An Integrated Solution for Secure Group Communication in Wide-Area Networks

D. Agarwal[*] O. Chevassut[*][†] M. R. Thompson[*] G. Tsudik[‡]

Abstract

Many distributed applications require a reliable group communication system to provide co-ordination among the application components. Such group communication is often conducted over a wide-area network like the global Internet. Messages sent over the Internet often traverse several independently controlled networks and are thus susceptible to attacks. Consequently, distributed applications need mechanisms to secure this communication.

This paper describes a secure group communication protocol. The secure group layer (SGL) bundles a reliable group communication system, a group authorization and access control mechanism, and a group key agreement protocol to provide a comprehensive and practical secure group communication platform. SGL also encapsulates the standard message security services (i.e, confidentiality, authenticity and integrity). A number of challenging issues encountered in the design of SGL are brought to light and experimental results obtained with a prototype implementation of SGL are discussed.

Keywords: security, group communication, group authorization and access control, group key agreement.

[*]Lawrence Berkeley National Laboratory, CA, *email:* {*DAAgarwal,OChevassut,MRThompson@lbl.gov*}*@lbl.gov*

[†]UCL Crypto Group, Université Catholique de Louvain, Belgium

[‡]Dept. of Information and Computer Science, UC Irvine, *email: gts@ics.uci.edu*

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, Mathematical Information and Computing Sciences Division, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. This document is report LBNL-47158.

1 Introduction

Many current applications are implemented as distributed systems. Some are distributed by nature (e.g., conferencing) while others are distributed to meet load-balancing and fault-tolerance requirements (e.g., content servers). Such applications often rely on reliable group communication to provide coordination between processes communicating as a *process group*. A typical group communication system provides reliable delivery of messages within the group, group membership maintenance and reporting, as well as ordered delivery of messages. These properties allow the application to track membership and send messages to the group with the assurance that all messages will be received by all members in the same order. Example group communication systems which provide these properties include Totem [18], Spread [4], and InterGroup [7].

One example of an application class that can benefit from, and make extensive use of, a reliable group communication platform is scientific collaboration software. Applications such as distributed white boards, remote instrument control, messaging systems, electronic notebooks, and data sharing are natural users of group communication. Applications of this type normally involve users spread across a wide-area network and may utilize multiple process groups. Unfortunately, few group communication systems can operate over a wide-area network and even fewer incorporate the access control and other security services that these applications require.

Although acceptable solutions (e.g., SSL [10], IPSec [12]) are available for securing unicast connections, they do not extend to securing group communication. One of the main reasons is key management. Unicast communication involves only two parties and consensus on a shared key is relatively easy to reach. Each time a new unicast connection is created the consensus process starts from scratch and, if either party in a unicast communication session quits, fails, or for any reason drops the connection, the other party also quits. In group communication, consensus on a shared key is more complex since group membership is dynamic. Once a group is formed, members may join or leave the group due to failures, network partitions and voluntary membership changes. This greatly complicates the key management.

Controlling access to a group requires authentication of users and definition of group access policy. Authentication and authorization for groups present a more complicated set of problems than the typical client-server access control. Authentication is more difficult because each group member must be able to authenticate all the other members. In a server-based access control model the policy normally only controls access and is only enforced by the server. The scope of group security policy is still a research topic as are the methods of group policy enforcement.

Another challenge in introducing group security services is how best to provide them to the application. One approach is to integrate them into the underlying group communication system. This approach makes the security services invisible to the applications but providing authenticity, authorization and access control at the granularity of the users is difficult. It would also not be portable across different group communication systems.

An alternate approach is to interpose a security layer between the application and the group communication system. This approach introduces minor changes in the application to convey a user's credentials (access privileges and user identity information) to the secure layer and has the advantage of being largely independent of a specific group communication system implementation. This approach also allows the security layer to leverage off the properties of the group communication system in transmitting its own messages to the group.

The main contribution of this paper is in the design of a secure group layer (SGL) aimed at WAN environments. The secure group layer protects against attacks like eavesdropping and spoofing[*] by integrating a reliable group communication system, a group authorization and access control mechanism to determine who knows the key, and a group key agreement protocol which facilitates the standard message security services (i.e. confidentiality, authenticity and integrity). However, denial of service attacks can still be a problem.

The remainder of this paper is organized as follow. Section 2 defines the group communication terminology used in the rest of the paper. Section 3 summarizes the related work. Section 4 describes the secure group layer architecture and its security protocols. Finally section 5 presents some experimental results obtained with a prototype implementation.

[*]Spoofing is an integral part of many network attacks. In a group communication setting, spoofing attacks refer to the impersonation of a group member.

2 Group Communication

Group communication systems are designed to support communication between processes cooperating in groups. The group communication system provides an underlying layer that does the work of maintaining membership of the process group and reliably delivering messages sent to the process group in an asynchronous distributed system.

There are several message ordering properties that group communication systems provide to the application. A very basic property is *causal ordering* of messages. Messages sent by the same application process are received by the application in the order they were sent and messages received by an application process before sending a new message (m1) are ordered before m1 at all the members of the group. Many systems also provide a *total order* on messages within the group so that messages are received by the application in a single linear total order that is the same at all the members of the group. This total order of messages allows the application to operate on the messages serially as they are received since all the group members are receiving the same stream of messages.

Group membership maintenance is a critical component of the group communication system since the membership of the group is the basis for the determination of reliable delivery of messages and message order. A particular instance of the group membership is referred to as a *view*. Each application receives messages within the context of a view. It is important that the delivery order of messages and view changes are consistent across the members of a particular view.

There are several consistency definitions that are in use by group communication systems. Some common consistency definitions are *sending view delivery*, *view synchrony*, and *extended virtual synchrony* (see [29]). *Sending view delivery* means that messages are received in the view in which they were sent. *Virtual synchrony* defines messages order, message delivery and view change [8]. *Extended virtual synchrony* [19] places system wide consistency constraints on message and view change reception properties and message order.

3 Related Work

In the research literature, three different aspects of secure group communication have been considered. One aspect is to secure the group communication system against Byzantine failures[†] [13, 15, 21]. In this category are Rampart [21], Immune [13] for LAN environments and, their extension to WAN environments done by D. Malkhi et al. [15]. The second aspect is to protect the messages generated by the application. Here the threat model assumes that the group communication servers will not be corrupted; hence, the focus is on attacks such as eavesdropping and spoofing. This approach is exemplified by Ensemble [23] and Secure Spread [3]. The third aspect is to address the various group policy issues such as requirements for group rekeying and levels of message security. An example is the Antigone framework[16].

The Rampart system [21, 22] was the first to demonstrate the feasibility of reliable and atomic group multicast in asynchronous distributed systems in the presence of Byzantine failures. It uses public key cryptography to establish authenticated communication between a pair of processors and implements the reliable and atomic group multicast protocols over a secure group membership protocol [22].

The Immune system [13, 20] also uses public key cryptography to secure the Totem [18] daemon. Immune secures the low-level ring protocol against Byzantine failure and hence maintains the reliable ordered message delivery and group membership services despite the corruption of some group communication servers by an attacker.

The Ensemble system [23] is descended from an earlier system named Horus, itself descended from the Isis system. The early work on group communication security was performed in Horus and then extended into Ensemble. Ensemble allows application-dependent trust models, and optimizes certain aspects of group key distribution protocols. The group key generation and distribution protocols used in Ensemble are extensions of symmetric (i.e. two-party) cryptographic tools such as PGP [17] or Kerberos [24]. However, Ensemble relies on a trusted group leader to perform and initiate key generation. The group leader is static and changes only in the event that the current group leader leaves the group voluntarily or becomes unreachable. Consequently, the security is

[†]In the Byzantine threat model the attacker can compromise the underlying group communication system and/or run fake group communication system.

based on the group leader's ability to generate good keys and its constant on-line availability.

Secure Spread [3] differs from Ensemble since it uses a fully distributed group key generation protocol [26]. The Spread secure group layer is placed above the Spread group communication system and relies on the property commonly known as *view synchrony.* However, we claim that view synchrony is a stronger property than is required for the secure group layer (The property *sending view delivery* is actually enough). Furthermore, we note that Secure Spread does not provide any authentication or group access control mechanisms, does not consider the Byzantine failure model, and focuses primarily on LAN and interconnected LAN environments.

The Antigone framework [16] provides interfaces for the definition and implementation of policies for secure groups. Policies are implemented by composition and configuration of mechanisms which provide basic services for secure groups.

4 Secure Group Layer

The design of our secure group layer (SGL) needs the properties provided by the underlying group communication system. These properties are a subset of those provided by *extended virtual synchrony* and ensure that messages are consistently ordered and delivered across the group. The *view change events* emanating from the group communication system notify SGL of membership changes due to a join, leave, fail, partition, or merge event.

In addition to the existing properties of the group communication system, SGL provides applications with the property of *sending view delivery.* This property is useful for implementing group security services since it guarantees the application that its group view when a message is sent is the same view in which the message will be delivered. By using sending view delivery, SGL can use one group key at a time and change keys with each new view.

The SGL architecture consists of four main components each implementing a separate protocol (see Fig. 1). The *record layer* provides standard message security services (i.e., confidentiality, integrity and authenticity). The *access control protocol* enforces restrictions on group membership. The *flush protocol* provides a mechanism for delineating membership views where each view corresponds to the lifetime of a specific secret group key and any keys derived from it. The flush protocol allows group members to discard previously used keys and other associated data. The *key agreement*

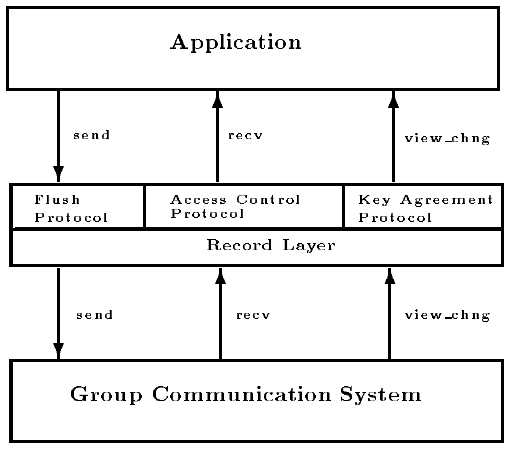

Figure 1: Protocol stack

protocol creates a shared group secret which is then used to derive a symmetric encryption key and an authentication (MAC) key. These two keys are subsequently made available to the record layer.‡

4.1 Record Layer

The record layer supports message transmission with confidentiality, integrity and authenticity. It takes an application message waiting to be sent, encrypts it using the symmetric group encryption key, applies an integrity algorithm using the MAC group key and sends it using the group communication system. On receipt, a message is decrypted with the symmetric group decryption key, verified using the MAC group key and delivered to the application. The current SGL implementation uses the Rijndael cipher [9] for encryption and the HMAC method [14] for MAC computation.

4.2 Flush Protocol

As mentioned above, the flush protocol implements sending view delivery for SGL and applications. It defines the end of a membership view and thus guarantees that no further messages encrypted with a particular session key will be received. Coordination is attained with special flush messages shown in Figure 2.

The flush protocol is invoked by a view change event. Recall that *sending view delivery* means that all messages that the application believes were sent in a given view must be received by the

‡The flush, access control, and key agreement protocols are invoked by each view change event.

Identifier:	*flush.id*
Group:	*flush.G*
View:	*flush.view*
Sender's Certificate:	*flush.cert*
Signature:	*flush.sign*

Figure 2: Flush message structure.

application in that same view. Consequently, the flush protocol must send pending messages (which have been accepted from the application) and block any new messages from the application before it sends the *flush* message.

The protocol waits until a flush message from every process in the new view is received and is verified (see Section 4.3.2 for details). Receipt of a flush message represents for the recipient a "promise" by the sender not to send any more messages encrypted with the group key corresponding to the old view. When all flush messages are received, the process can conclude that no further messages protected by the old group key will be received. Incidentally, the flush protocol treats a process failure message as a flush message from the failed process. Other types of view change events reset and restart the flush protocol.

It is important to note that, if SGL were to use an underlying group communication system that provided sending view delivery, the flush layer would still be needed. Otherwise, the group communication system would need to accept, for sending in the old view, application messages buffered (e.g., during encryption) in SGL and not yet passed to the group communication system.

4.3 Access Control Protocol

A group access control mechanism enforces restrictions on group membership. Without it, other security services (including key agreement and data integrity/privacy) are basically ineffective. Our access control approach uses *membership certificates* that authorize entry into the key agreement protocol and, hence, the group itself.

Exploring all the variables involved in defining group policies is an interesting research topic, but for the purpose of this paper we will assume that a simple policy exists. However, we still need to identify the repository for group policies.

4.3.1 Authorization Authority

The Authorization Authority is responsible for collecting group policies and using them to determine who is allowed to join a group. The Akenti server is such an authority. Akenti determines if a user is allowed to join a group and issues *membership certificates* containing that information.

A *membership certificate* (see Figure 3) associates a public key with the X.509 identity [30] of a user, contains the access privilege granted to the user with respect to the group, the validity period for the certificate as determined from the policy, the identification of the Authorization Authority that issued the certificate and additional information for the Authorization Authority's use, such as a serial number.

Subject:	Distinguished Name, Public Key
Access Privilege:	Group, Authorized or Denied Access
Issuer:	Distinguished Name, Signature
Administrative Information:	Version, Serial Number.
Period of Validity:	Start and Expire Dates/Times

Figure 3: Membership certificate structure.

The Akenti server not only issues membership certificates, it also manages them. It keeps a list of all non-expired certificates that have been issued for a group and revokes them when the group policy changes. Akenti also keeps a list of all the revoked certificates called the Certificate Revocation List (CRL). When examining membership certificates for validity, therefore, it is necessary to contact the issuing Akenti server to check the Certificate Revocation List. At this time this is not an automated part of the group access control protocol.

4.3.2 Group Access Control Protocol

A user needs to first obtain a membership certificate from the Akenti server or needs to request a new certificate if its certificate has expired or has been revoked - this is not currently an automated part of the protocol.

When a user wants to join the secure group, he will start by joining the reliable group. This join will cause a view change and intitate the flush protocol. During the flush protocol each member, including the new joinee, will broadcast its membership certicate as part of the flush message. For efficiency's sake, the membership certificate is included in the flush message. This does not increase the size of the flush message since the membership certificate can replace the *flush.cert* field. Each member will verify each other member certificate by checking that the message is signed by the subject of the certificate, that the membership certificate is signed by Akenti, is within its validity period and grants joining access.

Once the group controller has verified all the members, it will start a new key agreement protocol. The group controller is a group member who enforces group access control policy by creating and disseminating the group keying material to authorized members. The group controller role is determined by the group key agreement as we will see in the next section. If any member sees key agreement messages from a user that it does not trust, it can refuse to participate.

4.4 Key Agreement Protocol

In this section we briefly sumarize the group key agreement protocol and then motivate its choice as a key exchange protocol for SGL.

4.4.1 Group Key Agreement

A group key agreement mechanism establishes a secret key between members of a group. It allows the members to agree upon and begin efficiently computing a key without relying on any centralized trusted third party (TTP) which could be a single point of vulnerability for the overall system.

An example of group key agreement is the Cliques [6, 26] protocol suite. The Cliques protocols dynamically determine one of the members to serve as the group controller[§] whose main task is to coordinate the generation of partial keys and to disseminate them to other group members. Within the Cliques protocol suite we focus on two protocols: IKA.1 (depicted in Fig 4) and IKA.2 (described in [26] and in the appendix). These two Initial Key Agreement protocols can be extended to support single-member join operations and also key agreement following a merge event (see [25]).

[§]Group controller is always the newest (or most recent) group member. This selection criterion has an important benefit as it can be performed without any message exchange. Note that the concept of newest is not meaningful in an execution model where different processes observe group views in differents orders or with gaps. We postpone further discussion of this issue until Section 4.4.3.

IKA.1 is a member of the family of so-called *Group Diffie Hellman (GDH)* key agreement protocols which are shown secure against passive [26] and active [6] attacks. IKA.1 trades off minimal round (and number of messages) complexity in return for higher computational cost. In contrast, IKA.2 a "sibling" GDH protocol minimizes computational cost at the expense of more protocol rounds (and more messages). One contribution of the present work is the integration of the IKA.1 protocol with a group communication system, thus far IKA.1 has only been a theoretical interest.¶

Protocol IKA.1:
Let $\mathcal{M} = \{M_1, \ldots, M_n\}$ be a set of members wishing to share a key GK_n. IKA.1 executes in n rounds. In the first stage ($n-1$ rounds) contributions are collected from individual group members and, then, in the second stage (n-th round), the group keying material is broadcast. Each member then uses its own contribution to compute the group key. The actual protocol is as follows:
Initialization: Let p be a prime and q a prime divisor of $p-1$. Let G be the unique cyclic subgroup of $\mathbb{Z}_p^*$ of order q, and let α be a generator of G.
Round i $(0 < i < n)$:

1. M_i selects $r_i \in_R \mathbb{Z}_q^*$.
2. $M_i \longrightarrow M_{i+1}$: $\{\alpha^{\frac{r_1 \cdots r_i}{r_j}} | j \in [1, i]\}$, $\alpha^{r_1 \cdots r_i}$

Round n:

1. M_n selects $r_n \in_R \mathbb{Z}_q^*$.
2. $M_n \longrightarrow$ ALL M_i: $\{\alpha^{\frac{r_1 \cdots r_n}{r_i}} | i \in [1, n[\}$

Figure 4: Cliques Group Diffie-Hellman Protocol (IKA.1)

4.4.2 Performance Analysis

As mentioned above, IKA.1 aims to minimize round complexity and the number of messages while IKA.2 aims to minimize computational costs. The overall time-to-completion for each protocol is dominated by two factors: network communications and cryptographic processing times; primarily, exponentiation with large numbers which is quite costly. In order to compare the costs of the two protocols we need to consider the steps of each protocol.

IKA.1 (Fig 4) operates in k rounds and requires $k-1$ unicast messages followed by a single broadcast. Each round i $(1 \leq i < k)$ involves each member M_i performing i exponentiations. This is followed by M_i unicasting a set of $(i+1)$ partial keys on to M_{i+1}, except for the last (k-th) round when M_{n+k} broadcasts the partial keys. Finally, each M_i performs a single exponentiation upon

¶For more details see the technical report [2].

receipt of the broadcast. Assuming that all members exponentiate in approximately the same time, the total protocol delay is thus $COST(IKA.1)$. Similarly for IKA.2, the total protocol delay is $COST(IKA.2)$ (details in the appendix).

$$COST(IKA.1) = \frac{E}{2}\,k^2 \;+\; (En + \frac{E}{2} + D)\;k \;+\; D \;\;;\;\; COST(IKA.2) = (2E + D)\;k \;+ (En - E + 3D)$$

where D is the network delay, E the cost of a single exponentiation, n the number of members in the group and k the number of joining members.

We are now ready to compare the relationship between the cost of IKA.1 and the cost of IKA.2. We assume a 2ms exponentiation delay[||] and a 100ms wide-area network delay [**] and thus obtain the relation represented in Figure 5. The curve represents the values for which IKA.1 and IKA.2 have the same cost.

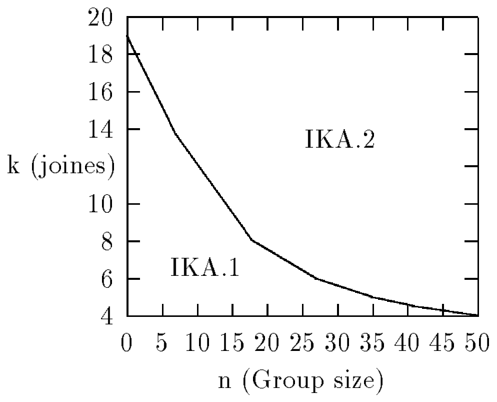

Figure 5: Tradeoff between group size and number of joining members. In the area above the curve, IKA.2 is faster than IKA.1.

In the typical underlying group communication system most view changes require consensus among the new view's. For this reason, the time to complete a membership change becomes prohibitive as the group size grows. This is particularly true when group members are spread across a wide-area network (WAN) since a WAN involves an increased round-trip time between group members and a greater likelihood of lost and thus resent messages due to the number of hops and sheer

[||]The performance for the 512-bit moduli exponentiation was obtained using the big number library in OpenSSL on a 450MHZ Pentium II PC.

[**]The average point-to-point delay for a US coast-to-coast round-trip at the application level.

distances traversed.

We postulate that a practical limit for process group membership size in a wide-area network is likely to be around 40. In our experience, current scientific collaborations typically involve even smaller groups for example less than 20 members. Thus, most membership increases in a 20 member group are likely to involve a relatively small number of members merging into an existing group (either new members or heals of prior network partitions). Figure 5 clearly demonstrates that, under these assumptions, IKA.1 offers better performance than IKA.2.

As shown in Figure 5, on one extreme all 20 members join the collaborative session as soon as it starts. At $n = 0$ and $k = 20$, IKA.1 is as fast as IKA.2. Later, members may leave and join the group or the group may become partitioned and such a partioned group may merge. Suppose a 5 member group and a 15 member group were to merge. As shown in Figure 5 at $n = 5$, $k = 15$ and n=15, $k = 5$, IKA.1 is faster than IKA.2.

Figure 5 above is based on our current measurements for exponentiation and wide-area network delay. The curve in figure 5 will move up as the time for exponentiation goes down. In the future, due to faster computers, the exponentiation delay is more likely to decrease than the wide-area network delay. Moreover faster exponentiation algorithms exist; Hankerson et al. [11] obtained an exponentiation delay of 1.5 ms[††] with a level of security equivalent to twice (i.e. 1024-bit security) the 512-bit security that we require for scientific collaboration software.

4.4.3 Group Controller

In the event of a network failure, a group may become partitioned into several disjoint components. These components may subsequently need to merge whenever the failure is repaired (i.e., a partition heals). However, this brings up an important question of how to select the group controller following a merge event.

In our framework, the new group controller is selected as the prior controller of the largest merging sub-group (largest in terms of number of *authorized* members). Adding members from the

[††]Hankerson et al. [11] obtain a 1.5ms exponentiation delay on NIST-recommended elliptic curves K-163 using the big number library in OpenSSL on a 400MHZ Pentium II PC.

smaller group into the larger one has some obvious advantages. As an example, consider the merge of a 5-member group and a 15-member group. Assuming all 20 members are previously authorized, a 2ms single exponentiation delay and a 100ms WAN round-trip delay, merging 5 members into a 15-member group costs 0.780 sec, while merging 15 members into a 5-member group costs 1.900 sec. More generally, $COST(IKA.1)$ grows linearly with n and is quadratic in k, thus, merging a smaller group into a larger one is always faster.

The problem now is how to agree on which group has the larger number of authorized members. Since the underlying group communication system is acting independently of the secure group layer (SGL) its membership may be a superset of the secure group membership. Consequently, relative sizes of the merging secure groups cannot be determined from the information provided by the underlying group communication system. Additionally, there may even be view changes where one of the merging groups has no authorized members.

With a small modification, our flush protocol can provide the information about sizes of merging groups. Each member adds to the flush (*flush.view*) a list of the processes in its secure group communication session that are also in the new unsecure group view. Thus, on receipt of a flush message from each member in the new view, all members can determine the largest secure group and hence dynamically determine a member to serve as the merged group controller.

5 Experimental results

A prototype implementation of the Secure Group Layer in the "C" programming language has been completed. It currently runs on Sun UltraSparc workstations with the Solaris 5.7 operating system. The Totem system [1, 18] is utilized as the underlying group communication system, the Akenti server [28] serves as the authorization server and the IKA.1 protocol has been implemented using the functions provided by the Cliques toolkit [5]. We also use the implementation of DSA provided by OpenSSL [27].

The Totem system [1, 18] provides all the properties required by the secure group layer and some additional properties such as totally ordered messages. The Totem system runs as a daemon and a light-weight user interface layer. The remote users connect via the light-weight layer to the Totem daemon using a TCP/IP connection - or through an SSL connection - across the high latency link since the Totem daemons were not designed to operate over a high latency link. One advantage of the Totem system is that it can be replaced, if needed, by its secure version called Immune [13] which is designed to protect against Byzantine failures.

The Akenti server issues the membership certificates used for group admission. For the sake of fault tolerance, it can be run as a set of mirrored servers. Note that Akenti can be administered independently.. Akenti provides a Java interface to allow stakeholders to create digitally signed policy certificates.

Initial performance tests with our prototype implementation were performed between sites in Berkeley, California (LBNL) and Argonne, Illinois (ANL). In these tests we measured the performance of the secure group layer (SGL) when one member joins the group (Fig 6), leaves the group (Fig 8) and group merge operation with various component sizes (Fig 7). In each case the graphs show the results from the worst case scenario (e.g. the joining member is separated from the group by a high-latency link).

We now describe our experiments. At Lawrence Berkeley Laboratory (Berkeley), one Sun UltraSparc 5s is running a Totem daemon and one group member. The second Sun UltraSparc 5s is running a Totem daemon and the rest of the Berkeley group members. At Argonne, a Sun UltraSparc 2 is running one user who connects to a Totem daemon at Berkeley. On a Sun UltraSparc 5, a 512-bit moduli exponentiation, DSA signature and DSA verification [‡‡] provided by OpenSSL [27] costs respectively 0.010 seconds, 0.010 seconds and 0.030 seconds.

Figure 6 shows the performance of SGL when the member in Argonne joins an existing group in Berkeley. As an example, adding one member to a group of 19 members takes 1.4 seconds. Examining the steps after the flush protocol is finished, the group controller in Berkeley computes 19 exponentiations, signs the message and sends the values to the joining member in Argonne; 0.200

[‡‡]It is worth to notice that DSA operations (i.e. signing and verification) are more symmetric than the RSA operations. RSA verification is roughly an order of magnitude faster than RSA signing and RSA signing is roughly as fast as DSA signing. For SGL, DSA is clearly a bottleneck.

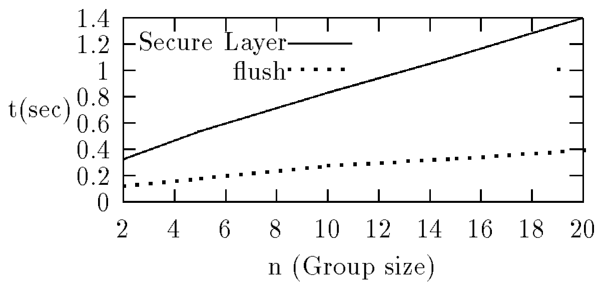

Figure 6: Performance of the Secure Group Layer when one member located at Argonne joins the group of size $n-1$.

seconds. Upon reception of the message, the joining member verifies the signature, computes 19 exponentiations, signs the message, and sends the values to the daemon in Berkeley, 0.303 seconds. At this point, the total is 0.503 seconds. When the daemon receives the message, it forwards it to the users. The user in Argonne gets the message after 0.070 seconds, verifies the signature, and computes one exponentiation; 0.110 seconds. The user in Argonne computes the group secret in a total time of 0.613 seconds. Adding the flush protocol we get 0.993 seconds. Each of the 18 users on the Sun UltraSparc 5 have to get the message, verify the signature and compute one exponentiation; $0.035 + 18 * 0.04 = 0.755$ seconds. Adding the cost of the flush protocol, the 1st member computes the group key in 0.958 seconds while the 18th member computes the group key in 1.638 seconds. The average experimental result obtained is 1.380 ms.

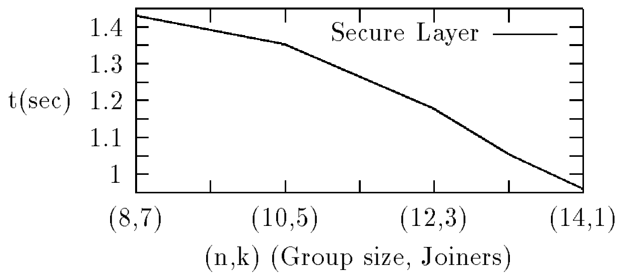

Figure 7: Performance of group merge with variable-size merging components. The main group size is constant at 15 members. The cost of the flush is not included.

Figure 7 shows the performance of the group merge operation with various partition sizes. As an example a group with k=3 members is added to an existing group with n=12 members; i.e. (12,3).

The 12 members in the existing group are on one computer and the other group has one member at Argonne and the other two on a computer in Berkeley. Once the flush protocol completes, the group controller of the larger group computes 12 exponentiations, signs the message and sends the value to the 1st member in the group with 3 members; 0.13 seconds. The 1st member receives the message, verifies the signature, computes 13 exponentiations, signs the message and sends it to the member located at Argonne; 0.17 seconds. The member located at Argonne receives the message, verifies the signature, computes 14 exponentiations, signs the message and sends it to the 3rd member; 0.215 seconds. The 3rd member receives the message, verifies the signature, computes 14 exponentiations, signs the message and sends it to the group; 0.215 seconds. At this point the total is 0.73 seconds. The member located in Argonne, computes the group secret in a total time of 0.805 seconds. Each of the first group's 12 members need to get the message, verify the signature and compute one exponentiation; 0.48 seconds. So, the 1st member computes the group key in 0.77 seconds while the 12th member computes the group key in 1.21 seconds. The other two members of the second group get the message, verify the signature and compute an exponentiation; 0.81 ms. The graph shows the average experimental value obtained for this group merge operation.

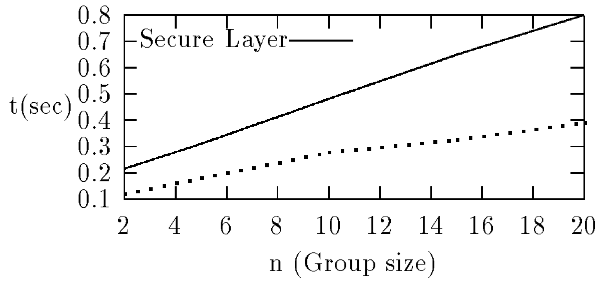

Figure 8: Performance of the Secure Group Layer when one member leaves the group and the group controller is located at Argonne.

The emphasis in building this first prototype was on correctness rather than performance. A significant performance improvement would result from an extensive study, and implementation of IKA.1 using elliptic curve cryptography [11]. Significant improvements can also be realized by exploiting faster platforms to speed-up the cryptographic operations such as exponentiation and signature. Using the Totem system is an intermediaite solution. In the future, we expect to replace

Totem with the InterGroup protocols [7], currently under development. The InterGroup protocols are intended for wide-area networks.

6 Conclusions

This paper presented the design of the secure group layer (SGL) aimed at WAN environments. SGL offers protection against attacks like eavesdropping and spoofing. SGL determines who is allowed to join the communication group by using membership certificates provided by an authorization server. Moreover SGL does not rely on a centralized server (e.g., key distribution center) by integrating a group key agreement protocol with a group communication system.

Since the group key agreement protocol is secure and the application messages are all encrypted, group information is passed securely inside the group. Even the lack of reliable and ordered delivery of messages will not disclose this information. However if the reliable group communication system is compromised, the application can no longer count on the reliable delivery of messaages, which may cause the communication to be useless. SGL could be used in conjunction with a group communication system resistant to Byzantine failures [13, 15, 21] to reduce the chances of the communication failing.

The result of this work is a prototype implementation of SGL using the Totem protocol. Implemented atop the Totem system [1, 18], the prototype provides protection against attacks like eavesdropping and spoofing. Additionally used in conjunction with the Immune system [13] SGL can offer protection against Byzantine failures.

Development of SGL presented several challenges. For example, access control introduced the potential for the group communication system and SGL to have different membership views. This, in turn, led to the need to gather information about old memberships in SGL rather than in the group communication system.

On-going work on SGL is investigating efficiency improvements and robustness features. In addition, we are considering replacing the Totem system with its more recent follow-on, the InterGroup [7] wide-area group communication protocol suite. This would allow for more efficient support of groups spread across wide-area networks.

References

[1] D. A. Agarwal. *Totem: A Reliable Ordered Delivery Protocol for Interconnected Local-Area Networks.* PhD thesis, University of California, Santa Barbara, December 1994.

[2] D.A. Agarwal, O. Chevassut, and G. Tsudik. Integrating Access Control with Secure Group Communication for Wide-Area Networks. Technical Report LBNL-46652, Lawrence Berkeley National Laboratory, USA, August 1999.

[3] 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 *20th IEEE ICDCS*, April 2000.

[4] Y. Amir, C. Danilov, and J. Stanton. A low latency, loss tolerant architecture and protocol for wide area group communication. In *30th IEEE FTCS*, June 2000.

[5] G. Ateniese, O. Chevassut, D. Hasse, Y. Kim, and G. Tsudik. The Design of a Group Key Agreement API. In *DARPA DISCEX 2000*, January 2000.

[6] G. Ateniese, M. Steiner, and G. Tsudik. New multi-party authentication services and key agreement protocols. *IEEE JSAC, special issue on Secure Communication*, May 2000.

[7] K. Berket, L. Moser, and P. Melliar-Smith. The InterGroup Protocols: Scalable Group Communication for the Internet. In *IEEE GLOBECOM'98*, 1998.

[8] K. Birman and T. Joseph. Reliable communication in the presence of failures. In *ACM Transactions on Computer Systems*, volume 5(1), pages 47–76, February 1987.

[9] J. Daemen and V. Rijmen. The Rijndael Block Cipher. In *AES Proposal*, 2000. Available at http://csrc.nist.gov/encryption/aes/.

[10] A. Freier, P. Karlton, and P. Kocher. *The SSL Protocol Version 3.0*, November 1996.

[11] D. Hankerson, J. Hernandez, and A. Menezes. Software implementation of elliptic curve cryptography over binary fields. In *CHES 2000*. Lecture Notes in Computer Science, 2000.

[12] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, November 1998.

[13] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. The SecureRing Protocols for Securing Group Communication. In *31st IEEE HICSS, Kona, Hawaii*, pages 317–326, January 1998.

[14] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. Internet RFC 2104, February 1997.

[15] Dahlia Malkhi, Michael Merritt, and Ohad Rodeh. Secure Reliable Multicast Protocols in a WAN. In *17th International Conference on Distributed Computing Systems (ICDCS'97)*, pages 87–94. IEEE Computer Society Press, May 1997.

[16] P. D. McDaniel, A. Prakash, and P. Honeyman. Antigone: A flexible framework for secure group communication. In *8th USENIX Security Symposium*, pages 99–114, August 1999.

[17] The MIT Press. *The Official PGP User's Guide. prz@acm.org.* Also in http://www.pegasus.esprit.ec.org/people/arne/pgp.html.

[18] L. Moser, P. Melliar-Smith, D. Agarwal, R. Budhia, and C. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, April 1996.

[19] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal. Extended Virtual Synchrony. In *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, pages 56–65. Springer-Verlag, June 1994.

[20] P. Narasimhan, K. Kihlstrom, L. Moser, and P. Melliar-Smith. Providing support for survivable corba applications with the immune system. In *19th IEEE ICDCS*, pages 507–516, May 1999.

[21] M. K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *2nd ACM Conference on Computer and Communication Security*, November 1994.

[22] M. K. Reiter. Secure Group Membership Protocol. In *IEEE Symposium on Research in Security and Privacy*, May 1994.

[23] O. Rodeh, K. Birman, M. Hayden, Z. Xiao, and D. Dolev. Ensemble security. Technical Report TR98-1703, Cornell, Departement of Computer Science, September 1998.

[24] J. Steiner, C. Neuman, and J. Schiller. Kerberos: An authentication service for open networks systems. In *Usenix Winter Conference*, pages 191–202, Jan 1998.

[25] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman Key Distribution Extended to Group Communication. In *3rd ACM Conference on Computer and Communications Security*, March 1996.

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

[27] OpenSSL Project team. Openssl user's guide, 2000. Available at http://www.openssl.org.

[28] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate-based access control for widely distributed resources. In *8th Usenix Security Symposium*, August 1999.

[29] R. Vitenberg, I. Keidar, G. Chockler, and D. Dolev. Group communication specifications: A comprehensive study. Technical Report MIT-LCS-TR-790, MIT, September 1999.

[30] ITU-Recommendation X.509. The Directory - Authentication Framework, 1998.

Appendix

Protocol IKA.2:

As before, let $\mathcal{M} = \{M_1, \ldots, M_n\}$ be a set of members wishing to share a key GK_n. IKA.2 executes in $n+2$ rounds. In the first stage ($n-1$ rounds) contributions are collected from each of the first $n-1$ group members. Then, in the n-th round, the accumulated keying material is broadcast by M_n. Next, each member factors out its contribution and returns the result to M_n. In the last round, M_n contributes its own share to each partial key received in the previous round and broadcasts the key-set to the group. The actual protocol is as follows:

Initialization:

Let p be a prime and q a prime divisor of $p-1$. Let G be the unique cyclic subgroup of $\mathbb{Z}_p^*$ of order q, and let α be a generator of G.

Round n:

1. $M_n \longrightarrow$ ALL M_i: $\alpha^{r_1 \cdots r_{n-1}}$
2. Each $M_i \longrightarrow M_n$: $\alpha^{\frac{r_1 \cdots r_{n-1}}{r_i}}$
3. M_n selects $r_n \in_R \mathbb{Z}_q^*$.
4. $M_n \longrightarrow$ ALL M_i: $\{\alpha^{\frac{r_1 \cdots r_n}{r_i}} | i \in [1, n[\}$

Figure 9: Cliques Group Diffie-Hellman Protocol (IKA.2)

The IKA.2 group key agreement protocol operates in $k+2$ rounds and requires $2(n+k-1)$ unicast messages as well as two broadcasts. However, $n+k-1$ of these are actually contained within a single round. The first $(k-1)$ rounds are similar to those in $IKA.1$ (depicted in Fig 4) except that each M_i only performs a single exponentiation. This is followed by M_{n+k} broadcasting the partial keys. Then, each M_i performs one exponentiation (to factor out its share) and unicasts the result back to M_{n+k}. M_{n+k} then performs $n+k-1$ exponentiations to add its share and broadcasts the resulting set. Finally, as in IKA.1, each M_i performs a single exponentiation and obtains a group key. The total protocol delay can be expressed as:

$$COST(IKA.2) = (2E + D)\,k \; + (En - E + 3D)$$