be voluntary, forced (by other members), or involuntary, for example, due to a processor crash or a disconnect. For the purpose of this discussion, we do not differentiate between the three possible causes of a leave event. (We assume that, regardless of the cause, the group key must be changed.)

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

Table I. Communication Cost

Protocols		Rounds	Messages	Unicast	Multicast
GDH	Join	4	$n+3$	$n+1$	2
	Leave	1	1	0	1
	Merge	$m+3$	$n+2m+1$	$n+2m-1$	2
	Partition	1	1	0	1
TGDH	Join, merge	2	3	0	3
	Leave	1	1	0	1
	Partition	h	$2h$	0	$2h$
STR	Join	2	3	0	3
	Leave, partition	1	1	0	1
	Merge	2	3	0	3
BD	Join	2	$2n+2$	0	$2n+2$
	Leave	2	$2n-2$	0	$2n-2$
	Merge	2	$2n+2m$	0	$2n+2m$
	Partition	2	$2n-2p$	0	$2n-2p$
CKD	Join	3	3	2	1
	Leave	1	1	0	1
	Merge	3	$m+2$	m	2
	Partition	1	1	0	1
	Controller leave	3	$3n-6$	$2n-4$	2

We are interested in two cost aspects: (1) communication in number of protocol rounds as well as number and type of messages, and (2) computation in number of fine-grained integer operations (such as modular exponentiations) as well as coarser-grained operations (such as signature generation and verifications). Although the cost of communication in modern high-speed LANs can appear negligible in comparison with the cost of, say, modular exponentiations, we consider it important since it becomes more meaningful even in LANs for protocols that trade off lower computation for higher communication costs. Of course, communication cost is very important in high-delay networks (e.g., WANs or low-bandwidth wireless). Because of the distributed nature of group communication systems, we consider only serial computation cost.[6] Thus, we stress that the number of cryptographic operations reflected in Table II is not the sum total for all participants.

Tables I and II summarize the communication and the computation costs for the five protocols we consider. 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] TGDH costs depend on the tree height, the balanceness of the key tree, the insertion point of the joining tree (or node) and the location of the leaving node(s). The results we present for TGDH are conservative, we compute the

[6]Computation that can be processed in parallel is collapsed accordingly.

[7]Instead of always maintaining a fully balanced key tree, TGDH uses a best-effort approach: it tries to balance the tree only upon additive events. Prior experiments [Kim et al. 2000] demonstrated that, in the presence of random events, the height of the key tree remains smaller than $2\log n$. The tree can be better balanced with the AVL tree management described in Rodeh et al. [2002]. However, AVL incurs a higher communication cost for subtractive events.

Table II. Computation Cost

Protocols		Exponentiations	Signatures	Verifications
GDH	Join	$n+3$	4	$n+3$
	Leave	$n-1$	1	1
	Merge	$n+2m+1$	$m+3$	$n+2m+1$
	Partition	$n-p$	1	1
TGDH	Join, merge	$\frac{3h}{2}$	2	3
	Leave	$\frac{3h}{2}$	1	1
	Partition	$3h$	h	h
STR	Join	7	2	3
	Leave, partition	$\frac{3n}{2}+2$	1	1
	Merge	$3m+4$	2	3
BD	Join	3	2	$n+3$
	Leave	3	2	$n+1$
	Merge	3	2	$n+m+2$
	Partition	3	2	$n-p+2$
CKD	Join	$n+2$	3	3
	Leave	$n-2$	1	1
	Merge	$n+2m$	3	$m+2$
	Partition	$n-p-1$	1	1
	Controller leaves	$2n-3$	$2n-2$	n

worst-case costs for the TGDH. We provide details on this cost will differ from average case cost when we discuss each group event.

The number of modular exponentiations in STR after a leave event is determined by the location of the lowest leaving leaf node. To be fair, we compute the average cost, i.e., for the case when lowest leaving leaf is in the middle of the tree (at level $\frac{n}{2}$). All other protocols, except TGDH and STR, show exact costs.

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.

The BD protocol has a hidden computation cost not reflected in Table II. In Step 3 (see Figure 9), BD incurs $n-1$ modular exponentiations with small exponents—ranging between p and 2 (where p is the modulus)—as well as $(n-1)$ modular multiplications. Although a single small exponentiation is negligible, the sum of $(n-1)$ exponentiations is clearly not. Because of this hidden cost, it is difficult to compare the computational overhead of BD to that of 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 messages is BD, which uses n broadcast messages for each round. The rest of the 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 (in terms of group size) number of exponentiations. TGDH is relatively efficient, scaling logarithmically in the number of exponentiations. We note that best case for TGDH for join is when the node will join at the root, while average case will be more expensive. STR has a constant number of exponentiations and BD requires the fewest exponentiations, but has the aforementioned hidden cost.

Leave: Table I shows that BD is most expensive for a leave operation in terms of communication. The cost order between 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 because of its logarithmic scaling with group size. Best case for TGDH leave is when the node leaving the group is positioned near the root.

STR, GDH, and CKD scale linearly with the group size. We note that the cost of CKD is actually higher than the cost reflected in Tables I and II, since in case of the controller leaving, the new controller must establish new secure channels with all group members. As in join, BD has a hidden cost thus making it difficult to compare it with the other protocols.

Merge: We first consider the communication cost. In terms of communication rounds, GDH scales linearly with the number of merging members (smaller group is assumed to merge into larger one). Whereas, BD, CKD, STR, and TGDH are more efficient as each requires a constant number of rounds. Since BD has n messages for each round and CKD uses $(m + 1)$ messages, STR and TGDH are the most communication efficient for merge events.

Looking at the computation, it seems that BD has the lowest cost amounting to only three exponentiations. However, the impact of the number of small exponent exponentiations is difficult to evaluate. TGDH scales logarithmically with the group size; it is clearly more efficient than STR, CKD, and GDH which all scale linearly with both the group size and the number of new members. The comparison is done best on worst case of TGDH. Note that for the average case, the performance of TGDH will be better.

Partition: Table I shows that GDH, STR, and CKD 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 upper bounded by the resulting tree height. As before, computationwise it is difficult to compare BD with others due to its hidden cost in Step 3. As TGDH requires a logarithmic number of exponentiations, it has a lower computation cost than GDH, STR, and CKD which scale linearly with the group size.

6. EXPERIMENTAL RESULTS

In this section, we present, compare, and evaluate experimental costs of the five protocols discussed above. We do so in the context of two network environments: high-speed LAN and high-delay WAN.

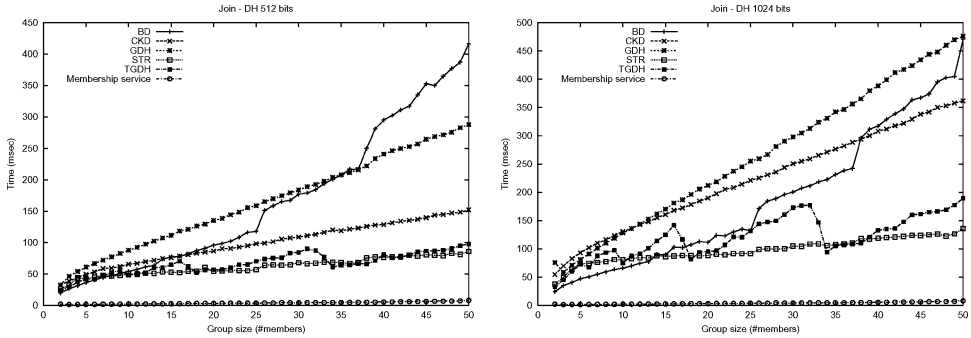

Fig. 10. Join—Average time (LAN).

We measure the "total elapsed time" from the moment the group membership event happens until the time when the group key agreement finishes and the application is notified about the group change and the new key. For all events initiated by clients (voluntary events), this time includes all communication and computation costs of the underlying key agreement protocol as well as the cost of the membership service provided by the group communication system. In other words, the total time represents the actual delay experienced by an application. For all events triggered by network connectivity changes (involuntary events), the total time includes all the same costs as for voluntary events. However, it does not include the time needed by Spread to detect that a network connectivity change occurred.

6.1 Experimental Results in LAN

We begin by presenting performance results of the five protocols discussed above in a LAN setting. We first describe the experimental testbed and discuss the particulars of the scenarios we considered. Then, we present results for join, leave, partition, and merge operations.

6.1.1 *Testbed and Basic Parameters.* We used an experimental testbed consisting of a cluster of thirteen 666 MHz Pentium III dual-processor PCs running Linux. A Spread server runs on each PC and group members are uniformly distributed across all 13 PCs. Therefore, more than one process can be running on a single machine, which is frequent in many collaborative applications.

For communication services we used FIFO and Agreed ordering. 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 to 0.92 ms for a group size ranging from 2 to 50 members. Also, in a scenario (similar to a single round in the BD protocol) where each member of the group sends an Agreed broadcast message and receives all the $(n - 1)$ similar messages from all other members, the average cost is about 2 ms for a group of 2 members and about 21 ms for a group of 50. The cost of the membership service (see Figures 10 and 11) is negligible with respect to the key agreement overhead, varying between 2 and 8 ms for a group between 2 and 50 members.

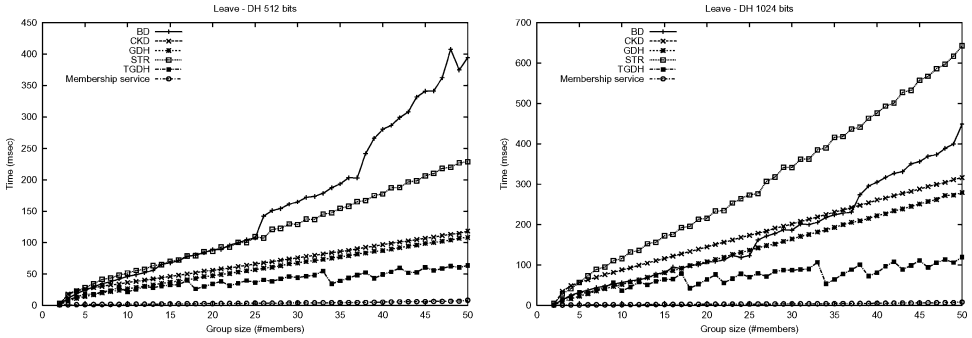

Fig. 11. Leave—Average time (LAN).

The results we present are the average time needed to recompute a key when the group changes. The size of the group determines indirectly the size of some of the messages part of the group key agreement (the size of the tree or the list structures used by TGDH, STR, and GDH protocols). For the group size we considered the size of the message is relatively small, therefore will not affect significantly the results. More details about Spread performance can be found in Amir et al. [2004].

Message origin and data authentication are obtained via RSA [Rivest et al. 1978] signatures. We chose RSA because it allows for very inexpensive signature verification and all group key agreement protocols described in this paper rely heavily on source authentication, that is, all protocol messages must be verified by all receivers. If all processes are located on different CPUs, verification is performed in parallel. In practice, however, a CPU may have multiple group processes and expensive signature verification (e.g., as in DSA [NIST 2000]) noticeably degrades performance.

We used 1024-bit RSA signatures with the public exponent of 3, to reduce verification costs, albeit, a quasi-standard in RSA parameter selection is to use 65, 537 as the public exponent. This is because (1) there are no security risks in using 3 as a public exponent in RSA signature scheme [Boneh 1999], (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 ms, respectively.[8]

For short-term group keys, 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 ms for a 512- and a 1024-bit modulus, respectively.

6.1.2 *Test Scenarios.* The cost of each protocol depends on a number of factors. We tried to design our tests such that we take into account all the factors, keeping experiments as similar and as simple as possible.

Some protocols maintain specific data structures: GDH and CKD maintain a list, TGDH, and STR—a tree, and BD is stateless. Communication and

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

computation costs vary depending on the operations performed on these data structures, For example, the costs of GDH and BD do not depend on the position of the joining or leaving member, all leave and all join operations cost the same in these protocols, while CKD can be expensive for a leave event if the leaving member is the current controller. For STR, the computation cost of a leave depends on the position (in the tree) of the leaving member. The cost of TGDH depends on many factors: location of the leaving or joining node, tree height, and the balanceness of the tree.

Based by the above observations, we designed our tests as follows. For CKD, we factor in the $1/n$ probability of the group controller leaving the group. Since the estimated cost presented for STR leave is the average cost, we tested the average case: the leaving member is the leaf node at height $n/2$, in 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 balanceness 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 must then be conducted on this random tree. However, such tests are very difficult to perform. Instead, we chose a less complex 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 would be lower. This is because a random tree is more sparse than an artificially balanced tree. Therefore, the joining point would be located closer to the root in the former. The best case for the join is when the joining node is added in tree at the root. On the other hand, leave cost can be greater since the height of a random tree would normally exceeds that of a balanced one. The way we add and remove clients from the group captures both the best case and the worst case.

In cases of partition and merge we ran the same scenarios for all protocols: we partition the group into two subgroups of the same size, and then merge them back. STR and TGDH exhibit a clustering effect (see Section 6.1.5) when partition events are repeated in the same configuration. To emphasize this property, we partitioned the group into two subgroups, merge it back, and then partition it again.

6.1.3 *Join Results.* Figure 10 shows the total average time for a group to establish secure membership following a join event. From the left-side graph (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 (roughly less than 7). Recall that BD involves only three full-blown exponentiations as opposed to STRs 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 9). 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 we use a 512-bit Diffie–Hellman modulus. For 1024-bit modulus, GDH is the worst due to the sharp increase in the cost of modular exponentiation.

Another interesting observation about BDs performance 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, all machines we used are dual processor, thus, up to a group size of 26 it can be assumed that there is one client per processor. For other protocols, this behavior is less obvious, since the most costly tasks are performed by a single member (controller or sponsor).

The right-side graph in Figure 10 (1024-bit modulus) does not show the same deterioration in BD. In fact, BD remains the best for groups up to 14 members. This is because the cost of exponentiations rises sharply as we go 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 the meantime, other protocols are more affected, since their cost is mainly determined by the number of exponentiations.

In both graphs, TGDH and STR are fairly close with the latter performing slightly better. Although the numbers in Table II show constant cost for STR, the measured cost increases slightly because a CPU may experience an increasing number of processes as the group increases, and other 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. We also note that the "saw-like" aspect of the TGDH graph is due to the fact that the tree is artificially balanced. The dip corresponds to the case when the group member joins at the root, while the highest cost corresponds to the case when the node joins at a leaf.

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.

6.1.4 *Leave Results.* Figure 11 shows the average time for a group to establish secure membership following a leave event. In line with the prior analysis, TGDH outperforms the rest, requiring the fewest ($O(\log n)$) modular exponentiations. This sublinear behavior becomes more evident for a group size greater than 30. Note that, for a random tree, a leave would be more expensive, however, it will still remain less expensive than a leave in GDH [Kim et al. 2000], which is second best.

BD is the worst performer in 512-bit leave. Recall that, leave and join incur the same cost in BD. STR, CKD, and GDH exhibit linear increase in cost. CKD

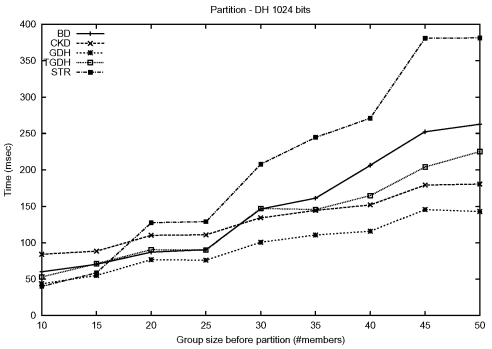

Fig. 12. Partition—Average time (LAN).

and GDH costs are very close, while STR's linear factor is $(2n)$ which makes the graph's slope steeper. In addition, while STR, TGDH, CKD, and GDH require only one broadcast, BD uses two rounds of n messages each.

In case of 1024-bit modulus, STR is the most expensive since it involves the most exponentiations. The cost of TGDH roughly doubles from that in the 512-bit case, however, TGDH remains the leader. Surprisingly, BD 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 low cost of RSA signature verifications and the commensurately small number of full-blown 1024-bit exponentiations in BD.

6.1.5 *Partition Results.* Figure 12 presents measurements of the elapsed time to establish a new secure membership when a group is split into two smaller groups of roughly equal size. The event is generated by simulating a network partition between the servers. The evaluated time does not include the time needed by Spread servers to detect a network partition.

GDH clearly outperforms all others. The main reason that STR and TGDH are very costly in this scenario is because they need to rebuild the key tree. STR is the most expensive because it also scales linearly in computation with respect to the new group size. The CKD cost is higher than we estimated in Table II. This is because, when the group is partitioned into two subgroups, the controller ends up in one of the partitions, while, in the other partition, the new controller needs to establish pair-wise secure channels with all remaining members.

TGDH and STR have an interesting advantage over other protocols in case of partition. Due to the tree structure, both TGDH and STR have a clustering effect (visible also in Figures 5 and 7) that basically decreases the cost of a partition when it occurs multiple times in the same configuration (statistical results show that this is a common case). To illustrate this feature, we ran the following experiment. We partitioned a group in two equal groups, such that, all odd-numbered members are in one group (G_1) and all even-numbered (in another (G_2)). We then merged the two groups and repeated the same partition event. In other words, the two partitions (P_1 and P_2) are identical. The time needed to establish a key and install secure membership in G_1 and G_2 after events

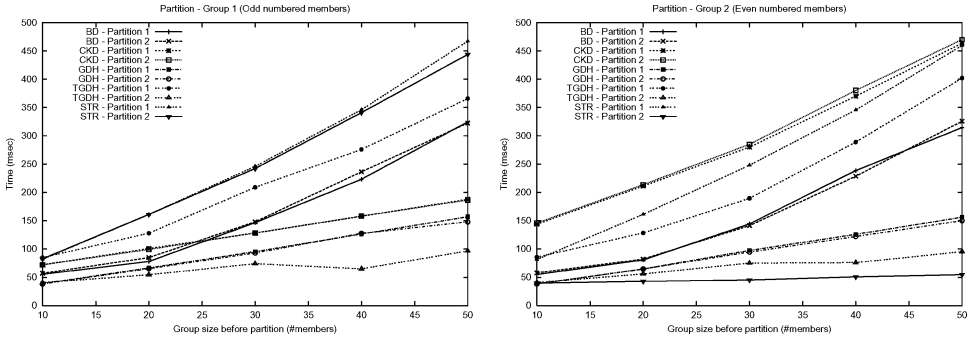

Fig. 13. Partition—Clustering effect.

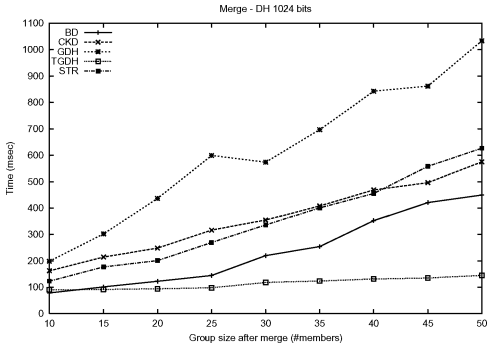

Fig. 14. Merge—Average time (LAN).

P_1 and P_2 are presented in Figure 13. For BD and GDH, the only important cost factor is the number of the remaining members. Hence, they take about the same time, since the cost of partition depends only on the group size and the number of leaving members. For CKD, there is a difference between the respective costs in G_1 and G_2, since in the latter the new controller needs to set up pair-wise secure channels.

Looking at STR, a cost decrease is evident for handling P_2 for G_2. This is because the merge event injected between P_1 and P_2 changed the structure of the key tree, and the key share change after P_2 happens at a higher level in the tree. However, there is no similar cost reduction for G_1 because the key share change takes place very low in the tree. For TGDH, the partition protocol may involve as many as $\log n$ rounds. When a partition heals, two previously separate groups are merged into a single key tree. However, they are still clustered along the lines of the first partition. When another partition happens on the same link, the partitioned members are no longer scattered across the leaves of the key tree. Therefore, subsequent partition on the same link take only one round to complete. This drastically improves the communication cost (and the number of signature computations).

6.1.6 *Merge Results.* Figure 14 presents the measurements of the time required to establish a new secure membership when two groups of about the

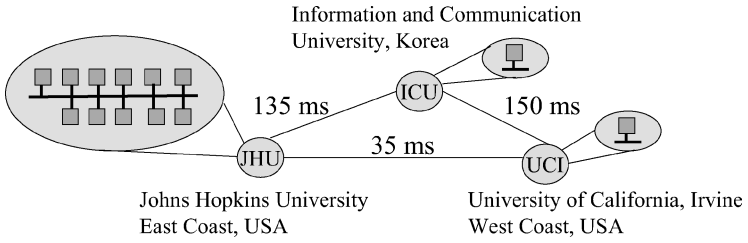

Fig. 15. WAN testbed.

same size merge. GDH is evidently the most expensive protocol as its computation cost scales linearly in the number of new members (in the smaller group), Moreover, GDH has the highest number of communication rounds, also related to the number of new members. CKD has fewer rounds, however, its computation cost is quite high since each new member needs to establish a secure channel with the controller. STR exhibits costs comparable to those of CKD owing to the fact that an STR tree actually resembles a list. Thus, when $n/2$ members are merged, the key share change happens very low in the tree and results in a heavy computational load. The high costs of STR and GDH, make BD look quite good in comparison. Nonetheless, balanced tree structure and small number of communication rounds make TGDH the best performer in this event class.

6.2 Experimental Results in WAN: An Extreme Case

In this section, we present experimental results obtained in a high-delay WAN environment. We first describe the testbed used in the WAN experiments and then present and discuss the actual results. We only conducted experiments for join and leave operations. Merge and partition were not included in our experiments because of the logistical difficulty in replicating the same partition/merge scenarios as used in our LAN setting. We provide an analysis of the behavior of merge and partition based on the experimental results for join and leave.

6.2.1 *Testbed and Basic Parameters.* Figure 15 shows the network configuration used in the WAN experiments. To achieve the same computation distribution as in the LAN experiments, we configured an experimental testbed of 13 PCs running Linux: ten 666 MHz Pentium III dual-processor PCs, one 1.1 MHz Athlon and one 930 MHz Pentium III PCs. The PCs were located as follows: 11 machines at Johns Hopkins University (JHU), Maryland, one machine at University of California at Irvine (UCI) and one at Information and Communications University (ICU), Korea. As before, members were uniformly distributed among the 13 PCs often with multiple group member processes running on a single PC. Each PC ran a Spread server. Approximate round-trip latencies as measured via the ping program are (in ms): JHU–UCI 70, UCI–ICU 300, and ICU–JHU 270. We emphasize that the determining factor in the performance of our communication infrastructure is the diameter of the network.

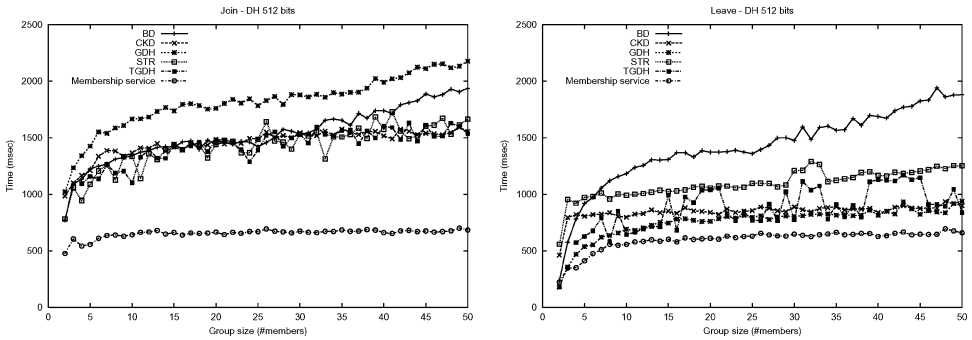

Fig. 16. Join and Leave—Average time (WAN).

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

We point out that, in a LAN, the cost of the group membership service provided by Spread (about 7 ms) is negligible in comparison with the key agreement overhead (hundreds of ms). However, the former cost becomes significantly higher in a WAN. As can be seen in Figure 16, the cost varies between 400 and 670 ms for a join and between 250 and 650 for a leave, for a group size ranging from 2 to 50.

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

6.2.2 *Join Results.* The left side of Figure 16 presents the average time results for a join event. 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, both communication and computation.

The first observation is that GDH performs significantly worse than others. The main difference between GDH and other protocols comes from communication. First, the number of rounds is greater than in other protocols, as shown in Table I. GDH requires 4 rounds while others require only 2.

Other protocols are more or less in the same range, with BD becoming more expensive for group sizes over 30, while STR and TGDH are very similar. It is interesting to note that STR and TGDH come closer to BD. This is mainly due to the communication aspect of the protocols. As evident from Table I and Sections 4.4 and 4.3, STR and TGDH each have two rounds and the first round consists of two "simultaneous" broadcasts. In our implementation, these broadcasts are not simultaneous, since, to achieve ordered message delivery, Spread

uses a mechanism, whereby a token is passed between participants and only the entity that has the token is allowed to send. Recall that our WAN setup has three wide-area sites: JHU, UCI, and ICU. (The cost of passing the token inside a site is significantly less than the cost of doing the same between sites.) Therefore, STR and TGDH costs—2 members sending broadcasts and all other members receiving them—is close to the cost of BD—n members broadcast and each member receives $(n - 1)$ messages.[9] BD deteriorates faster than other protocols due to the number of broadcast messages. Though CKD, has three rounds, two involve single-message unicasts. This helps CKD to remain competitive with respect to other protocols.

We can clearly conclude that communication costs of group key agreement (number of rounds and numbers of messages sent in each round) greatly influence the overall performance on a high-delay WAN.

6.2.3 *Leave Results.* Our leave results are shown on the right side of Figure 16. BD is evidently the most expensive protocol in our WAN setup, due to its two rounds of n broadcasts and its computation 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 in comparison to the other "contenders."

TGDH exhibits more dynamic behavior 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 PC. 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, and CKD would have almost identical costs.

6.3 Partition and Merge Results

Although merge and partition were not included in our experiments because of the logistical difficulty in replicating the same partition/merge scenarios as used in our LAN setting, it is relatively easy to guess the behavior of partition and merge for WAN from join and leave results for WAN.

The latter results clearly show that the number of rounds is the determinant factor of the overall cost. Therefore, in case of partitions, protocols with a small number of rounds (e.g. STR, GDH) will outperform other protocols. Note that the expensive computational cost of STR and GDH would not be seen easily as in the join or leave case for WAN. TGDH would have poor performance for partition mainly because it requires multiple rounds, while self-clustering effect would help amortize its expensive partition cost. For CKD, at least one partition needs to elect new controller, and then the new controller has to set up secure channels with all of the current members, which requires several communication rounds. The partition/merge cost of BD would not be

[9]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.

different from the join/leave cost, since the same protocol is used in all these cases.

Any merge takes at least two communication rounds. STR, TGDH, and BD achieve this bound, and therefore will be the most efficient for merge events. The performance of BD will deteriorate as the number of users increases as shown in Figure 16, since the probability of message loss increases as the group size grows.

6.4 Summary of Experimental Results

Our experiment results show some notable differences between theoretical and measured (experimental) complexity of the protocols:

—The importance of the communication costs: Our experiments clearly show that communication cost for group-oriented cryptographic protocols over long delay network can dominate the computational cost.

—Cost of simultaneous n broadcast messages: When designing group-oriented cryptographic protocols, most cryptographers focused on computational overhead and number of rounds (even the most recent results [Katz and Yung 2003]). We show that simultaneous n broadcast message for relatively large n is also very expensive in practice, and it, therefore, is recommended to be avoided.

—Cost of BD: The cost of BD roughly doubles as the group size grows in increment of the total number of machines and degrades significantly when the group size hits the number of processors. This is not visible from the table also.

—Cost of TGDH: The best and worst case costs for TGDH can be theoretically analyzed. (Most notably, its worst case communication cost is quite expensive compared to STR.) The results show that TGDH is the best overall protocol in practice, if only one protocol has to be selected. We showed that its self-clustering effect (not visible from the theoretical cost table) can reduce its complexity.

7. COMMON APPLICATIONS CLASSES AND GROUP KEY AGREEMENT

Our long-term experience with group communication systems (in particular, with the Spread toolkit) shows that there are several common communication patterns used by group-oriented applications. Below, we introduce major Spread applications focusing on the type of groups they use, and, for each, identify the most suitable group key agreement protocol. Since 512-bit key size is not considered secure enough for many mission-critical applications, we only consider experimental results for 1024-bit keys. (Note that we do not consider other security services, such as message authentication or authorization.)

Peer groups of long-running servers: This group type usually connects replicated servers that provide a service, as if they were one logical server. The entire group of servers can reside on one LAN, or they may be spread across a WAN. These servers join the group upon startup, and never voluntarily leave the group. The most prevalent membership events in such a group are partitions

and merges. There may also be a limited number of server shutdown and startups, usually for maintenance reasons. When a small number of servers is considered, BD would fit best. However, when the number of servers increases, TGDH would perform much better. Especially in a WAN setting, the self-clustering effect of TGDH would play a major role in reducing its inefficient partition cost. When servers are distributed over a high-delay WAN, STR can be also considered, as it has the most efficient communication cost.

Conferencing: In this group type, membership is built over time as participants join the conference, with an occasional participant joining or leaving. The group usually dissolves when most participants leave at roughly the same time, although the time to complete the mass leave event is not very important to the participants. Again, for small number of participants, BD would fit best. Since most hosts in this application type are expected to run a single member process, BD would not show stepping behavior. However, STR would fit better for a large number of participants. Note that the cost of a subtractive event does not matter much in this case, since the group dissolves almost at the same time.

One-to-many broadcast: In this group type there is one source multicasting to many receivers. The group has no value without the presence of the source; while receivers join and leave at will. It is obvious that group key distribution protocols are most appropriate for this application class. However, CKD would fit better for applications that require strong security properties such as PFS.

Distributed logging: This group type has several logging servers that accept updates from many participants, which may frequently join or leave the group. For example, we have observed such systems with hundreds of participants, each of which with a lifespan of several minutes. This translates into several join or leave operations per second globally. This model requires the most scalable and efficient solution; therefore, TGDH would fit best.

Mobile state transfer: This group type has up to a few tens of participants that share soft state. From time to time, participants join the group, exchange state, stay connected for a while, and temporarily leave the group. In such a setting, there are no long-term participants in the group, but the group changes are usually less frequent than in the distributed logging case. However, this also requires relatively frequent joins and leaves. Therefore, once again, TGDH would work best.

8. CONCLUSIONS

We presented a framework for cost evaluation of group key agreement protocols in a realistic network setting. Our 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. Based on these extensive experiments, we confirm that computation is the most important cost factor in LANs and communication is the most cost factor in high-delay WANs.

In a LAN setting, TGDH performs the best overall. However, we also note that for small groups—no greater than, say, a dozen members—BD is a 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 nontrivial tree management [Kim et al. 2000]. An additional factor can skew the relative 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. At the same time, the leave cost would have been more expensive, yet still less expensive than that in GDH [Kim et al. 2000].

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 WAN (70–100 ms round-trip).

ACKNOWLEDGMENTS

We would like to thank Dang Nguyen Duc and Taekyoung Kwon for providing machines at ICU and Sejong University (Korea) used in our experiments. We also thank Ryan Caudy for providing a tool to generate partitions and merges for Spread servers. Finally, we are very grateful to the ACM TISSEC anonymous referees for their insightful comments and suggestions on a previous version of this paper.

REFERENCES

AMIR, Y., DANILOV, C., MISKIN-AMIR, M., SCHULTZ, J., AND STANTON, J. 2004. The Spread Toolkit: Architecture and Performance. Tech. rep., CNDS-2004-1, Johns Hopkins University.

AMIR, Y., DOLEV, D., KRAMER, S., AND MALKI, D. 1992. Transis: A communication sub-system for high availability. In *Digest of Papers, The 22nd International Symposium on Fault-Tolerant Computing Systems*. 76–84.

AMIR, Y., KIM, Y., NITA-ROTARU, C., SCHULTZ, J., STANTON, J., AND TSUDIK, G. 2001. Exploring robustness in group key agreement. In *The 21st IEEE International Conference on Distributed Computing Systems*. IEEE Computer Society Press, 399–408.

AMIR, Y., KIM, Y., NITA-ROTARU, C., SCHULTZ, J., STANTON, J., AND TSUDIK, G. 2004. Secure group communication using robust contributory key agreement. *IEEE Trans. Parallel and Distrib. Syst. 15*, 5, 468–480.

AMIR, Y., KIM, Y., NITA-ROTARU, C., AND TSUDIK, G. 2002. On the performance of group key agreement protocols (short paper). In *The 22nd IEEE International Conference on Distributed Computing Systems*. IEEE Computer Society Press.

AMIR, Y., MOSER, L. E., MELLIAR-SMITH, P. M., AGARWAL, D., AND CIARFELLA, P. 1995. The Totem single-ring ordering and membership protocol. *ACM Trans. Comput. Syst. 13*, 4 (Nov.), 311–342.

AMIR, Y., NITA-ROTARU, C., STANTON, J., AND TSUDIK, G. 2003. Scaling secure group communication systems: Beyond peer-to-peer. In *The 3rd DARPA Information Survivability Conference and Exposition (DISCEX III)*, Washington, D.C.

AMIR, Y. AND STANTON, J. 1998. The Spread wide area group communication system. Tech. rep., 98-4, Johns Hopkins University.

ANKER, T., CHOCKLER, G. V., DOLEV, D., AND KEIDAR, I. 1998. Scalable group membership services for novel applications. In *Workshop on Networks in Distributed Computing*.

BIRMAN, K. P. AND JOSEPH, T. 1987. Exploiting virtual synchrony in distributed systems. In *The 11th Annual Symposium on Operating Systems Principles*. 123–138.

BIRMAN, K. P. AND RENESSE, R. V. 1994. *Reliable Distributed Computing with the ISIS Toolkit*. IEEE Computer Society Press.

BONEH, D. 1998. The decision Diffie-Hellman problem. In *Third Algorithmic Number Theory Symposium*. Lecture Notes in Computer Science, vol. 1423. Springer-Verlag, Berlin Germany, 48–63.

BONEH, D. 1999. Twenty years of attacks on the RSA cryptosystem. *Not. Am. Math. Soc. (AMS) 46*, 2, 203–213.

BRESSON, E., CHEVASSUT, O., AND POINTCHEVAL, D. 2001a. Provably authenticated group Diffie-Hellman key exchange—The dynamic case. In *Asiacrypt 2001*. Lecture Notes in Computer Science.

BRESSON, E., CHEVASSUT, O., POINTCHEVAL, D., AND QUISQUATER, J.-J. 2001b. Provably authenticated group Diffie-Hellman key exchange. In *The 8th ACM Conference on Computer and Communications Security*. ACM Press.

BURMESTER, M. AND DESMEDT, Y. 1994. A secure and efficient conference key distribution system. *Advances in Cryptology—EUROCRYPT'94*.

CARONNI, G., WALDVOGEL, M., SUN, D., WEILER, N., AND PLATTNER, B. 1999. The VersaKey framework: Versatile group key management. *IEEE J. Select. Areas Commun. 17*, 9 (Sep.).

DIFFIE, W. AND HELLMAN, M. E. 1976. New directions in cryptography. *IEEE Trans. Inform. Theory IT-22*, 644–654.

FEKETE, A., LYNCH, N., AND SHVARTSMAN, A. 1997. Specifying and using a partitionable group communication service. In *The 16th ACM Symposium on Principles of Distributed Computing,* Santa Barbara, CA. 53–62.

FLOYD, S., JACOBSON, V., LIU, C., MCCANNE, S., AND ZHANG, L. 1997. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Trans. Netw. 5*, 6 (Dec.), 784–803.

GONG, L. 1997. Enclaves: Enabling secure collaboration over the Internet. *IEEE J. Select. Areas Commun. 15*, 3 (Apr.), 567–575.

HARNEY, H., COLEGROVE, A., AND MCDANIEL, P. 2001. Principles of policy in secure groups. In *Network and Distributed Systems Security Symposium*.

HILTUNEN, M. A. AND SCHLICHTING, R. D. 1996. Adaptive distributed and fault-tolerant systems. *Int. J. Comput. Syst. Sci. Engng. 11*, 5 (Sep.), 125–133.

HILTUNEN, M. A., SCHLICHTING, R. D., AND UGARTE, C. 2001. Enhancing survivability of security services using redundancy. In *International Conference on Dependable Systems and Networks*.

KATZ, J. AND YUNG, M. 2003. Scalable protocols for authenticated group key exchange. *Advances in Cryptology—CRYPTO'03*.

KEIDAR, I., MARZULLO, K., SUSSMAN, J., AND DOLEV, D. 2000. A client-server oriented algorithm for virtually synchronous group membership in WANs. In *The 20th International Conference on Distributed Computing Systems*. 356–365.

KIHLSTROM, K. P., MOSER, L. E., AND MELLIAR-SMITH, P. M. 1998. The SecureRing protocols for securing group communication. In *The 31st Hawaii International Conference on System Sciences,* Vol. 3. Kona, Hawaii, 317–326.

KIM, Y. 2002. Group Key Agreement—Theory and Practice. Ph.D. thesis, Department of Computer Science, University of Southern California.

KIM, Y., PERRIG, A., AND TSUDIK, G. 2000. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *The 7th ACM Conference on Computer and Communications Security*. ACM Press, 235–244.

KIM, Y., PERRIG, A., AND TSUDIK, G. 2001. Communication-efficient group key agreement. In *IFIP SEC 2001*.

KIM, Y., PERRIG, A., AND TSUDIK, G. 2004a. Group key agreement efficient in communication. *IEEE Trans. Comput. 33*, 7.

KIM, Y., PERRIG, A., AND TSUDIK, G. 2004b. Tree-based group key agreement. *ACM Trans. Inf. Syst. Secur. 7*, 1.

MCDANIEL, P., PRAKASH, A., AND HONEYMAN, P. 1999. Antigone: A flexible framework for secure group communication. In *The 8th USENIX Security Symposium*. 99–114.

MENEZES, A., VAN OORSCHOT, P., AND VANSTONE, S. 1996. *Handbook of Applied Cryptography*. CRC Press.

MOSER, L. E., AMIR, Y., MELLIAR-SMITH, P. M., AND AGARWAL, D. A. 1994. Extended virtual synchrony. In *The 14th International Conference on Distributed Computing Systems*. IEEE Computer Society Press, Los Alamitos, CA, 56–65.

NATIONAL INSTITUTE FOR STANDARDS AND TECHNOLOGY (NIST). 2000. *Digital Signature Standard (DSS)*. Number FIPS 186-2. National Institute for Standards and Technology (NIST). http://csrc.nist.gov/publications/fips/fips186-2/fips186-2.pdf.

NITA-ROTARU, C. 2003. High Performance Secure Group Communication. Ph.D. thesis, Department of Computer Science, Johns Hopkins University.

OPENSSL PROJECT TEAM. 1999. OpenSSL. http://www.OpenSSL.org/.

REITER, M. K. 1994. Secure agreement protocols: reliable and atomic group multicast in RAMPART. In *The 2nd ACM Conference on Computer and Communications Security*. 68–80.

RENESSE, R. V., BIRMAN, K., AND MAFFEIS, S. 1996. Horus: A flexible group communication system. *Commun. ACM 39*, 76–83.

RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L. M. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21*, 2 (Feb.), 120–126.

RODEH, O., BIRMAN, K., AND DOLEV, D. 2001. The architecture and performance of security protocols in the Ensemble Group Communication System. *ACM Trans. Inf. Syst. Secur. 4*, 3 (Aug.), 289–319.

RODEH, O., BIRMAN, K., AND DOLEV, D. 2002. Using AVL trees for fault tolerant group key management. *Int. J. Inf. Secur. 1*, 2 (Feb.).

SCHULTZ, J. 2001. Partitionable Virtual Synchrony using Extended Virtual Synchrony. M.S. thesis, Department of Computer Science, Johns Hopkins University.

SETIA, S., KOUSSIH, S., JAJODIA, S., AND HARDER, E. 2000. Kronos: A scalable group re-keying approach for secure multicast. In *The 2000 IEEE Symposium on Security and Privacy*. IEEE, 215–218. Oakland, CA.

SHERMAN, A. T. AND MCGREW, D. A. 2003. Key establishment in large dynamic groups using one-way function trees. *IEEE Trans. Softw. Engng.* 444–458.

STEER, D., STRAWCZYNSKI, L., DIFFIE, W., AND WIENER, M. 1990. A secure audio teleconference system. *Advances in Cryptology—CRYPTO'88*.

STEINER, M., TSUDIK, G., AND WAIDNER, M. 2000. Key agreement in dynamic peer groups. *IEEE Trans. Parallel Distrib. Syst.*.

TZENG, W.-G. AND TZENG, Z.-J. 2000. Round-efficient conference-key agreement protocols with provable security. In *Advances in Cryptology—ASIACRYPT '2000*. Lecture Notes in Computer Science. Springer-Verlag, Kyoto, Japan.

WALLNER, D., HARDER, E., AND AGEE, R. 1999. Key management for multicast: Issues and architectures. RFC 2627.

WHETTEN, B., MONTGOMERY, T., AND KAPLAN, S. 1994. A high performance totally ordered multicast protocol. In *Theory and Practice in Distributed Systems, International Workshop*. Lecture Notes in Computer Science, vol. 938.

WONG, C. K., GOUDA, M. G., AND LAM, S. S. 2000. Secure group communications using key graphs. *Trans. Netw. 8*, 1, 16–30.

Received November 2003; revised May 2004 and June 2004; accepted June 2004